Redbooks
ibm.com/redbooks

Implementing an IBM High-Performance Computing Solution on IBM Power System S822LC

Dino Quintero

Luis Carlos Cruz Huertas

Tsuyoshi Kamenoue

Wainer dos Santos Moschetta

Mauricio Faria de Oliveira

Georgy E Pavlov

Alexander Pozdneev

Power Systems

IBM

Redbooks

International Technical Support Organization

Implementing an IBM High-Performance Computing Solution on IBM Power System S822LC

July 2016

SG24-8280-00

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

First Edition (July 2016)

This edition applies to the following products:
- Red Hat Enterprise Linux (RHEL) Server 7.2 (little-endian)
- Linux kernel version 3.10.0-327
- Extreme Cluster/Cloud Administration Toolkit (xCAT) 2.11
- Compute Unified Device Architecture (CUDA) Toolkit 7.5 (7.5-23)
- Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux 3.2 (3.2-1.0.1.1)
- XL C/C++ Compiler for Linux V13.1.2
- XL Fortran Compiler for Linux V15.1.2
- Advance Toolchain 8.0 (8.0-5)
- GNU Compiler Collection (GCC) 4.8.5 (RHEL)
- IBM Parallel Environment Runtime Edition (PE RTE) 2.3
- IBM Parallel Environment Developer Edition (PE DE) 2.2
- IBM Engineering and Scientific Subroutine Library (ESSL) 5.4
- IBM Parallel ESSL (PESSL) 5.2
- IBM Spectrum Scale (formerly IBM GPFS) 4.1.1.3
- IBM Spectrum LSF (formerly IBM Platform LSF) 9.1.3
- OpenPower Abstraction Layer (OPAL) firmware OP810.10 (OP8_v1.7_1.13)
- NAS Parallel Benchmarks version 3.3.1

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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 grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

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 jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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 actual people or business enterprises 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. The sample programs are provided “AS IS”, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

Find and read thousands of IBM Redbooks publications

- Search, bookmark, save and organize favorites
- Get personalized notifications of new content
- Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App

iOS **Download Now** Android

Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!

ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication demonstrates and documents that IBM Power Systems™ high-performance computing and technical computing solutions deliver faster time to value with powerful solutions. Configurable into highly scalable Linux clusters, Power Systems offer extreme performance for demanding workloads such as genomics, finance, computational chemistry, oil and gas exploration, and high-performance data analytics.

This book delivers a high-performance computing solution implemented on the IBM Power System S822LC. The solution delivers high application performance and throughput based on its built-for-big-data architecture that incorporates IBM POWER8® processors, tightly coupled Field Programmable Gate Arrays (FPGAs) and accelerators, and faster I/O by using Coherent Accelerator Processor Interface (CAPI). This solution is ideal for clients that need more processing power while simultaneously increasing workload density and reducing datacenter floor space requirements. The Power S822LC offers a modular design to scale from a single rack to hundreds, simplicity of ordering, and a strong innovation roadmap for graphics processing units (GPUs).

This publication is targeted toward technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) responsible for delivering cost effective high-performance computing (HPC) solutions that help uncover insights from their data so they can optimize business results, product development, and scientific discoveries.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is a Complex Solutions Project Leader and an IBM Level 3 Certified Senior IT Specialist with the ITSO in Poughkeepsie, New York. His areas of knowledge include enterprise continuous availability, enterprise systems management, system virtualization, technical computing, and clustering solutions. He is an Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Luis Carlos Cruz Huertas is an Executive Technology Architect with a specialization in transition and transformation solution architectecture with IBM GTS Delivery. During his over three years with IBM, he has performed research in BigData analytics, mobility, and cloud, also has held several positions with Midrange and Storage Technical Solution Architecture. Before Luis came to IBM, he worked at GBM, a strategic IBM Alliance company in Latin America where he held positions in strategy, IBM Tivoli® Architecture, and project management. He primarily workied with Tivoli Service Management capabilities, management systems, data warehouse infrastructure, information integration, database administration, performance management, and database development technology. He has been a prominent speaker at industry events such as IBM Edge, Interconnect, and customer briefings and a frequent contributor to industry articles, analyst research, and other publications.

Tsuyoshi Kamenoue is a Senior IT specialist in IBM Power Systems Technical Sales in IBM Japan. He has over 10 years of experience of working on pSeries, System p, and Power Systems products. He has led numerous HPC business opportunities in Japan with his

technical expertise of this area. He also participated in the development of IBM publications about the Power Systems 775 cluster solution. He holds a Bachelor's degree in System Information from the University of Tokyo.

Wainer dos Santos Moschetta is a Staff Software Engineer in the IBM Linux Technology Center, Brazil. He initiated and formerly led the IBM Software Development Kit (SDK) project for the IBM PowerLinuxTM project. He has more than seven years of experience with designing and implementing software development tools for Linux on IBM Power Systems. Wainer holds a Bachelor's degree in Computer Science from the University of São Paulo. He co-authored the IBM publications, *IBM Parallel Environment (PE) Developer Edition*, SG24-8075, *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*, SG24-8171, and `Implementing an IBM High-Performance Computing Solution on IBM POWER8`, SG24-8264. He has published articles and videos for the IBM developerWorks® website, and contributes to the IBM Linux on Power technical community blog.

Mauricio Faria de Oliveira is an Advisory Software Engineer at the Linux Technology Center at IBM Brazil. His areas of expertise include Linux performance analysis and optimization, Debian, and Ubuntu for IBM PowerPC® 64-bit Little-Endian, and Multipath I/O on IBM Power and OpenPower Systems. He also worked with official benchmark publications for Linux on IBM Power Systems and early development (bootstrap) of Debian on PowerPC 64-bit Little-Endian. Mauricio holds a Master of Computer Science and Technology degree and a Bachelor of Engineering degree in Computer Engineering from Federal University of Itajuba, Brazil.

Georgy E Pavlov is a Staff Software Engineer in ESSL development team at Science and Technology center in IBM Russia. He has about 5 years of experience in development and optimization of mathematical code for IBM Power Systems. He holds a Master's degree in Computer Science and Mathematics from Lomonosov Moscow State University.

Alexander Pozdneev is a Research Software Engineer at IBM Science and Technology Center, Moscow, Russia. He has 12 years of experience in HPC. He holds a Ph.D. degree in Mathematical Modeling, Numerical Methods, and Software from Lomonosov Moscow State University. His areas of expertise include parallel computing and application performance optimization.

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

Ella Bushlovic
Richard Conway
International Technical Support Organization, Poughkeepsie Center

John Dunham
Victor Hu
John Lemek
Serban Maerean
Joan McComb
Mark Perez
Mike Schiffer
Bob Sciortino
Donna Upright
Duane Witherspoon
IBM Poughkeepsie

James Woodbury
IBM Rochester

Sameh S Sharkawi
IBM Austin

Wei Li
IBM Canada

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

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

1

Introduction to the IBM Power System S822LC for high performance computing workloads

Built on industry standards and incorporating community innovation from the OpenPOWER Foundation, the IBM Power System S822LC delivers high application performance and throughput based on its built-for-big-data architecture. This architecture incorporates POWER8 processors, tightly coupled Field Programmable Gate Array (FPGA) and accelerators, and faster input/output (I/O) by using the Coherent Accelerator Processor Interface (CAPI).

The Power System S822LC is ideal for clients that need more processing power while simultaneously increasing workload density and reducing data center floor space requirements. It offers a modular design to scale from a single rack to hundreds, simplicity of ordering, and a strong innovation road map for graphics processing units (GPUs).

This chapter provides an overview of the IBM POWER8 technology, OpenPOWER Foundation, and Power System S822LC as the solution targeted for the next generation of high-performance computing (HPC) workloads.

The following sections are presented in this chapter:

- IBM POWER8 technology
- OpenPOWER
- IBM Power System S822LC

1.1 IBM POWER8 technology

It is no secret that disruptive trends in technology are rapidly changing how organizations do business. Technology is advancing so rapidly, in fact, that dynamic communities of collaboration are forming just to harness it all. The growing torrent of data from within and outside your organization, mobile employees, customers, and prospects present an unprecedented opportunity to gain valuable insights and apply these insights to improve your business results.

Making the transition to advanced capabilities requires an integrated infrastructure that supports your key IT initiatives. IBM investments to bring new optimized solutions in the area of advanced analytics, cloud, and mobile access are designed to simplify and accelerate your journey to address today's market opportunities.

The next generation of IBM Power Systems, with IBM POWER8 technology, is the first family of systems built with innovations that transform the power of big data and analytics, mobile, and cloud into competitive advantages in ways never before possible. New scale-out systems offered by IBM provide a powerful, scalable, and economical means of putting data to work for you.

IBM Power Systems are designed for big data and deliver the performance and throughput of POWER8 combined with the cost optimization of industry standardization, all without the wait.

Businesses are collecting a wealth of data and IBM Power Systems solutions can help store it, help secure it, and most importantly extract actionable insight from it in a short time frame. Power Systems and POWER8 processor, in particular, are designed for big data. From predictive analytics and data warehouses to unstructured big data processing and cognitive IBM Watson™ solutions, Power Systems servers are optimized for the compute-intensive performance demands of database and analytics applications, and can flexibly scale to support the demands of rapidly growing data.

1.2 OpenPOWER

The IBM Power System S822LC is manufactured as OpenPOWER system. The OpenPOWER Foundation is an open technical community based on the IBM POWER® architecture. It was incorporated in December 2013.

The main goal of the OpenPOWER Foundation is to create an open ecosystem to build customized servers, networking, and storage hardware for future data centers and cloud computing. IBM has opened the POWER architecture specifications such as processor, firmware, and software to its partners. To become a partner, a company needs to contribute intellectual property to the OpenPOWER Foundation and pay an annual fee.

There is some differentiation by members level:

1. The highest level of membership is the *Platinum* Level. IBM, NVIDIA, Mellanox, Google, Samsung, and Ubuntu are among platinum members.
2. The next level of membership is the *Gold* Level. The following companies are among the gold level members: Hitachi, Avnet, ZTE, and Wistron.
3. The other level of corporate membership is the *Silver* Level. Among others, silver level members are QLogic, Memblaze, IDT, and Asetek.
4. An additional level is not available for corporations and its members that are not paying an annual fee, called *Associate and Academic* Level. Many universities and laboratories have

these levels and participate in the OpenPOWER Foundation. Some examples include Oak Ridge National Laboratory, Louisiana State University, Bauman Moscow State Technical University, and FreeBSD project.

The word *open* within the name of the Foundation has the following meaning:

- IBM is *openly* sharing blueprints about software and hardware with their partners. After that the partners can hire IBM or other manufacturing companies to produce their own chips or processors.
- Members benefit from *open* licensing of processors.
- Members collaborate *openly* and share contributed intellectual properties and innovations between each other.

For more information about the OpenPOWER Foundation, see the following website:

http://openpowerfoundation.org

1.3 IBM Power System S822LC

The Power System S822LC computing server is designed to deliver superior performance and throughput for high-value Linux workloads such as industry applications, big data, and LAMP (Linux, Apache, MariaDB and PHP) workloads.

Power System S822LC is ideal for clients that need more processing power while simultaneously increasing workload density and reducing data center floor space. It offers a modular design to scale from single racks to hundreds, simplicity of ordering, and a strong innovation road map for GPUs.

Built on industry standards and incorporating community innovation from the OpenPOWER Foundation, the Power System S822LC computing server delivers higher application performance and throughput based on its built-for-big-data architecture. It incorporates POWER8 processors, tightly coupled FPGAs and accelerators, and faster I/O by using CAPI.

The Power System S822LC server is co-designed by OpenPOWER Foundation members IBM and Wistron Corporation, and it is commercialized in two different models:

- The Power System S822LC (8335-GCA) commercial computing server supports two POWER8 processor sockets offering 16-core 3.32 GHz or 20-core 2.92 GHz configurations in a 19-inch rack-mount, 2U (EIA units) drawer configuration. All the cores are activated.
- The Power System S822LC (8335-GTA) technical computing server supports two POWER8 processor sockets offering 16-core 3.32 GHz or 20-core 2.92 GHz configurations in a 19-inch rack-mount, 2U (EIA units) drawer configuration. All the cores are activated. It includes two NVIDIA K80 GPUs.

Figure 1-1 shows the front, rear, and top view of a Power System S822LC server.

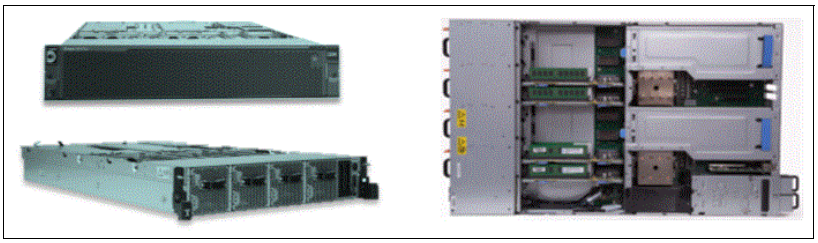

Figure 1-1 IBM Power System S822LC server

1.3.1 Differences between 8335-GCA and 8335-GTA models

The following are the detailed differences between the 8335-GCA and the 8335-GTA:

- Power System S822LC server model 8355-GCA features

  This summary describes the standard features of the Power System S822LC model 8355-GCA:

  - Rack-mount (2U) chassis
  - Two processor modules:
    - 8-core 3.32 GHz processor module
    - 10-core 2.92 GHz processor module
  - Up to 1024 GB of 1333 MHz DDR3 error-correcting code (ECC) memory
  - Two small form-factor (SFF) bays for two hard disk drives (HDDs) or two solid-state drives (SSDs)
  - Integrated SATA controller
  - Five PCIe Gen 3 slots:
    - One PCIe x8 Gen3 Low Profile slot, CAPI enabled
    - One PCIe x16 Gen3 Low Profile slot, CAPI enabled
    - One PCIe x8 Gen3 Low Profile slot
    - Two PCIe x16 Gen3, CAPI enabled and supports GPUs or PCIe adapters
  - Integrated features:
    - IBM EnergyScale™ technology
    - Hot-swap and redundant cooling
    - One front USB 2.0 port for general usage
    - One rear USB 3.0 port for general usage
    - One system port with RJ45 connector
  - Two hot-plug, redundant power supplies

- Power System S822LC server model 8355-GTA features

  This summary describes the standard features of the Power System S822LC model 8355-GTA:

  - Rack-mount (2U) chassis
  - Two POWER8 processor modules:
    - 8-core 3.32 GHz processor module
    - 10-core 2.92 GHz processor module
  - Up to 1024 GB of 1333 MHz DDR3 ECC memory

- Two SFF bays for two HDDs or two SSDs
- Integrated SATA controller
- Five PCIe Gen 3 slots:
  - One PCIe x8 Gen3 Low Profile slot, CAPI enabled
  - One PCIe x16 Gen3 Low Profile slot, CAPI enabled
  - One PCIe x8 Gen3 Low Profile slot
  - Two PCIe x16 Gen3, CAPI enabled and dedicated to NVIDIA K80 GPU
- Two Compute Intensive Accelerator GPU K80
- Integrated features:
  - EnergyScale technology
  - Hot-swap and redundant cooling
  - One front USB 2.0 port for general usage
  - One rear USB 3.0 port for general usage
  - One system port with RJ45 connector
- Two power supplies

These servers support NVIDIA GPU accelerators, which are optional for model 8335-GCA and included for model 8355-GTA. This book focuses on Model 8355-GTA.

For more information, see the Model 8355-GTA page in the IBM Knowledge Center, with the following navigation:

IBM Knowledge Center → Power Systems → POWER8 → 8335-GTA (Power System S822LC)

http://www.ibm.com/support/knowledgecenter/HW4M4/p8hdx/8335_gta_landing.htm

2

Reference architecture

This chapter provides information about the reference architecture of a high-performance computing (HPC) solution. The architecture addresses the most common cases. This reference architecture can serve as a basis for solutions targeted to more specific usage scenarios.

This chapter covers the following topics:

- Hardware components of an HPC system

  This section describes a high-level logical structure of the hardware components of an HPC system.

- Software components of an HPC system

  This section focuses on a high-level logical structure of an HPC system . It also explains the layout of the HPC cluster software components on the hardware.

- HPC system solution

  This section provides the names of specific hardware offerings.

This chapter includes the following sections:

- Hardware components of an HPC system
- Software components of an HPC system
- HPC system solution

2.1 Hardware components of an HPC system

An HPC system is a computer system that includes multiple servers and is able to execute parallel programs on these machines. In this context, a *parallel program* is a piece of software that is specifically designed to take advantage of running simultaneously on multiple servers. For more details about the nature of parallel programs, see 6.5, “Development models” on page 180.

An HPC system typically consists of the following server components[1]:

- Login nodes
- Management nodes
- Compute nodes
- Parallel file system

Server components of an HPC system are coupled together with the following networks:

- High performance interconnect
- Management network

Note: A storage area network (SAN) is not an intrinsic component of an HPC system. Typically, the SAN is hidden at the level of the parallel file system.

A simplified logical overview of an HPC system is presented in Figure 2-1.

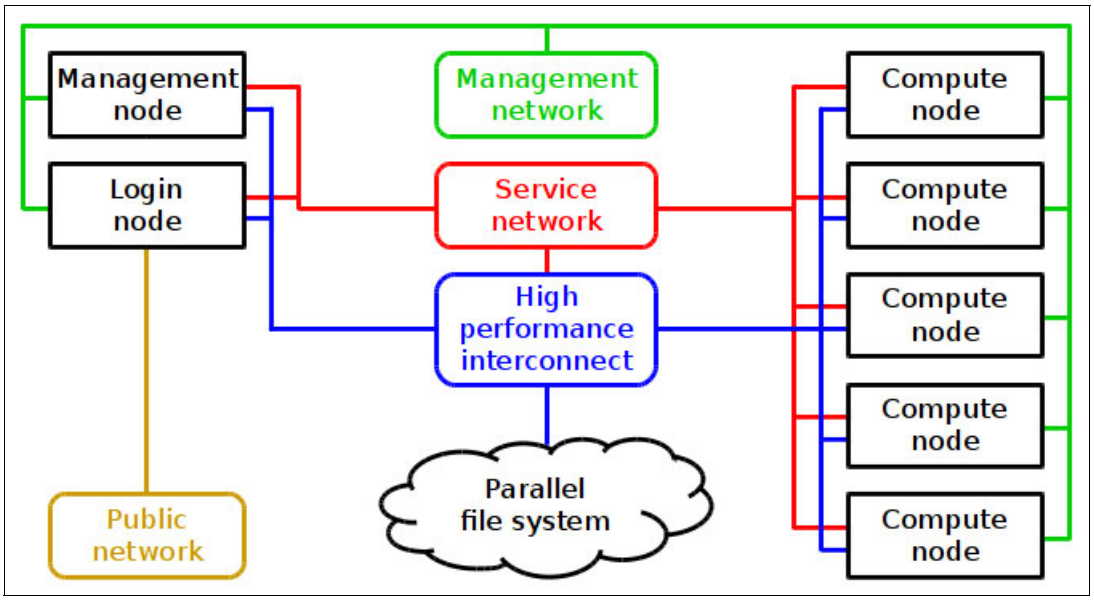

Figure 2-1 A simplified view of an HPC system

The following sections describe each of these components in detail. For examples of specific hardware offerings, see Chapter 3, “Hardware components” on page 23.

[1] In IBM Blue™ Gene solution, *login nodes* were known as *frontend nodes*, and *management nodes* were known as *service nodes.*

2.1.1 Login nodes

The login node is a point of entry for users of an HPC system. Login nodes are typically made accessible from external networks (see Figure 2-1 on page 8). However, for security reasons, login nodes are typically hidden behind a firewall. Sometimes login nodes are made available only through virtual private network (VPN) connection. Other components of an HPC system are typically made inaccessible from external networks.

The login node is the only component of an HPC system that is directly accessible by a user. Only system administrators have direct access to other components of an HPC system.

The file system that contains user data is typically kept physically separated from the login node hardware. Usually, this file system is mounted to the login node as the `/home` directory.

A user interacts with a login node according to the following typical scenario:

1. Copies data from some other system (local workstation, another remote system) to an HPC system.
2. Logs in to the HPC system.
3. Works with the HPC system in an interactive mode:
   - Edit source code files
   - Build, debug, and profile applications
   - Prepare input data for applications
   - Submit computing jobs (see “Workload management software” on page 16)
   - Postprocess results of computations of previously completed jobs
4. Logs out of the HPC system.
5. Copies data from the HPC system to some other system (local workstation, another remote system).

2.1.2 Management nodes

Often, application users and application developers are not aware of the existence of management nodes. These nodes are for use by system administrators only (see Figure 2-1 on page 8). The following system software components are typically on management nodes:

- System management software (see “System management software” on page 14)
- Workload management software (see “Workload management software” on page 16)

Thus, management nodes are mainly used for the following purposes:

- Deployment of compute nodes
- Managing resources and scheduling jobs

2.1.3 Compute nodes

Compute nodes constitute most of the server components of an HPC system (see Figure 2-1 on page 8). The purpose of compute nodes is to run parallel compute intensive user tasks. Typically, all compute nodes of an HPC system have identical hardware and software configuration. This configuration is used to ensure that the execution of a computation with some specific data takes the same amount of time regardless of a compute node that it is executed on.

Usually, compute nodes are not directly accessible by a user, and the user puts the compute tasks for execution through a job scheduler (see “Workload management software” on page 16).

Compute node of an HPC system typically has the following features:

- A processor is installed in each socket of a server.
- All memory slots of a server are populated with memory modules.
- The server has at least one compute accelerator (for example, graphics processing unit (GPU)).
- The server has a high-performance network adapter (see 2.1.4, “High performance interconnect” on page 10).

Note: If you need to choose between a hardware configuration with higher *memory bandwidth* and a server option with higher *volume of memory,* the choice is typically made in favor of higher *memory bandwidth.*

2.1.4 High performance interconnect

Technical computing workloads that involve interprocess communication are usually characterized by a large volume of data that is transferred between processes. A delay between the request for data transfer and the actual data transfer affects the performance of applications that frequently send and receive small chunks of data. This delay means that the network interconnect between compute nodes of an HPC system needs to have the following features:

- High bandwidth
- Low latency

Modern high performance interconnects typically implement the Remote Direct Memory Access (RDMA) feature. RDMA helps to minimize the processor overhead by allowing remote processor to directly access system memory with no operating system (OS) involvement.

In addition to connecting compute nodes with each other, the high performance interconnect also provides access to the parallel file system (see Figure 2-1 on page 8 and 2.1.6, “Parallel file system” on page 12). This configuration allows high throughput operations on files.

High performance interconnect includes the following hardware components:

- Host channel adapters (HCA)
- Network switches
- Cables

The high performance interconnect network is also known as *application network* because application processes that run on compute nodes use this network to communicate with each other.

2.1.5 Management, service, and site (public) networks

The following networks are used for hardware control, OS deployment, and system management:

- Service network
- Management network

In addition to these networks, the site (public) network is used to provide external access to the servers (see Figure 2-1 on page 8).

The hardware infrastructure for these networks includes the following components:

- Local network area (LAN) adapters built-in into servers
- LAN adapters attached to PCI slots
- Network switches
- Cables

Service network

The service network provides access to the service processors of the hardware (see Figure 2-1 on page 8). The management node uses this network to control the network attached devices in an *out-of-band* manner. For example, the IBM Power System S822LC server can be controlled through a baseboard management controller (BMC). The service network allows you to perform the following actions remotely:

- Access to firmware
- Boot process troubleshooting
- Installation of operating system

Management network

The management network is used by the management node to support all management activities that do not involve service processors of the hardware (see Figure 2-1 on page 8). This network has network interface controllers (NICs) as endpoints of the management node, login node, and compute nodes. The management node uses this network to control the network attached devices in an *in-band* manner. For example, servers can be managed through Secure Shell (SSH) or Virtual Network Computing (VNC). This configuration means that the remote end first needs connectivity software installed.

The management network allows management node (see 2.1.2, “Management nodes” on page 9) to perform the following activities:

- Deployment of compute nodes
  - Installation of the OS to the nodes
  - Managing the OS of the nodes
  - Installation and configuration of drivers and applications
- Managing resources and scheduling jobs

The following network services are usually set up in the management network[2]:

- Domain Name Servers (DNS)
- Hypertext Transfer Protocol (HTTP)
- Dynamic Host Configuration Protocol (DHCP)
- Trivial File Transfer Protocol (TFTP)
- Network File System (NFS)
- Network Time Protocol (NTP)

Networking technology considerations

The most common type of interconnect for management and service networks is the Gigabit Ethernet (1 GigE). This choice is dictated by the type of network supported by managed devices. For instance, the typical network interface that is provided by a service processor of a server is 1 GigE.

[2] This list originates from the list of network services that are needed by xCAT.

If you plan large data transfer between an HPC system and external world, you can consider a 10 Gigabit Ethernet (10 GigE) option for the site (public) network. However, 1 GigE is usually enough for the site (public) network too.

Security considerations

Typically, the access to an HPC system is guarded externally by the following security technologies:

- Firewalls
- VPNs

Generally, keep the login node (see 2.1.1, “Login nodes” on page 9) as a single point of entry to an HPC system. In this scenario, all the networks (application, management, service) are configured only within an HPC system and are not visible from the outside. Particularly, the management node (see 2.1.2, “Management nodes” on page 9) becomes accessible only through a login node.

One option to isolate networks is to use dedicated network switches and dedicated network interfaces for each network. However, a server built-in network port is often shared between the service processor network interface and the in-band server network interface[3]. This configuration means that the separation of networks can be achieved within one network switch by the use of virtual local area network (VLAN) technology.

2.1.6 Parallel file system

A shared file system accessible from the compute nodes and from a login node is an essential component of an HPC system (see Figure 2-1 on page 8). It is challenging to access the data that needs to be available for every node if there is no shared file system within an HPC system. For more information about the implications, see “Distributed execution environment” on page 15.

Technical computing workloads often require multiple processes of a distributed application to operate simultaneously on the same file. The parallel file system middleware hides the complexity of such operations and implements it in a performance efficient way.

When multiple processes access a file system at the same time, the file system performance can become a bottleneck. Parallel file systems are designed in a such way to take advantage of distributed storage servers and a high performance interconnect. Therefore, parallel file systems help to minimize the performance implications of parallel input/output operations.

To summarize, a parallel file system is simultaneously mounted on multiple nodes and provides the following features for an HPC system:

- Shared file system with common space of file names
- Simultaneous access to a file from different processes
- High bandwidth of input/output operations

Most portions of the following software components can be located in a parallel file system:

- Distributed execution environment (see “Distributed execution environment” on page 15)
- Application development software (see 2.2.2, “Application development software” on page 17)
- Application software (see 2.2.3, “Application software” on page 20)

[3] Power S822LC servers have service processor network interface and in-band server network interface separated.

2.2 Software components of an HPC system

There are three main groups of software components of an HPC system solution:

- System software
- Application development software
- Application software

Figure 2-2 provides a simple extension of Figure 2-1 on page 8 with the mapping of software components to hardware.

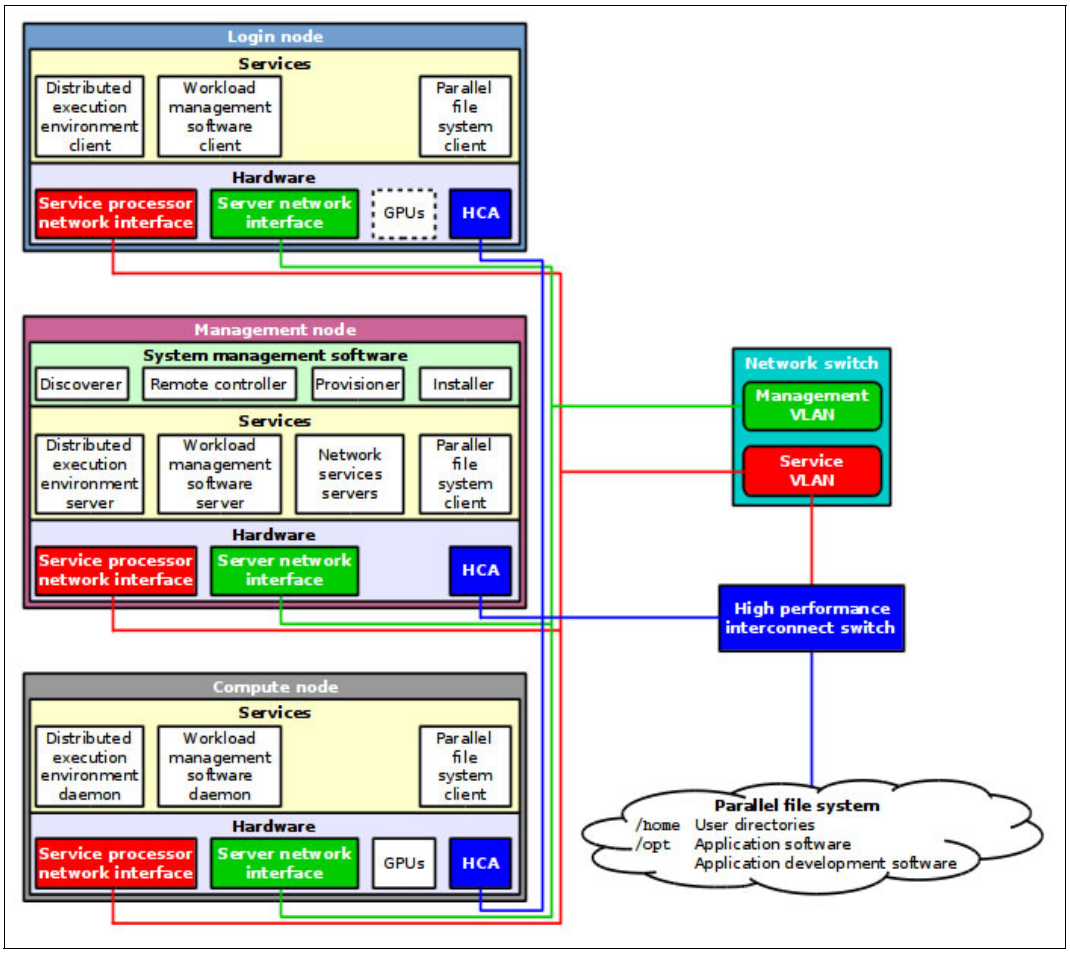

Figure 2-2 Software components of an HPC system mapped to the hardware

The following sections describe each of these components in detail. For examples of specific software offerings, see Chapter 4, “Software stack” on page 43.

2.2.1 System software

The system software lies at the lowest level of software stack. It consists of the components that are responsible for the system deployment, basic system functionality, operation of the high performance computing pieces of hardware, access to parallel file system, execution environment that supports distributed programs, workload management, and system monitoring.

System management software

The system management software (see “Management node” in Figure 2-2 on page 13) is a cornerstone of the system software stack. It helps to automate the process of deployment and maintenance of an HPC system. Usually, this is the first piece of software that is deployed when installing an HPC system. All the other software components are installed at a later stage.

> **Note:** The choice of system management software product is an important architectural decision. After chosen and implemented, it can be quite time consuming to switch to another product.

The following are typical tasks performed with the system management software[4]:

- Discovery of the hardware servers
- Remote system management against the discovered server:
  - Remote power control
  - Remote console support
  - Remote inventory information query
- Provisioning OS on physical (bare-metal) or virtual machines
- Installation and configuration of software
  - During OS installation
  - After the OS installation
- System management in a parallel manner
  - Parallel shell (that is, running shell command against nodes in parallel)
  - Parallel copy

That means that the system management software is crucial for the automation of the following routine tasks:

- Provisioning and deployment of multiple identical compute servers
- Maintenance of multiple identical system images:
  - Applying OS updates
  - System-wide software settings change
- Replacement of a failing node with a new one

The following are typical examples of system management software:

- xCAT (Extreme Cloud/Cluster Administration Toolkit)

  http://xcat.org

  Chapter 5, “Software deployment” on page 53 shows how to use xCAT as a tool for an HPC system deployment.

- IBM Platform Cluster Manager

  http://www.ibm.com/support/knowledgecenter/SSDV85/product_welcome_pcm.html

For the overview of the IBM Spectrum™ Computing products family, visit the following link:

http://www.ibm.com/systems/spectrum-computing/

Operating system

The OS of choice in an HPC cluster is typically Linux. Also, typically all servers of an HPC system run the same Linux version. In rare cases, some components of an HPC system are required to run a specific version of Linux.

[4] This list of tasks partially originates from the list of xCAT features, but does not cover all of them.

The HPC solution described in this book is based on Red Hat Enterprise Linux (RHEL) Server OS. For information about how to install RHEL with xCAT, see 5.6.2, “RHEL Server” on page 111.

Device drivers

Most of the hardware components of a server commonly have built-in support in an OS and do not require special handling. However, the device drivers that operate the high performance computing pieces of hardware are not typically shipped with the OS and must be installed separately.

A modern HPC solution usually needs drivers for the following devices:

- Hardware accelerator, such as a GPU.

  For example, drivers for the NVIDIA Tesla GPU of the IBM Power System S822LC server are installed as part of NVIDIA CUDA Toolkit (see 5.6.3, “CUDA Toolkit” on page 113).

- High performance interconnect host channel adapter that supports RDMA technology.

  5.6.4, “Mellanox OFED for Linux” on page 117 demonstrates how to install the Mellanox Open Fabrics Enterprise Distribution (OFED) package that enables the Mellanox InfiniBand adapter.

Parallel file system

Unlike compute accelerators and network adapters, a parallel file system is not a piece of hardware that is directly installed in a machine. You can think of a parallel file system as a software service that is external to the machine (see Figure 2-2 on page 13). Typically, the interaction between machines and a parallel file system is organized in a client-server manner:

- A parallel file system exports its services by running server software components
- The machines that need access to a parallel file system run a client software component

Therefore, parallel file system client software needs to be installed and configured on all machines that use a parallel file system. A parallel file system client software helps an operating system to make a parallel file system available for a user by mounting it to a directory tree. As a result, from the user perspective, the interaction with the parallel file system does not differ from the interaction with any other file system mounted to a machine.

This book shows how to couple an HPC system with a parallel file system: IBM Spectrum Scale™ product. IBM Spectrum Scale is a proven, scalable, high-performance data and file management solution. IBM Spectrum Scale is based on the IBM General Parallel File System (GPFS™) technology. For the technical details about the deployment, see 5.6.12, “Spectrum Scale (formerly GPFS)” on page 129.

Distributed execution environment

The need for a distributed execution environment emerges when an application developer wants to use multiple compute nodes within a single application. HPC systems are used exactly for this purpose (running parallel programs). Therefore, multiple frameworks are available that facilitate the development and execution of this sort of computer codes.

The following sections elaborate on the view of a distributed execution environment purely from a perspective of application development (see “Message passing interface” on page 18). This section outlines how the distributed execution environment removes the burden of distributed application loading and execution from application users and application developers.

Typically, a distributed execution environment (see Figure 2-2 on page 13) facilitates the following routine tasks related to the execution of parallel applications:

- Runs an executable file on specific multiple compute nodes simultaneously
- Monitors the runtime status of a parallel program
- Terminates a parallel program and cleans up compute nodes
- Exports the OS environment variables to compute nodes before executing a program
- Manages standard input, output, and error streams (`stdin`, `stdout`, and `stderr`)
- Controls the binding and affinity of parallel processes and threads to logical processors
- Selects a type of interconnect to use and tunes its parameters
- Provides distributed debugging tools
- Interacts with workload management software (see “Workload management software” on page 16)

Additionally, if a shared file system is not available in an HPC system, some distributed execution environments provide help with the following routine file operations:

- Copy a specified executable to compute nodes before starting remote processes and delete it upon completion of a job
- Preinstall files to compute nodes where processes will be executed just before launching those processes

The following are examples of distributed execution environments:

- IBM Parallel Environment Runtime Edition (for more information, see “PE RTE” on page 122)
- Open the message passing interface (MPI) project

Workload management software

Usually, an HPC system is shared by many users and multiple computing tasks running at the same time. A workload management software automates the task of handling system resources and user jobs in such circumstances.

The following case provides an insight into a typical scenario where a workload management software becomes useful:

- An HPC system has limited number of compute nodes
- Application user needs to run parallel applications that require several compute nodes
- User has several computing jobs to be executed
- Many users work with an HPC system at the same time

Workload management software (see Figure 2-2 on page 13) utilizes the following two logical components to cope with that scenario:

- *Resource manager* monitors and controls compute nodes. A resource manager is aware of compute nodes that are idle or busy with user applications.
- *Job scheduler* maintains a queue of tasks from application users. Job scheduler uses information from resource manager to schedule the tasks for execution.

The interaction between application user, workload management software, and its components is based on the following scenario:

1. User submits a task to a job of the scheduler. The minimal specification of a task typically includes the following information:
   - Application name (path to an executable file)
   - Number of compute nodes to be utilized
   - Maximum time that is expected to be taken by a program to run
2. Job scheduler places the task into a job queue.
3. Workload management software schedules computing resources for the task and arranges a time slot for the execution.
4. At some moment in time, the workload management software sends the task for execution.
5. When the task completes, it is removed from the job queue, and the user can collect the results.

The job scheduler uses multiple criteria to arrange jobs. Modern workload management software provides flexible options for tuning the configuration of a job queue and a scheduler to adhere to local site policies.

Application users can also do the following actions with the workload management software:

- Request the status of a job queue
- Inquire the status of a particular job from a job queue
- Change the specification of a task submitted to a job queue
- Cancel a task submitted to a job queue

At the core, the workload management software provides the following basic advantages for application users, system administrators, and HPC systems owners:

- Automation of task management
- Better system utilization

This book focuses on the IBM Spectrum LSF® workload management software. The overview of this product is available at the following web page:

http://www.ibm.com/systems/spectrum-computing/products/lsf/

For the details of the IBM Spectrum LSF deployment, see 5.6.13, “IBM Spectrum LSF” on page 134.

2.2.2 Application development software

As its name implies, application development software (see Figure 2-2 on page 13) is used to develop software. However, application development software is needed by all categories of users:

- Application developers
- Application users
- System administrators

If application software or system software is distributed in source code package form, *application users* and *system administrators* utilize application development tools to create ready-to-use binary packages. Nevertheless, the main target audience of application development software is *application developers*.

This chapter only provides a brief overview of the application development software stack. You can find more details in the following sections.

Compilers

A compiler is a tool that converts source code into a binary executable. The most popular languages in the area of HPC are C, C++, and Fortran. Most major distributions of Linux provide compilers from these languages. System vendors provide state-of-the-art compilers that can take full advantage of the underlying hardware. The following C, C++, and Fortran compilers are relevant for HPC systems based on IBM POWER processors:

- GNU Compiler Collection (GCC)
- IBM Advance Toolchain for Linux on Power
- IBM XL compiler products
- NVIDIA compiler for GPU

For a detailed overview of these compilers, see 4.7, “IBM XL compilers, GCC, and Advance Toolchain” on page 46. 5.6.5, “XL C/C++ Compiler” on page 119, 5.6.6, “XL Fortran Compiler” on page 120, and 5.6.7, “Advance Toolchain” on page 121 show how to deploy compilers. Chapter 6, “Application development and tuning” on page 155 demonstrates how to utilize them for application development.

Parallel computing application interfaces

To take advantage of an HPC system, an application needs to be able to run in parallel mode. It is a responsibility of an application developer to write the program in such a way to make it possible to utilize the HPC system. If a program has not been designed to run in an HPC system, the program cannot be easily parallelized.

You can make a program run in parallel by using three different methods, although these methods can be intermixed with each other. For a deeper description, see 6.5, “Development models” on page 180.

OpenMP

OpenMP is perhaps the simplest way to enable parallelism in an application. The OpenMP standard defines a set of directives to be embedded into source code. However, the application developer still needs substantial efforts to identify parallelism in an application domain, and design algorithms and data structures to fit OpenMP parallel programming model.

OpenMP can be used to utilize the parallel capabilities of a single server. Parallel threads created by OpenMP require shared address space. Pool of OpenMP threads spawned by a process cannot span multiple compute nodes.

GCC and IBM compilers enable support of OpenMP through a compiler option.

NVIDIA CUDA

CUDA is a parallel computing platform and programming model by NVIDIA. CUDA is a way to utilize GPUs by NVIDIA. GPUs are especially efficient in solving data parallel problems. CUDA programs run within a single machine. A compiler that is a part of CUDA Toolkit is needed to produce binary files that utilize GPUs.

Message passing interface

MPI is probably the most widespread parallel programming interface to develop applications that span execution across multiple compute nodes.

Application written with MPI runs multiple threads on different compute nodes. Processes of an application coordinate their execution by sending messages to each other or by utilizing remote memory access techniques.

> **Note:** In contrast to problems that can be solved with widely accepted map-reduce type programming model, HPC problems typically are highly sensitive to the latency of individual operations. Ideally, processes of an HPC application need to run in sync with each other and communicate with minimal latency and maximum bandwidth. MPI facilitates the development in this programming model and also takes advantage of the underlying high performance interconnect.

6.5.1, “MPI programs with IBM Parallel Environment” on page 180 shows how to develop MPI programs with IBM Parallel Environment Runtime Edition.

Mathematical libraries

Software libraries of mathematical routines are essential part of an HPC system. A mathematical library is a software package that implements some numerical algorithms. Application developers access the algorithms through a programming interface exposed by a library. Hardware vendors often supply libraries optimized for a particular architecture. Many libraries also implement parallel algorithms.

By using mathematical libraries, application developer receives the following benefits:

- Saves time on implementing standard mathematical routines
- Takes advantage of optimized implementation supplied by a hardware vendor
- Utilizes parallelism hidden inside a library

This book focuses on the IBM Engineering and Scientific Subroutine Library (ESSL) offering. For more details, see the following sections:

- For an overview, see “IBM Engineering and Scientific Subroutine Library and Parallel ESSL” on page 49.
- For information about deployment, see “ESSL” on page 127 and “PESSL” on page 128.
- For details about usage, see “Engineering and Scientific Subroutine Library” on page 160 and “Parallel ESSL” on page 167.

Integrated development environments

Integrated development environment (IDE) provides a convenient graphical user interface for application developer. IDE typically features the following tools:

- Code editor with syntax highlighting and code completion
- Building tools
- Remote application launcher
- Debugger
- Code analyzer
- Profiler

For more details about IDE options, see 6.7, “Tools for development and tuning of applications” on page 199.

Debuggers

A code debugger is an application development tool that facilitates the process of eliminating programming errors from a source code. Parallel applications provide more challenges for debugging compared to serial applications. To learn about the tools for debugging MPI and CUDA programs, see 6.7, “Tools for development and tuning of applications” on page 199.

Performance analysis tools

Profilers, or performance analysis tools, automate the process of finding hotspots in a code. These tools help to evaluate the following metrics:

- Stalls of processor core units
- Effective memory bandwidth
- Cache hits and misses
- Graphical processing unit performance
- Network performance

The section 6.7, “Tools for development and tuning of applications” on page 199 provides a brief overview of performance analysis tools. For more information about performance optimization with the help of performance analysis tools, see the following publications:

- *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*, SG24-8171
- *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263

2.2.3 Application software

Running application software is essentially the ultimate purpose of an HPC system’s existence. Application software (see Figure 2-2 on page 13) is a tool that users employ to solve problems from application domains.

However, this book generally targets system administrators and application developers. So it does not cover application software in detail. Nevertheless, for the purpose of completeness, Appendix A, “Applications and performance” on page 263 provides some examples of application software.

2.3 HPC system solution

The previous sections presented a generic overview of the hardware and the software components of an HPC system. This section revisits the schemes presented in Figure 2-1 on page 8 and Figure 2-2 on page 13 and provides the specific names of the IBM products.

For more information about HPC system solution implementation, see Chapter 5, “Software deployment” on page 53.

2.3.1 Compute nodes

Generally, use the IBM Power System S822LC (model 8335-GTA) server offering for high-performance computing as compute nodes. Consider the option to augment the server with the following devices:

- Two NVIDIA Tesla K80 GPUs (see “NVIDIA GPU” on page 38)
- One 100Gb EDR InfiniBand Adapter (see “Mellanox InfiniBand” on page 40)

Processor options and system memory

This IBM Power System S822LC model has two sockets, and all memory slots are populated with memory modules. When choosing the processor option and the amount of system memory, take into account the anticipated workloads.

Disk features

Generally, a compute node does not need large and fast disks because it only stores the operating system, drivers, and minor pieces of system software (see Figure 2-2 on page 13). However, if you plan to extensively utilize local disk space during computations, consider the option of larger and faster disks.

2.3.2 Management node

A management node does not need GPUs and high memory bandwidth because it does not consume many processor cycles. Therefore, even an entry-level server option has enough resources to satisfy the needs of a management node. Consider the IBM Power System S812LC or the IBM Power System S812L offerings as a possible management node.

2.3.3 Login node

Generally, a login node is not used as a computing resource. Therefore, the most basic solution can be built based on the IBM Power System S812LC or the IBM Power System S812L offerings as well. However, if the GPU is required in a login node, consider using an IBM Power System S822LC or an IBM Power System S824L for it.

2.3.4 Combining the management and the login node

Consider the following scenario:

- No large workload expected on the management and login nodes
- Login node does not need GPU

In this case, consider using only one physical machine (IBM Power System S812LC or IBM Power System S812L). In such scenario, the management and login node can coexist as PowerKVM guests.

2.3.5 Parallel file system

To implement the parallel file system, consider one of the following options:

- IBM Elastic Storage™ Server

  http://www.ibm.com/systems/storage/spectrum/ess/index.html

- IBM Spectrum Scale (built on the IBM GPFS)

  http://www.ibm.com/systems/storage/spectrum/scale/

2.3.6 High performance interconnect switch

The implementation of the high performance interconnect is built around an InfiniBand switch. You can consider an offering from Mellanox. Check the hardware compatibility matrix for a suitable product.

3

Hardware components

This chapter describes the hardware components and related technologies adopted in the implementation of an IBM high-performance computing (HPC) solution in an IBM POWER8 cluster with the IBM Power System S822LC server.

The following topics are described in this chapter:

- IBM Power System S822LC
- Mellanox InfiniBand
- IBM System Storage

3.1 IBM Power System S822LC

This section describes the overall system architecture for the IBM Power System S822LC computing servers, focusing on the topics that are closely related to HPC.

The bandwidths that are provided throughout the section are theoretical maximums that are only used for reference. The speeds that are shown are at an individual component level. Multiple components and application implementation are key to achieving the best performance. Always do performance sizing at the application workload environment level and evaluate performance by using real-world performance measurements and production workloads. For more information about the IBM Power System S822LC, see *IBM Power System S822LC Technical Overview and Introduction*, REDP-5283-01.

3.1.1 IBM POWER8 processor

This section introduces the latest processor in the IBM Power Systems product family and describes its main characteristics and features in general.

Processor chip overview

The POWER8 processor is manufactured by using the IBM 22 nm Silicon-On-Insulator (SOI) technology. Each chip is 649 mm^2 and contains 4.2 billion transistors. As shown in Figure 3-1, the chip contains up to 12 cores[1], two memory controllers, Peripheral Component Interconnect Express (PCIe) Gen3 I/O controllers, and an interconnection system that connects all components within the chip. Each core has 512 KB of L2 cache, and all 12 cores share 96 MB of L3 embedded DRAM (eDRAM). The interconnect also extends through module and board technology to other POWER8 processors in addition to DDR3 memory and various I/O devices.

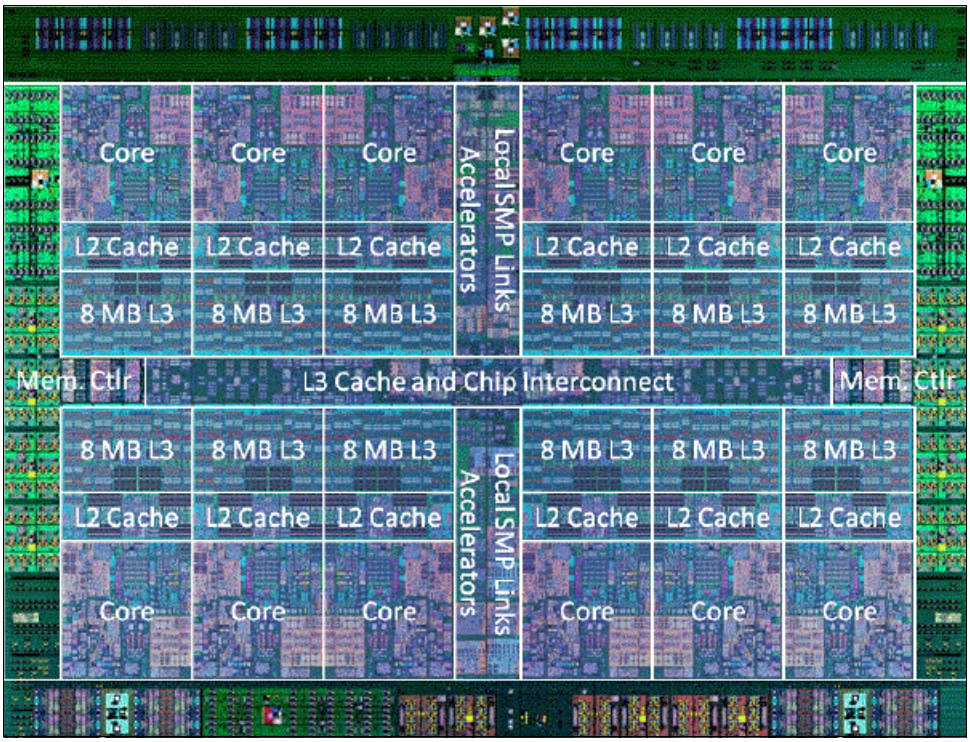

Figure 3-1 The POWER8 processor chip

[1] Power S822LC servers have 8 or 10 cores activated per processor.

POWER8 processor-based systems use memory buffer chips to interface between the POWER8 processor and DDR3 or DDR4 memory[2]. Each buffer chip also includes an L4 cache to reduce the latency of local memory accesses.

The POWER8 processor is for system offerings from single-socket servers to multi-socket enterprise servers. The processor incorporates a triple-scope broadcast coherence protocol over local and global Symmetric Multiprocessing (SMP) links to provide superior scaling attributes. Multiple-scope coherence protocols reduce the amount of SMP link bandwidth that is required by attempting operations on a limited scope (single chip or multi-chip group) when possible. If the operation cannot complete coherently, the operation is reissued by using a larger scope to complete the operation.

The following are additional features that can help augment the performance of the POWER8 processor:

- Support for DDR3 and DDR4 memory through memory buffer chips that offload the memory support from the POWER8 memory controller.
- An L4 cache within the memory buffer chip that reduces the memory latency for local access to memory behind the buffer chip. The operation of the L4 cache is not apparent to applications that run on the POWER8 processor. A full-featured POWER8 processor with four memory channels per memory controller can have 16 MB of L4 cache per memory channel and up to 128 MB of L4 cache in total[3].
- Hardware transactional memory.
- On-chip accelerators, including on-chip encryption, compression, and random number generation accelerators.
- Coherent Accelerator Processor Interface (CAPI), which allows accelerators plugged into a PCIe slot to access the processor bus by using a low latency, high-speed protocol interface.
- Adaptive power management.

Table 3-1 summarizes the technology characteristics of the POWER8 processor.

Table 3-1 Summary of POWER8 processor technology[4]

Technology	POWER8 processor
Die size	649 mm^2
Fabrication technology	► 22 nm lithography ► Copper interconnect ► SOI ► eDRAM
Maximum number of processor cores	12
Maximum number of execution threads per core/chip	8 / 96
Maximum volume of L2 cache per core/chip	512 KB / 6 MB
Maximum volume of on-chip L3 cache per core/chip	8 MB / 96 MB
Maximum volume of L4 cache per channel/chip	16 MB / 128 MB

[2] At the time of writing, the available POWER8 processor-based systems use DDR3 memory.
[3] Processors of Power S822LC servers have two memory channels activated per memory controller and up to 64 MB of L4 cache is available for each POWER8 processor.
[4] Information in the table is provided for a general full-featured 12-core POWER8 processor with four memory channels per memory controller. For Power S822LC servers, adjust for a 8- or 10-core POWER8 processor with two memory channels per memory controller.

Technology	POWER8 processor
Maximum number of memory controllers	2
SMP design-point	16 sockets with IBM POWER8 processors
Compatibility	With prior generation of POWER processors

Processor core overview

The POWER8 processor core is a 64-bit implementation of the IBM Power Instruction Set Architecture (ISA) Version 2.07 and has the following features:

- Multi-threaded design, capable of up to eight-way simultaneous multithreading (SMT)
- 32 KB, eight-way set-associative L1 instruction cache
- 64 KB, eight-way set-associative L1 data cache
- Enhanced prefetch, with instruction speculation awareness and data prefetch depth awareness
- Enhanced branch prediction, using both local and global prediction tables with a selector table to choose the best predictor
- Improved out-of-order execution
- Two symmetric fixed-point execution units
- Two symmetric load and store units and two load units, all four of which can also run simple fixed-point instructions
- Two integrated, multi-pipeline vector-scalar floating point units for running both scalar and SIMD-type instructions, including the Vector Multimedia Extension (VMX) instruction set and the improved Vector Scalar Extension (VSX) instruction set. Each is capable of up to eight single precision floating point operations per cycle (four double precision floating point operations per cycle)
- In-core Advanced Encryption Standard (AES) encryption capability
- Hardware data prefetching with 16 independent data streams and software control
- Hardware decimal floating point (DFP) capability

More information about Power ISA Version 2.07 can be found at the following websites:

http://www.power.org/wp-content/uploads/2013/05/PowerISA_V2.07_PUBLIC.pdf
http://www.power.org/documentation/power-isa-v-2-07b/

Figure 3-2 shows a picture of the POWER8 core with some of the functional units highlighted.

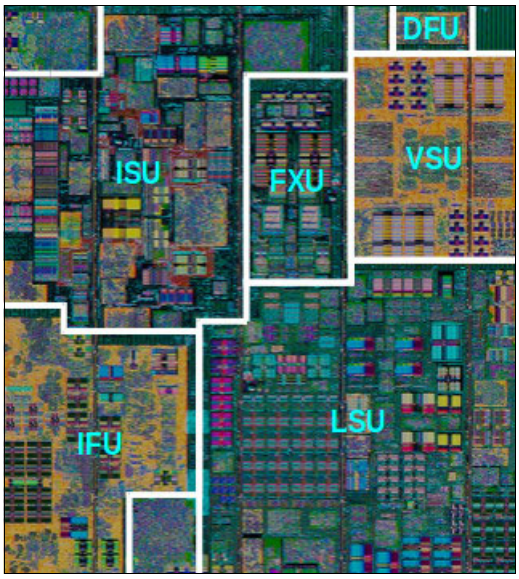

Figure 3-2 POWER8 processor core

Simultaneous multithreading

POWER8 processor advancements in multi-core and multi-thread scaling are remarkable. A significant performance opportunity comes from parallelizing workloads to enable the full potential of the microprocessor, and the large memory bandwidth. Application scaling is influenced by both multi-core and multi-thread technology.

SMT capability allows a single physical processor core to simultaneously dispatch instructions from more than one hardware thread context. With SMT, each POWER8 core can present eight hardware threads. Because there are multiple hardware threads per physical processor core, multiple instructions can run at the same time. SMT is primarily beneficial in commercial environments where the speed of an individual transaction is not as critical as the total number of transactions that are performed. SMT typically increases the throughput of workloads with large or frequently changing working sets, such as database servers and web servers.

Table 3-2 shows a comparison between the different POWER processors options for IBM Power S822LC and the number of threads that are supported by each SMT mode.

Table 3-2 SMT levels that are supported by a Power S822LC server

Cores per system	SMT mode	Hardware threads per system
16	Single Thread	16
16	SMT2	32
16	SMT4	64
16	SMT8	128
20	Single Thread	20
20	SMT2	40
20	SMT4	80
20	SMT8	160

The architecture of the POWER8 processor, with its larger caches, larger cache bandwidth, and faster memory, allows threads to have faster access to memory resources. This capability translates to a more efficient usage of threads. Therfore, POWER8 allows more threads per core to run concurrently, increasing the total throughput of the processor and the system.

Memory access

On the Power S822LC, each POWER8 module has two memory controllers, each connected to two memory channels. Each memory channel operates at 1600 MHz and connects to a memory riser card. Each memory riser card has a memory buffer that is responsible for many functions that were previously on the memory controller, such as scheduling logic and energy management. The memory buffer also has 16 MB of L4 cache. The memory riser card also houses four industry standard RDIMMs.

At the time of writing, each memory channel can address RDIMM memory modules of up to 32 GB. Therefore, the Power S822LC can address up to 1 TB of total memory.

Figure 3-3 shows a POWER8 processor connected to four memory riser cards and its components.

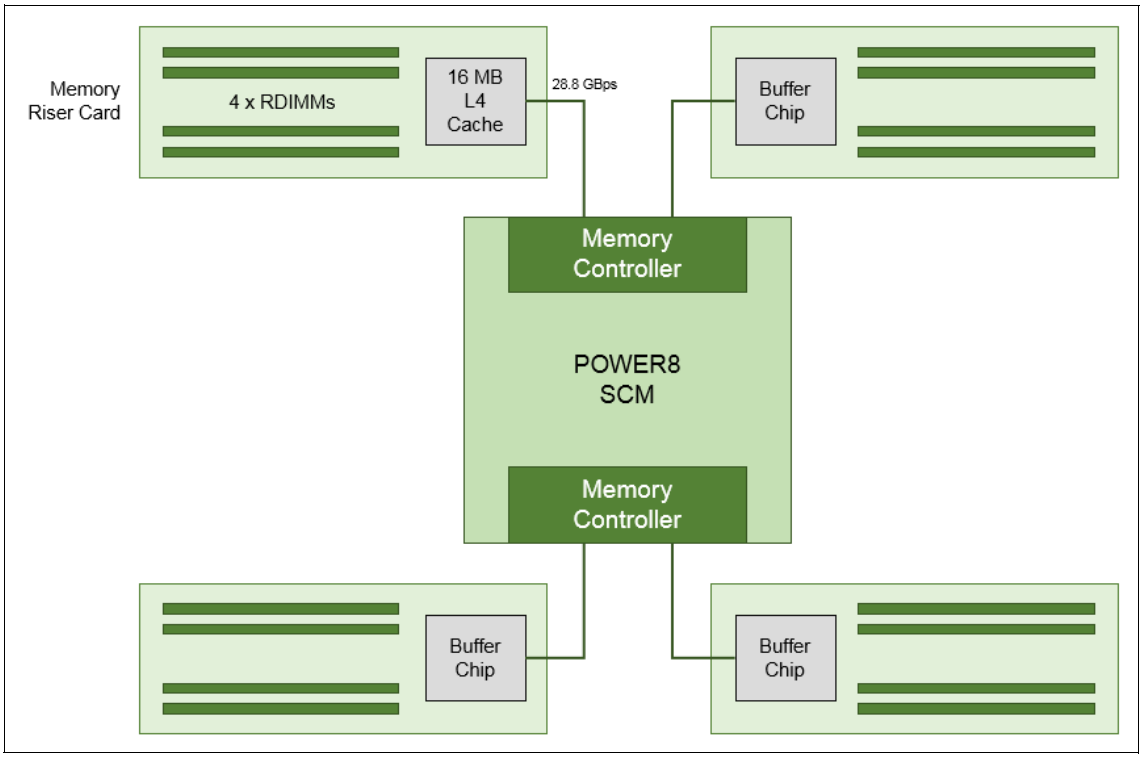

Figure 3-3 Logical diagram of a POWER8 processor connected to four memory riser cards

On-chip L3 cache innovation and Intelligent Cache

The POWER8 processor uses a breakthrough in material engineering and microprocessor fabrication to implement the L3 cache in eDRAM and place it on the processor die. L3 cache is critical to a balanced design, as is the ability to provide good signaling between the L3 cache and other elements of the hierarchy, such as the L2 cache or SMP interconnect.

The on-chip L3 cache is organized into separate areas with differing latency characteristics. Each processor core is associated with a fast 8 MB local region of L3 cache (FLR-L3), and also has access to other L3 cache regions as shared L3 cache. Additionally, each core can negotiate to use the FLR-L3 cache that is associated with another core, depending on

reference patterns. Data can also be cloned to be stored in more than one core's FLR-L3 cache, again depending on reference patterns. This Intelligent Cache management enables the POWER8 processor to optimize the access to L3 cache lines and minimize overall cache latencies.

Figure 3-1 on page 24 show the on-chip L3 cache, and highlights the fast 8 MB L3 region that is closest to a processor core.

The innovation of using eDRAM on the POWER8 processor die is significant for several reasons:

- Latency improvement

  A six-to-one latency improvement occurs by moving the L3 cache on-chip compared to L3 accesses on an external (on-ceramic) application-specific integrated circuit (ASIC).

- Bandwidth improvement

  A twofold bandwidth improvement occurs with on-chip interconnect. Frequency and bus sizes are increased to and from each core.

- No off-chip driver or receivers

  Removing drivers or receivers from the L3 access path lowers interface requirements, conserves energy, and lowers latency.

- Small physical footprint

  The performance of eDRAM when implemented on-chip is similar to conventional SRAM but requires far less physical space. IBM on-chip eDRAM uses only a third of the components than conventional SRAM, which has a minimum of six transistors to implement a 1-bit memory cell.

- Low energy consumption

  The on-chip eDRAM uses only 20% of the standby power of SRAM.

L4 cache and memory buffer

POWER8 processor-based systems introduce an extra level in memory hierarchy. The L4 cache is implemented together with the memory buffer in the memory riser cards. Each memory buffer contains 16 MB of L4 cache. On a Power S822LC, you can have up to 128 MB of L4 cache by using all the eight memory riser cards.

Figure 3-4 shows a picture of the memory buffer, where you can see the 16 MB L4 cache and processor links and memory interfaces.

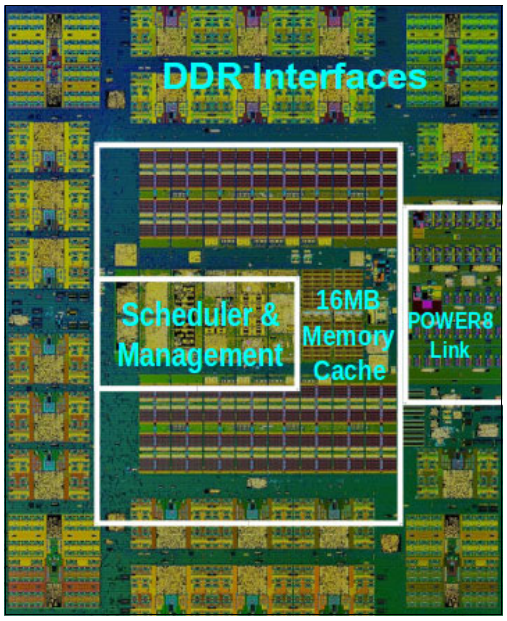

Figure 3-4 Memory buffer chip

Table 3-3 shows a comparison of the different levels of cache in the IBM POWER7®, IBM POWER7+™, and POWER8 processors.

Table 3-3 POWER8 cache hierarchy

Cache	POWER7	POWER7+	POWER8
L1 instruction cache: Capacity/associativity	32 KB, 4-way	32 KB, 4-way	32 KB, 8-way
L1 data cache: Capacity/associativity	32 KB, 8-way	32 KB, 8-way	64 KB, 8-way
L2 cache: Capacity/associativity	256 KB, 8-way Private	256 KB, 8-way Private	512 KB, 8-way Private
L3 cache: Capacity/associativity	On-Chip 4 MB/core, 8-way	On-Chip 10 MB/core, 8-way	On-Chip 8 MB/core, 8-way
L4 cache: Capacity/associativity	N/A	N/A	Off-Chip 16 MB/buffer chip, 16-way Up to eight buffer chips per socket[a]

a. For Power S822LC server, up to four buffer chips per socket.

3.1.2 Memory subsystem

The Power S822LC is a two socket system that supports two POWER8 processor single chip modules (SCMs). The server supports a maximum of 32 DDR3 RDIMMs slots that are housed in eight memory riser cards.

Memory features equate to a riser card with four memory DIMMs. Memory feature codes that are supported are 16 GB, 32 GB, 64 GB, and 128 GB, and run at speeds of 1333 MHz, allowing for a maximum system memory of 1024 GB.

Memory riser cards

Memory riser cards are designed to house up to four industry-standard DRAM memory DIMMs and include a set of components that allow for higher bandwidth and lower latency communications:

- Memory scheduler
- Memory management (reliability, availability, and serviceability (RAS) decisions and energy management)
- Buffer cache

By adopting this architecture, several decisions and processes about memory optimizations are executed outside the processor, saving bandwidth and allowing for faster processor to memory communications. It also allows for more robust RAS.

A detailed diagram of the memory riser card that is available for the Power S822LC and its location on the server are shown in Figure 3-5.

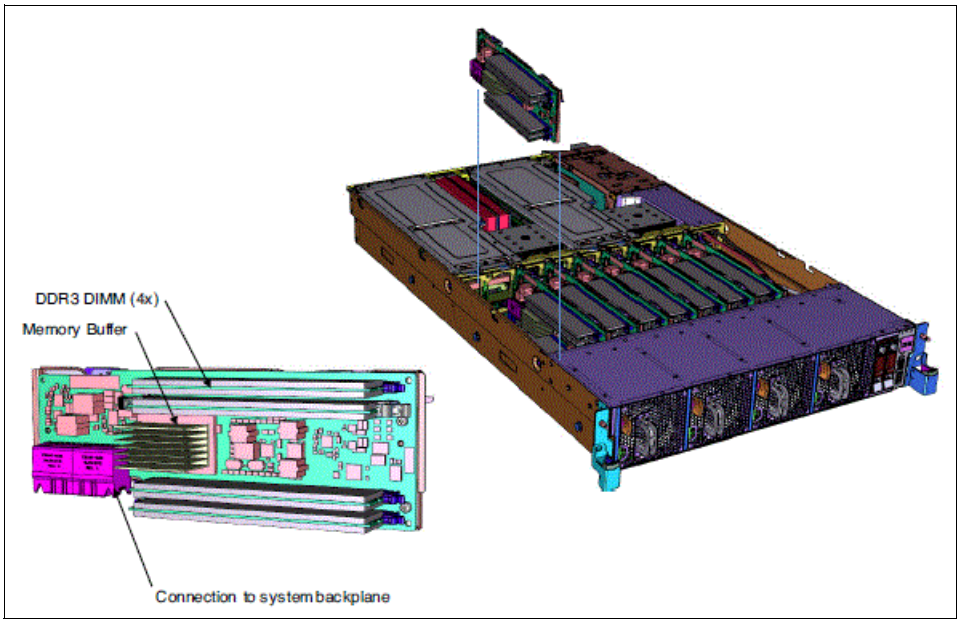

Figure 3-5 Memory riser card components and server location

The buffer cache is an L4 cache and is built on eDRAM technology (same as the L3 cache), which has lower latency than regular SRAM. Each memory riser card has a buffer chip with 16 MB of L4 cache. A fully populated Power S822LC server (two processors and eight memory riser cards) has 128 MB of L4 cache. The L4 cache performs several functions that have a direct impact on performance and brings a series of benefits for the Power S822LC:

- Reduces energy consumption by reducing the number of memory requests
- Increases memory write performance by acting as a cache and by grouping several random writes into larger transactions

- Partial write operations that target the same cache block are “gathered” within the L4 cache before written to memory, becoming a single write operation
- Reduces latency on memory access. Memory access for cached blocks has up to 55% lower latency than non-cached blocks

Memory bandwidth

The POWER8 processor has exceptional cache, memory, and interconnect bandwidths.

For the entire Power S822LC system populated with the two processor modules, the overall bandwidths are shown in Table 3-4.

Table 3-4 Power S822LC total bandwidth estimates

Total bandwidths	**8335-GCA and 8335-GTA**	
	20 cores @ 2.92 GHz	**16 cores @ 3.32 GHz**
Total memory	230 GBps	230 GBps
PCIe interconnect (raw / effective)	128 GBps / 126 GBps	128 GBps / 126 GBps

Where:

- Total memory bandwidth: Each POWER8 processor has four memory channels that run at 9.6 GTransfers per second (GTps) capable of reading 2 bytes and writing 1 byte at each transfer. The total bandwidth is calculated as follows:

  `4 channels × 9.6 GTps × 3 bytes = 115.2 GBps per processor module`

- SMP interconnect: Two POWER8 processors of a Power S822LC server are connected to each other with three SMP buses, each of which is 2 bytes wide and runs at 6.4 GTps. The combined bandwidth of SMP connection is given by the following formula:

  `3 buses × 2 bytes × 6.4 GTps = 38.4 GBps`

- PCIe interconnect: Each POWER8 processor has 32 PCIe Gen3 lanes running at raw bandwidth of 8 Gbps full-duplex[5]. The bandwidth is calculated as follows:

  `32 lanes × 2 processors × 8 Gbps × 2 = 128 GBps`

3.1.3 Input and output

This section introduces the I/O system bus, slot configuration, and PCI adapters in an IBM Power S822LC server.

System bus

This section provides more information about the internal buses.

The Power S822LC servers have internal I/O connectivity through PCIe Gen3 (PCI Express Gen3 or PCIe Gen3) slots.

The internal I/O subsystem on the systems is connected to the PCIe controllers in a POWER8 processor in the system. Each POWER8 processor has a bus that has 32 PCIe lanes running at 8 Gbps full-duplex. Each processor provides 64 GBps of raw I/O connectivity to the PCIe slots, SAS internal adapters, and USB ports.

[5] The effective bandwidth of a PCIe Gen3 lane is approximately 985 MBps in each direction.

Some PCIe devices are connected directly to the PCIe Gen3 buses on the processors, and other devices are connected to these buses through a PCIe Gen3 Switch. The PCIe Gen3 Switch is a high-speed device that allows for the optimal usage of the processor's PCIe Gen3 bus by grouping slower devices that do not use the full bandwidth of the bus. Figure 3-6 shows the Power S822LC server buses and logical architecture.

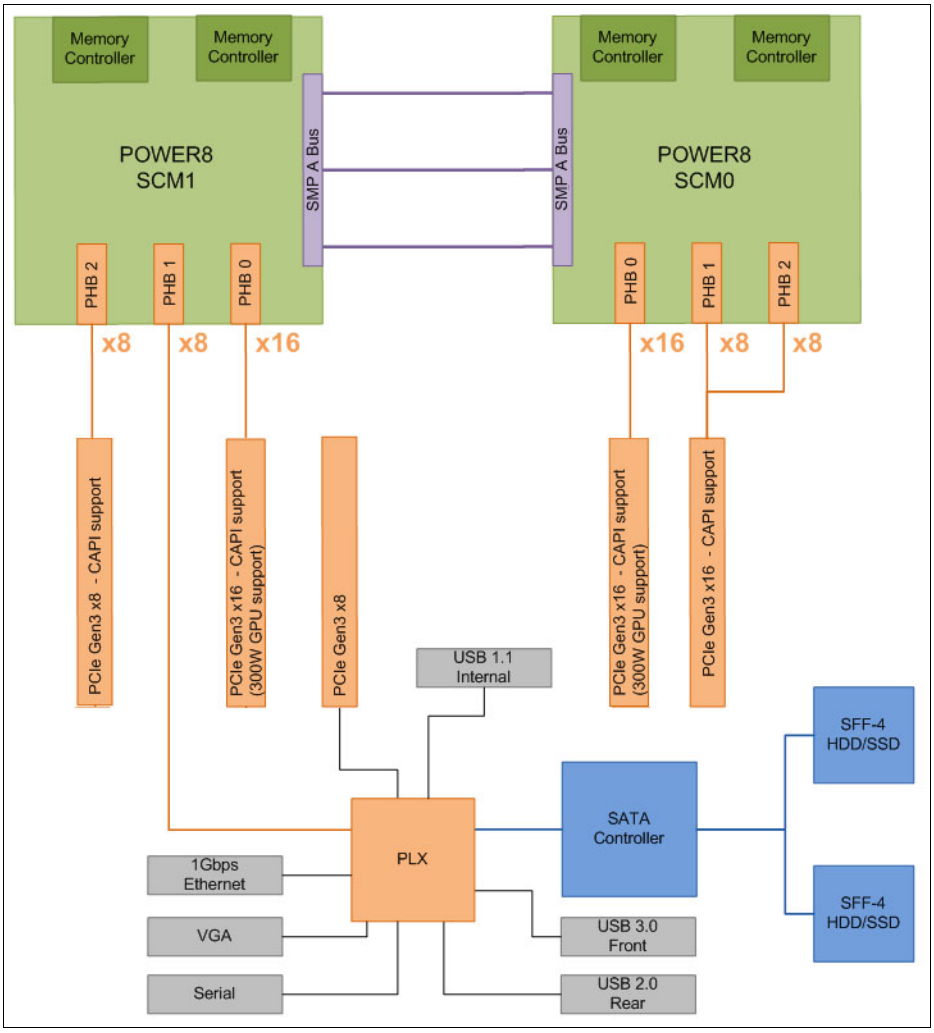

Figure 3-6 Power S822LC server buses and logical architecture

Each processor has 32 PCIs lanes that are split into three channels: Two channels are PCIe Gen3 x8 and one channel is PCIe Gen3 x16.

The PCIe Gen3 x16 channels are connected to the PCIe slots, which can support graphs processing units (GPUs) and other high-performance adapters, such as InfiniBand.

Table 3-5 lists the total I/O bandwidth of a Power S822LC server.

Table 3-5 I/O bandwidth

I/O	I/O bandwidth (maximum theoretical)
Total I/O bandwidth (raw / effective)	► Simplex: 64 GBps / 63 GBps ► Duplex: 128 GBps / 126 GBps

For the PCIe Gen3 interconnect, each POWER8 processor has 32 PCIe Gen3 lanes that run at 8 Gbps full-duplex[6]. The raw bandwidth is calculated as follows:

```
32 lanes × 2 processors × 8 Gbps × 2 = 128 GBps
```

Internal I/O subsystem

The internal I/O subsystem is on the system board, which supports PCIe slots. PCIe adapters on the Power S822LC server are not hot-pluggable.

Slot configuration

The Power S822LC server has five PCIe Gen3 slots. Two of the PCIe Gen3 slots are reserved for GPU usage in the Power S822LC model 8335-GTA technical computing server.

Figure 3-7 is a rear view diagram of the PCIe slots for the Power S822LC server.

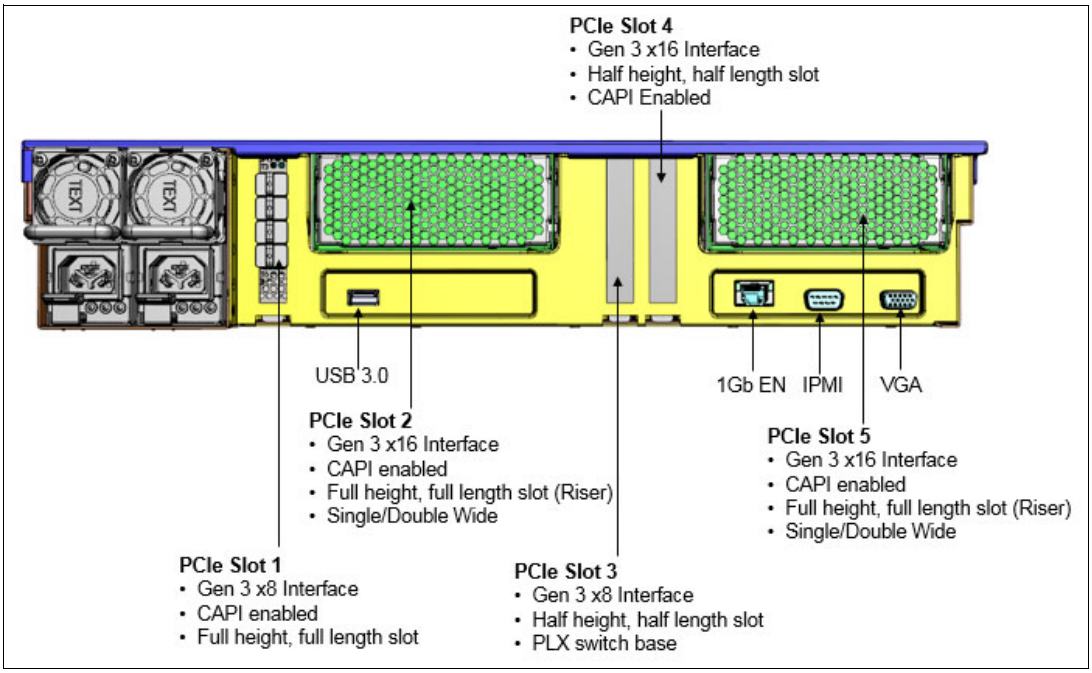

Figure 3-7 Power S822LC server rear view PCIe slots and connectors

Table 3-6 shows the PCIe Gen3 slot configuration for the Power S822LC server.

Table 3-6 Power S822LC server PCIe slot properties

Slot	Description	Card size	CAPI capable	Power limit
Slot 1	PCIe Gen3 x8	Half height, half length	Yes	50 W
Slot 2[a]	PCIe Gen3 x16	Full height, full length	Yes	300 W (GPU) 75 W (Full height full length adapters)
Slot 3	PCIe Gen3 x8	Half height, half length	No	50 W

[6] The effective bandwidth of a PCIe Gen3 lane is approximately 985 MBps in each direction.

Slot	Description	Card size	CAPI capable	Power limit
Slot 4	PCIe Gen3 x16	Full height, full length	Yes	75 W
Slot 5[a]	PCIe Gen3 x16	Full height, full length	Yes	300 W (GPU) 75 W (Full height full length adapters)

a. This slot is made available through a PCIe riser.

The two x16 slots that are provided by the internal PCIe riser (Slot 2 and Slot 5) can be populated with GPU adapters (NVIDIA) or can be used for any high-profile (not low-profile (LP)) adapters. Mixing of GPU and other high-profile adapters on the internal PCIe riser is supported.

Only LP adapters can be placed in LP slots. An x8 adapter can be placed in an x16 slot, but an x16 adapter cannot be placed in an x8 slot. One LP slot must be used for a required Ethernet adapter (#5260, #EL3Z, or #EN0T).

System ports

The system board has one 1 Gbps Ethernet port, one Intelligent Platform Management Interface (IPMI) port and a VGA port, as shown in Figure 3-7 on page 34.

The integrated system ports are supported for modem and asynchronous terminal connections with Linux. Any other application that uses serial ports requires a serial port adapter to be installed in a PCI slot. The integrated system ports do not support IBM PowerHA® configurations. The VGA port does not support cable lengths that exceed 3 meters.

PCI adapters

This section covers the various types and functions of the PCI adapters that are supported by the Power S822LC servers.

PCI Express

PCIe uses a serial interface and allows for point-to-point interconnections between devices by using a directly wired interface between these connection points. A single PCIe serial link is a dual-simplex connection that uses two pairs of wires (one pair for transmit and one pair for receive), and can transmit only one bit per cycle. These two pairs of wires are called a lane. A PCIe link can consist of multiple lanes. In these configurations, the connection is labeled as x1, x2, x8, x12, x16, or x32, where the number is effectively the number of lanes.

The PCIe interfaces that are supported on this server are PCIe Gen3, which are capable of 16 GBps simplex (32 GBps duplex) on a single x16 interface[7]. PCIe Gen3 slots also support previous generation (Gen2 and Gen1) adapters, which operate at lower speeds, according to the following rules:

- Place x1, x4, x8, and x16 speed adapters in the same size connector slots first, before mixing adapter speed with connector slot size.
- Adapters with lower speeds are allowed in larger sized PCIe connectors, but larger speed adapters are not compatible with smaller connector sizes (that is, a x16 adapter cannot go in an x8 PCIe slot connector).

[7] The effective bandwidth of a PCIe Gen3 lane is approximately 985 MBps in each direction.

PCIe adapters use a different type of slot than PCI adapters. If you attempt to force an adapter into the wrong slot type, you might damage the adapter or the slot.

POWER8 based servers can support two different form factors of PCIe adapters:

- PCIe LP cards, which are used with the Power S822L server.
- PCIe full height and full high cards are designed for the 4 EIA scale-out servers, such as the Power S824L server.

Before adding or rearranging adapters, use the System Planning Tool to validate the new adapter configuration. For more information, see the System Planning Tool website:

http://www.ibm.com/systems/support/tools/systemplanningtool/

If you are installing a new feature, ensure that you have the software that is required to support the new feature and determine whether there are any existing update prerequisites to install. To obtain this information, use the IBM prerequisite website:

https://www-912.ibm.com/e_dir/eserverprereq.nsf

The following sections describe the supported adapters and provide tables of orderable feature code numbers.

LAN adapters

To connect the Power S822LC servers to a local area network (LAN), you can use the LAN adapters that are supported in the PCIe slots of the system unit. Table 3-7 lists the supported LAN adapters for the Power S822LC servers.

Table 3-7 Supported LAN adapters

Feature code	CCIN	Description	GCA/GTA support	Max	OS support
5260	576F	PCIe2 LP 4-port 1 GbE Adapter	GCA and GTA	3	Linux
5899	576F	PCIe2 4-port 1 GbE Adapter	GCA	2	Linux
EC3A	57BD	PCIe3 LP 2-Port 40 GbE NIC RoCE QSFP+ Adapter	GCA and GTA	2	Linux
EC3B	57BD	PCIe3 2-Port 40 GbE NIC RoCE QSFP+ Adapter	GCA	2	Linux
EC3E	2CEA	PCIe3 LP 2-port 100Gb EDR InfiniBand Adapter x16	GCA and GTA	1	Linux
EC3T	2CEB	PCIe3 LP 1-port 100Gb EDR InfiniBand Adapter x16	GCA and GTA	1	Linux
EL3Z		PCIe2 LP 2-port 10/1 GbE BaseT RJ45 Adapter	GCA and GTA	3	Linux
EL4L		PCIe2 x4 LP capable 4-port (UTP) 1 GbE Adapter	GCA	2	Linux
EL4M		PCIe2 x4 LP 4-port (UTP) 1 GbE Adapter	GCA and GTA	3	Linux
EL55		PCIe2 2-port 10/1 GbE BaseT RJ45 Adapter	GCA	2	Linux

Feature code	CCIN	Description	GCA/GTA support	Max	OS support
EN0S	2CC3	PCIe2 4-Port (10 Gb+1 GbE) SR+RJ45 Adapter	GCA	2	Linux
EN0T		PCIe2 LP 4-Port (10 Gb+1 GbE) SR+RJ45 Adapter	GCA and GTA	3	Linux

Compute Intensive Accelerator

Compute Intensive Accelerators are GPUs that were developed by NVIDIA. With NVIDIA GPUs, the Power S822LC server can offload processor-intensive operations to a GPU accelerator and boost performance. The Power S822LC server aims to deliver a new class of technology that maximizes performance and efficiency for all types of scientific, engineering, Java, big data analytics, and other technical computing workloads.

Table 3-8 shows the supported Compute Intensive Accelerators.

Table 3-8 Graphics processing units adapters that are supported

Feature code	Description	GCA/GTA support	Max	Operating system support
EC49	Compute Intensive Accelerator GPU K80	GCA and GTA	2	Linux

Fibre Channel adapters

The servers support direct or SAN connection to devices that use Fibre Channel adapters. Table 3-9 summarizes the available Fibre Channel adapters, which all have LC connectors.

If you are attaching a device or switch with an SC type fiber connector, then an LC-SC 50 Micron Fibre Converter Cable (#2456) or an LC-SC 62.5 Micron Fibre Converter Cable (#2459) is required.

Table 3-9 Fibre Channel adapters supported

Feature code	CCIN	Description	GCA/GTA support	Max	OS support
EL43	577F	PCIe3 LP 16Gb 2-port Fibre Channel Adapter	GCA and GTA	2	Linux
EL5B	577F	PCIe3 16Gb 2-port Fibre Channel Adapter	GCA	2	Linux

Internal storage

The internal storage on the Power S822LC server has the following features:

- A storage backplane for two 2.5-inch SFF Gen4 SATA HDDs or SDDs

> **Limitation:** The disks use an SFF-4 carrier. Disks that are used in other Power Systems usually have SFF-3 or SFF-2 carriers, and are not compatible with this system.

- One integrated SATA disk controller without RAID capability
- The storage split backplane feature is not supported

Table 3-10 presents a summarized view of these features.

Table 3-10 Summary of features for the integrated SATA disk controller

Option	Integrated SATA disk controller
Supported RAID types	JBOD
Disk bays	Two SFF Gen4 (HDDs/SDDs)
SATA controllers	Single
IBM Easy Tier® capable controllers	No
External SAS ports	No
Split backplane	No

The 2.5-inch or small form factor (SFF) SAS bays can contain SATA drives (HDD or SSD) that are mounted on a Gen4 tray or carrier (also knows as SFF-4). SFF-2 or SFF-3 drives do not fit in an SFF-4 bay. All SFF-4 bays support concurrent maintenance or hot-plug capability.

External I/O subsystems

The Power S822LC server does not support external PCIe Gen3 I/O expansion drawers or EXP24S SFF Gen2-bay drawers.

External storage

For information about external storage options, see 3.3, “IBM System Storage” on page 41.

3.1.4 NVIDIA GPU

At the time of writing, the NVIDIA Tesla K80 product is the latest GPU model of the Tesla family[8] for HPC servers. In comparison with the previous Tesla K40 generation, the Tesla K80 nearly doubles overall resources and performance.

The Tesla K80 is a dual-GPU card that is composed of two GK210 GPUs sitting on different slots at board. It interfaces with the host system by way of one PCI Express Gen3 (PCIe 3.0) connector for x16 slot (fully specification bandwidth). The GK210 GPUs are connected with each other and the host system by the PLX (an on-board PCIe switch).

Each GK210 GPU implements the Tesla Kepler architecture and has these capabilities:

- 13 SMX multiprocessors, each of which features:
  - 192 CUDA cores
  - 64 double-precision units
  - 32 special function units
  - 32 load/store units
  - 16 texture filtering units
  - 128 KB of configurable memory (shared memory and L1 cache)
  - 64 KB read-only memory (also known as constant memory)
- 12 GB of memory (SDRAM)
  - GDDR5 technology
  - Bus width of 384-bit
  - Clock rate of 2505 Mhz
  - Bandwidth of 240 GBps

[8] The NVIDIA GPU Tesla family landing webpage is http://www.nvidia.com/object/tesla-servers.html

- 1536 KB of L2 cache memory
- Power Cap of 150 W
- Base core clock rate of 560 MHz
  - Maximum core clock rate of 875 MHz

So combined the GK210 GPUs on the K80 board provide a total of 24 GB of SDRAM, 480 GBps of memory bandwidth, and power capacity of 300 W.

The Tesla K80 uses error-correcting code (ECC) check codes to ensure the protection of the memory SDRAM and read-only memories against failures. The ECC mechanism can also be disabled when the application requires more memory bandwidth because this error checking consumes some memory amount.

The Tesla K80 GPU has four compute modes:

- Prohibited: Not available for compute applications
- Exclusive Thread: Only one process and thread can use the GPU at a time
- Exclusive Process: Only one process can use the GPU at a time, but its threads can create work concurrently
- Default: Multiple processes/threads can use the GPU simultaneously

Example 3-1 shows more details of the Tesla K80 GPU of an IBM Power System S822LC.

Example 3-1 Details of the Tesla K80 GPU

```
Device 0: "Tesla K80"
  CUDA Driver Version / Runtime Version          7.5 / 7.5
  CUDA Capability Major/Minor version number:    3.7
  Total amount of global memory:                 11520 MBytes (12079136768 bytes)
  (13) Multiprocessors, (192) CUDA Cores/MP:     2496 CUDA Cores
  GPU Max Clock rate:                            824 MHz (0.82 GHz)
  Memory Clock rate:                             2505 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536),
3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 3 / 0
```

```
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >
```

See the NVIDIA Tesla K80 GPU specification document for further information:

http://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf

3.1.5 BMC

The IBM Power System S822LC server features a baseboard management controller (BMC) for out-of-band system management. Some functions are also available by way of in-band methods. This differs from previous lines of IBM Power System servers, which traditionally featured a flexible service processor (FSP) for that purpose, with increased focus on RAS functions.

The BMC offers functionality that is more focused on scale-out environments, such as HPC clusters and cloud platform infrastructure, where factors like automation, and simplicity for deployment and maintenance play an important role. For example, the BMC provides these functions:

- Power management
- Console (or terminal) sessions
- Network and boot device configuration
- Sensors information (for example, temperature, fan speed, and voltage)
- Firmware and vital product data (VPD) information (for example, firmware components version, serial number, and machine-type model)
- Virtual hard-disk and optical drives (for example, for installing operating systems)
- System firmware upgrade

The BMC functions are available by way of several methods, for example:

- IPMI, both in-band and out-of-band
- Advanced System Management Interface (ASMI)
- Secure Shell (SSH)

The BMC provides the following ports:

- Ethernet port, for out-of-band IPMI, ASMI (web), and SSH access
- Video Graphics Adapter (VGA) port, for graphics display (functional from Petitboot onward)
- Serial port
- Internal (in-chassis) serial port

3.2 Mellanox InfiniBand

The IBM Power System S822LC server can include one-port or two-port high bandwidth and low latency Mellanox InfiniBand host channel adapters (HCAs). These HCAs connect to the bus through PCIe 3.0 x16, which is able to deliver 100 Gbps of data at each EDR port.

InfiniBand provides these significant features, among others:

- Delivers more than 100 Gb per second overall throughput
- Implements Virtual Protocol Interconnect (VPI)
- Takes over transport operations to offload the CPU
- Supports noncontinuous memory transfers
- Supported by IBM Parallel Environment (PE)

Device drivers and support tools are available with the Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux.

3.3 IBM System Storage

The IBM System Storage® disk systems products and offerings provide compelling storage solutions with value for all levels of business, from entry-level to high-end storage systems. For more information about the offerings, see the following website:

http://www.ibm.com/systems/storage/disk

The following section highlights a few of the offerings.

3.3.1 IBM Storwize family

The IBM Storwize® family is the ideal solution to optimize the data architecture for business flexibility and data storage efficiency. Different models, such as the IBM Storwize V3700, IBM Storwize V5000, and IBM Storwize V7000, offer storage virtualization, IBM Real-time Compression™, Easy Tier, and many more functions. For more information, see the following website:

http://www.ibm.com/systems/storage/storwize

3.3.2 IBM FlashSystem family

The IBM FlashSystem® family delivers extreme performance to derive measurable economic value across the data architecture (servers, software, applications, and storage). IBM offers a comprehensive flash portfolio with the IBM FlashSystem family. For more information, see the following website:

http://www.ibm.com/systems/storage/flash

3.3.3 IBM XIV Storage System

The IBM XIV® Storage System is a high-end disk storage system that helps thousands of enterprises meet the challenge of data growth with hotspot-free performance and ease of use. Simple scaling, high service levels for dynamic, heterogeneous workloads, and tight integration with hypervisors and the OpenStack platform enable optimal storage agility for cloud environments.

XIV Storage Systems extend ease of use with integrated management for large and multi-site XIV deployments, reducing operational complexity and enhancing capacity planning. For more information, see the following website:

http://www.ibm.com/systems/storage/disk/xiv/index.html

4

Software stack

This chapter describes the software stack used in the implementation of an IBM High Performance Computing (HPC) solution on IBM POWER8 with the IBM Power System S822LC server.

For more information and details, see the following IBM HPC software announcement[1]: *IBM High Performance Computing software supports IBM Power Systems S822LC servers running Red Hat Enterprise Linux (RHEL) 7.2 in little-endian mode*.

This chapter includes the following sections:

- System management
- OPAL firmware
- xCAT
- RHEL server
- NVIDIA CUDA Toolkit
- Mellanox OFED for Linux
- IBM XL compilers, GCC, and Advance Toolchain
- IBM Parallel Environment
- IBM Engineering and Scientific Subroutine Library and Parallel ESSL
- IBM Spectrum Scale (formerly IBM GPFS)
- IBM Spectrum LSF (formerly IBM Platform LSF)

[1] IBM United States Software Announcement 215-396 (December 8, 2015) http://www.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/6/897/ENUS215-396/index.html

4.1 System management

The baseboard management controller (BMC) provides the system management functions by way of several methods:

- The Advanced System Management Interface (ASMI)
- The Secure Shell (SSH) protocol
- The Intelligent Platform Management Interface (IPMI) protocol

You can access each method by way of one or more BMC IP addresses with the following software:

- ASMI: Any standards-compliant web browser, by way of both (Secure) Hypertext Transfer Protocols (HTTP and HTTPS)
- SSH: Any standards-compliant SSH client
- IPMI: The IPMItool utility, version 1.8.15 and later (requirement for some functions)

  The IPMI method is available both out-of-band (from other systems, by way of network), and in-band (from the current system, by way of internal communication).

  You can find build instructions for IPMItool later in 5.4.8, “IPMI authentication credentials” on page 91.

  For more information and details about the IPMItool, see the project page at the following website:

  http://sourceforge.net/projects/ipmitool

You can configure access credentials for each method on the ASMI.

4.2 OPAL firmware

The Open Power Abstraction Layer (OPAL) firmware is available on the IBM Power System S822LC server and several other IBM Power Systems servers with POWER8 processors. The OPAL firmware supports running Linux in non-virtualized (or bare-metal) mode and virtualized mode (guest or virtual-machine) with kernel-based virtual machine (KVM) acceleration.

With the OPAL firmware, several of the system management functions are performed by way of IPMI rather than Hardware Management Console (HMC), which was used in previous generations of IBM Power Systems servers. This makes systems with OPAL firmware more suited for a wider range of environments and system management tools, allowing for more choice and integration between software and hardware components.

For more information about the OPAL firmware, see its open source project page:

https://github.com/open-power

4.3 xCAT

The Extreme Cluster/Cloud Administration Toolkit (xCAT) performs the roles of deployment and management of the software stack. Among other things, it controls the node discovery process, power management, console sessions, operating system provisioning, and software stack installation and configuration.

xCAT is open source software, and relatively recently moved toward more openness, adopting significant changes to its development process and documentation pages. The development process is now hosted on GitHub (with public milestones and schedules, issue reporting and tracking, and source-code pull requests), and the documentation pages are refreshed, and hosted on Read The Docs (a GitHub service).

xCAT also provides community support, and support options are available from IBM.

xCAT 2.11 release introduces support for the IBM Power System S822LC server, which is accompanied with Linux distributions, installation modes, and other features:

- Red Hat Enterprise Linux (RHEL) Server 7.2 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode
- CUDA Toolkit for NVIDIA graphics processing units (GPUs)
- Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux
- IBM HPC software support with xCAT kits
- System management: Hardware discovery, hardware control, and console sessions
- Diskful and diskless installation

For more information and details, see the xCAT project page, documentation page, and release notes at the following websites:

http://xcat.org
http://xcat-docs.readthedocs.org
https://github.com/xcat2/xcat-core/wiki/XCAT_2.11_Release_Notes

4.4 RHEL server

RHEL is the world's leading enterprise Linux platform[2], which runs on highly scalable, multi-core systems that support the most demanding workloads. Collaboration between Red Hat and engineers from major hardware vendors ensure that the operating system takes advantage of the newest hardware innovations available in chip design, system architecture, and device drivers to improve performance and reduce power utilization.

For more information and details, see the RHEL documentation at the following website:

https://access.redhat.com/documentation/en/red-hat-enterprise-linux

4.5 NVIDIA CUDA Toolkit

The Compute Unified Device Architecture (CUDA) is a programming model and application programming interface (API) conceived to fully utilize the NVIDIA General Purpose Graphics Processing Unit (GPGPU), which provides a highly scalable parallel computing platform.

The toolkit offers a multi-platform apparatus to develop, compile, debug, profile, and run CUDA programs on GPUs. In addition to APIs and runtime libraries, the following are the main resources that are bundled:

- nvcc: A CUDA compiler (nvcc)
- cuda-gdb: A command-line program debugger
- cuda-memcheck: Suite of tools for dynamic error detection

[2] According to the Red Hat Enterprise Linux datasheet published by Red Hat, Inc.® (#12182617_V1_0514).

- profiler: A command-line program profiling tool (nvprof) and GUI-oriented (Visual Profiler)
- Binary utilities
- Development libraries
- Nsight Eclipse Edition: An Eclipse-based integrated development environment (IDE)

Note: At the time of writing, the latest CUDA Toolkit version 7.5 is fully supported on the IBM Power S822LC System with RHEL version 7.2 for Linux on POWER8 Little-endian.

In many sections of this book, there are references about some aspects of the CUDA Toolkit usage on the context of HPC on IBM Power Systems. However, see the following website to read about topics on the toolkit:

http://docs.nvidia.com/cuda/index.html

4.6 Mellanox OFED for Linux

Mellanox OFED for Linux is a version of the OFED distribution from the OpenFabrics Alliance that is tested and packaged by Mellanox, and supports Remote Direct Memory Access (RDMA) and kernel bypass APIs (OFED verbs) over InfiniBand and Ethernet.

The Mellanox OFED for Linux comprises the following components, among others:

- Drivers for InfiniBand, RDMA over Converged Ethernet (RoCE), L2 network interface controller (NIC)
- Access Layers and common verbs interface
- Virtual Protocol Interconnect (VPI)
- IP-over-IB
- Subnet Manager (OpenSM)
- Installation, administration, and diagnostics tools
- Performance test suites

For more details and information, see the Mellanox website:

http://www.mellanox.com

4.7 IBM XL compilers, GCC, and Advance Toolchain

This section describes some of the compiler options available for the software stack such as the IBM XL compilers, the GNU Compiler Collection (GCC), and the IBM Advance Toolchain (basically, more recent GCC and libraries than in the Linux distribution).

4.7.1 XL compilers

IBM XL compilers enhancements help to increase application performance and developer productivity. XL Fortran v15.1.2 and XLC/C++ v13.1.2 compilers support the latest Linux distributions, including RHEL 7.2 and all the new features of the POWER8 processor, including the latest built-in vector intrinsics.

The XL Fortran compiler improves support of the following features from release to release:

- Intrinsic procedures, which help to increase utilization of POWER8 processors
- New compiler options
- Fortran 2008 features
- Language interoperability, which lets developers write programs that contain parts written in Fortran and parts written in the C language
- OpenMP (Version 15.1.2 fully supports the OpenMP Application Program Interface Version 3.1 specification and partially supports the OpenMP Application Program Interface Version 4.0 specification)

The following changes are included in the latest releases of the XL C/C++ compilers:

- Support of new built-in functions for POWER8 processors
- Additional compiler options
- Increase support of the following C/C++ standards:
  - C++14
  - C++11
  - C11
- Partial support of OpenMP 4.0 (Version 13.1.2 fully supports only the OpenMP Application Program Interface Version 3.1 specification).

For more information about XL Fortran support for POWER8 processor, see the following website:

http://www.ibm.com/support/knowledgecenter/?lang=en#!/SSAT4T_15.1.2/com.ibm.compilers.linux.doc/welcome.html

For more information about XL C/C++ compilers, see the following website:

http://www.ibm.com/support/knowledgecenter/?lang=en#!/SSXVZZ_13.1.2/com.ibm.compilers.linux.doc/welcome.html

4.7.2 GCC and Advance Toolchain

GCC 4.8.5 is included in the RHEL 7.2 distribution. It contains features and optimizations in the common parts of the compiler that improve support for the POWER8 processor architecture.

The IBM Advance Toolchain for Linux on Power is a set of open source development tools and runtime libraries that allows users to take advantage of the latest IBM POWER8 hardware features on Linux. It supports big endian (ppc64) and little endian (ppc64le).

The latest release includes current versions of the following packages, which can help with porting and tuning applications for POWER8:

- GNU Compiler Collection (gcc, g++, gfortran), including individually optimized gcc runtime libraries for supported POWER8 processor
- GNU C library (glibc), individually optimized for supported POWER8 processor
- GNU Binary Utilities (binutils)
- AUXV Library (libauxv)
- GNU Debugger (gdb)

- Performance analysis tools (oprofile, valgrind, itrace)
- Multi-core exploitation libraries (Intel TBB, Userspace RCU, SPHDE)
- Plus several support libraries (libhugetlbfs, Boost, zlib, and more)

Note: For some specific workloads, GCC 5.2 provided with the IBM Advance Toolchain and GCC 4.8.5 provided with the RHEL 7.2 distribution have different performance.

For more information about GCC support on POWER8 for RHEL 7.2, see this website:

https://gcc.gnu.org/gcc-4.8/

For more information about the IBM Advance Toolchain, see the following website:

http://ibm.co/1CsNsDs

4.8 IBM Parallel Environment

The IBM Parallel Environment (PE) is an added-value layer of components that provide a complete solution for high performance and technical computing on clusters of Linux with IBM Power Systems.

The IBM Parallel Environment comprises of two components:

4.8.1 IBM PE Runtime Edition

IBM PE Runtime Edition provides runtime environment, libraries, compiling scripts and tools that target development, debugging and execution of programs written on C, C++ and Fortran with the Message Passing Interface (MPI), Parallel Active Message Interface (PAMI) and OpenSHMEM parallel programming models.

The execution of parallel batch jobs is managed by the Parallel Operating Environment (POE). It provides a rich set of variables to manage the execution of parallel jobs and controls to fine-tune the environment for performance. POE also can be evoked with the IBM Spectrum Load Sharing Facility (LSF), which is a robust workload manager.

Some tools are also available. For instance, the runtime package comes with the Parallel Environment Shell (PESH) command-line tool and parallel applications can be debugged with the parallel debugger (PDB).

4.8.2 IBM PE Developer Edition

It provides tools for development, debugging, and performance analysis of parallel applications in C, C++b and Fortran. The environment integrates with PE and IBM Spectrum LSF, allowing you to spawn parallel jobs from within the GUI.

The following analysis and measurements can be achieved with PE Developer Edition tools:

- Hardware performance counter profiling
- GPU hardware counter profiling
- MPI profiling and trace
- MPI I/O profiling
- Call graph analysis
- openMP profiling

Note: New in PE Developer Edition version 2.3 is support for GPU hardware counter profiling.

The following components can also be installed:

- The HPC Toolkit: Provides libraries and tools that are used to prepare the parallel application and collect performance data
- The hpctView: A GUI that eases the processes of gathering and analysis of the data
- Plug-ins: Used to extend an Eclipse IDE installation and provides integration with the HPC Toolkit

Find more about PE Devloper Edition in 6.7.1, “The Parallel Environment Developer Edition” on page 200.

4.9 IBM Engineering and Scientific Subroutine Library and Parallel ESSL

The Engineering and Scientific Subroutine Library (ESSL) family of products is a state-of-the-art collection of mathematical subroutines. Running on IBM POWER8 servers and clusters, the ESSL family provides a wide range of high-performance mathematical functions for various scientific and engineering applications.

The collection contains two types of libraries:

- ESSL 5.4, which contains over 600 high-performance serial and symmetric multiprocessing (SMP) mathematical subroutines that are tuned for POWER8.
- Parallel ESSL 5.2, which contains over 125 high-performance single program, multiple data (SPMD) mathematical subroutines. These subroutines are designed to exploit the full power of clusters of POWER8 servers that are connected with a high performance interconnect.

IBM ESSL includes IBM implementation of BLAS/LAPACK and IBM Parallel ESSL includes IBM implementation of ScaLAPACK, which are the industry standards for linear algebra subroutines. If the application utilizes BLAS/LAPACK/ScaLAPACK functions, you just recompile your application on IBM POWER8 and link it with IBM ESSL to get optimized performance.

Additionally, IBM ESSL implements some linear algebra routines that are not included in BLAS/LAPACK. Examples of such routines: `_GETMI` (General Matrix Transpose (In-Place)) and `_GETMO` (General Matrix Transpose (Out-of-Place)).

For POWER8 servers and clusters, which have NVIDIA GPUs within, performance can be significantly improved by using environment variables, which enables GPU usage inside ESSL routines. There are two options:

- GPU-only mode: All computations will be performed on GPU
- Hybrid mode: All computations will be distributed between GPU and CPU to increase utilization of the system

Additionally, eSSL provides support of the following libraries:

- Fastest Fourier Transform in the West (FFTW)
- CBLAS

All routines from the ESSL family are callable from Fortran, C, and C++.

For more information about the IBM ESSL, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSFHY8/essl_welcome.html

For more information about the IBM Parallel ESSL, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSNR5K/pessl_welcome.html

4.10 IBM Spectrum Scale (formerly IBM GPFS)

The IBM Spectrum Scale (formerly the IBM General Parallel File System (GPFS)) is a distributed, high-performance, massively scalable enterprise file system solution that addresses the most challenging demands in high-performance computing. It is a proven solution that is used to store the data for thousands of mission-critical commercial installations worldwide.

The IBM Spectrum Scale is software-defined storage for high performance, large-scale workloads on-premises or in the cloud. This scale-out storage solution provides file, object, and integrated data analytics for these items:

- Compute clusters (technical computing, and high-performance computing)
- Big data and analytics
- Hadoop Distributed File System (HDFS)
- Private cloud
- Content repositories
- File Placement Optimizer (FPO)

For more information about IBM Systems Software for high performance computing, in particular IBM Spectrum Scale, see the following website:

http://www.ibm.com/systems/power/hardware/hpc.html

For more information about IBM Spectrum Scale, see the following website:

http://www.ibm.com/systems/storage/spectrum/scale/index.html

4.11 IBM Spectrum LSF (formerly IBM Platform LSF)

The IBM Spectrum LSF is a workload management that is employed to coordinate shared access and optimized use of computing resources of an HPC cluster. It provides the following features, among others:

- Policy-driven work scheduling and load balancing
- Compute resources allocation
- Cluster resources administration
- Cluster monitoring
- Supports heterogeneous resources and multi-cluster
- Fault tolerance
- Security

At the time of writing, its latest version (9.1.3) supports the IBM Power System S822LC server with RHEL version 7.2. Spectrum LSF is also fully integrated with the IBM Parallel Environment.

Cluster users and administrators can interact with Spectrum LSF by way of the command-line tools, web interface (provided by the IBM Platform Center) or application programming interface (API).

Many topics about Spectrum LSF in the context of the HPC solution that are described in this book are discussed in 7.3, “Using the IBM Spectrum LSF” on page 240 and Chapter 8, “Cluster monitoring” on page 247.

For more information about IBM Spectrum LSF, see the following website:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_welcome.html

5

Software deployment

This chapter describes the software deployment of an Extreme Cluster/Cloud Administration Toolkit (xCAT) cluster with the IBM High Performance Computing (HPC) software. The cluster runs on Red Hat Enterprise Linux (RHEL) Server 7.2 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode on a IBM Power System S822LC server.

The following sections provide a summary of the software stack, and describe the system management procedures adopted in this chapter:

- Software stack
- System management

The next sections describe the concepts and deployment of the xCAT cluster and the software stack:

- xCAT overview
- xCAT Management Node
- xCAT Node Discovery
- xCAT Compute Nodes
- xCAT Login Nodes

5.1 Software stack

The following software stack components and versions are referenced in this chapter:

- Extreme Cluster/Cloud Administration Toolkit (xCAT) 2.11
- Red Hat Enterprise Linux (RHEL) Server 7.2
- Compute Unified Device Architecture (CUDA) Toolkit 7.5 (7.5-23)
- Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux 3.2 (3.2-1.0.1.1)
- XL C/C++ Compiler for Linux V13.1.2
- XL Fortran Compiler for Linux V15.1.2
- Advance Toolchain 8.0 (8.0-5)
- GNU Compiler Collection (GCC) 4.8.5 (provided with RHEL)
- Parallel Environment (PE) Runtime Edition (PE RTE) 2.3
- Parallel Environment Developer Edition (PE DE) 2.2
- Engineering and Scientific Subroutine Library (ESSL) 5.4
- Parallel ESSL (PESSL) 5.2
- Spectrum Scale (formerly GPFS) 4.1.1.3
- Spectrum LSF (formerly Platform LSF) 9.1.3
- Open Power Abstraction Layer (OPAL) firmware OP810.10 (OP8_v1.7_1.13)
- Intelligent Platform Management Interface (IPMI) tool (IPMItool) 1.8.15

5.2 System management

This chapter describes system management functions (or operations) for the IBM Power System S822LC server based on the IPMI protocol, using the IPMItool utility. Some functions require IPMItool version 1.8.15 or later, such as system firmware upgrade.

This section provides the following instructions:

- Build IPMItool 1.8.15 from the source on RHEL Server 7.2
- Frequently used commands for IPMItool
- Configure the boot order in the Petitboot bootloader
- Upgrade the system firmware of the IBM Power System S822LC server

5.2.1 Build instructions for IPMItool

To build IPMItool version 1.8.15 (or later), complete the following steps:

1. Install the required build dependencies:

   ```
   # yum install gcc make bzip2 openssl-devel
   ```

1. Check the project page for more recent versions, and the download link for the source-code tarball (file: `ipmitool-`*`version`*`.tar.bz2`):

   https://sourceforge.net/projects/ipmitool/

2. Download, and extract the tarball:

```
# mkdir /tmp/ipmitool
# cd /tmp/ipmitool

# curl -sOL http://sourceforge.net/projects/ipmitool/files/ipmitool/1.8.15/
ipmitool-1.8.15.tar.bz2

# tar xf ipmitool-1.8.15.tar.bz2
# cd ipmitool-1.8.15
```

3. Run the **configure** script to configure the build on the current system.

   The following summary table is listed at the end of the script execution. The `lan` and `lanplus` interfaces are listed as `yes` (that is, enabled). The `lanplus` interface requires the `openssl-devel` package.

```
# ./configure
<...>
ipmitool 1.8.15

Interfaces
  lan     : yes
  lanplus : yes
  open    : yes
  free    : no
  imb     : yes
  bmc     : no
  lipmi   : no
  serial  : yes
  dummy   : no

Extra tools
  ipmievd   : yes
  ipmishell : no
```

4. Run the **make** command to start the build.

   If the system has multiple processors, you can increase the number of parallel build jobs with the `-j`*`number`* option, for example:

```
# make -j160
```

5. Run the **make install** command to install the build result in the system:

```
# make install
```

6. Verify the **ipmitool** command points to the locally installed program (at the `/usr/local/bin` directory) with the **which** command, at the expected version:

```
# which ipmitool
/usr/local/bin/ipmitool

# ipmitool -V
ipmitool version 1.8.15
```

5.2.2 Frequently used commands with the IPMItool

The following arguments are present in every command in the following list, replaced by the <*arguments*> word:

`-I lanplus -H` *`bmc-address`* `-U` *`ipmi-username`* `-P` *`ipmi-password`*

The `lanplus` interface is suggested. Some commands can work with the `lan` interface.

The default IPMI username is `ADMIN`, and the default IPMI password is `admin`.

The following list contains frequently used **`ipmitool`** commands:

- Power status:

  ```
  $ ipmitool <arguments> chassis power status
  ```

- Power on:

  ```
  $ ipmitool <arguments> chassis power on
  ```

- Power off:

  ```
  $ ipmitool <arguments> chassis power off
  ```

- Power cycle (power off, then power on):

  ```
  $ ipmitool <arguments> chassis power cycle
  ```

- Open console session:

  ```
  $ ipmitool <arguments> sol activate
  ```

- Close console session:

  ```
  $ ipmitool <arguments> sol deactivate
  ```

  You can also close an active console session with the following keystrokes:

  – On a non-Secure Shell (SSH) connection:

    `Enter, ~ (tilde[1]), . (period)`

    > **Note:** This command can close an SSH connection, which can leave the console session open.

  – On an SSH connection:

    `Enter, ~ (tilde), ~ (tilde), . (period)`

    > **Note:** This command leaves the SSH connection open and closes the console session.

- Reset the BMC:

  ```
  $ ipmitool <arguments> bmc reset cold
  ```

- Values of sensors:

  ```
  $ ipmitool <arguments> sdr list full
  ```

- Display the BMC Ethernet Port network configuration:

  ```
  $ ipmitool <arguments> lan print 1
  ```

- Set the BMC Ethernet Port for Dynamic Host Configuration Protocol (DHCP) IP address:

  ```
  $ ipmitool <arguments> lan set 1 ipsrc dhcp
  ```

- Set the BMC Ethernet Port for Static IP address:

  ```
  $ ipmitool <arguments> lan set 1 ipsrc static
  $ ipmitool <arguments> lan set 1 ipaddr a.b.c.d
  ```

[1] On keyboards with dead keys (for example, on some non-English languages), the tilde mark requires two keystrokes: tilde and space.

```
$ ipmitool <arguments> lan set 1 netmask e.f.g.h
$ ipmitool <arguments> lan set 1 defgw i.j.k.l
```

- Display machine type and model, serial number, and other information:

```
$ ipmitool <arguments> fru print 3
```

- Display system firmware version information:

```
$ ipmitool <arguments> fru print 47
```

5.2.3 Boot order configuration

The Petitboot bootloader can automatically boot (or *autoboot*) from several types of devices in a certain order (that is, falling back to later devices if earlier devices cannot be booted from).

The scenario presented here requires a specific configuration of device boot order (specifically, for Genesis-based node discovery, and diskful installation, which are described in 5.3, “xCAT overview” on page 61). It is important for Petitboot to first attempt to boot from the network devices by way of DHCP. Only they are not available should it attempt to boot from the disk devices. In such order, the node can obtain its network and boot configuration from the xCAT management node (for example, the Genesis image or diskless installation), or fall back to boot an operating system from disk if network boot is not specified (for example, diskful installation).

In order to configure the boot order in the Petitboot bootloader, perform the following steps:

1. Power on the system:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power on
```

2. Open the console:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    sol activate
```

3. Wait for the Petitboot panel (Example 5-1).

Example 5-1 Petitboot panel

```
Petitboot (dev.20151015)                    8335-GTA         0000000000000000

System information
System configuration
Language
Rescan devices
Retrieve config from URL
*Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
Welcome to Petitboot
```

4. Configure boot order for Petitboot. In the **Petitboot** panel, select **System configuration**.
5. In the **Petitboot System Configuration** panel (Example 5-2 on page 58), complete these steps:
   a. In the **Boot order** section, complete the following steps:
      i. Select **Clear**.
      i. Select **Add Device**.

ii. In the **Select a boot device to add** panel (Example 5-3), select **Any Network device**, and press **OK**.

iii. Select **Add Device** again.

iv. In the **Select a boot device to add** panel again, select **Any Disk device**, and press **OK**.

v. Verify that the **Boot order** section is identical to Example 5-4 on page 59.

b. In the **Network** section, select the **DHCP on a specific interface** option.

> **Note:** You can select the **DHCP on all active interfaces** option, but that might slow the boot process unnecessarily if multiple network ports are cabled and active.

c. In the **Device** section, select the network interface for accessing the xCAT Management Node (if you selected *DHCP on a specific interface* in the Network section).

d. Press **OK**.

Example 5-2 Petitboot System Configuration panel, DHCP on a specific interface setting

```
Petitboot System Configuration

Boot Order:    (None)

               [     Add Device     ]
               [  Clear & Boot Any  ]
               [       Clear        ]

Network:       ( ) DHCP on all active interfaces
               (*) DHCP on a specific interface
               ( ) Static IP configuration

Device:        ( ) enP1p3s0f0 [98:be:94:59:fa:24, link down]
               ( ) enP1p3s0f1 [98:be:94:59:fa:25, link down]
               ( ) enP1p3s0f2 [98:be:94:59:fa:26, link up]
               (*) enP1p3s0f3 [98:be:94:59:fa:27, link up]
               ( ) tunl0 [00:00:00:00:08:00, link up]

DNS Server(s): ______________________________ (eg. 192.168.0.2)
               (if not provided by DHCP server)

Disk R/W:      ( ) Prevent all writes to disk
               (*) Allow bootloader scripts to modify disks

               [    OK    ]  [   Help   ]  [  Cancel  ]

tab=next, shift+tab=previous, x=exit, h=help
```

Example 5-3 shows the panel to select a boot device.

Example 5-3 Select a boot device to add panel (any Network Device setting)

```
Select a boot device to add

         ( ) net:  enP1p3s0f0 [mac: 98:be:94:59:fa:24]
         ( ) net:  enP1p3s0f1 [mac: 98:be:94:59:fa:25]
```

```
        ( ) net:  enP1p3s0f2 [mac: 98:be:94:59:fa:26]
        ( ) net:  enP1p3s0f3 [mac: 98:be:94:59:fa:27]
        ( ) net:  tunl0 [mac: 00:00:00:00:08:00]
        (*) Any Network device
        ( ) Any Disk device
        ( ) Any USB device
        ( ) Any CD/DVD device
        ( ) Any Device

        [    OK    ]  [  Cancel  ]

 tab=next, shift+tab=previous, x=exit, h=help
```

Example 5-4 shows the petitboot system configuration option panel.

Example 5-4 Petitboot System Configuration panel, Boot Order configuration

```
Petitboot System Configuration

Boot Order:    (0) Any Network device
               (1) Any Disk device

               [     Add Device     ]
               [  Clear & Boot Any  ]
               [       Clear        ]

<...>
```

On the next boot, the Petitboot bootloader can automatically boot from the network and disk devices in the specified order. On this boot, no automatic boot attempt is made due to user intervention.

5.2.4 System firmware upgrade

The IPMItool version 1.8.15 or later can be used to upgrade the system firmware of the IBM Power System S822LC. Run the IPMItool command from the same or a close local area network (LAN) to the target system or Baseboard Management Controller (BMC) to avoid network instability problems.

To upgrade the system firmware, complete the following steps:

1. Download the system firmware image:
   a. Go to the IBM Support page:

      http://ibm.com/support

   b. In **Product Finder**, enter **8335-GTA**.
   c. In the results list, select **Scale-out LC 8335-GTA**.
   d. Under **Downloads**, select **Downloads (drivers, firmware, PTFs)**.
   e. In the results list, click the desired version (for example, **OP8_v1.7_1.62_F**). Usually, the latest version is suggested for the general case.
   f. Proceed with the sign-in process.
   g. Select a download option (for example, HTTPS).

h. Click the **8335_<*version*>_update.hpm** link to download the file (for HTTPS; for other methods, follow the instructions that are provided in the website).

i. *Optional*: Click the **Description** link for more details about the firmware version.

2. Install the system firmware image:

a. Power off the system:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power off
Chassis Power Control: Down/Off
```

Wait until the system is powered off. You can verify it with the following command:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power status
Chassis Power is off
```

b. Reset the BMC:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    mc reset cold
Sent cold reset command to MC
```

Wait until the BMC is back online. You can verify it with the following command (repeat as necessary):

```
# ping -c1 bmc-address
<...>
1 packets transmitted, 1 received, 0% packet loss, time 0ms
<...>
```

c. Protect the BMC memory content (for example, network configuration) during upgrade:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    raw 0x32 0xba 0x18 0x00
```

d. Upgrade the system firmware image (Example 5-5). Note the following about the process:

- You might be asked to confirm the operation; press *y* and *enter*.
- The output can vary depending on the old and new firmware versions used.
- In case of segmentation fault errors, try to change the **-z** argument to 25000.
- If you lose the network configuration, establish a serial connection to the internal serial port and configure it with the **ipmitool** command on the 127.0.0.1 IP address.
- For more information and details about errors, see the **Description** link in the system firmware image download page for more information and instructions.

Example 5-5 One step of the system firmware upgrade with the ipmitool command

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    -z 30000 hpm upgrade 8335_<version>_update.hpm force
Setting large buffer to 30000

PICMG HPM.1 Upgrade Agent 1.0.9:

Validating firmware image integrity...OK
Performing preparation stage...
Services  be affected during upgrade. Do you wish to continue? (y/n): y
OK
```

```
Performing upgrade stage:

--------------------------------------------------------------------------------
|ID  | Name     |                     Versions                          | %  |
|    |          |      Active     |      Backup     |      File       |    |
|----|----------|-----------------|-----------------|-----------------|----|
|*  2|BIOS      |   0.00 00000000 | ---.-- -------- |   1.00 3E010701 |100%|
|    |Upload Time: 00:27          | Image Size: 33554584 bytes                |
|*  0|BOOT      |   2.13 7B4E0100 | ---.-- -------- |   2.13 AB660100 |100%|
|    |Upload Time: 00:00          | Image Size:  262296 bytes                 |
|*  1|APP       |   2.13 7B4E0100 | ---.-- -------- |   2.13 AB660100 |100%|
|    |Upload Time: 00:16          | Image Size: 33292440 bytes                |
--------------------------------------------------------------------------------
(*) Component requires Payload Cold Reset

Firmware upgrade procedure successful
```

e. Power on the system:

Note: The system firmware upgrade only completes *after* the system is powered on.

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power on
Chassis Power Control: Up/On
```

f. Open the console:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
sol activate
```

g. Wait for the Petitboot panel (Example 5-1 on page 57).

Note: Only around twenty ISTEP lines are required (earlier than Petitboot).

If the IPMI console is not responsive to any keys, try to reset the BMC again.

3. Verify that the system firmware version matches the desired or downloaded version:

```
# ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    fru print 47
Product Name          : OpenPOWER Firmware
Product Version       : IBM-firestone-ibm-OP8_v1.7_1.62
Product Extra         :        hostboot-bc98d0b-1a29dff
Product Extra         :        occ-0362706-16fdfa7
Product Extra         :        skiboot-5.1.13
Product Extra         :        hostboot-binaries-43d5a59
Product Extra         :        firestone-xml-e7b4fa2-c302f0e
Product Extra         :        capp-ucode-105cb8f
```

5.3 xCAT overview

This section provides an overview of the architecture and concepts involved in a cluster that is administered with xCAT (that is, an *xCAT cluster*), and the scenario described in this chapter.

xCAT manages the nodes in a cluster by using continuous configuration and event handling. The xCAT database contains the information required to perform the configuration steps. Several services and commands (for example, DHCP server, and xCAT administration commands) trigger and respond to events such as node booting, node discovery, and installation of operating system (OS) and other software.

For more information and details, see the xCAT project and documentation pages at the following websites:

http://xcat.org
http://xcat-docs.readthedocs.org

5.3.1 xCAT cluster: Nodes and networks

An xCAT cluster is a number of nodes that are interconnected by one or more networks.

The type of node depends on its function in the cluster (for example, management, compute, login, or service node). The type of network depends on its traffic and interconnected nodes (for example, operating system-level management and provisioning, service processor-level hardware management, application-level intercommunication, and public/Internet access).

The following list describes the types of nodes in an xCAT cluster:

- Management node: Performs administrative operations on compute nodes, for example power control, software deployment (operating system provisioning, application installation, and updates), configuration, command execution, and monitoring.
- Compute nodes: Perform operations that are specified by the management node, for example operating system provisioning and command execution, and runs the runtime and application software. Compute nodes are sometimes referred to simply as *nodes*.
- Login nodes: Perform the role of an interface to the users of the clusters, allowing for tasks such as job submission and source-code compilation.
- Service nodes: Perform operations that are delegated by the management node on groups of nodes, and respond to requests from a set of other nodes, acting as intermediary management nodes on large clusters. An xCAT cluster with service nodes is known as a *hierarchical cluster*.

The following list describes the types of networks in an xCAT cluster:

- Management network:
  - Used for in-band operations (that is, through the system's network interface), for example, node discovery, operating system provisioning, and management
  - Interconnects the management node, service nodes (if any), login nodes (if any), and compute nodes (all by way of the in-band network interface controller (NIC))
  - Possibly segregated into separate virtual LANs (VLANs) for each service node
  - Usually an Ethernet network of high transfer speed, depending on the number of compute nodes and frequency of operating system provisioning, software download, and installation operations
- Service network:
  - Used for out-of-band operations (that is, through the BMC's network interface), for example, power control, console sessions, and platform-specific functions
  - Interconnects the management node and service nodes (by way of the in-band NIC), and the other nodes (by way of the out-of-band NIC)

  - Possibly combined with the management network (same physical network)
  - Usually an Ethernet network, but do not need to be as high transfer speed as the management node because the network traffic of out-of-band operations is usually of smaller size and lower frequency
- Application network:
  - Used by applications that are running on compute nodes
  - Interconnects the compute nodes
  - Usually an InfiniBand network for HPC applications
- Optional: Site (public) network:
  - Used for directly accessing the management node, and other nodes
  - Interconnects the site gateway, and nodes (by way of in-band NIC)
  - Can provide the cluster with access to the Internet
  - Can be combined with the management node (same physical network)
  - Usually an Ethernet network

5.3.2 xCAT database: Objects and tables

The xCAT database contains all information that relates to the cluster. This information is either defined by an administrator or automatically obtained from the cluster, such as nodes, networks, services, and configuration for any services used by xCAT (for example, DNS, DHCP, HTTP, and NFS).

All data is stored in tables, and can be manipulated either directly as tables (for example, `nodelist`, `networks`, `nodegroup`, `osimage`, or `site`) or logically as objects (or object definitions) of certain types (for example, `node`, `network`, `group`, `osimage`, or `site`) with the following commands:

- Objects:
  - View: **`lsdef`**
  - Create: **`mkdef`**
  - Change: **`chdef`**
  - Remove: **`rmdef`**
- Tables:
  - View: **`tabdump`**
  - Change: **`tabedit`** or **`chtab`**

On certain tables or object attributes (typically on per-node attributes), you can use *regular expressions* to set an attribute's value according to the name of the respective object (for example, the IP address of compute nodes).

The xCAT database is stored in a SQLite database by default, but other database management systems can be used (for example, for higher scalability on larger clusters).

For more information, see the xCAT database manual page by using the following command:

```
$ man 5 xcatdb
```

5.3.3 xCAT node booting

The xCAT management node can control the boot method (or device) of the compute nodes with several methods:

- Change the temporary boot device configuration of the bootloader by way of IPMI.
- Change the network boot configuration that is provided to the bootloader by way of DHCP.
- Do not provide a network boot configuration to the bootloader, allowing it to boot from other device than network adapters.

The methods based on the network boot configuration require the correct automatic boot (or autoboot) configuration in the bootloader, and dynamic configuration of DHCP and Trivial File Transfer Protocol (TFTP) servers with general and per-node settings.

It is possible to boot new or undiscovered nodes into node discovery, and known or discovered nodes into arbitrary stages (for example, operating system provisioning, installed operating system, basic shell environment, or node discovery again).

For that purpose, the nodes' bootloader needs to be configured to automatically boot with the following device order: Primarily, from the network interface on the xCAT management network (to obtain network and boot configuration by way of DHCP/TFTP from the xCAT management node), and secondarily from local disks. For more details, see 5.2.3, "Boot order configuration" on page 57.

On that foundation, any boot image can be specified by the xCAT management node through DHCP to a node, retrieved by the node through TFTP, and booted. If no boot image is specified, the node proceeds to boot from disk, which can contain an operating system installation that is already provisioned.

5.3.4 xCAT node discovery

The xCAT node discovery (or node discovery process) consists of booting a new (or undiscovered) node, and letting the management node identify (or *discover*) the node by using some particular method.

For example, during boot, the node can be offered a special-purpose boot image by the management node. This process is called *genesis* (basically, a Linux *kernel* and a custom initial RAM file system, or *initramfs*). Genesis collects identification and hardware information about the node, informs the management node about it, and waits for instructions. The management node can then configure and control the discovered node (for example, based on existing object definitions).

Several node discovery methods are available in xCAT, ranging between more manual and more automatic:

- Manual node definition (not really "discovery")
- MTMS-based (Machine Type and Model, and Serial number) discovery (adopted in this chapter)
- Sequential discovery
- Switch-based discovery

For more information and details about node discovery methods, see the xCAT documentation page:

http://xcat-docs.readthedocs.io/en/2.11/guides/admin-guides/manage_clusters/ppc64le/discovery/index.html

For example, the MTMS-based discovery can be summarized in the following sequence:

1. The new or undiscovered node is powered on, and the bootloader requests a network address and boot configuration.
2. The xCAT management node does not recognize the node (that is, the Media Access Control (MAC) address in the request). It provides the node with a temporary network address, and pointers to the node discovery boot image.
3. The node applies the network configuration, downloads the node discovery boot image, and boots it.
4. The boot image collects hardware information (for example, the system's machine type and model, serial number, network interfaces' MAC address, and processor and memory information) and reports it back to the xCAT management node.
5. The xCAT management node attempts to match part of the reported hardware information to a node object definition (currently, the system's machine-type and model, and serial number).
6. If a match is identified, the xCAT management node then configures that node according to its object definition (for example, network configuration and next stages to perform) and updates that object definition with any new hardware information.

The node can then respond to in-band or operating system-level management operations through SSH on its IP address (available on the running Genesis image).

After node discovery, the new or undiscovered node becomes a known or discovered node and supports in-band operations, for example, operating system provisioning, software installation, and configuration, from the xCAT management node.

Note: The Genesis image contains the IPMItool utility, which is available for performing IPMI commands both out-of-band (by way of the BMC IP address) and in-band (by way of the internal connection between the system and BMC, independently of network configuration).

This is specially useful in case of problems during the network configuration of the BMC, which can render it unresponsive out-of-band, but still responsive in-band.

5.3.5 xCAT BMC discovery

The xCAT BMC discovery occurs automatically after node discovery, and it can be summarized in the following sequence:

1. The xCAT management node attempts to match the node object definition to a temporary BMC object definition (with machine type and model, and serial number information) for new or undiscovered BMCs.
2. If a match is identified, the xCAT management node configures the BMC according to the BMC attributes in the node object definition, for example network configuration, and removes the temporary BMC object.

The node can then respond to out-of-band management operations through IPMI on its BMC's IP address.

The BMC becomes a discovered BMC, and the corresponding node supports out-of-band operations, for example power control and monitoring, by way of its object definition in the xCAT management node.

5.3.6 xCAT operating system installation types: Disks and state

The xCAT management node can support different methods for provisioning an OS and providing persistent state (data) to the compute nodes according to availability of disks and persistency requirements:

- Diskful and Stateful: The operating system is installed to disk and loaded from disk. Changes are written to disk (persistent).
- Diskless and Stateless: The operating system is installed by using a different and contained method in the management node and loaded from the network. Changes are written to memory and discarded (not persistent).
- Diskless and Statelite: An intermediary type between stateless and stateful. The operating system is installed by using a different and contained method in the management node and loaded from the network. Changes are written to memory and can be stored (persistent) or discarded (not persistent). This method is not supported by xCAT 2.11.

For more information about operating system provisioning and state persistency, see the xCAT documentation page:

http://xcat-docs.readthedocs.io/en/2.11/guides/admin-guides/manage_clusters/ppc64le/index.html

5.3.7 xCAT network interfaces: Primary and additional

In xCAT terminology, a network adapter or interface[2] in a node can be either *primary* or *additional* (also known as *secondary*). Only one primary network adapter exists, and zero or more additional network adapters can exist.

The primary network interface of a node connects to the xCAT management network (that is, to the management node), and therefore is used to provision, boot, and manage that node.

An additional network interface connects to an xCAT network other than the xCAT management network and xCAT service network. Therefore, it is used by xCAT application networks, xCAT site networks or public networks, or for other purposes.

For more information, see the xCAT documentation page:

http://xcat-docs.readthedocs.io/en/2.11/guides/admin-guides/manage_clusters/ppc64le/diskless/customize_image/cfg_second_adapter.html

5.3.8 xCAT software kits

The xCAT provides support to a software package format that is called *xCAT Software Kits* (also known as *xCAT kits* or simply *kits*) that is tailored for installing software in xCAT clusters. Kits are used with some products of the IBM HPC Software stack.

An xCAT software kit can include a software product's distribution packages, configuration information, scripts, and other files. It also includes xCAT-specific information for installing the

[2] The term *network interface* can be more precise and unambiguous because a single network *adapter* can provide multiple network *interfaces* (for example, one interface per port, or virtual interfaces) to the operating system.

appropriate product pieces, which are referred to as the *kit components,* to a particular node according to its environment and role in the xCAT cluster.

The kits are distributed as *tarballs*, which are files with the *.tar* extension. The kits can be either *complete* or *incomplete* (which are also known as *partial*), which indicates whether a kit contains the packages of a product (complete) or not (incomplete/partial).

The incomplete or partial kits are indicated by filenames with the *NEED_PRODUCT_PKGS* string, which can be converted to complete kits when combined with the product's distribution packages. Incomplete or partial kits are usually distributed separately from the product's distribution media.

After a complete kit is added to the xCAT management node, its kit components can be assigned or installed to new and existing operating system installations.

For more information about xCAT Software Kits, see the xCAT documentation page:

http://xcat-docs.readthedocs.io/en/2.11/advanced/kit/index.html

5.3.9 xCAT version

This section describes xCAT version 2.11, which includes the following functionalities:

- RHEL Server 7.2 support for PowerPC 64-bit Little-Endian (ppc64le)
  - Provisioning types: Diskful/stateful and diskless/stateless
  - Support for CUDA Toolkit for NVIDIA GPUs
  - Support for Mellanox OFED for Linux for Mellanox InfiniBand adapters
  - Support for kits with the IBM HPC Software
  - Support for non-virtualized (or bare-metal) mode
- Power System S822LC server with OPAL firmware:
  - Hardware discovery for BMCs
  - Hardware management with IPMI

For more information about the xCAT 2.11 release, see the following website:

https://github.com/xcat2/xcat-core/wiki/XCAT_2.11_Release_Notes

Update: After this publication was written, xCAT 2.11.1 was announced. For more information and details, see the release notes at the following website:

https://github.com/xcat2/xcat-core/wiki/XCAT_2.11.1_Release_Notes

5.3.10 xCAT scenario

This chapter adopts the following xCAT networks and network addresses:

- Network address scheme: 10.*network-number*.0.0/16 (16-bit network mask)
- Management network (1 Gigabit Ethernet, or 1 GbE): 10.*1*.0.0/16
- Service network (1 Gigabit Ethernet): 10.*2*.0.0/16
- Application network (10 Gigabit Ethernet, or 10 GbE): 10.*3*.0.0/16
- Application network (10 Gigabit Ethernet): 10.*4*.0.0/16

- Application network (InfiniBand): 10.*5*.0.0/16
- Application network (InfiniBand): 10.*6*.0.0/16
- Site network (Ethernet): 9.114.37.0/24

> **Note:** Depending on the adapters in the systems, the number and type of network interfaces (for example, 2x 1 GbE, 2x 10 GbE, and 2x InfiniBand) can differ.
>
> This chapter describes the configuration of all the mentioned network interfaces, and the usage of 1 GbE and InfiniBand network interfaces.

The management and service networks are combined in a single network interface. They can be configured on different network interfaces. The network switch VLAN configuration can ensure that the management node can access all nodes' both in-band and out-of-band (BMC) network interfaces, and the non-management nodes can only access in-band network interfaces (but not out-of-band network interfaces).

The IP addresses are assigned to the nodes according to the following scheme:

- IP address: 10.*network-number.rack-number.node-number-in-rack*
- xCAT management node: 10.*network-number*.0.1
- Temporary IP addresses (the dynamic range): 10.*network-number*.254.*sequential-number*

The hostnames (or node names) are assigned to the POWER8 (thus, *p8*) compute nodes according to the following naming scheme:

- Node naming scheme: p8r<*rack-number*>n<*node-number-in-rack*>
- For example, for five racks and six systems per rack: `p8r1n1`, `p8r1n2`, ..., `p8r2n1`, `p8r2n2`,... `p8r5n6`

The following IPMI credentials are adopted for the BMCs:

- Username: `ADMIN`
- Password: `admin`

5.4 xCAT Management Node

This section describes the deployment of the xCAT Management Node with RHEL Server 7.2 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode on the IBM Power System S822LC server.

After you complete the steps in this section, the Management Node will be ready for the configuration and execution of the xCAT Node Discovery process on other nodes in the cluster.

For more information and details, see the xCAT *Installation Guide for Red Hat Enterprise Linux*, and the *Configure xCAT* section, at the following location and steps:

http://xcat-docs.readthedocs.org

- **Install Guides** → **Installation Guide for Red Hat Enterprise Linux**
- **Admin Guide** → **Manage Clusters** → **IBM Power LE / OpenPOWER** → **Configure xCAT**

5.4.1 RHEL server

You can install RHEL server with one of the following methods:

- Virtual media (with the BMC ASM interface)
- Network boot server
- USB device
- Network installation (for example, by way of HTTP)

Note: It is not possible to use optical media because optical drives are not supported.

The Knowledge Center provides instructions for the first three methods (virtual media, network boot server, and USB device), at this location:

http://ibm.com/support/knowledgecenter/linuxonibm/liabw/liabwinstalllc.htm

This chapter describes the last method (network installation by way of HTTP), which is officially supported by RHEL. It consists of making the ISO contents available in an HTTP server.

For more information and details, see the RHEL 7 *Installation Guide* (section 2.3.3.2 *Installation Source on an HTTP, HTTPS or FTP Server*), at this location:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Installation_Guide/sect-making-media-additional-sources.html#sect-making-media-sources-network

Note: The network installation method described requires another system with RHEL 7.

In case another system is not available, see the USB device method, and the RHEL 7 *Installation Guide* (section 2.2.1 *Making Installation USB Media*) for more instructions.

Instructions can differ for other Linux distributions.

Install the HTTP server

Install an HTTP server with the yum package manager:

```
# yum install httpd
```

Open ports in the firewall

Complete these steps to open ports in the firewall:

1. Check the firewall zone of the network interface for the HTTP server:

   ```
   # firewall-cmd --get-zone-of-interface=enP3p3s0f3
   ```

2. Add the HTTP service to that firewall zone:

   ```
   # firewall-cmd --permanent --zone=public --add-service=http
   success
   ```

3. Restart the firewall service:

   ```
   # systemctl restart firewalld
   ```

4. Confirm that the HTTP service is listed for that zone and network interface:

   ```
   # firewall-cmd --zone=public --list-all
   public (default, active)
   interfaces: enP3p3s0f3
   sources:
   ```

```
services: dhcpv6-client http ssh
ports:
masquerade: no
forward-ports:
icmp-blocks:
```

Download the installation ISO

Download the RHEL Server 7.2 installation ISO at this location:

https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.2/x86_64/product-software

> **Note:** The *Download Red Hat Enterprise Linux* page did not provide ISO images for the PowerPC 64-bit Little-Endian (ppc64le) architecture at the time this publication was written.

Copy ISO contents to HTTP server

Complete these steps to copy the ISO contents to the HTTP server:

1. Mount the ISO in a temporary directory:

```
# mkdir /mnt/rhel7.2-install

# mount -o loop,ro -t iso9660 \
   RHEL-7.2-20151030.0-Server-ppc64le-dvd1.iso \
   /mnt/rhel7.2-install/
```

2. Copy the ISO contents to the HTTP server's document root:

```
# cp -r /mnt/rhel7.2-install/ /var/www/html/
```

3. Unmount the ISO:

```
# umount /mnt/rhel7.2-install/
```

4. Remove the temporary directory:

```
# rmdir /mnt/rhel7.2-install/
```

Create a Petitboot configuration file

Complete these steps to create a Perirboot configuration file:

1. Confirm that the kernel and initramfs files are present:

```
# ls -1 /var/www/html/rhel7.2-install/ppc/ppc64/
initrd.img
TRANS.TBL
upgrade.img
vmlinuz
```

2. Create a Petitboot configuration file, pointing to the HTTP server's network address and ISO contents:

```
# server=<ip-address>
# dir=/rhel7.2-install

# cat <<EOF >/var/www/html/rhel7.2-install.cfg
label RHEL Server 7.2 Network Installation via HTTP
    kernel http://$server/$dir/ppc/ppc64/vmlinuz
    initrd http://$server/$dir/ppc/ppc64/initrd.img
    append inst.repo=http://$server/$dir/
EOF
```

3. Confirm that the ISO contents are available in the HTTP server:

```
# curl http://$server/$dir
...
<html>
   <head>
      <title>Index of /rhel7.2-install</title>
   </head>
...
```

Power on

Complete these steps to power on the system:

1. Power on the system:

```
# ipmitool -I lanplus -H <bmc-ip> -U <username> -P <password> chassis power on
```

2. Open the console:

```
# ipmitool -I lanplus -H <bmc-ip> -U <username> -P <password> sol activate
```

3. Wait for the Petitboot panel (Example 5-6).

Example 5-6 Petitboot panel

```
Petitboot (dev.20151015)                    8335-GTA         0000000000000000


System information
System configuration
Language
Rescan devices
Retrieve config from URL
*Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
Welcome to Petitboot
```

Configure network interface for Petitboot

Complete these steps to configure network interface for Petitboot:

1. In the **Petitboot** panel, select **System configuration**.
2. In the **Petitboot System Configuration** panel (Example 5-7 on page 72), complete these steps:
   a. In the **Network** section, select one of the following options, according to your network configuration (either dynamic or static IP address):
      - DHCP on all active interfaces
      - DHCP on a specific interface
      - Static IP configuration
   b. In the **Device** section, select the network interface for accessing the HTTP server. (if you selected either **DHCP on a specific interface** or **Static IP configuration**)
   c. In the **IP/mask, Gateway, and DNS Server(s)** fields, provide the values according to your network configuration (if you selected **Static IP configuration**).
   d. Press **OK**.

Example 5-7 Petitboot System Configuration panel, Static IP configuration

```
Petitboot System Configuration

Network:       ( ) DHCP on all active interfaces
               ( ) DHCP on a specific interface
               (*) Static IP configuration

Device:        ( ) enP1p3s0f0 [98:be:94:59:fa:24, link down]
               ( ) enP1p3s0f1 [98:be:94:59:fa:25, link down]
               ( ) enP1p3s0f2 [98:be:94:59:fa:26, link up]
               (*) enP1p3s0f3 [98:be:94:59:fa:27, link up]
               ( ) tunl0 [00:00:00:00:08:00, link up]

IP/mask:       9.114.37.179     / 24            (eg. 192.168.0.10 / 24)
Gateway:       9.114.37.254                     (eg. 192.168.0.1)
DNS Server(s): 9.12.16.2                        (eg. 192.168.0.2)

Disk R/W:      ( ) Prevent all writes to disk
               (*) Allow bootloader scripts to modify disks

               [    OK    ]  [   Help   ]  [  Cancel  ]

tab=next, shift+tab=previous, x=exit, h=help
```

Note: Alternatively, you can perform the network configuration steps in the Petitboot shell.

For the static IP network configuration in Example 5-7:

```
# ip link set enP1p3s0f3 up
# ip addr add 9.114.37.179/24 dev enP1p3s0f3
# ip route add default via 9.114.37.254
# echo nameserver 9.12.16.2 >/etc/resolv.conf
```

Download Petitboot configuration file

Complete these steps to download the Petitboot configuration file:

1. In the **Petitboot** panel, select **Retrieve config from URL**.
2. In the **Petitboot Config Retrieval** panel (Example 5-8).
   a. In the **Configuration URL** field, provide the address of the Petitboot configuration file on the HTTP server.
   b. Press **OK**.

Example 5-8 Petitboot Config Retrieval panel

```
Petitboot Config Retrieval

Configuration URL:     http://9.114.37.180/rhel7.2-install.cfg

                       [    OK    ]  [   Help   ]  [  Cancel  ]

tab=next, shift+tab=previous, x=exit, h=help
```

3. In the **Petitboot** panel, the boot entry for the Petitboot configuration file is present, if it can be correctly downloaded and parsed as shown in Example 5-9.

Example 5-9 Petitboot configuration information

```
Petitboot (dev.20151015)                    8335-GTA         0000000000000000

[Network: enP1p3s0f3 / 98:be:94:59:fa:27]
       RHEL Server 7.2 Network Installation via HTTP

System information
System configuration
Language
Rescan devices
*Retrieve config from URL
Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
Info: Config file rhel7.2-install.cfg parsed
```

Configure network boot options for RHEL installation

To configure network boot options for RHEL installation, complete these steps:

1. In the **Petitboot** panel, place the cursor in **RHEL Server 7.2 Network Installation via HTTP**, and press the *e* key (edit).
2. In the **Petitboot Option Editor** panel (Example 5-10), complete these steps:
   a. In the **Boot arguments** field, provide the *network boot options* (for the installer) for the network interface accessing the HTTP server, according to your network configuration.

      The boot options used are: **`ip`**, **`nameserver`** (optional), **`ifname`** (optional). The boot option **`inst.repo`** has been provided in the Petitboot configuration file.

      > **Note:** The network boot options are required in order for the `initramfs` to configure the network interface at boot time, and download files for starting the installation.
      >
      > For more information and details about network boot options, see the RHEL 7 *Installation Guide* (chapter 20 *Boot options*, section *Network boot options*) at the following website:
      >
      > http://red.ht/25OX76a

      For example, for the network configuration in Example 5-7 on page 72:

      - **`ifname`**=**eth0**:98:be:94:59:fa:27
      - **`ip`**=**9.114.37.179**::**9.114.37.254**:**255.255.255.0**::**eth0**:none
      - **`nameserver`**=**9.12.16.2**

        The options are listed in multiple lines for clarity, but are entered in a single line in the **Boot arguments** field.

   b. Press **OK**.

Example 5-10 Petitboot Option Editor panel

```
Petitboot Option Editor

Device:       (*) Specify paths/URLs manually
```

```
Kernel:         http://9.114.37.180/rhel7.2-install/ppc/ppc64/vmlinuz
Initrd:         http://9.114.37.180/rhel7.2-install/ppc/ppc64/initrd.img
Device tree:
Boot arguments: inst.repo=http://9.114.37.180/rhel7.2-install/ <other options>

               [    OK    ]  [   Help   ]  [  Cancel  ]

tab=next, shift+tab=previous, x=exit, h=help
```

Boot network installation

Complete these steps to boot the network installation:

1. In the **Petitboot** panel, select **RHEL Server 7.2 Network Installation via HTTP**.
2. Wait for the installer to start (Example 5-11).

Example 5-11 Petitboot panel, booting the RHEL Server 7.2 Network Installation

```
Petitboot (dev.20151015)                    8335-GTA         0000000000000000

[Network: enP1p3s0f3 / 98:be:94:59:fa:27]
*  RHEL 7.2 Network Install via HTTP

System information
System configuration
Language
Rescan devices
Retrieve config from URL
Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
The system is going down NOW!t
Sent SIGTERM to all processes
Sent SIGKILL to all processes
<...>
[...] OPAL V3 detected !
<...>
[...] Using PowerNV machine description
<...>
[...] systemd[1]: Detected architecture ppc64-le.
[...] systemd[1]: Running in initial RAM disk.

   Welcome to Red Hat Enterprise Linux ComputeNode 7.2 (Maipo)
dracut-033-359.el7 (Initramfs)!

<...>
[...] dracut-initqueue[1708]: % Total    % Received % Xferd  <...>
[...] dracut-initqueue[1708]: Dload  Upload   Total   Spent    Left  Speed
100  260M  100  260M    0     0   112M      0  <...>
<...>

Welcome to Red Hat Enterprise Linux ComputeNode 7.2 (Maipo)!

<...>
Starting installer, one moment...
```

```
anaconda 21.48.22.56-1 for Red Hat Enterprise Linux 7.2 started.
<...>
```

Note: Alternatively, you can skip the steps related to downloading Petitboot configuration file and configuring network boot options, and boot the network installation with the network boot options from the Petitboot shell.

For the configuration in Example 5-10 on page 73:

```
# wget http://9.114.37.180/rhel7.2-install/ppc/ppc64/vmlinuz
# wget http://9.114.37.180/rhel7.2-install/ppc/ppc64/initrd.img
# kexec \
   -l vmlinuz \
   -i initrd.img \
   -c 'inst.repo=http://9.114.37.180/rhel7.2-install/
   ip=9.114.37.179::9.114.37.254:255.255.255.0::eth0:none
   nameserver=9.12.16.2
   ifname=eth0:98:be:94:59:fa:27' \
   -e
```

Notice that the argument of the `-c` option has quotes, and no line breaks (just spaces).

Start VNC mode installation

The installation offers the option to start the VNC mode (remote graphical-based installation). You can start the VNC mode with the following steps (Example 5-12):

1. Select option **1) Start VNC** (type **1**, and press **Enter**).
2. Provide the VNC password, and confirm it.

The installation starts a VNC server.

Example 5-12 RHEL Server 7.2 Network Installation: Start VNC

```
Starting installer, one moment...
anaconda 21.48.22.56-1 for Red Hat Enterprise Linux 7.2 started.
<...>
================================================================================
================================================================================
VNC

Text mode provides a limited set of installation options. It does not offer cust
om partitioning for full control over the disk layout. Would you like to use VNC
mode instead?

1) Start VNC

2) Use text mode

Please make your choice from above ['q' to quit | 'c' to continue |
'r' to refresh]: 1
================================================================================
================================================================================
VNC Password

Please provide VNC password (must be six to eight characters long).
You will have to type it twice. Leave blank for no password
```

```
Password:
Password (confirm):
<...> Starting VNC...
<...> The VNC server is now running.
<...>

You chose to execute vnc with a password.

<...> Please manually connect your vnc client to <hostname>:1 (9.114.37.179:1) to
begin the install.
<...> Attempting to start vncconfig
```

Connect to VNC mode installation

You can connect to the VNC mode installation with a VNC client (viewer) from your workstation with the following command:

- Direct connection from workstation to system:

  ```
  $ vncviewer <system-address>:<display number>
  ```

  For example, using the values in Example 5-12 on page 75:

  ```
  $ vncviewer 9.114.37.179:1
  ```

- Indirect connection from workstation to system, with an intermediary system (SSH tunnel):

  ```
  $ ssh -N -L <local-port>:<system-address>:<5900+display> <intermediary-address>
  ```

  For example, using the system with the HTTP server as an intermediary system:

  ```
  $ ssh -N -L 55901:9.114.37.179:5901 9.114.37.180
  ```

  Then, connect to the local host/port:

  ```
  $ vncviewer 127.0.0.1:55901
  ```

Note: Linux distributions usually include packages that provide a VNC client.

You can use the package manager to search for VNC-related packages and install an example VNC client with the following commands:

Fedora-based Linux distributions (for example, RHEL, CentOS):

```
$ yum search vnc
$ sudo yum install tigervnc
```

Debian-based Linux distributions (for example, Ubuntu):

```
$ apt-cache search vnc
$ sudo apt-get install xtightvncviewer
```

The example VNC packages provide the **vncviewer** command.

After connected, the VNC client prompts for the password, and, upon successful authentication, provides some output (Example 5-13), and display a window with the graphical installation.

Example 5-13 RHEL Server 7.2 Network Installation: VNC client output

```
$ vncviewer 127.0.0.1:59001
Connected to RFB server, using protocol version 3.8
```

```
Performing standard VNC authentication
Password:
Authentication successful
Desktop name "Red Hat Enterprise Linux 7.2 installation on host
redbook01.pok.stglabs.ibm.com"
<...>
```

Install RHEL Server 7.2

Proceed with the RHEL Server 7.2 installation. For more information and details, see the RHEL 7 *Installation Guide* at the following website:

http://red.ht/25OX76a

When the installation is complete, it prompts you to reboot the system.

Boot the RHEL Server 7.2

After the system boot reaches the Petitboot bootloader, it lists boot entries for RHEL Server 7.2 (rescue and default modes), and automatically boots it after a few seconds, according to the system configuration for automatic boot (Example 5-14).

Example 5-14 RHEL Server 7.2: boot process from Petitboot to the Login prompt

```
Petitboot (dev.20151015)                    8335-GTA         0000000000000000

[Disk: sda2 / 1fa92838-dfd3-403b-994a-2c4c2ecee5cf]
     Red Hat Enterprise Linux Server (0-rescue-6e9bbfaeaa954952a3c89155d5746cb6)
   *  Red Hat Enterprise Linux Server (3.10.0-327.el7.ppc64le) 7.2 (Maipo)

System information
System configuration
Language
Rescan devices
Retrieve config from URL
Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
Info: Booting in 3 sec: Red Hat Enterprise Linux Server (3.10.0-327.el7.ppc64le
<...>
[...] OPAL V3 detected !
[...] Reserving 4096MB of memory at 128MB for crashkernel (System RAM: 262144MB)
[...] Using PowerNV machine description
<...>
[...] systemd[1]: Detected architecture ppc64-le.
<...>
Welcome to Red Hat Enterprise Linux Server 7.2 (Maipo)!
<...>
Red Hat Enterprise Linux Server 7.2 (Maipo)
Kernel 3.10.0-327.el7.ppc64le on an ppc64le

redbook01 login:
```

Configure RHEL Server 7.2 package repository

To install additional packages and satisfy package dependencies for xCAT, configure a *yum* package repository for the RHEL Server 7.2 packages.

You can configure the system for the RHEL regular package update channels, or at least, the RHEL Server 7.2 DVD 1 (or Binary DVD). For simplicity, this chapter describes the latter.

You can configure the package repository for RHEL Server 7.2 DVD 1 in one of the following ways:

- Contents in the HTTP server:

  a. Install the RPM GPG key:

  ```
  # rpm --import http://9.114.37.180/rhel7.2-install/RPM-GPG-KEY-redhat-release
  ```

  b. Create the yum package repository file:

  ```
  # cat <<EOF > /etc/yum.repos.d/rhel72-dvd1.repo
  [rhel-7.2-dvd1]
  name=RHEL Server 7.2 DVD1 (HTTP)
  baseurl=http://9.114.37.180/rhel7.2-install/
  enabled=1
  gpgcheck=1
  EOF
  ```

- Contents in local ISO file:

  a. Configure a mountpoint for the ISO:

  ```
  # cp /path/to/RHEL-7.2-20151030.0-Server-ppc64le-dvd1.iso /mnt/

  # echo '/mnt/RHEL-7.2-20151030.0-Server-ppc64le-dvd1.iso /mnt/rhel7.2-dvd1
  iso9660 defaults,loop,ro 0 0' >> /etc/fstab

  # mkdir /mnt/rhel7.2-dvd1
  # mount /mnt/rhel7.2-dvd1
  ```

  b. Install the RPM GPG key:

  ```
  # rpm --import /mnt/rhel7.2-dvd1/RPM-GPG-KEY-redhat-release
  ```

  c. Create the yum package repository file:

  ```
  # cat <<EOF >/etc/yum.repos.d/rhel7.2-dvd1.repo
  [rhel-7.2-dvd1]
  name=RHEL 7.2 Server DVD1 (ISO)
  baseurl=file:///mnt/rhel7.2-dvd1
  enabled=1
  gpgcheck=1
  EOF
  ```

You can verify that the package repository is configured correctly with the following command:

```
# yum repolist
<...>
rhel-7.2-dvd1                                  | 4.1 kB  00:00:00
(1/2): rhel-7.2-dvd1/group_gz                  | 132 kB  00:00:00
(2/2): rhel-7.2-dvd1/primary_db                | 2.9 MB  00:00:00
repo id          repo name                    status
rhel-7.2-dvd1    RHEL 7.2 Server DVD1 (ISO)   3,398
repolist: 3,398
```

Enable the rpcbind service

The xCAT installs and attempts to start the NFS server. However, the `nfs` service fails to start if the `rpcbind` service is not yet started as shown in Example 5-15. To resolve this problem, complete the following steps:

1. Install the `nfs-utils` package for the NFS server and services:

   ```
   # yum install nfs-utils
   ```

2. Enable the `rpcbind` service to start automatically on system boot:

   ```
   # systemctl enable rpcbind.service
   ```

3. Start the `rpcbind` service:

   ```
   # systemctl start rpcbind.service
   ```

Example 5-15 RHEL Server 7.2: nfs service fails to start if the rpcbind service is not running

```
# systemctl start nfs.service
Job for nfs-server.service failed <...>
See "systemctl status nfs-server.service" <...>

# systemctl status nfs-server.service
<...>
   Active: failed (Result: exit-code) since ...; 9s ago
  Process: ... ExecStart=/usr/sbin/rpc.nfsd ... (code=exited, status=1/FAILURE)
<...>
<...> systemd[1]: Starting NFS server and services...
<...> rpc.nfsd[...]: rpc.nfsd: writing fd to kernel failed: errno 111 (Connection
refused)
<...>
<...> systemd[1]: Failed to start NFS server and services.
<...>
```

The `nfs` service then starts correctly (Example 5-16).

Example 5-16 RHEL Server 7.2: nfs service starts correctly if the rpcbind service is running

```
# systemctl start nfs.service

# systemctl status nfs-server.service
<...>
   Active: active (exited) since ...; 3s ago
  Process: ... ExecStart=/usr/sbin/rpc.nfsd $... (code=exited, status=0/SUCCESS)
<...>
<...> systemd[1]: Starting NFS server and services...
<...> systemd[1]: Started NFS server and services.
```

Disable SELinux

The xCAT 2.11 requires SELinux to be disabled because it is enabled by default on the RHEL Server 7.2.

You can disable it with a change to the `/etc/selinux.conf` file, and a reboot (Example 5-15):

1. Verify SELinux is enabled (current status and configuration file):

   ```
   # getenforce
   Enforcing
   ```

```
# grep ^SELINUX= /etc/selinux/config
SELINUX=enforcing
```

2. Disable SELinux in the configuration file:

```
# sed 's/^SELINUX=.*/SELINUX=disabled/' -i /etc/selinux/config

# grep ^SELINUX= /etc/selinux/config
SELINUX=disabled
```

3. Reboot the system:

```
# reboot
```

4. Verify SELinux is disabled (current status):

```
# getenforce
Disabled
```

5.4.2 xCAT packages

The xCAT is a collection of packages that are available for download in the xCAT project page at the following website:

http://xcat.org/download.html

The xCAT packages are organized into two package repositories:

- xCAT Core Packages: Packages with the xCAT.

  This package repository is available in three streams (or types):

  - Release (or Stable) builds: The latest, officially released (general availability) version of xCAT.
  - Snapshot Builds: Unreleased changes for the next refresh of current version of xCAT.
  - Development Builds: Unreleased changes for the next version of xCAT.
- xCAT Dependency Packages: Required packages that are not provided by the Linux distribution.

Each package repository is available either as an online repository or local (or offline) repository, both as RPM packages (for RHEL and SUSE Linux Enterprise Server (SLES)) and Debian packages (for Ubuntu).

This chapter describes the local repository method with RPM packages.

> **Note:** For the online repository method, download the `xCAT-core.repo` and `xCAT-dep.repo` files into the `/etc/yum.repos.d` directory.

Create local package repositories by using these steps:

1. Install the `bzip2` package, which is not installed with the minimal installation package set:

```
# yum install bzip2
```

2. Create a local package repository for xCAT Core Packages (xCAT-core) Release tarball by using these steps:

   a. Download and extract the xCAT-core tarball:

```
# mkdir -p /root/software/xcat
# cd /root/software/xcat
```

```
# curl -sOL http://xcat.org/files/xcat/xcat-core/2.11.x_Linux/xcat-core/
xcat-core-2.11-linux.tar.bz2

# tar xf xcat-core-2.11-linux.tar.bz2
```

b. Run the `mklocalrepo.sh` script:

```
# ./xcat-core/mklocalrepo.sh
/root/software/xcat
```

This process creates the `xCAT-core.repo` file in the `/etc/yum.repos.d` directory:

```
# cat /etc/yum.repos.d/xCAT-core.repo
[xcat-2-core]
name=xCAT 2 Core packages
baseurl=file:///root/software/xcat/xcat-core
enabled=1
gpgcheck=1
gpgkey=file:///root/software/xcat/xcat-core/repodata/repomd.xml.key
```

c. Import the RPM GPG key:

```
# rpm --import xcat-core/repodata/repomd.xml.key
```

d. Verify that the package repository is configured correctly:

```
# yum repolist
<...>
repo id          repo name                    status
rhel-7.2-dvd1    RHEL 7.2 Server DVD1 (ISO)   3,398
xcat-2-core      xCAT 2 Core packages         18
repolist: 3,416
```

3. Create a local package repository for xCAT Dependency Packages (xCAT-deps) tarball:

a. Download and extract the xCAT-deps tarball:

```
# curl -sOL http://xcat.org/files/xcat/xcat-dep/2.x_Linux/
xcat-dep-2.11.tar.bz2

# tar xf xcat-dep-2.11.tar.bz2
```

b. Run the `mklocalrepo.sh` script for this Linux distribution and processor architecture:

```
# ./xcat-dep/rh7/ppc64le/mklocalrepo.sh
/root/software/xcat
```

This process creates the `xCAT-dep.repo` file in the `/etc/yum.repos.d` directory:

```
# cat /etc/yum.repos.d/xCAT-dep.repo
[xcat-dep]
name=xCAT 2 depedencies
baseurl=file:///root/software/xcat/xcat-dep/rh7/ppc64le
enabled=1
gpgcheck=1
gpgkey=file:///root/software/xcat/xcat-dep/rh7/ppc64le/repodata/repomd.xml.k
ey
```

c. Import the RPM GPG key:

```
# rpm --import xcat-dep/rh7/ppc64le/repodata/repomd.xml.key
```

d. Verify that the package repository is configured correctly:

```
# yum repolist
<...>
```

```
repo id          repo name                    status
rhel-7.2-dvd1    RHEL 7.2 Server DVD1 (ISO)   3,398
xcat-2-core      xCAT 2 Core packages         18
xcat-dep         xCAT 2 depedencies           30
repolist: 3,446
```

Install xCAT packages

Complete these steps to install the xCAT packages:

1. Install the `xCAT` package.

   This process installs the xCAT and any required package dependencies, and performs some initialization steps for the xCAT (Example 5-17):

   Example 5-17 xCAT: Package installation

```
# yum install xCAT
<...>
Is this ok [y/d/N]: y
<...>
  Installing : xCAT-2.11-snap201511300543.ppc64le
<...>
Generating new node hostkeys...
<...>
NFS has been restarted.
<...>
Created xCAT certificate.
<...>
Restarting xcatd (via systemctl):  [  OK  ]
dns server has been enabled on boot.
<...>
httpd has been restarted.
xCAT is now running, it is recommended to tabedit networks
<...>
Running '/opt/xcat/sbin/mknb ppc64', triggered by the installation/update of
xCAT-genesis-scripts-ppc64 ...
Creating genesis.fs.ppc64.gz in /tftpboot/xcat
The 'mknb ppc64' command completed successfully.
<...>
Installed:
  xCAT.ppc64le 0:2.11-snap201511300543
<...>
Complete!
```

2. Verify that the xCAT service is running:

```
# systemctl status xcatd.service
xcatd.service - LSB: xCATd
   Loaded: loaded (/etc/rc.d/init.d/xcatd)
   Active: active (running) since ...; 4min 37s ago
<...>
```

3. Verify the version information and node type:

```
# source /etc/profile.d/xcat.sh

# lsxcatd -a
```

```
Version 2.11 (git commit 9ea36ca6163392bf9ab684830217f017193815be, built Mon
Nov 30 05:43:11 EST 2015)
This is a Management Node
dbengine=SQLite
```

Note: The **source** command for the `xcat.sh` file is only required on current login shells.

Configure logging to /var/log/messages

At the time of writing, an xCAT issue[3] disabled any logging to the `/var/log/messages` file at installation time. In order to restore the default behavior, complete the following steps:

1. Verify logging to */var/log/messages* is disabled (line commented with leading # symbol):

```
# grep '\*\.info' /etc/rsyslog.conf
#*.info;mail.none;authpriv.none;cron.none                /var/log/messages
```

2. Remove the leading # symbol from the line:

```
# sed '/^#\*\.info/ s/^#//' -i /etc/rsyslog.conf
```

3. Verify that the leading # symbol is removed from the line:

```
# grep '\*\.info' /etc/rsyslog.conf
*.info;mail.none;authpriv.none;cron.none                /var/log/messages
```

4. Restart the `rsyslog` service:

```
# systemctl restart rsyslog.service
```

5. Verify that logging is restored to `/var/log/messages`:

```
# tail /var/log/messages
<...>
<...> systemd: Started System Logging Service.
```

5.4.3 Static IP network configuration

The xCAT requires static IP network configuration for the Management Node.

Note: This requirement applies to any xCAT networks with communication between the Management Node and other nodes (for example, Management and Service Networks).

You can configure an Ethernet network interface with static IP address in one of many ways. This chapter describes the method that uses `sysconfig` or `ifcfg` files, and the **nmcli** command (Network Manager Command Line Interface).

For more information and details, see the RHEL 7 *Networking Guide*, at this location and navigation steps:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/index.html

This section employs content from the following sections of the RHEL 7 *Networking Guide*:

- Section 1.9 Network configuration using sysconfig files
- Section 2.4.1 Configuring a network interface with ifcfg files

[3] https://github.com/xcat2/xcat-core/issues/438

In order to configure a network interface with static IP address, complete the following steps:

- For the Management Network:

  a. Create the `/etc/sysconfig/network-scripts/ifcfg-<network-interface>` file.

     For the scenario described in this chapter, the file looks like this:

     ```
     # cat <<EOF >/etc/sysconfig/network-scripts/ifcfg-enP3p3s0f2
     DEVICE=enP3p3s0f2
     ONBOOT=yes
     BOOTPROTO=none
     IPADDR=10.1.0.1
     PREFIX=16
     IPV6INIT=yes
     EOF
     ```

  b. Verify that the network configuration is not in effect immediately (no IPv4 or IPv6 address):

     ```
     # ip addr show enP3p3s0f2
     4: enP3p3s0f2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
     qlen 1000
         link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
     ```

  c. Reload the network configuration for that network interface with the **`nmcli`** command.

     The network configuration is loaded automatically on system boot.

     ```
     # nmcli connection load /etc/sysconfig/network-scripts/ifcfg-enP3p3s0f2
     ```

  d. Verify that the network configuration is in effect (including an IPv6 link-local address):

     ```
     # ip addr show enP3p3s0f2
     4: enP3p3s0f2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
     qlen 1000
         link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
         inet 10.1.0.1/16 brd 10.1.255.255 scope global enP3p3s0f2
            valid_lft forever preferred_lft forever
         inet6 fe80::9abe:94ff:fe59:fa26/64 scope link
            valid_lft forever preferred_lft forever
     ```

- For the Service Network:

  Depending on your network environment and configuration, the Management Node can use different or shared network interfaces for the Management and Service Networks.

  – For different network interfaces, perform the steps for the Management Network, and replace the network interface and configuration for the Service Network.

  – For shared network interface, modify the respective `ifcfg` file, and define `IPADDR<n>` and `PREFIX<n>` (or `NETMASK<n>`) options for each network.

  For more information and details, see the Red Hat Knowledgebase Solution article[4] *How to configure multiple IP addresses on a single interface?* and the documentation for `ifcfg` files in the `/usr/share/doc/initscripts-*/sysconfig.txt` file.

  For the scenario described in this chapter, the file looks like this:

  ```
  # cat <<EOF >/etc/sysconfig/network-scripts/ifcfg-enP3p3s0f2
  DEVICE=enP3p3s0f2
  ONBOOT=yes
  BOOTPROTO=none
  IPADDR0=10.1.0.1
  ```

[4] https://access.redhat.com/solutions/915193

```
PREFIX0=16
IPADDR1=10.2.0.1
PREFIX1=16
IPV6INIT=yes
EOF
```

a. Reload the network configuration for that network interface with the **nmcli** command.

   The network configuration is loaded automatically on system boot.

```
# nmcli connection load /etc/sysconfig/network-scripts/ifcfg-enP3p3s0f2
```

b. Apply the network configuration changes with the **ifdown** and **ifup** commands:

```
# ifdown enP3p3s0f2
# ifup enP3p3s0f2
```

c. Verify that the network configuration is in effect (including the other IPv4 address):

```
# ip addr show enP3p3s0f2
4: enP3p3s0f2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
qlen 1000
    link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
    inet 10.1.0.1/16 brd 10.1.255.255 scope global enP3p3s0f2
       valid_lft forever preferred_lft forever
    inet 10.2.0.1/16 brd 10.2.255.255 scope global enP3p3s0f2
       valid_lft forever preferred_lft forever
    inet6 fe80::9abe:94ff:fe59:fa26/64 scope link
       valid_lft forever preferred_lft forever
```

5.4.4 Hostname and aliases

Configure the hostname to be resolved to the IP address in the Management Network by completing these steps:

1. Configure the hostname in the `/etc/hostname` file:

```
# echo 'xcat-mn.xcat-cluster' > /etc/hostname
```

2. Add the host aliases in the `/etc/hosts` file:

```
# echo '10.1.0.1 xcat-mn.xcat-cluster xcat-mn' >> /etc/hosts
```

3. Verify the short and long (fully qualified domain name) hostnames are detected:

```
# hostname --short
xcat-mn

# hostname --long
xcat-mn.xcat-cluster
```

4. Verify that the short hostname resolves to the long hostname, and the **ping** test works:

```
# ping -c1 xcat-mn
PING xcat-mn.xcat-cluster (10.1.0.1) 56(84) bytes of data.
64 bytes from xcat-mn.xcat-cluster (10.1.0.1): icmp_seq=1 ttl=64 time=0.025 ms

--- xcat-mn.xcat-cluster ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.025/0.025/0.025/0.000 ms
```

5.4.5 xCAT networks

The xCAT networks configuration is stored the `networks` table, and is also available as objects of the `network` type.

The **`makenetworks`** command populates the `networks` table based on the current configuration of the network interfaces. It is run automatically during the installation of xCAT packages.

You can use the following commands to create, list, modify, and remove, `network` objects, and list, and edit, the `networks` table, respectively:

- **`mkdef`** `-t network`
- **`lsdef`** `-t network`
- **`chdef`** `-t network`
- **`rmdef`** `-t network`
- **`tabdump`** `networks`
- **`tabedit`** `networks`

To configure the xCAT networks, populate the `networks` table with the **`makenetworks`** command, remove any non-xCAT networks, rename the xCAT networks (optional), and create any other xCAT networks.

For the scenario described in this chapter, complete these steps:

1. Populate the `networks` table with the **`makenetworks`** command:

```
# makenetworks

# lsdef -t network
10_1_0_0-255_255_0_0  (network)
10_2_0_0-255_255_0_0  (network)
9_114_37_0-255_255_255_0  (network)
fd55:faaf:e1ab:33e::/64  (network)
```

2. Remove any non-xCAT networks:

```
# rmdef -t network 9_114_37_0-255_255_255_0
1 object definitions have been removed.

# rmdef -t network fd55:faaf:e1ab:33e::/64
1 object definitions have been removed.

# lsdef -t network
10_1_0_0-255_255_0_0  (network)
10_2_0_0-255_255_0_0  (network)
```

3. Rename the xCAT networks (optional):

```
# chdef -t network 10_1_0_0-255_255_0_0 -n net-mgmt
Changed the object name from 10_1_0_0-255_255_0_0 to net-mgmt.

# chdef -t network 10_2_0_0-255_255_0_0 -n net-svc
Changed the object name from 10_1_0_0-255_255_0_0 to net-mgmt.

# lsdef -t network
net-mgmt  (network)
net-svc (network)
```

4. Create any other xCAT networks:

```
# mkdef -t network net-app-10gbe-port1 net=10.3.0.0 mask=255.255.0.0
1 object definitions have been created or modified.

# mkdef -t network net-app-10gbe-port2 net=10.4.0.0 mask=255.255.0.0
1 object definitions have been created or modified.

# mkdef -t network net-app-ib-port1 net=10.5.0.0 mask=255.255.0.0
1 object definitions have been created or modified.

# mkdef -t network net-app-ib-port2 net=10.6.0.0 mask=255.255.0.0
1 object definitions have been created or modified.

# lsdef -t network
net-app-10gbe-port1  (network)
net-app-10gbe-port2  (network)
net-app-ib-port1  (network)
net-app-ib-port2  (network)
net-mgmt  (network)
net-svc  (network)
```

Note: The Application Networks lack the `mgtifname` attribute (*management interface name*) because the Management Node is not connected to them (that is, only some other nodes are, such as the Compute Nodes, and Login Nodes).

However, Application Networks must be defined in the Management Node for it to perform their network configuration on other nodes (via Management Network).

5. Verify the xCAT networks configuration.

   You can use the **`lsdef`** command to list the configuration of a specific xCAT network or networks. The following example lists the Management Network and Application Network for InfiniBand port 1:

```
# lsdef -t network net-mgmt,net-app-ib-port1
Object name: net-mgmt
    gateway=<xcatmaster>
    mask=255.255.0.0
    mgtifname=enP3p3s0f2
    net=10.1.0.0
    tftpserver=10.1.0.1
Object name: net-app-ib-port1
    mask=255.255.0.0
    net=10.5.0.0
```

   You can use the **`tabdump`** command to list the configuration of all xCAT networks:

```
# tabdump networks
#netname,net,mask,mgtifname,<...>
"net-mgmt","10.1.0.0","255.255.0.0","enP3p3s0f2",<...>
"net-svc","10.2.0.0","255.255.0.0","enP3p3s0f2",<...>
"net-app-10gbe-port1","10.3.0.0","255.255.0.0",,,,,,,,,,,,,,,
"net-app-10gbe-port2","10.4.0.0","255.255.0.0",,,,,,,,,,,,,,,
"net-app-ib-port1","10.5.0.0","255.255.0.0",,,,,,,,,,,,,,,
"net-app-ib-port2","10.6.0.0","255.255.0.0",,,,,,,,,,,,,,,
```

5.4.6 DNS server

The xCAT configures the DNS server based on attributes of the `site` table, the `/etc/hosts` file, and node definitions. The **makedns** command applies the configuration.

The following attributes of the `site` table are used to configure the DNS server:

- `dnsinterfaces`: hostname (optional) and network interfaces for the DNS server to listen on.
- `domain`: DNS domain name for the cluster.
- `forwarders`: DNS servers for resolving non-cluster names, that is, the site's or external DNS servers.
- `master`: IP address of the xCAT management node on the management network, as known by the nodes.
- `nameservers`: DNS servers that are used by the compute nodes. Usually, the IP address of the management node. The value <*xcatmaster*> indicates that the management node or service node that is managing a node (automatically defined to the correct IP address in the respective xCAT network) is more portable.

For more information, see the manual page of the `site` table with the following command:

```
# man 5 site
```

The **makedns** command generates the following configuration files for the DNS server, and reloads it:

- `/etc/named.conf`: main configuration file (generated by **makedns -n**).
- `/var/named/*`: zone files for network names and addresses (generated by **makedns -n**)

To perform the configuration of the DNS server, complete the following steps:

1. Set the `dnsinterfaces`, `domain`, `forwarders`, `master`, and `nameservers` attributes of the `site` table with the **chdef** command.

   For the scenario described in this chapter:

   ```
   # chdef -t site \
        dnsinterfaces='xcat-mn|enP3p3s0f2' \
        domain=xcat-cluster \
        forwarders=9.12.16.2 \
        master=10.1.0.1 \
        nameservers=10.1.0.1
   1 object definitions have been created or modified.
   ```

   You can verify the attributes with the **lsdef** command:

   ```
   # lsdef -t site -i dnsinterfaces,domain,forwarders,master,nameservers
   Object name: clustersite
       dnsinterfaces=xcat-mn|enP3p3s0f2
       domain=xcat-cluster
       forwarders=9.12.16.2
       master=10.1.0.1
       nameservers=10.1.0.1
   ```

2. Generate new configuration files for the DNS server with the **makedns -n** command.

   The DNS server is automatically (re)started.

   ```
   # makedns -n
   Handling xcat-mn.xcat-cluster in /etc/hosts.
   ```

```
Handling localhost in /etc/hosts.
Handling localhost in /etc/hosts.
Getting reverse zones, this  take several minutes for a large cluster.
Completed getting reverse zones.
Updating zones.
Completed updating zones.
Restarting named
Restarting named complete
Updating DNS records, this  take several minutes for a large cluster.
Completed updating DNS records.
```

> **Note:** It is important that no errors are reported in the output of the **makedns** command. The proper functioning of the DNS server is essential to several features in the xCAT (for example, node discovery).
>
> If any errors are reported, check the messages, the contents of the `/etc/hosts` file, and any existing node definitions (with the **lsdef** command) for errors or inconsistencies.

3. Verify that the DNS server is resolving internal and external names with the **host** command by completing these steps:
   a. Install the `bind-utils` package (not installed with the minimal installation package set):

      ```
      # yum install bind-utils
      ```

   b. Verify the name resolution of internal names:

      For example, the Management Node (that is, its short and long hostnames are associated, and are resolved to its IP address in the Management Network):

      ```
      # host xcat-mn 10.1.0.1
      Using domain server:
      Name: 10.1.0.1
      Address: 10.1.0.1#53
      Aliases:

      xcat-mn.xcat-cluster has address 10.1.0.1
      ```

   c. Verify the name resolution of external names as well:

      ```
      # host example.com 10.1.0.1
      Using domain server:
      Name: 10.1.0.1
      Address: 10.1.0.1#53
      Aliases:

      example.com has address 93.184.216.34
      example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
      ```

5.4.7 DHCP server

The xCAT configures the DHCP server based on attributes of the `site` and `networks` tables, and the node definitions (for reservation of IP address leases based on MAC address). The **makedhcp** command applies the configuration.

The following attributes of the `site` and `networks` tables are used to configure the DHCP server:

- `dhcpinterfaces` (`site` table)

  Hostname (optional) and network interfaces for the DHCP server to listen on.

- `dynamicrange` (`networks` table)

  Range of IP addresses that are *temporarily* assigned during the node discovery process, which are required in the xCAT management and service networks.

The **makedhcp** command generates the configuration files for the DHCP server, and reloads it:

- `/etc/dhcp/dhcpd.conf`: Main configuration file (generated by **makedhcp -n**)
- `/var/lib/dhcpd/dhcpd.leases`: IP address leases (generated by **makedhcp -a**)

To configure of the DHCP server, complete the following steps:

1. Set the `dhcpinterfaces` attribute of the `site` table with the **chdef** command.

   For the scenario described in this chapter:

   ```
   # chdef -t site dhcpinterfaces='xcat-mn|enP3p3s0f2'
   1 object definitions have been created or modified.
   ```

   You can verify the attributes with the **lsdef** command:

   ```
   # lsdef -t site -i dhcpinterfaces
   Object name: clustersite
       dhcpinterfaces=xcat-mn|enP3p3s0f2
   ```

2. Generate the configuration file for the DHCP server with the **makedhcp -n** command.

   The DHCP server is automatically (re)started.

   ```
   # makedhcp -n
   Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak

   The dhcp server must be restarted for OMAPI function to work
   Warning: No dynamic range specified for 10.1.0.0. If hardware discovery is
   being used, a dynamic range is required.
   Warning: No dynamic range specified for 10.2.0.0. If hardware discovery is
   being used, a dynamic range is required.
   ```

   Despite the message related to the need to restart the DHCP server, it is automatically restarted, as noticed in the `/var/log/messages` file:

   ```
   # tail /var/log/messages
   <...>
   <...> xcat[...]: xCAT: Allowing makedhcp -n for root from localhost
   <...> systemd: Starting DHCPv4 Server Daemon...
   <...> dhcpd: Internet Systems Consortium DHCP Server 4.2.5
   <...>
   ```

3. Generate the leases file for the DHCP server with the **makedhcp -a** command. This step is only required if any node definitions exist (they do not yet exist in the scenario in this chapter).

   ```
   # makedhcp -a
   ```

5.4.8 IPMI authentication credentials

The xCAT configures the authentication credentials for IPMI commands (for example, power management, console sessions, BMC discovery, and network configuration) based on attributes from node definitions, node groups definitions, and the `passwd` table (in this order).

To configure IPMI authentication credentials on individual nodes or on node groups, set the `bmcusername` and `bmcpassword` attributes on the node or node group object with the **chdef** command:

```
# chdef <node or group> bmcusername=<IPMI username> bmcpassword=<IPMI password>
```

If the IPMI authentication credentials are common across some or all of the systems' baseboard management controllers (BMCs), you can set the common credentials in the `passwd` table. Any different credentials can be set in the respective node or node group objects.

To configure the IPMI authentication credentials, complete the following steps:

1. Set the `username` and `password` attributes of the `ipmi` key/row in the `passwd` table with either one of the **chtab** or **tabedit** commands.

   For the scenario adopted in this chapter:

   ```
   # chtab key=ipmi passwd.username=ADMIN passwd.password=admin
   ```

2. Verify the setting with the **tabdump** command:

   ```
   # tabdump -w key==ipmi passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "ipmi","ADMIN","admin",,,,
   ```

   You can use the **tabdump** command without filter arguments to list the configuration of all entries in the `passwd` table:

   ```
   # tabdump passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "omapi","xcat_key","<...>=",,,,
   "ipmi","ADMIN","admin",,,,
   ```

   Note: If the IPMI authentication credentials are not set or invalid, some IPMI-based commands can show errors like the following:

   ```
   # rpower node status
   node: Error: Unauthorized name
   ```

5.5 xCAT Node Discovery

This section describes the xCAT Node Discovery (or Hardware Discovery) process. It covers the configuration steps that are required in the Management Node, and instructions for performing the discovery of nodes in the cluster. For more information, see 5.3.4, "xCAT node discovery" on page 64.

The xCAT provides the following methods for node discovery:

- Manual definition: Manual hardware information collection and node object definition. This example includes required node-specific and xCAT/platform-generic attributes:

  ```
  # mkdef node1 \
     groups=all,s822lc \
  ```

```
ip=10.1.1.1 mac=6c:ae:8b:6a:d4:e installnic=mac primarynic=mac \
bmc=10.2.1.1 bmcusername=ADMIN bmcpassword=admin \
mgt=ipmi cons=ipmi netboot=petitboot
```

- **M**achine **T**ype and **M**odel, and **S**erial Number (MTMS)-based discovery: Automatically collects MTM and SN information from the node's BMC and operating system (Genesis), and match it with a minimal manually defined node object (with `mtm` and `serial` attributes). This process automatically stores the hardware information in the matched node object.
- Sequential discovery: Automatically stores hardware information in a list of minimal node objects (with no attributes) in a sequential manner, in the order the nodes are booted.
- Switch-based discovery: Automatically identifies the network switch and port for the node (with the SNPMv3 protocol), and matches it with a minimal manually defined node object (with `switch` and `switchport` attributes). This process automatically stores the hardware information in the matched node object.

This chapter describes the MTMS-based discovery method.

5.5.1 Verification of network boot configuration and Genesis image files

The xCAT node discovery requires the node to boot the Genesis image from the network. For more information, see 5.3.4, "xCAT node discovery" on page 64. It is important to verify that the files for network boot configuration and Genesis image are correctly in place.

The **mknb** (*make network boot*) command generates the network boot configuration file (specified in the `/etc/dhcp/dhcpd.conf` file) and Genesis image files. It is run automatically during the installation of xCAT packages.

In order to verify and generate the files for network boot configuration and Genesis image, perform the following steps:

1. Verify the location of the platform-specific network boot configuration file in the `/etc/dhcp/dhcpd.conf` file.

   The scenario adopted in this chapter uses OPAL firmware:

   ```
   # grep -A1 OPAL /etc/dhcp/dhcpd.conf
      } else if option client-architecture = 00:0e { #OPAL-v3
         option conf-file = "http://10.1.0.1/tftpboot/pxelinux.cfg/p/10.1.0.0_16";
   --
      } else if option client-architecture = 00:0e { #OPAL-v3
         option conf-file = "http://10.2.0.1/tftpboot/pxelinux.cfg/p/10.2.0.0_16";
   ```

   Note: The network boot configuration and Genesis image files are required only for the xCAT Management Network.

2. Verify that the file exists.

   The specified file might not exist in some cases, such as if the **mknb** command is not run after a change in the xCAT networks configuration, which is the scenario in this chapter.

   ```
   # cat /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   cat: /tftpboot/pxelinux.cfg/p/10.1.0.0_16: No such file or directory
   ```

3. If the file does not exist, you can generate it and the Genesis image with the **mknb** command for the `ppc64` architecture:

   ```
   # mknb ppc64
   Creating genesis.fs.ppc64.gz in /tftpboot/xcat
   ```

4. Verify that the file exists. It contains pointers to the platform-specific Genesis image files.

```
# cat /tftpboot/pxelinux.cfg/p/10.1.0.0_16
default xCAT
   label xCAT
   kernel http://10.1.0.1:80//tftpboot/xcat/genesis.kernel.ppc64
   initrd http://10.1.0.1:80//tftpboot/xcat/genesis.fs.ppc64.gz
   append "quiet xcatd=10.1.0.1:3001 "
```

5. Verify that the files of the Genesis image exist:

```
# ls -lh /tftpboot/xcat/genesis.kernel.ppc64 /tftpboot/xcat/genesis.fs.ppc64.gz
-rw-r--r-- 1 root root 48M Nov 24 22:47 /tftpboot/xcat/genesis.fs.ppc64.gz
-rwxr-xr-x 1 root root 23M Nov 16 08:19 /tftpboot/xcat/genesis.kernel.ppc64
```

Tip: To increase the verbosity of the node discovery in the nodes (useful for educational and debugging purposes), remove the `quiet` argument from the `append` line in the network boot configuration file. You can do that with the following command:

```
# sed 's/quiet//' -i /tftpboot/pxelinux.cfg/p/10.1.0.0_16
```

5.5.2 Configuration of the DHCP dynamic range

The xCAT node discovery requires temporary IP addresses for nodes and BMCs until the association with the respective node objects, and permanent network configuration occur. The IP address range that is reserved for that purpose is known as *dynamic range*, an attribute of network objects, reflected in the configuration file of the DHCP server.

You need to provide temporary IP addresses for the Management and Service Networks to handle both the *in-band* network interface (used by the Petitboot bootloader, and Genesis image) and *out-of-band* network interface (used by the BMC).

Depending on your network environment and configuration, the Management Node can use different or shared network interfaces for the Management and Service Networks. This is an important consideration because the dynamic range is defined *per network interface* in the configuration file of the DHCP server.

- For different network interfaces, set the `dynamicrange` attribute on both the Management and Service Networks.
- For shared network interface, set the `dynamicrange` attribute in either one of the Management or Service Networks.

To set the dynamic range, complete the following steps:

1. Set the `dynamicrange` attribute for the Management Network with the **`chdef`** command.

   For the scenario adopted in this chapter:

```
# chdef -t network net-mgmt dynamicrange=10.1.254.1-10.1.254.254
1 object definitions have been created or modified.
```

   You can verify it with the **`lsdef`** command:

```
# lsdef -t network net-mgmt -i dynamicrange
Object name: net-mgmt
    dynamicrange=10.1.254.1-10.1.254.254
```

2. Set the `dynamicrange` attribute for the Service Network with the **`chdef`** command (only required for different network interfaces).

   This step is not required for the scenario in this chapter.

For example:

```
# chdef -t network net-svc dynamicrange=<start-address>-<end-address>
```

You can verify it with the **lsdef** command:

```
# lsdef -t network net-svc -i dynamicrange
```

3. Generate the configuration file for the DHCP server with the **makedhcp -n** command.

```
# makedhcp -n
Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak

Warning: No dynamic range specified for 10.2.0.0. If hardware discovery is
being used, a dynamic range is required.
```

4. Verify the dynamic range in the configuration of the DHCP server:

```
# grep 'network\|subnet\|_end\|range' /etc/dhcp/dhcpd.conf
shared-network enP3p3s0f2 {
  subnet 10.1.0.0 netmask 255.255.0.0 {
    range dynamic-bootp 10.1.254.1 10.1.254.254;
  } # 10.1.0.0/255.255.0.0 subnet_end
  subnet 10.2.0.0 netmask 255.255.0.0 {
  } # 10.2.0.0/255.255.0.0 subnet_end
} # enP3p3s0f2 nic_end
```

5.5.3 Configuration of BMCs to DHCP mode

The xCAT node discovery requires the BMCs' network configuration to occur in DHCP mode (until the association with the respective node objects, and permanent network configuration occur).

> **Note:** If the BMCs' network configuration cannot be changed (for example, due to network restrictions or maintenance requirements), skip the steps required for the BMC network configuration in the node discovery process. For more information, see 5.5.3, "Configuration of BMCs to DHCP mode" on page 94, and 5.5.4, "Definition of temporary BMC objects" on page 96. Then manually set the `bmc` attribute of one or more nodes to the respective BMC IP address.

In order to set the network configuration of the BMCs to DHCP mode, perform the following steps:

- For BMCs already in **DHCP** mode:

  No action required.

- For BMCs with **Static (IP) Address** mode and **known IP address**:

  a. Set the *IP Address Source* attribute to DHCP with the **ipmitool** command.

     Notice that the IP Address Source attribute is currently set to *Static Address*:

```
# ipmitool -I lanplus -H <ip> -U <user> -P <pass> lan print 1
<...>
IP Address Source       : Static Address
IP Address              : 192.168.101.29
Subnet Mask             : 255.255.255.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

Set it to DHCP:

```
# ipmitool -I lanplus -H <ip> -U <user> -P <pass> lan set 1 ipsrc dhcp
```

Notice the network configuration changes do not take effect immediately:

```
# ipmitool -I lanplus -H <ip> -U <user> -P <pass> lan print 1
<...>
IP Address Source       : DHCP Address
IP Address              : 192.168.101.29
Subnet Mask             : 255.255.255.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

b. Reboot the BMC with the **ipmitool** command, which is required for the network configuration changes to take effect:

```
# ipmitool -I lanplus -H <ip> -U <user> -P <pass> mc reset cold
```

c. Wait for the BMC to perform initialization and network configuration.

In order to determine when the BMC is back online and acquired an IP address through DHCP, you can watch the `/var/log/messages` file for DHCP server log messages.

For example:

```
# tail -f /var/log/messages
<...>
<...> dhcpd: DHCPDISCOVER from 70:e2:84:14:02:54 via enP3p3s0f2
<...> dhcpd: DHCPOFFER on 10.1.254.1 to 70:e2:84:14:02:54 via enP3p3s0f2
<...> dhcpd: DHCPREQUEST for 10.1.254.1 (10.1.0.1) from 70:e2:84:14:02:54
via enP3p3s0f2
<...> dhcpd: DHCPACK on 10.1.254.1 to 70:e2:84:14:02:54 via enP3p3s0f2
<...>
```

It is also possible to determine when the BMC is back online with an approach based on its IPv6 link-local address, which does not change across power cycles, network configuration steps, and so on (Example 5-18). To identify the IPv6 link-local address of each BMC in a local network, refer to the next bullet (*For BMCs with unknown IP address*).

Example 5-18 Waiting for the BMC with the ping6 command and IPv6 link-local address

```
# while ! ping6 -c 1 <IPv6-link-local-address>%<network-interface>; do echo
Waiting; done; echo Finished
PING fe80::72e2:84ff:fe14:254%enP3p3s0f2(fe80::72e2:84ff:fe14:254) ...
<...>
Waiting
PING fe80::72e2:84ff:fe14:254%enP3p3s0f2(fe80::72e2:84ff:fe14:254)...
<...>
Waiting
<...>
PING fe80::72e2:84ff:fe14:254%enP3p3s0f2(fe80::72e2:84ff:fe14:254) ...
64 bytes from fe80::72e2:84ff:fe14:254: icmp_seq=1 ttl=64 time=0.669 ms
<...>
Finished
```

d. Verify that the network configuration changes are in effect with the **ipmitool** command:

```
# ipmitool -I lanplus -H <ip> -U <user> -P <pass> lan print 1
<...>
```

```
IP Address Source       : DHCP Address
IP Address              : 10.1.254.1
Subnet Mask             : 255.255.0.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

- For BMCs with **unknown IP address** (either in DHCP or Static Address mode):
  a. Install the `nmap` package:

     ```
     # yum install nmap
     ```

  b. Discover one or more IPv6 link-local addresses of one or more BMCs with the **`nmap`** command (Example 5-19).

     Example 5-19 Discovering the IPv6 link-local address of BMCs with the nmap command

     ```
     # nmap -6 --script=targets-ipv6-multicast-echo -e enP3p3s0f2

     Starting Nmap 6.40 ( http://nmap.org ) at <...>
     Pre-scan script results:
     | targets-ipv6-multicast-echo:
     |   IP: fe80::9abe:94ff:fe59:f0f2  MAC: 98:be:94:59:f0:f2  IFACE: enP3p3s0f2
     |   IP: fe80::280:e5ff:fe1b:fc99   MAC: 00:80:e5:1b:fc:99  IFACE: enP3p3s0f2
     |   IP: fe80::280:e5ff:fe1c:9c3    MAC: 00:80:e5:1c:09:c3  IFACE: enP3p3s0f2
     |   IP: fe80::72e2:84ff:fe14:259   MAC: 70:e2:84:14:02:59  IFACE: enP3p3s0f2
     |   IP: fe80::72e2:84ff:fe14:254   MAC: 70:e2:84:14:02:54  IFACE: enP3p3s0f2
     |_  Use --script-args=newtargets to add the results as targets
     WARNING: No targets were specified, so 0 hosts scanned.
     Nmap done: 0 IP addresses (0 hosts up) scanned in 2.37 seconds
     ```

  c. Perform the steps that are described in the *Static (IP) Address mode and known IP address* case. Replace the BMC's IPv4 address in the **`ipmitool`** command for the IPv6 link-local address with the network interface as *zone index*, separated by the *percent* symbol. For example, (also described in Example 5-18 on page 95):

     ```
     <IPv6-link-local-address>%<network-interface>
     ```

5.5.4 Definition of temporary BMC objects

The xCAT node discovery requires temporary node objects for BMCs until the association with the respective node objects, and permanent network configuration occurs.

The temporary BMC objects are created by the **`bmcdiscover`** command, which scans an IP address range for BMCs, and collects information such as machine type and model, serial number, and IP address. It can provide that information as either objects in the xCAT database or the respective *stanzas* (a text-based description format with object name, type, and attributes). The objects are named after their MTM and SN information (obtained through IPMI).

The temporary BMC objects are automatically removed during the node discovery process after the respective node objects are matched and ready to refer to the BMCs. It is a simple means to have xCAT objects to refer to the BMCs still using temporary IP addresses (not yet associated with the respective node objects). This configuration allows for running xCAT commands (for example, power management, and console sessions) before the network configuration of the BMC occurs.

The **bmcdiscover** command has the following requirements:

- The BMCs' IP addresses to be within a known range (satisfied with the *dynamic range* configuration, in 5.5.2, "Configuration of the DHCP dynamic range" on page 93, and BMCs in DHCP mode, in 5.5.3, "Configuration of BMCs to DHCP mode" on page 94).
- The IPMI authentication credentials to be defined either in the `passwd` table (satisfied in 5.4.8, "IPMI authentication credentials" on page 91) or by using command arguments.

The **bmcdiscover** command can be used with the following arguments:

- `-z`: Provides object definition stanza.
- `-t`: Provides object with attributes for BMC node type and hardware type.
- `-w`: Writes objects to the xCAT database.

To define temporary objects for the BMCs, complete the following steps:

1. Run the **bmcdiscover** command on the dynamic range of IP addresses:

```
# bmcdiscover --range 10.1.254.1-254 -t -w
node-8335-gta-0000000000000000:
        objtype=node
        groups=all
        bmc=10.1.254.1
        cons=ipmi
        mgt=ipmi
        mtm=8335-GTA
        serial=0000000000000000
        nodetype=mp
        hwtype=bmc
```

> **Note:** The serial number is set to zeroes in the early system revision that is used for this book. This information is present on other systems (for example, from normal customer orders).

2. Verify that the BMC objects are listed as node objects with the **lsdef** command.

   Note the attributes/values `nodetype=mp` and `hwtype=bmc`.

```
# lsdef
node-8335-gta-0000000000000000  (node)

# lsdef node-8335-gta-0000000000000000
Object name: node-8335-gta-0000000000000000
    bmc=10.1.254.1
    cons=ipmi
    groups=all
    hwtype=bmc
    mgt=ipmi
    mtm=8335-GTA
    nodetype=mp
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=0000000000000000
```

When BMC objects, and IPMI authentication credentials are defined, you can run xCAT commands on the BMC objects, such as these examples:

- **rpower** for power management
- **rcons** for console sessions (requires the **makeconservercf** command first)
- **rsetboot** for boot-method selection

Note: It is always possible to run **ipmitool** commands to the BMC IP address as well.

5.5.5 Definition of node objects

The xCAT node discovery requires minimal node objects that can *match* the Machine Type and Model, and Serial Number information, which can be collected either automatically by the Genesis image (with the `mtm` and `serial` attributes) or manually. For more information, see 5.2.2, “Frequently used commands with the IPMItool” on page 55. The node discovery also tries to match the information with the temporary BMC objects to associate the node objects with their respective BMCs, and perform the network configuration of the BMCs.

This section covers creating a node group to set the attributes that are common among nodes or based on regular expressions.

To define a node group, complete the following steps:

1. Create the `s822lc` node group with the **mkdef** command:

```
# mkdef -t group s822lc \
  ip='|p8r(\d+)n(\d+)|10.1.($1+0).($2+0)|' \
  bmc='|p8r(\d+)n(\d+)|10.2.($1+0).($2+0)|' \
  mgt=ipmi \
  cons=ipmi
Warning: Cannot determine a member list for group 's822lc'.
1 object definitions have been created or modified.
```

2. Verify the node group with the **lsdef** command:

```
# lsdef -t group s822lc
Object name: s822lc
    bmc=|p8r(\d+)n(\d+)|10.2.($1+0).($2+0)|
    cons=ipmi
    grouptype=static
    ip=|p8r(\d+)n(\d+)|10.1.($1+0).($2+0)|
    members=
    mgt=ipmi
```

To create a node that is part of the node group, complete the following steps:

1. Create a node with the **mkdef** command, and include the node group in the `groups` attribute. Similarly, you can also modify an existing node, and make it part of a group with the **chdef** command.

```
# mkdef p8r1n1 groups=all,s822lc
1 object definitions have been created or modified.
```

   To create many nodes at the same time, you can use a node range, for example:

```
# mkdef p8r[1-5]n[1-6] groups=all,s822lc
30 object definitions have been created or modified.
```

2. Verify that the node inherits the node group's attributes with the **lsdef** command. Notice the attributes based on regular expressions are evaluated according to the name of the node. Some other attributes are set by xCAT by default.

```
# lsdef p8r1n1
Object name: p8r1n1
    bmc=10.2.1.1
    cons=ipmi
    groups=all,s822lc
    ip=10.1.1.1
    mgt=ipmi
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
```

To set the `mtm` and `serial` attributes to match the BMC object, complete the following steps:

1. Set the attributes with the **chdef** command. Similarly, you can also set the attributes when creating the node object with the **mkdef** command.

```
# chdef p8r1n1 mtm=8335-GTA serial=0000000000000000
1 object definitions have been created or modified.
```

> **Note:** The `mtm` and `serial` attributes are case-sensitive.

2. Verify the attributes with the **lsdef** command:

```
# lsdef p8r1n1
Object name: p8r1n1
    bmc=10.2.1.1
    cons=ipmi
    groups=all,s822lc
    ip=10.1.1.1
    mgt=ipmi
    mtm=8335-GTA
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=0000000000000000
```

> **Note:** By default, the network mask of the BMC might be configured incorrectly due to an issue in xCAT 2.11. The value might be set to 255.255.255.0 regardless of the value in the object definition.
>
> This is reported to be a problem with the `rspconfig` script, and the `bmcsetup` script is used for BMC-based servers. To enable the `bmcsetup` script for all nodes in the `s822lc` group, issue the following command:
>
> ```
> # chdef s822lc chain="runcmd=bmcsetup"
> ```
>
> For more details, see the following xCAT issue and documentation page:
>
> https://github.com/xcat2/xcat-core/issues/494
> http://xcat-docs.readthedocs.io/en/2.11/guides/admin-guides/manage_clusters/ppc64le/discovery/mtms_discovery.html

5.5.6 Configuration of host table, DNS, and DHCP servers

The xCAT requires the node objects to be present and up-to-date in configuration files for the host table, DNS server, and DHCP server.

> **Note:** This is particularly important for the node discovery process, which otherwise can show errors that are difficult to trace to specific misconfiguration steps.

It is required to update the configuration files after changes such as the following, which are reflected in the configuration files:

- Adding or removing node objects
- Adding, modifying, or removing hostnames, IP addresses, or aliases for network interfaces

The order of the commands is relevant, as some commands depend on changes performed by other commands. For more information and details, see the manual pages of the **makehosts**, **makedns**, and **makedhcp** commands:

```
# man makehosts
# man makedns
# man makedhcp
```

To update the configuration files with the node objects, complete the following steps:

1. Update the host table with the **makehosts** command.

   ```
   # makehosts s822lc
   ```

2. Verify that the node objects are present on the host table:

   ```
   # cat /etc/hosts
   127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
   ::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
   10.1.0.1 xcat-mn.xcat-cluster xcat-mn
   10.1.1.1 p8r1n1 p8r1n1.xcat-cluster
   ```

3. Update the DNS server configuration with the **makedns** command:

   ```
   # makedns -n s822lc
   Handling p8r1n1 in /etc/hosts.
   Getting reverse zones, this  take several minutes for a large cluster.
   Completed getting reverse zones.
   Updating zones.
   Completed updating zones.
   Restarting named
   Restarting named complete
   Updating DNS records, this  take several minutes for a large cluster.
   Completed updating DNS records.
   ```

4. Verify that the DNS server can resolve the name of the node with the **host** command:

   ```
   # host p8r1n1 10.1.0.1
   Using domain server:
   Name: 10.1.0.1
   Address: 10.1.0.1#53
   Aliases:

   p8r1n1.xcat-cluster has address 10.1.1.1
   ```

5. Update the DHCP server's configuration file with the **makedhcp** command:

   ```
   # makedhcp -n
   ```

6. Update the DHCP server's leases file with the **makedhcp** command:

```
# makedhcp -a
```

5.5.7 Boot into Node discovery

Finally, you can boot the nodes into node discovery with power on (or cycle), provided the boot order configuration is correct. For more information, see 5.2.3, "Boot order configuration" on page 57.

You can watch the progress of the node discovery process in the `/var/log/messages` file, which is described with comments as shown in Example 5-20.

Example 5-20 Contents and comments for the `/var/log/messages` file during node discovery

```
# tail -f /var/log/messages
...

Petitboot (DHCP client acquires an IP address, and releases it before booting):

... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPOFFER on 10.1.254.4 to 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPREQUEST for 10.1.254.4 (10.1.0.1) from 98:be:94:59:f0:f2 via
enP3p3s0f2
... dhcpd: DHCPACK on 10.1.254.4 to 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPRELEASE of 10.1.254.4 from 98:be:94:59:f0:f2 via enP3p3s0f2 (found)

Genesis (DHCP client acquires an IP address):

... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPOFFER on 10.1.254.5 to 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPREQUEST for 10.1.254.5 (10.1.0.1) from 98:be:94:59:f0:f2 via
enP3p3s0f2
... dhcpd: DHCPACK on 10.1.254.5 to 98:be:94:59:f0:f2 via enP3p3s0f2

Genesis (Communication with the xCAT Management Node; some error messages and
duplicated steps are apparently OK):

... xcat[30280]: xCAT: Allowing getcredentials x509cert
... xcat[17646]: xcatd: Processing discovery request from 10.1.254.5
... xcat[17646]: Discovery Error: Could not find any node.
... xcat[17646]: Discovery Error: Could not find any node.
... xcat[17646]: xcatd: Processing discovery request from 10.1.254.5

Genesis (The respective BMC object is identified, used for configuring the BMC
according to the node object, and then removed):

... xcat[17646]: Discovery info: configure password for
pbmc_node:node-8335-gta-0000000000000000.
... xcat[39159]: xCAT: Allowing rspconfig to node-8335-gta-0000000000000000
password= for root from localhost
... xcat[39168]: xCAT: Allowing chdef node-8335-gta-0000000000000000 bmcusername=
bmcpassword= for root from localhost
... xcat[17646]: Discover info: configure ip:10.2.1.1 for
pbmc_node:node-8335-gta-0000000000000000.
```

```
... xcat[39175]: xCAT: Allowing rspconfig to node-8335-gta-0000000000000000
ip=10.2.1.1 for root from localhost
... xcat[17646]: Discovery info: remove pbmc_node:node-8335-gta-0000000000000000.
... xcat[39184]: xCAT: Allowing rmdef node-8335-gta-0000000000000000 for root from
localhost
... xcatd: Discovery worker: fsp instance: nodediscover instance: p8r1n1 has been
discovered
... xcat[17646]: Discovery info: configure password for
pbmc_node:node-8335-gta-0000000000000000.
... xcat[39217]: xCAT: Allowing chdef node-8335-gta-0000000000000000 bmcusername=
bmcpassword= for root from localhost
... xcat[17646]: Discover info: configure ip:10.2.1.1 for
pbmc_node:node-8335-gta-0000000000000000.
... xcat[17646]: Discovery info: remove pbmc_node:node-8335-gta-0000000000000000.

Genesis (DHCP client releases the temporary IP address, and acquires the permanent
IP address):

... dhcpd: DHCPRELEASE of 10.1.254.5 from 98:be:94:59:f0:f2 via enP3p3s0f2 (found)
... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPOFFER on 10.1.1.1 to 98:be:94:59:f0:f2 via enP3p3s0f2
... dhcpd: DHCPREQUEST for 10.1.1.1 (10.1.0.1) from 98:be:94:59:f0:f2 via
enP3p3s0f2
... dhcpd: DHCPACK on 10.1.1.1 to 98:be:94:59:f0:f2 via enP3p3s0f2

Genesis (Cleanup of the BMC discovery):

... xcat[39230]: xCAT: Allowing rmdef node-8335-gta-0000000000000000 for root from
localhost
... xcatd: Discovery worker: fsp instance: nodediscover instance: Failed to notify
10.1.254.5 that it's actually p8r1n1.

Genesis (Further communication with the xCAT Management Node):

... xcat[39233]: xCAT: Allowing getcredentials x509cert from p8r1n1
... xcat: credentials: sending x509cert
```

The Genesis image remains waiting for further instructions from the xCAT Management Node, and is accessible through SSH.

> **Note:** During the BMC network configuration steps, it can lose network connectivity (including the IPv6 link-local address). In this case, reset it by using in-band IPMI with the **ipmitool** command included in the Genesis image. This limitation might be addressed in a future xCAT or firmware version.

The steps to reset the BMC by using in-band IPMI on the Genesis image, and wait for the BMC to come back online are described in Example 5-21.

Example 5-21 Resetting the BMC via in-band IPMI on the Genesis image

```
Reset the BMC via in-band IPMI on the Genesis image:

# ssh p8r1n1 'ipmitool mc reset cold'

Wait for the BMC to come back online:
```

```
(This example is based on the link-local IPv6 address; you can also use the IPv4
address assigned on node discovery; e.g., ping 10.2.1.1)

# while ! ping6 -c 1 -q fe80::72e2:84ff:fe14:254%enP3p3s0f2; do echo Waiting;
done; echo; echo Finished
PING fe80::72e2:84ff:fe14:254%enP3p3s0f2(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enP3p3s0f2 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

Waiting
PING fe80::72e2:84ff:fe14:254%enP3p3s0f2(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enP3p3s0f2 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

<...>
Waiting
PING fe80::72e2:84ff:fe14:254%enP3p3s0f2(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enP3p3s0f2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.606/0.606/0.606/0.000 ms

Finished
```

You can watch the node discovery on the node console through IPMI. The BMC IP address can change as a result of the process. Therefore, use the BMC IPv6 link-local address (which does not change) for that purpose (Example 5-22).

Example 5-22 Node discovery on the console via IPMI with the rcons on ipmitool commands

```
With the rcons command:
(require initial configuration w/ the makeconservercf command):
# makeconservercf
# rcons node-8335-gta-0000000000000000

With the ipmitol command (via IPv6 link-local address):
# ipmitool -I lanplus -H fe80::72e2:84ff:fe14:254%enP3p3s0f2 -U ADMIN -P admin sol
activate
```

You can verify that the node object now contains attributes obtained during node discovery (for example, hardware characteristics), and other xCAT attributes with the **lsdef** command (Example 5-23).

Example 5-23 Node object with attributes obtained during node discovery

```
# lsdef p8r1n1
Object name: p8r1n1
    arch=ppc64
    bmc=10.2.1.1
    cons=ipmi
    cpucount=160
    cputype=POWER8 (raw), altivec supported
    disksize=sda:1000GB,sdb:1000GB
    groups=all,s822lc
```

```
ip=10.1.1.1
mac=98:be:94:59:f0:f2
memory=261482MB
mgt=ipmi
mtm=8335-GTA
netboot=petitboot
nodetype=mp
postbootscripts=otherpkgs
postscripts=syslog,remoteshell,syncfiles
serial=0000000000000000
status=standingby
statustime=11-25-2015 23:13:50
supportedarchs=ppc64
```

5.6 xCAT Compute Nodes

This section describes the deployment of an xCAT Compute Node with the IBM High Performance Computing (HPC) software running on RHEL Server 7.2 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode on the IBM Power System S822LC server.

The steps described in this section cover the diskful installation. They install all the software stack (except for Spectrum LSF) in a single provisioning stage (that is, with the **nodeset** command after all components are marked for installation). You can install the software stack incrementally by first provisioning the Linux distribution (with the **nodeset** command), and later installing each component of the software stack (with the **updatenode** command).

5.6.1 Network interfaces

The network interface associated with the management network is known as primary network interface (or adapter). The network interfaces associated with other networks are known as secondary (or additional) network interfaces (or adapters).

For more information and details, see the following xCAT documentation page:

http://xcat-docs.readthedocs.org/en/latest/guides/admin-guides/manage_clusters/ppc64le/diskless/customize_image/cfg_second_adapter.html

Primary network interface

The primary network interface is the network interface connected to the Management Network. Two attributes are important for this network interface:

- `primarynic`: Identifies the primary network interface. Set it to `mac` to use the network interface with the MAC address specified by the `mac` attribute (collected during node discovery).
- `installnic`: Identifies the network interface used for OS installation, which is usually the primary network interface, so set it to `mac`.

To set the attributes for the primary network interface to `mac`, issue the following commands:

1. Set the `primarynic` and `installnic` with the **chdef** command:

```
# chdef -t group s822lc \
    installnic=mac \
```

```
    primarynic=mac
1 object definitions have been created or modified.
```

2. Verify the attributes with the **lsdef** command:

```
# lsdef -t group s822lc -i installnic,primarynic
Object name: s822lc
    installnic=mac
    primarynic=mac
```

Secondary network interfaces

The xCAT employs the information from the `nics` table to configure the network interfaces in the nodes to be part of the xCAT networks that are defined in the `networks` table. For example, the following attributes are used:

- `nicips` for IP addresses
- `nicnetworks` for xCAT networks (defined in the `networks` table)
- `nictypes` for the type of networks (for example, Ethernet or InfiniBand)
- `nichostnamesuffixes` (*optional*) for appending per-network suffixes to hostnames

The attribute format for the `nics` table uses several types of field separators, which allows for each field to relate to multiple network interfaces with multiple values per network interface (for example, IP addresses and hostnames). The format is a *comma-separated list* of *interface!values* pairs (that is, one pair per network interface), where *values* is a *pipe-separated list* of values (that is, all values assigned to that network interface). For values with regular expressions, include the xCAT regular expression delimiters/pattern around the value (that is, *leading pipe, regular expression pattern, separator pipe, value*, and *trailing pipe*).

For an illustration, consider the following example:

- Two network interfaces: `eth0` and `eth1`
- Two IP addresses each (`eth0` with 192.168.0.1 and 192.168.0.2; `eth1` with 192.168.0.3 and 192.168.0.4):

  ```
  nicips='eth0!192.168.0.1|192.168.0.2,eth1!192.168.0.3|192.168.0.4'
  ```

- Two hostname suffixes each (`eth0` with `-port0ip1` and `-port0ip2`; `eth1` with `-port1ip1` and `-port1ip2`):

  ```
  nichostnamesuffixes='eth0!-port0ip1|-port0ip2,eth1!-port1ip1|-port1ip2'
  ```

To configure the network interfaces of the compute nodes, complete the following steps:

1. Set the attributes of the nics table in the node group with the **chdef** command:

```
# chdef -t group s822lc \
    nictypes='enP3p3s0f0!Ethernet,enP3p3s0f1!Ethernet,ib0!Infiniband,ib1!Infini
band' \
    nicnetworks='enP3p3s0f0!net-app-10gbe-port1,enP3p3s0f1!net-app-10gbe-port2,
ib0!net-app-ib-port1,ib1!net-app-ib-port2' \
    nichostnamesuffixes='enP3p3s0f0!-10gbe-1,enP3p3s0f1!-10gbe-2,ib0!-ib-1,ib1!
-ib-2' \
    nicips='|p8r(\d+)n(\d+)|enP3p3s0f0!10.3.($1+0).($2+0),enP3p3s0f1!10.4.($1+0
).($2+0),ib0!10.5.($1+0).($2+0),ib1!10.6.($1+0).($2+0)|'
1 object definitions have been created or modified.
```

2. Verify the attributes with the **lsdef** command. They are represented with per-interface subattributes (that is, *<attribute>.<interface>=<values-for-the-interface>*):

```
# lsdef -t group s822lc
Object name: s822lc
    <...>
    nichostnamesuffixes.enP3p3s0f0=-10gbe-1
    nichostnamesuffixes.enP3p3s0f1=-10gbe-2
    nichostnamesuffixes.ib0=-ib-1
    nichostnamesuffixes.ib1=-ib-2
    nicips.|p8r(\d+)n(\d+)|enP3p3s0f0=10.3.($1+0).($2+0)
    nicips.enP3p3s0f1=10.4.($1+0).($2+0)
    nicips.ib0=10.5.($1+0).($2+0)
    nicips.ib1=10.6.($1+0).($2+0)|
    nicnetworks.enP3p3s0f0=net-app-10gbe-port1
    nicnetworks.enP3p3s0f1=net-app-10gbe-port2
    nicnetworks.ib0=net-app-ib-port1
    nicnetworks.ib1=net-app-ib-port2
    nictypes.enP3p3s0f0=Ethernet
    nictypes.enP3p3s0f1=Ethernet
    nictypes.ib0=Infiniband
    nictypes.ib1=Infiniband
    <...>
```

Notice that the `nicips` per-interface subattributes for the node group include the head and tail of the regular expression in the first and last interfaces. This issue is expected to be resolved in a future xCAT version (xCAT issue #476).

3. Verify the attributes based on regular expressions have correct values on a particular node with the **lsdef** command:

```
# lsdef p8r1n1
Object name: p8r1n1
<...>
    nichostnamesuffixes.enP3p3s0f0=-10gbe-1
    nichostnamesuffixes.enP3p3s0f1=-10gbe-2
    nichostnamesuffixes.ib0=-ib-1
    nichostnamesuffixes.ib1=-ib-2
    nicips.enP3p3s0f0=10.3.1.1
    nicips.enP3p3s0f1=10.4.1.1
    nicips.ib0=10.5.1.1
    nicips.ib1=10.6.1.1
    nicnetworks.enP3p3s0f0=net-app-10gbe-port1
    nicnetworks.enP3p3s0f1=net-app-10gbe-port2
    nicnetworks.ib0=net-app-ib-port1
    nicnetworks.ib1=net-app-ib-port2
    nictypes.enP3p3s0f0=Ethernet
    nictypes.enP3p3s0f1=Ethernet
    nictypes.ib0=Infiniband
    nictypes.ib1=Infiniband
<...>
```

4. Update the configuration files for the host table, DNS server, and DHCP server with the **makehosts**, **makedns**, and **makedhcp** commands:

```
# makehosts s822lc

# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
```

```
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.1.0.1 xcat-mn.xcat-cluster xcat-mn
10.1.1.1 p8r1n1 p8r1n1.xcat-cluster
10.5.1.1 p8r1n1-ib-1 p8r1n1-ib-1.xcat-cluster
10.6.1.1 p8r1n1-ib-2 p8r1n1-ib-2.xcat-cluster
10.4.1.1 p8r1n1-10gbe-2 p8r1n1-10gbe-2.xcat-cluster
10.3.1.1 p8r1n1-10gbe-1 p8r1n1-10gbe-1.xcat-cluster
```

Note: If a network interface prefix/suffix is renamed or removed, you can update the host table by removing the node entry (or nodes, or group entries), and adding it (them) back with the following commands:

```
# makehosts -d nodes
# makehosts nodes
```

```
# makedns -n s822lc
Handling p8r1n1-10gbe-2 in /etc/hosts.
Handling p8r1n1 in /etc/hosts.
Handling p8r1n1-10gbe-1 in /etc/hosts.
Handling p8r1n1-ib-2 in /etc/hosts.
Handling p8r1n1-ib-1 in /etc/hosts.
Getting reverse zones, this  take several minutes for a large cluster.
Completed getting reverse zones.
Updating zones.
Completed updating zones.
Restarting named
Restarting named complete
Updating DNS records, this  take several minutes for a large cluster.
Completed updating DNS records.

# makedhcp -n
# makedhcp -a
```

5. Add the `confignics` script to the list of postscripts for configuring the network interfaces. The InfiniBand network interfaces (2-port adapter) require the argument `--ibaports=2`.

```
# chdef -t group s822lc --plus postscripts='confignics --ibaports=2'
1 object definitions have been created or modified.

# lsdef -t group s822lc -i postscripts
Object name: s822lc
    postscripts=confignics --ibaports=2
```

For more information and details about the `confignics` script and the configuration of InfiniBand adapters, see the following xCAT documentation pages:

- http://xcat-docs.readthedocs.org/en/latest/guides/admin-guides/manage_clusters/ppc64le/diskless/customize_image/cfg_second_adapter.html
- http://xcat-docs.readthedocs.org/en/latest/advanced/networks/infiniband/index.html
- http://xcat-docs.readthedocs.org/en/latest/advanced/networks/infiniband/network_configuration.html

Public or site network connectivity (optional)

The connectivity to the public or site networks for the compute nodes can be provided by one of these methods:

- By way of the Management Node: By using network address translation (NAT).
- By way of Compute Nodes: By using an additional xCAT network.

To perform the required network configuration, follow the steps of either method:

- By way of the Management Node
- By way of Compute Nodes

By way of the Management Node

This method requires that the `gateway` attribute of the xCAT Management Network be set to the Management node (default setting, with value "`<xcatmaster>`"), and NAT rules be configured in the firewall. To connect, complete these steps:

1. Verify that the `gateway` attribute of the Management Network is set to `<xcatmaster>` with the **`lsdef`** command:

   ```
   # lsdef -t network net-mgmt -i gateway
   Object name: net-mgmt
       gateway=<xcatmaster>
   ```

   If not, set it with the **`chdef`** command:

   ```
   # chdef -t network net-mgmt gateway='<xcatmaster>'
   ```

2. Verify the `routers` option of the DHCP server configuration file (based on the `gateway` attribute) in the Management Network reflects the IP address of the Management Node:

   ```
   # grep 'subnet\|routers' /etc/dhcp/dhcpd.conf
     subnet 10.1.0.0 netmask 255.255.0.0 {
       option routers  10.1.0.1;
     } # 10.1.0.0/255.255.0.0 subnet_end
     subnet 10.2.0.0 netmask 255.255.0.0 {
       option routers  10.2.0.1;
     } # 10.2.0.0/255.255.0.0 subnet_end
   ```

   If not, regenerate the DHCP server configuration files with the **`makedhcp`** command:

   ```
   # makedhcp -n
   # makedhcp -a
   ```

3. Verify the default route on the compute nodes is set to the IP address of the Management Node with the **`ip`** command:

   ```
   # xdsh p8r1n1 'ip route show | grep default'
   p8r1n1: default via 10.1.0.1 dev enP3p3s0f1
   ```

   If not, restart the network service with the **systemctl** command:

   ```
   # xdsh p8r1n1 'systemctl restart network'
   ```

4. Configure the `iptables` firewall rules for NAT in the `rc.local` script, configure it to start automatically on boot, and start the service manually this time.

   For the network scenario adopted in this chapter:

   – Management network on network interface `enP3p3s0f2` in the management node

   – Public/site network on network interface `enP3p3s0f3` in the management node

   ```
   # cat <<EOF >>/etc/rc.d/rc.local
   iptables -t nat --append POSTROUTING --out-interface enP3p3s0f3 -j MASQUERADE
   iptables --append FORWARD --in-interface enP3p3s0f2 -j ACCEPT
   ```

```
EOF

# chmod +x /etc/rc.d/rc.local
# systemctl start rc-local
```

> **Note:** The default firewall in RHEL Server 7.2 is `firewalld`, which is disabled by xCAT.
>
> For more information and details about firewalld, see the following RHEL documentation page:
>
> https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/sec-Using_Firewalls.html

5. You can verify the network connectivity on an already running node (if any) with the **`ping`** command:

```
# xdsh p8r1n1 'ping -c1 example.com'
p8r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p8r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=4.07 ms
<...>
```

By way of Compute Nodes

This method requires the `gateway` attribute of the xCAT Management Network *not* to be set, and an additional xCAT network (for the public or site network) to define the network interface, address, and gateway to be used.

1. Reset the `gateway` attribute of the Management Network with the **`chdef`** command:

```
# chdef -t network net-mgmt gateway=''

# lsdef -t network net-mgmt -i gateway
Object name: net-mgmt
    gateway=
```

2. Define a new xCAT network for the public or site network (for example, `net-site`):

```
# mkdef -t network net-site net=9.114.37.0 mask=255.255.255.0
1 object definitions have been created or modified.

# lsdef -t network
net-app-10gbe-port1  (network)
net-app-10gbe-port2  (network)
net-app-ib-port1  (network)
net-app-ib-port2  (network)
net-mgmt  (network)
net-site  (network)
net-svc  (network)

# lsdef -t network net-site
Object name: net-site
    mask=255.255.255.0
    net=9.114.37.0
```

3. Modify the `nics` table to include the attributes of the new xCAT network with either the **`tabedit`** command or the **`chdef`** command. The network scenario described in this chapter uses a public or site network on network interface `enP3p3s0f3` in the compute nodes.

```
# tabedit nics

OR:
```

```
# chdef -t group s822lc \
    nictypes='enP3p3s0f0!Ethernet,enP3p3s0f1!Ethernet,enP3p3s0f3!Ethernet,ib0!I
nfiniband,ib1!Infiniband' \
    nicnetworks='enP3p3s0f0!net-app-10gbe-port1,enP3p3s0f1!net-app-10gbe-port2,
enP3p3s0f3!net-site,ib0!net-app-ib-port1,ib1!net-app-ib-port2' \
    nichostnamesuffixes='enP3p3s0f0!-10gbe-1,enP3p3s0f1!-10gbe-2,enP3p3s0f3!-si
te,ib0!-ib-1,ib1!-ib-2' \
    nicips='|p8r(\d+)n(\d+)|enP3p3s0f0!10.3.($1+0).($2+0),enP3p3s0f1!10.4.($1+0
).($2+0),enP3p3s0f3!9.114.37.(181-$2),ib0!10.5.($1+0).($2+0),ib1!10.6.($1+0).($
2+0)|' \
    nicextraparams='enP3p3s0f3!GATEWAY=9.114.37.254'
1 object definitions have been created or modified.
```

4. Verify the attributes with either the **tabdump** or the **lsdef** commands:

```
# tabdump nics
#node,nicips,nichostnamesuffixes,nichostnameprefixes,nictypes,niccustomscripts,
nicnetworks,nicaliases,nicextraparams,comments,disable
"s822lc","|p8r(\d+)n(\d+)|enP3p3s0f0!10.3.($1+0).($2+0),enP3p3s0f1!10.4.($1+0).
($2+0),enP3p3s0f3!9.114.37.(181-$2),ib0!10.5.($1+0).($2+0),ib1!10.6.($1+0).($2+
0)|","enP3p3s0f0!-10gbe-1,enP3p3s0f1!-10g
be-2,enP3p3s0f3!-site,ib0!-ib-1,ib1!-ib-2",,"enP3p3s0f0!Ethernet,enP3p3s0f1!Eth
ernet,enP3p3s0f3!Ethernet,ib0!Infiniband,ib1!Infiniband",,"enP3p3s0f0!net-app-1
0gbe-port1,enP3p3s0f1!net-app-10gbe-port2
,enP3p3s0f3!net-site,ib0!net-app-ib-port1,ib1!net-app-ib-port2",,"enP3p3s0f3!GA
TEWAY=9.114.37.254",,

# lsdef -t group s822lc
Object name: s822lc
<...>
    nicextraparams.enP3p3s0f3=GATEWAY=9.114.37.254
<...>
    nichostnamesuffixes.enP3p3s0f3=-site
<...>
    nicips.enP3p3s0f3=9.114.37.(181-$2)
<...>
    nicnetworks.enP3p3s0f3=net-site
<...>
    nictypes.enP3p3s0f3=Ethernet
<...>

# lsdef p8r1n1
Object name: p8r1n1
<...>
    nicextraparams.enP3p3s0f3=GATEWAY=9.114.37.254
<...>
    nichostnamesuffixes.enP3p3s0f3=-site
<...>
    nicips.enP3p3s0f3=9.114.37.180
<...>
    nicnetworks.enP3p3s0f3=net-site
<...>
    nictypes.enP3p3s0f3=Ethernet
<...>
```

5. Update the host table with the **makehosts** command:

```
# makehosts -d s822lc
# makehosts s822lc

# cat /etc/hosts
<...>
10.1.1.1 p8r1n1 p8r1n1.xcat-cluster
10.5.1.1 p8r1n1-ib-1 p8r1n1-ib-1.xcat-cluster
10.6.1.1 p8r1n1-ib-2 p8r1n1-ib-2.xcat-cluster
10.4.1.1 p8r1n1-10gbe-2 p8r1n1-10gbe-2.xcat-cluster
9.114.37.180 p8r1n1-site p8r1n1-site.xcat-cluster
10.3.1.1 p8r1n1-10gbe-1 p8r1n1-10gbe-1.xcat-cluster
<...>
```

6. Update the DNS server configuration with the **makedns** command:

```
# makedns -n s822lc
<...>
Handling p8r1n1-site in /etc/hosts.
<...>
Completed updating DNS records.
```

7. Update the DHCP server configuration with the **makedhcp** command:

```
# makedhcp -n
# makedhcp -a
```

8. You can update the network configuration on an already running node (if any) with the **confignics** script of the **updatenode** command:

```
# updatenode p8r1n1 -P confignics
```

9. You can verify the network connectivity on an already running node (if any) with the **ping** command:

```
# xdsh p8r1n1 'ping -c1 example.com'
p8r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p8r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=3.94 ms
<...>
```

5.6.2 RHEL Server

The xCAT stores the configuration for installing operating systems in objects of type `osimage` (that is, *operating system image*). The **copycds** command can be used to create `osimage` objects based on an OS installation disc image (for example, an ISO file).

Set the password for the root user

To set the root password, complete the following steps:

1. Set the `username` and `password` attributes of the `system` key/row in the `passwd` table with the **chtab** command:

```
# chtab key=system passwd.username=root passwd.password=cluster
```

2. Verify the attributes with the **tabdump** command:

```
# tabdump -w key==system passwd
#key,username,password,cryptmethod,authdomain,comments,disable
"system","root","cluster",,,,
```

Note: If the attributes are not correctly set, the **nodeset** command shows the following error message:

```
# nodeset p8r1n1 osimage=rh72-hpc-diskful
p8r1n1: Error: Unable to find requested filed <password> from table
<passwd>, with key <key=system,username=root>
Error: Some nodes failed to set up install resources on server
xcat-mn.xcat-cluster, aborting
```

Create an osimage object

Initially, there are no `osimage` objects:

```
# lsdef -t osimage
Could not find any object definitions to display.
```

To create `osimage` objects for the RHEL Server 7.2 installation disc image, complete the following steps:

1. Run the **copycds** command on the RHEL Server 7.2 ISO file:

   ```
   # copycds /mnt/RHEL-7.2-20151030.0-Server-ppc64le-dvd1.iso
   Copying media to /install/rhels7.2/ppc64le
   Media copy operation successful
   ```

2. Verify that the `osimage` objects are present with the **lsdef** command.

   ```
   # lsdef -t osimage
   rhels7.2-ppc64le-install-compute  (osimage)
   rhels7.2-ppc64le-install-service  (osimage)
   rhels7.2-ppc64le-netboot-compute  (osimage)
   rhels7.2-ppc64le-stateful-mgmtnode  (osimage)
   ```

   For more information about the `osimage` objects, see 5.3.6, “xCAT operating system installation types: Disks and state” on page 66.

3. Create a copy of the original `osimage` object, named *rh72-hpc-diskful*.

   It is optional, but useful in case multiple `osimage` objects are maintained (for example, for multiple/different configurations of the same OS).

   You can use the **lsdef -z** command, which provides the object stanza, modify it (for example, with the **sed** command), and create an object based on it with the **mkdef -z** command.

   ```
   # osimage=rh72-hpc-diskful

   # lsdef -t osimage rhels7.2-ppc64le-install-compute -z \
       | sed "s/^[^# ].*:/$osimage:/" \
       | mkdef -z
   1 object definitions have been created or modified.
   ```

4. Verify the copy `osimage` object with the **lsdef** command:

   ```
   # lsdef -t osimage
   rh72-hpc-diskful  (osimage)
   rhels7.2-ppc64le-install-compute  (osimage)
   rhels7.2-ppc64le-install-service  (osimage)
   rhels7.2-ppc64le-netboot-compute  (osimage)
   rhels7.2-ppc64le-stateful-mgmtnode  (osimage)
   ```

Note: It is already possible to install the OS (without other components of the software stack) with the **nodeset** command:

```
# nodeset p8r1n1 osimage=rh72-hpc-diskful
```

Change the pkglist attribute

The usage of multiple package lists is convenient for independently organizing the required packages for each component of the software stack. However, the xCAT currently does not support multiple package lists in the pkglist attribute, but it does support a package list to reference the contents of other package lists. This feature provides a way to achieve multiple package lists.

To change the pkglist attribute for a different package list, complete the following steps:

1. Verify the current pkglist attribute, and assign it to the old_list variable:

```
# lsdef -t osimage rh72-hpc-diskful -i pkglist
Object name: rh72-hpc-diskful
    pkglist=/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist

# old_list=/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist
```

2. Create a new list that includes the old list:

```
# new_list=/install/custom/install/rh/rh72-hpc.pkglist

# mkdir -p $(dirname $new_list)
# echo "# RHEL Server 7.2 (original pkglist)" > $new_list
# echo "#INCLUDE:${old_list}#" >> $new_list
```

3. Verify the contents of the new list:

```
# cat $new_list
# RHEL Server 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist#
```

4. Change the pkglist attribute to the new list with the **chdef** command:

```
# chdef -t osimage rh72-hpc-diskful pkglist=$new_list
1 object definitions have been created or modified.
```

5. Verify the pkglist attribute with the **lsdef** command:

```
# lsdef -t osimage rh72-hpc-diskful -i pkglist
Object name: rh72-hpc-diskful
    pkglist=/install/custom/install/rh/rh72-hpc.pkglist
```

5.6.3 CUDA Toolkit

The CUDA Toolkit can be installed with the RPM packages that are contained in the local repository package available for download in the NVIDIA website. The packages are marked for installation with the pkgdir and pkglist mechanism so to be installed during the Linux distribution installation. This setting ensures that a reboot occurs between installation and utilization (requirement).

For more information, see the following xCAT documentation page:

http://xcat-docs.readthedocs.org/en/latest/advanced/gpu/nvidia/osimage/rhels.html

To install the CUDA Toolkit, complete the following steps:

1. Install the `createrepo` package for creating package repositories:

```
# yum install createrepo
```

2. Download and extract the CUDA Toolkit RPM package:

> **Note:** The CUDA Toolkit is available for download at the following location and navigation steps:
>
> https://developer.nvidia.com/cuda-downloads
>
> **(Operating Systems) Linux** → **(Architecture) ppc64le** → **(Distribution) RHEL** → **(Version) 7** → **(Installer Type) rpm (local)** → **Download**.

```
# dir=/tmp/cuda
# mkdir -p $dir
# cd $dir

# curl -sOL http://.../cuda-repo-rhel7-7-5-local-7.5-23.ppc64le.rpm
# rpm2cpio cuda-repo-rhel7-7-5-local-7.5-23.ppc64le.rpm | cpio -id
2043179 blocks
```

3. Create a package repository for its contents with the **`createrepo`** command:

```
# dir=/install/post/otherpkgs/rhels7.2/ppc64le/cuda/7.5/
# mkdir -p $dir

# cp -r var/cuda-repo-7-5-local/* $dir
# createrepo $dir
<...>

# ls -1d $dir/*
/install/post/otherpkgs/rhels7.2/ppc64le/cuda/7.5/cuda-7.5-23.ppc64le.rpm

<...>
/install/post/otherpkgs/rhels7.2/ppc64le/cuda/7.5/repodata
<...>
```

4. Include the package repository's directory in the `pkgdir` attribute of the `osimage` with the **`chdef`** command:

```
# lsdef -t osimage rh72-hpc-diskful -i pkgdir
Object name: rh72-hpc-diskful
    pkgdir=/install/rhels7.2/ppc64le

# chdef -t osimage rh72-hpc-diskful --plus pkgdir=$dir
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i pkgdir
Object name: rh72-hpc-diskful
    pkgdir=/install/rhels7.2/ppc64le,/install/post/otherpkgs/rhels7.2/ppc64le/c
uda/7.5/
```

5. Download the `dkms` RPM package (dependency of the CUDA Toolkit):

```
# dir=/install/post/otherpkgs/rhels7.2/ppc64le/cuda/deps/
# mkdir -p $dir
# cd $dir
```

```
# repo_url='http://download.fedoraproject.org/pub/epel/7/ppc64/'
# dkms_path="$(curl -sL ${repo_url}/repoview/dkms.html | grep -o
'href=".*/dkms-.*.noarch.rpm"' | cut -d'"' -f2)"
# curl -sOL ${repo_url}/${dkms_path}

# ls -1
dkms-2.2.0.3-30.git.7c3e7c5.el7.noarch.rpm
```

6. Create a package repository for it with the **createrepo** command:

```
# createrepo .
<...>

# ls -1
dkms-2.2.0.3-30.git.7c3e7c5.el7.noarch.rpm
repodata
```

7. Include the package repository's directory in the `pkgdir` attribute of the `osimage` with the **chdef** command:

```
# chdef -t osimage rh72-hpc-diskful --plus pkgdir=$dir
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i pkgdir
Object name: rh72-hpc-diskful
    pkgdir=/install/rhels7.2/ppc64le,/install/post/otherpkgs/rhels7.2/ppc64le/c
uda/7.5/,/install/post/otherpkgs/rhels7.2/ppc64le/cuda/deps/
```

8. Include one of the two CUDA Toolkit package lists in the `osimage` package list.

   The following package lists are available for the CUDA Toolkit:

   – `cudafull.rhels7.ppc64le.pkglist`: Includes runtime and development tools

   – `cudaruntime.rhels7.ppc64le.pkglist`: Includes runtime only

```
# cuda_list=/opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist

# lsdef -t osimage rh72-hpc-diskful -i pkglist
Object name: rh72-hpc-diskful
    pkglist=/install/custom/install/rh/rh72-hpc.pkglist

# pkglist=/install/custom/install/rh/rh72-hpc.pkglist

# cat <<EOF >>$pkglist

# CUDA Toolkit 7.5 for RHEL 7.2 (original pkglist)
#INCLUDE:${cuda_list}#
EOF

# cat $pkglist
# RHEL Server 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist#

# CUDA Toolkit 7.5 for RHEL 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist#
```

9. Create this work-around package list because of the relative addressing that is used in the CUDA Toolkit package lists:

```
# list=/install/custom/install/rh/compute.rhels7.pkglist

# cat <<EOF >$list
# Empty list for fixing the INCLUDE from CUDA list.
EOF

# cat $list
# Empty list for fixing the INCLUDE from CUDA list.
```

This step is necessary due to the following #INCLUDE# line currently present in the CUDA package list:

```
# cat /opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist
#INCLUDE:compute.rhels7.pkglist#

#For Cuda 7.5
kernel-devel
gcc
pciutils
dkms
cuda
```

Note: If the work-around package list is not in place, the `ospkgs` script in the **updatenode** command can fail with the following error message (as listed in the `/var/log/messages` log file), which causes some of the packages not to be installed:

```
<...> xcat:
OSPKGS=wget,ntp,nfs-utils,net-snmp,rsync,yp-tools,openssh-server,util-linux,
net-tools,#INCLUDEBAD:cannot open
/install/custom/install/rh/compute.rhels7.pkglist#,kernel-devel,gcc,pciutils
,dkms,cuda,pciutils-libs,pciutils,tcl,tk,tcsh,gcc-gfortran,lsof,libnl,libxml
2-python,python-devel,redhat-rpm-config,rpm-build,kernel-devel,gtk2,atk,cair
o,ksh
```

This problem is reported in xCAT issue #484.

10. Create a script to configure features such as persistence mode and power limit for the GPUs during boot:

```
# script=/install/postscripts/nvidia-power-limit

# cat <<"EOF" >$script
#!/bin/bash

RC_LOCAL='/etc/rc.d/rc.local'

cat <<EORCLOCAL >>$RC_LOCAL
nvidia-smi --persistence-mode=1
nvidia-smi --power-limit=175
EORCLOCAL

chmod +x $RC_LOCAL

exit 0
```

```
EOF

# chmod +x $script
```

11. Include the script in the list of `postscripts` of the `osimage` object with the **chdef** command:

```
# lsdef -t osimage rh72-hpc-diskful -i postscripts
Object name: rh72-hpc-diskful
    postscripts=

# chdef -t osimage rh72-hpc-diskful --plus postscripts="$(basename $script)"
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i postscripts
Object name: rh72-hpc-diskful
    postscripts=nvidia-power-limit
```

5.6.4 Mellanox OFED for Linux

To install the Mellanox OFED for Linux, complete the following steps.

For more information and details, see the following xCAT documentation pages:

- http://xcat-docs.readthedocs.org/en/latest/advanced/networks/infiniband/mlnxofed_ib_install_v2_preparation.html
- http://xcat-docs.readthedocs.org/en/latest/advanced/networks/infiniband/network_configuration.html

1. Download and copy the ISO file to the xCAT installation directory:

> **Note:** The Mellanox OFED is available for download at the following location and navigation steps:
>
> http://www.mellanox.com/
>
> **Products → InfiniBand/VPI Drivers (under Software) → Mellanox OFED Linux (MLNX_OFED) → Download → (Version) 3.2-1.0.1.1 → (OS Distribution) RHEL/CentOS → (OS Distribution Version) RHEL/CentOS 7.2 → (Architecture) ppc64le → (Download/Documentation) ISO MLNX_OFED_LINUX-<...>.iso**.

```
# dir=/install/mofed/rh/ppc64le/
# mkdir -p $dir

# curl -sOL http://.../MLNX_OFED_LINUX-3.2-1.0.1.1-rhel7.2-ppc64le.iso
# cp MLNX_OFED_LINUX-3.2-1.0.1.1-rhel7.2-ppc64le.iso $dir
```

2. Copy the `mlnxofed_ib_install.v2` script into the `postscripts` directory (with the required file name `mlnxofed_ib_install`):

```
# script=/install/postscripts/mlnxofed_ib_install
# cp /opt/xcat/share/xcat/ib/scripts/Mellanox/mlnxofed_ib_install.v2 $script
# chmod +x $script
# ls -l $script
-rwxr-xr-x 1 root root 14747 Feb 13 13:11
/install/postscripts/mlnxofed_ib_install
```

3. Include the script in the list of `postbootscripts` of the node group.

   Include the `--add-kernel-support` argument to the list of default arguments (`--without-32bit --without-fw-update --force`, according to the xCAT documentation) to rebuild kernel modules for the installed kernel version.

```
# lsdef -t group s822lc -i postbootscripts
Object name: s822lc
    postbootscripts=

# file=MLNX_OFED_LINUX-3.2-1.0.1.1-rhel7.2-ppc64le.iso
# args='--add-kernel-support --without-32bit --without-fw-update --force'

# chdef -t group s822lc --plus postbootscripts="mlnxofed_ib_install -p
$dir/$file -m $args -end-"
1 object definitions have been created or modified.

# lsdef -t group s822lc -i postbootscripts
Object name: s822lc
    postbootscripts=mlnxofed_ib_install -p
/install/mofed/rh/ppc64le//MLNX_OFED_LINUX-3.2-1.0.1.1-rhel7.2-ppc64le.iso -m
--add-kernel-support --without-32bit --without-fw-update --force -end-
```

4. Add the InfiniBand package list (and `createrepo package` dependency) to the package list:

```
# ib_list=/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist

# lsdef -t osimage rh72-hpc-diskful -i pkglist
Object name: rh72-hpc-diskful
    pkglist=/install/custom/install/rh/rh72-hpc.pkglist

# pkglist=/install/custom/install/rh/rh72-hpc.pkglist

# cat <<EOF >>$pkglist

# Infiniband with Mellanox OFED for RHEL 7.2 (original pkglist)
createrepo
#INCLUDE:${ib_list}#
EOF

# cat $pkglist
# RHEL Server 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist#

# CUDA Toolkit 7.5 for RHEL 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist#

# Infiniband with Mellanox OFED for RHEL 7.2 (original pkglist)
createrepo
#INCLUDE:/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist#
```

Note: If the `createrepo` package is not specified in the list, the following error message is shown during the Mellanox OFED installation:

```
ERROR: createrepo is not installed!
createrepo package is needed for building a repository from MLNX_OFED rpms
```

5.6.5 XL C/C++ Compiler

To install the XL C/C++ Compiler xCAT kit, complete the following steps. For more information and details, see the following xCAT documentation page:

http://xcat-docs.readthedocs.org/en/2.11/advanced/kit/hpc/software/compilers.html

1. Download the partial kit (distributed by the xCAT project):

```
# mkdir /tmp/xlc
# cd /tmp/xlc

# curl -sOL
http://xcat.org/files/kits/hpckits/2.11/rhels7.2/ppc64le/xlc-13.1.2-0-ppc64le.N
EED_PRODUCT_PKGS.tar.bz2
```

2. Build the complete kit by combining the partial kit and the product packages with the **buildkit** command:

```
# dir=/path/to/xlc-rpm-packages/

# ls -1 $dir
libxlc-13.1.2.0-150526a.ppc64le.rpm
libxlc-devel.13.1.2-13.1.2.0-150526a.ppc64le.rpm
libxlmass-devel.8.1.2-8.1.2.0-150526.ppc64le.rpm
libxlsmp-4.1.2.0-150526.ppc64le.rpm
libxlsmp-devel.4.1.2-4.1.2.0-150526.ppc64le.rpm
xlc.13.1.2-13.1.2.0-150526a.ppc64le.rpm
xlc-license.13.1.2-13.1.2.0-150526a.ppc64le.rpm

# buildkit addpkgs xlc-13.1.2-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir $dir
Extracting tar file /tmp/xlc/xlc-13.1.2-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2.
<...>
Kit tar file /tmp/xlc/xlc-13.1.2-0-ppc64le.tar.bz2 successfully built.

# ls -1
xlc-13.1.2-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
xlc-13.1.2-0-ppc64le.tar.bz2
```

3. Add the kit to xCAT with the **addkit** command:

```
# lsdef -t kit
Could not find any object definitions to display.

# addkit xlc-13.1.2-0-ppc64le.tar.bz2
Adding Kit xlc-13.1.2-0-ppc64le
Kit xlc-13.1.2-0-ppc64le was successfully added.

# lsdef -t kit
xlc-13.1.2-0-ppc64le  (kit)
```

4. Verify its `kitcomponent` objects (and `description` fields) with the **lsdef** command:

```
# lsdef -t kitcomponent -w kitname==xlc-13.1.2-0-ppc64le -i description
Object name: xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le
    description=XLC13 for compiler kitcomponent
Object name: xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le
    description=XLC13 license kitcomponent
Object name: xlc.rte-compute-13.1.2-0-rhels-7.2-ppc64le
    description=XLC13 for runtime kitcomponent
```

5. Add the following `kitcomponent` (and dependencies) to the `osimage` object with the **addkitcomp** command:

```
# lsdef -t osimage rh72-hpc-diskful -i kitcomponents,otherpkglist
Object name: rh72-hpc-diskful
    kitcomponents=
    otherpkglist=

# addkitcomp --adddeps -i rh72-hpc-diskful \
    xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le
Assigning kit component xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le to
osimage rh72-hpc-diskful
Kit components xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le were added to
osimage rh72-hpc-diskful successfully

# lsdef -t osimage rh72-hpc-diskful -i kitcomponents,otherpkglist
Object name: rh72-hpc-diskful
    kitcomponents=xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le,xlc.rte-comput
e-13.1.2-0-rhels-7.2-ppc64le,xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le
    otherpkglist=/install/osimages/rh72-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh72-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist
```

5.6.6 XL Fortran Compiler

To install the XL Fortran Compiler xCAT kit, complete the following steps. For more information and details, see the following xCAT documentation page:

http://xcat-docs.readthedocs.org/en/2.11/advanced/kit/hpc/software/compilers.html

1. Download the partial kit (distributed by the xCAT project):

```
# mkdir /tmp/xlf
# cd /tmp/xlf

# curl -sOL
http://xcat.org/files/kits/hpckits/2.11/rhels7.2/ppc64le/xlf-15.1.2-0-ppc64le.N
EED_PRODUCT_PKGS.tar.bz2
```

2. Build the complete kit by combining the partial kit and the product packages distributed in the installation media with the **buildkit** command:

```
# dir=/path/to/xlf-rpm-packages/

# ls -1 $dir
libxlf-15.1.2.0-150526a.ppc64le.rpm
libxlf-devel.15.1.2-15.1.2.0-150526a.ppc64le.rpm
libxlmass-devel.8.1.2-8.1.2.0-150526.ppc64le.rpm
libxlsmp-4.1.2.0-150526.ppc64le.rpm
libxlsmp-devel.4.1.2-4.1.2.0-150526.ppc64le.rpm
xlf.15.1.2-15.1.2.0-150526a.ppc64le.rpm
xlf-license.15.1.2-15.1.2.0-150526a.ppc64le.rpm

# buildkit addpkgs xlf-15.1.2-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir $dir
Extracting tar file /tmp/xlf/xlf-15.1.2-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2.
<...>
Kit tar file /tmp/xlf/xlf-15.1.2-0-ppc64le.tar.bz2 successfully built.
```

```
# ls -1
xlf-15.1.2-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
xlf-15.1.2-0-ppc64le.tar.bz2
```

3. Add the kit to xCAT with the **addkit** command:

```
# addkit xlf-15.1.2-0-ppc64le.tar.bz2
Adding Kit xlf-15.1.2-0-ppc64le
Kit xlf-15.1.2-0-ppc64le was successfully added.

# lsdef -t kit
xlc-13.1.2-0-ppc64le  (kit)
xlf-15.1.2-0-ppc64le  (kit)
```

4. Verify its `kitcomponent` objects (and `description` fields) with the **lsdef** command:

```
# lsdef -t kitcomponent -w kitname==xlf-15.1.2-0-ppc64le -i description
Object name: xlf.compiler-compute-15.1.2-0-rhels-7.2-ppc64le
    description=XLF15 for compiler kitcomponent
Object name: xlf.license-compute-15.1.2-0-rhels-7.2-ppc64le
    description=XLF15 license kitcomponent
Object name: xlf.rte-compute-15.1.2-0-rhels-7.2-ppc64le
    description=XLF15 for runtime kitcomponent
```

5. Add the following `kitcomponent` (and dependencies) to the `osimage` object with the **addkitcomp** command:

```
# addkitcomp --adddeps -i rh72-hpc-diskful \
    xlf.compiler-compute-15.1.2-0-rhels-7.2-ppc64le
Assigning kit component xlf.compiler-compute-15.1.2-0-rhels-7.2-ppc64le to
osimage rh72-hpc-diskful
Kit components xlf.compiler-compute-15.1.2-0-rhels-7.2-ppc64le were added to
osimage rh72-hpc-diskful successfully

# lsdef -t osimage rh72-hpc-diskful -i kitcomponents
Object name: rh72-hpc-diskful
    kitcomponents=xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le,xlc.rte-comput
e-13.1.2-0-rhels-7.2-ppc64le,xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le,xl
f.license-compute-15.1.2-0-rhels-7.2-ppc64le,xlf.rte-compute-15.1.2-0-rhels-7.2
-ppc64le,xlf.compiler-compute-15.1.2-0-rhels-7.2-ppc64le
```

5.6.7 Advance Toolchain

To install the Advance Toolchain, complete the following steps:

1. Download the product packages:

```
# dir=/install/post/otherpkgs/rhels7.2/ppc64le/at8.0/
# mkdir -p $dir
# cd $dir

# wget
'ftp://ftp.unicamp.br/pub/linuxpatch/toolchain/at/redhat/RHEL7/at8.0/advance-to
olchain-at8.0-*-8.0-5.ppc64le.rpm'
<...>
=> advance-toolchain-at8.0-devel-8.0-5.ppc64le.rpm
<...>
=> advance-toolchain-at8.0-mcore-libs-8.0-5.ppc64le.rpm
<...>
```

```
=> advance-toolchain-at8.0-perf-8.0-5.ppc64le.rpm
<...>
=> advance-toolchain-at8.0-runtime-8.0-5.ppc64le.rpm
<...>
=> advance-toolchain-at8.0-selinux-8.0-5.ppc64le.rpm
<...>
```

2. Create a package repository with the **createrepo** command:

```
# createrepo .
<...>
```

3. Create a package list with a combination of the **rpm**, **awk**, **sed**, and **grep** commands:

```
# list=/install/custom/rhels7.2/at8.0.otherpkgs.pkglist

# mkdir -p $(dirname $list)
# rpm -qip *.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename $(pwd))/:"
| grep -v selinux >$list
<...>

# cat $list
at8.0/advance-toolchain-at8.0-devel
at8.0/advance-toolchain-at8.0-mcore-libs
at8.0/advance-toolchain-at8.0-perf
at8.0/advance-toolchain-at8.0-runtime
```

4. Include the package list in the `osimage` object with the **chdef** command:

```
# chdef -t osimage rh72-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i otherpkglist
Object name: rh72-hpc-diskful
    otherpkglist=/install/osimages/rh72-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh72-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.2/at8.0.otherpkgs.pkglist
```

5.6.8 PE RTE

To install PE RTE, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT with the **addkit** command:

```
# addkit /path/to/pe-rte-files/pperte-2.3.0.0-1547a-ppc64le.tar.bz2
Adding Kit pperte-2.3.0.0-1547a-ppc64le
Kit pperte-2.3.0.0-1547a-ppc64le was successfully added.

# lsdef -t kit
pperte-2.3.0.0-1547a-ppc64le  (kit)
xlc-13.1.2-0-ppc64le  (kit)
xlf-15.1.2-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) with the **lsdef** command:

```
# lsdef -t kitcomponent -w kitname==pperte-2.3.0.0-1547a-ppc64le -i description
Object name: min-pperte-compute-2.3.0.0-1547a-rhels-7.2-ppc64le
    description=Minimal PE RTE for compute nodes
Object name: pperte-compute-2.3.0.0-1547a-rhels-7.2-ppc64le
    description=PE RTE for compute nodes
```

```
Object name: pperte-license-2.3.0.0-1547a-rhels-7.2-ppc64le
    description=PE RTE License
Object name: pperte-login-2.3.0.0-1547a-rhels-7.2-ppc64le
    description=PE RTE for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object with the **addkitcomp** command:

```
# addkitcomp --adddeps -i rh72-hpc-diskful \
    pperte-compute-2.3.0.0-1547a-rhels-7.2-ppc64le
Assigning kit component pperte-compute-2.3.0.0-1547a-rhels-7.2-ppc64le to
osimage rh72-hpc-diskful
Kit components pperte-compute-2.3.0.0-1547a-rhels-7.2-ppc64le were added to
osimage rh72-hpc-diskful successfully

# lsdef -t osimage rh72-hpc-diskful -i kitcomponents
Object name: rh72-hpc-diskful
    kitcomponents=xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le,xlc.rte-comput
e-13.1.2-0-rhels-7.2-ppc64le,xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le,xl
f.license-compute-15.1.2-0-rhels-7.2-ppc6
4le,xlf.rte-compute-15.1.2-0-rhels-7.2-ppc64le,xlf.compiler-compute-15.1.2-0-rh
els-7.2-ppc64le,pperte-license-2.3.0.0-1547a-rhels-7.2-ppc64le,pperte-compute-2
.3.0.0-1547a-rhels-7.2-ppc64le
```

4. Create a script to perform the configuration changes suggested by the `pe_node_diag` utility (distributed with PE RTE):

```
# script=/install/postscripts/config_pe_node_diag

# cat <<"EOF" >$script
#!/bin/sh

# Script: config_pe_node_diag: fix issues reported by pe_node_diag
# Author: Mauricio Faria de Oliveira <mauricfo@linux.vnet.ibm.com>
# Changelog:
# - 20151121: initial version; support for RHEL Server 7.2 LE.

# Sample output:
#
#    # /opt/ibmhpc/pecurrent/ppe.poe/bin/pe_node_diag
#
#    Issue 1: net.core.wmem_max is 229376 but 1048576 is recommended.
#        (A) sysctl -w net.core.wmem_max=1048576
#
#    Issue 2: net.core.rmem_max is 229376 but 8388608 is recommended.
#        (A) sysctl -w net.core.rmem_max=8388608
#
#    Issue 3: net.ipv4.ipfrag_high_thresh is 4194304 but 8388608 is
recommended.
#        (A) sysctl -w net.ipv4.ipfrag_high_thresh=8388608
#
#    ATTENTION: kernel.shmall is calculated by dividing kernel.shmmax by
#    the system default memory page size of 65536 bytes.
#    If parallel jobs will be using a different page size,
#    kernel.shmall must be adjusted accordingly.
#
#    Issue 4: ulimit for nofile is 1024 but 4096 is recommended.
#        (M) Update nofile to be 4096 in /etc/security/limits.conf.
```

```
#
#    Issue 5: ulimit for nproc is 977056 but 2067554 is recommended.
#        (M) Update nproc to be 2067554 in /etc/security/limits.conf.
#
#    Issue 6: ulimit for memlock is 64 but unlimited is recommended.
#        (M) Update memlock to be unlimited in /etc/security/limits.conf.
#        (M) Restart xinetd with 'service xinetd restart'.
#
#    Issue 7: per_source is 10 in /etc/xinetd.conf but 160 is recommended.
#        (M) Change per_source to 160 in /etc/xinetd.conf.
#        (M) Restart xinetd with 'service xinetd restart'.

PE_NODE_DIAG='/opt/ibmhpc/pecurrent/ppe.poe/bin/pe_node_diag'

# Config files
SYSCTL_CONF='/etc/sysctl.d/90-ibm-pe-rte.conf'
LIMITS_CONF='/etc/security/limits.d/90-ibm-pe-rte.conf'
XINETD_CONF='/etc/xinetd.conf' # no effect with /etc/xinetd.d/
XINETD_CONF_ORIG="$XINETD_CONF.orig"

remove() {
    echo "$(basename $0) :: Removing changes.."
    rm -f $SYSCTL_CONF
    rm -f $LIMITS_CONF
    [ -f "${XINETD_CONF_ORIG}" ] && mv ${XINETD_CONF_ORIG} $XINETD_CONF || true
}

# Error handling
error() {
    echo "$(basename $0) :: Error on line $1."
    remove
    echo "$(basename $0) :: Exiting."
    exit 1
}

trap 'error $LINENO' ERR
set -e

# Remove option
if [ "$1" = '-r' ]; then
    remove
    exit 0
fi

# sysctl.conf
# -----------

$PE_NODE_DIAG | sed -n "s/.*sysctl -w //p" > $SYSCTL_CONF

echo "Changes on $SYSCTL_CONF"
cat $SYSCTL_CONF
echo

echo 'Loading changes..'
sysctl --system
```

```
# limits.conf
# -----------

$PE_NODE_DIAG | awk '/limits.conf/ { print "* - " $3 " " $6 }' > $LIMITS_CONF

echo "Changes on $LIMITS_CONF"
cat $LIMITS_CONF
echo


# xinetd.conf
# -----------

cp -a $XINETD_CONF ${XINETD_CONF_ORIG}
$PE_NODE_DIAG \
    | awk "/Change [^ ]+ to [^ ]+ in \/etc\/xinetd.conf/ { print \$3, \$5 }" \
    | while read variable value; do
        sed "/[ \t]*${variable}/ { H; s/^/#/; p; x; s/= .*/= ${value}/; } " -i
$XINETD_CONF
    done

echo "Changes on $XINETD_CONF"
diff ${XINETD_CONF_ORIG} $XINETD_CONF || true

echo 'Restarting xinetd..'
systemctl restart xinetd.service

exit 0
EOF

# chmod +x $script

# ls -l $script
-rwxr-xr-x 1 root root 3098 Feb 13 14:35
/install/postscripts/config_pe_node_diag
```

5. Include the script in the `postbootscripts` list.

> **Note:** The `postbootscripts` are executed after `postscripts` (the point at which kits are installed during the `osimage` installation). Therefore, the PE RTE contents are already installed by the time the script runs.
>
> For more information and details, see the following xCAT documentation page:
>
> http://xcat-docs.readthedocs.org/en/latest/guides/admin-guides/manage_clusters/common/deployment/prepostscripts/post_script.html

```
# chdef -t osimage rh72-hpc-diskful --plus postbootscripts=$(basename $script)
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i postbootscripts
Object name: rh72-hpc-diskful
```

```
    postbootscripts=KIT_pperte-compute-2.3.0.0-1547a-rhels-7.2-ppc64le_pperte_p
ostboot,config_pe_node_diag
```

6. Include the `ksh` package dependency (for the PE RTE Installation Verification Program) in the package list:

```
# list=/install/custom/install/rh/rh72-hpc.pkglist

# cat <<"EOF" >>$list

# PE RTE IVP
ksh
EOF

# cat /install/custom/install/rh/rh72-hpc.pkglist
# RHEL Server 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist#

# CUDA Toolkit 7.5 for RHEL 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist#

# Infiniband with Mellanox OFED for RHEL 7.2 (original pkglist)
createrepo
#INCLUDE:/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist#

# PE RTE IVP
ksh
```

5.6.9 PE DE

To install PE DE, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT with the **`addkit`** command:

```
# addkit /path/to/pe-de-files/ppedev-2.2.0-0.tar.bz2
Adding Kit ppedev-2.2.0-0
Kit ppedev-2.2.0-0 was successfully added.

# lsdef -t kit
ppedev-2.2.0-0  (kit)
pperte-2.3.0.0-1547a-ppc64le  (kit)
xlc-13.1.2-0-ppc64le  (kit)
xlf-15.1.2-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) with the **`lsdef`** command:

```
# lsdef -t kitcomponent -w kitname==ppedev-2.2.0-0 -i description
Object name: ppedev.compute-2.2.0-0-rhels-7.2-ppc64le
    description=Parallel Environment Developer Edition for compute nodes
Object name: ppedev.license-2.2.0-0-rhels-7.2-ppc64le
    description=Parallel Environment Developer Edition license package
Object name: ppedev.login-2.2.0-0-rhels-7.2-ppc64le
    description=Parallel Environment Developer Edition for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object with the **addkitcomp** command:

```
# addkitcomp --adddeps -i rh72-hpc-diskful \
    ppedev.compute-2.2.0-0-rhels-7.2-ppc64le
Assigning kit component ppedev.compute-2.2.0-0-rhels-7.2-ppc64le to osimage
rh72-hpc-diskful
Kit components ppedev.compute-2.2.0-0-rhels-7.2-ppc64le were added to osimage
rh72-hpc-diskful successfully

# lsdef -t osimage rh72-hpc-diskful -i kitcomponents
Object name: rh72-hpc-diskful
    kitcomponents=xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le,xlc.rte-comput
e-13.1.2-0-rhels-7.2-ppc64le,xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le,xl
f.license-compute-15.1.2-0-rhels-7.2-ppc6
4le,xlf.rte-compute-15.1.2-0-rhels-7.2-ppc64le,xlf.compiler-compute-15.1.2-0-rh
els-7.2-ppc64le,pperte-license-2.3.0.0-1547a-rhels-7.2-ppc64le,pperte-compute-2
.3.0.0-1547a-rhels-7.2-ppc64le,ppedev.license-2.2.0-0-rhels-7.2-ppc64le,ppedev.
compute-2.2.0-0-rhels-7.2-ppc64le
```

> **Note:** To perform development/profiling tasks on a compute node, install the following `kitcomponent`: `ppedev.login-2.2.0-0-rhels-7.2-ppc64le`.
>
> You can also install an official RHEL package update to the `glibc` packages to avoid the `monstartup: out of memory` error when running debug applications (not covered in this book).

5.6.10 ESSL

To install ESSL, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT with the **addkit** command:

```
# addkit /path/to/essl-files/essl-5.4.0-0-ppc64le.tar.bz2
Adding Kit essl-5.4.0-0-ppc64le
Kit essl-5.4.0-0-ppc64le was successfully added.

# lsdef -t kit
essl-5.4.0-0-ppc64le  (kit)
ppedev-2.2.0-0  (kit)
pperte-2.3.0.0-1547a-ppc64le  (kit)
xlc-13.1.2-0-ppc64le  (kit)
xlf-15.1.2-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) with the **lsdef** command:

```
# lsdef -t kitcomponent -w kitname==essl-5.4.0-0-ppc64le -i description
Object name: essl-computenode-3264rte-5.4.0-0-rhels-7.2-ppc64le
    description=essl for compute nodes with 3264 rte only
Object name: essl-computenode-3264rtecuda-5.4.0-0-rhels-7.2-ppc64le
    description=essl for compute nodes with 3264 rte cuda only
Object name: essl-computenode-5.4.0-0-rhels-7.2-ppc64le
    description=essl for compute nodes
Object name: essl-computenode-6464rte-5.4.0-0-rhels-7.2-ppc64le
    description=essl for compute nodes with 6464 rte only
Object name: essl-license-5.4.0-0-rhels-7.2-ppc64le
    description=essl license for compute nodes
```

```
Object name: essl-loginnode-5.4.0-0-rhels-7.2-ppc64le
    description=essl for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object with the **`addkitcomp`** command:

```
# addkitcomp --adddeps -i rh72-hpc-diskful \
    essl-computenode-5.4.0-0-rhels-7.2-ppc64le
Assigning kit component essl-computenode-5.4.0-0-rhels-7.2-ppc64le to osimage
rh72-hpc-diskful
Kit components essl-computenode-5.4.0-0-rhels-7.2-ppc64le were added to osimage
rh72-hpc-diskful successfully

# lsdef -t osimage rh72-hpc-diskful -i kitcomponents
Object name: rh72-hpc-diskful
    kitcomponents=xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le,xlc.rte-comput
e-13.1.2-0-rhels-7.2-ppc64le,xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le,xl
f.license-compute-15.1.2-0-rhels-7.2-ppc6
4le,xlf.rte-compute-15.1.2-0-rhels-7.2-ppc64le,xlf.compiler-compute-15.1.2-0-rh
els-7.2-ppc64le,pperte-license-2.3.0.0-1547a-rhels-7.2-ppc64le,pperte-compute-2
.3.0.0-1547a-rhels-7.2-ppc64le,ppedev.lic
ense-2.2.0-0-rhels-7.2-ppc64le,ppedev.compute-2.2.0-0-rhels-7.2-ppc64le,essl-li
cense-5.4.0-0-rhels-7.2-ppc64le,essl-computenode-5.4.0-0-rhels-7.2-ppc64le
```

5.6.11 PESSL

To install PESSL, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT with the **`addkit`** command:

```
# addkit /path/to/pessl-files/pessl-5.2.0-0-ppc64le.tar.bz2
Adding Kit pessl-5.2.0-0-ppc64le
Kit pessl-5.2.0-0-ppc64le was successfully added.

# lsdef -t kit
essl-5.4.0-0-ppc64le  (kit)
pessl-5.2.0-0-ppc64le  (kit)
ppedev-2.2.0-0  (kit)
pperte-2.3.0.0-1547a-ppc64le  (kit)
xlc-13.1.2-0-ppc64le  (kit)
xlf-15.1.2-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) with the **`lsdef`** command:

```
# lsdef -t kitcomponent -w kitname==pessl-5.2.0-0-ppc64le -i description
Object name: pessl-computenode-3264rtempich-5.2.0-0-rhels-7.2-ppc64le
    description=pessl for compute nodes with ESSL non-cuda runtime
Object name: pessl-computenode-5.2.0-0-rhels-7.2-ppc64le
    description=pessl for compute nodes
Object name: pessl-license-5.2.0-0-rhels-7.2-ppc64le
    description=pessl license for compute nodes
Object name: pessl-loginnode-5.2.0-0-rhels-7.2-ppc64le
    description=pessl for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object with the **`addkitcomp`** command:

```
# addkitcomp --adddeps -i rh72-hpc-diskful \
    pessl-computenode-5.2.0-0-rhels-7.2-ppc64le
```

```
Assigning kit component pessl-computenode-5.2.0-0-rhels-7.2-ppc64le to osimage
rh72-hpc-diskful
Kit components pessl-computenode-5.2.0-0-rhels-7.2-ppc64le were added to
osimage rh72-hpc-diskful successfully

# lsdef -t osimage rh72-hpc-diskful -i kitcomponents
Object name: rh72-hpc-diskful
    kitcomponents=xlc.license-compute-13.1.2-0-rhels-7.2-ppc64le,xlc.rte-comput
e-13.1.2-0-rhels-7.2-ppc64le,xlc.compiler-compute-13.1.2-0-rhels-7.2-ppc64le,xl
f.license-compute-15.1.2-0-rhels-7.2-ppc6
4le,xlf.rte-compute-15.1.2-0-rhels-7.2-ppc64le,xlf.compiler-compute-15.1.2-0-rh
els-7.2-ppc64le,pperte-license-2.3.0.0-1547a-rhels-7.2-ppc64le,pperte-compute-2
.3.0.0-1547a-rhels-7.2-ppc64le,ppedev.lic
ense-2.2.0-0-rhels-7.2-ppc64le,ppedev.compute-2.2.0-0-rhels-7.2-ppc64le,essl-li
cense-5.4.0-0-rhels-7.2-ppc64le,essl-computenode-5.4.0-0-rhels-7.2-ppc64le,pess
l-license-5.2.0-0-rhels-7.2-ppc64le,pessl-computenode-5.2.0-0-rhels-7.2-ppc64le
```

5.6.12 Spectrum Scale (formerly GPFS)

This section provides the instructions to install the packages for Spectrum Scale 4.1.1.0, the package updates for Spectrum Scale 4.1.1.3, and to build and install the GPFS Portability Layer (GPL) packages.

The process to build the GPL requires an already provisioned node (for example, compute, login, or management node) with Spectrum Scale packages installed (specifically, `gpfs.gpl`). Therefore, if not using the management node to build the GPL, you cannot install Spectrum Scale with the GPL (requirement) in a single provisioning stage for one of the nodes, which is used to build the GPL. It is possible, afterward, provided the built GPL package is made available for download in the management node (like other Spectrum Scale packages) for installation on other nodes.

For more information and details about the installation process, see the *Installing GPFS on Linux nodes* page in the Knowledge Center:

http://www.ibm.com/support/knowledgecenter/STXKQY_4.1.1/com.ibm.spectrum.scale.v4r11.ins.doc/bl1ins_loosein.htm

In order to install Spectrum Scale 4.1.1.3 (on top of 4.1.1.0), perform the following steps:

- Install a script for environment configuration
- Install the packages for Spectrum Scale 4.1.1.0
- Install the packages for Spectrum Scale 4.1.1.3
- Perform the GPL build

Install a script for environment configuration

To install the script for environment configuration, complete these steps:

1. Create a script to set the `PATH` environment variable:

```
# script=/install/postscripts/gpfs-path

# cat <<EOF >$script
#!/bin/bash

profile='/etc/profile.d/gpfs.sh'
echo 'export PATH=\$PATH:/usr/lpp/mmfs/bin' >\$profile
EOF
```

```
# chmod +x $script

# ls -l $script
-rwxr-xr-x 1 root root 99 Nov 30 17:24 /install/postscripts/gpfs-path
```

2. Include it in the `postscripts` list of the `osimage` object with the **`chdef`** command:

```
# lsdef -t osimage rh72-hpc-diskful -i postscripts
Object name: rh72-hpc-diskful
    postscripts=nvidia-power-limit

# chdef -t osimage rh72-hpc-diskful --plus postscripts='gpfs-path'
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i postscripts
Object name: rh72-hpc-diskful
    postscripts=nvidia-power-limit,gpfs-path
```

Install the packages for Spectrum Scale 4.1.1.0

To install the packages, complete the following steps:

1. Extract the product packages of Spectrum Scale **4.1.1.0**:

```
# dir=/install/post/otherpkgs/rhels7.2/ppc64le/gpfs-4110
# mkdir -p $dir

# /path/to/Spectrum_Scale_install-4.1.1.0_ppc64le_standard --silent --dir $dir
<...>
Extracting Product RPMs to /install/post/otherpkgs/rhels7.2/ppc64le/gpfs-4110
<...>
```

2. Create the respective package repository with the **`createrepo`** command:

```
# createrepo $dir
<...>

# ls -1 $dir
gpfs.base_4.1.1-0_ppc64el.deb
gpfs.base-4.1.1-0.ppc64le.rpm
gpfs.docs_4.1.1-0_all.deb
gpfs.docs-4.1.1-0.noarch.rpm
gpfs.ext_4.1.1-0_ppc64el.deb
gpfs.ext-4.1.1-0.ppc64le.rpm
gpfs.gpl_4.1.1-0_all.deb
gpfs.gpl-4.1.1-0.noarch.rpm
gpfs.gskit_8.0.50-40_ppc64el.deb
gpfs.gskit-8.0.50-40.ppc64le.rpm
gpfs.hadoop-2-connector-4.1.1-0.ppc64le.rpm
gpfs.msg.en-us_4.1.1-0_all.deb
gpfs.msg.en_US-4.1.1-0.noarch.rpm
license
manifest
repodata
```

3. Create the respective package list with a combination of commands:

```
# list=/install/custom/rhels7.2/gpfs-4110.otherpkgs.pkglist

# rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" | grep -v hadoop >$list

# cat $list
gpfs-4110/gpfs.base
gpfs-4110/gpfs.docs
gpfs-4110/gpfs.ext
gpfs-4110/gpfs.gpl
gpfs-4110/gpfs.gskit
gpfs-4110/gpfs.msg.en_US
```

4. Include the respective package list in the `otherpkglist` attribute of the `osimage` object with the **chdef** command:

```
# chdef -t osimage rh72-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i otherpkglist
Object name: rh72-hpc-diskful
    otherpkglist=/install/osimages/rh72-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh72-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.2/at8.0.otherpkgs.pkglist,/install/custom/rhels7
.2/gpfs-4110.otherpkgs.pkglist
```

Install the packages for Spectrum Scale 4.1.1.3

To install the packages, complete the following steps:

1. Extract the product packages of Spectrum Scale **4.1.1.3**:

```
# dir=/install/post/otherpkgs/rhels7.2/ppc64le/gpfs-4113
# mkdir -p $dir

# /root/software/gpfs/Spectrum_Scale_Standard-4.1.1.3-ppc64LE-Linux-update
--silent --dir $dir
<...>
Product rpms successfully extracted to
/install/post/otherpkgs/rhels7.2/ppc64le/gpfs-4113
```

2. Create the respective package repository with the **createrepo** command:

```
# createrepo $dir
<...>

# ls -1 $dir
gpfs.base_4.1.1-3_ppc64el_update.deb
gpfs.base-4.1.1-3.ppc64le.update.rpm
gpfs.docs_4.1.1-3_all.deb
gpfs.docs-4.1.1-3.noarch.rpm
gpfs.ext_4.1.1-3_ppc64el_update.deb
gpfs.ext-4.1.1-3.ppc64le.update.rpm
gpfs.gpl_4.1.1-3_all.deb
gpfs.gpl-4.1.1-3.noarch.rpm
gpfs.gskit_8.0.50-47_ppc64el.deb
gpfs.gskit-8.0.50-47.ppc64le.rpm
gpfs.hadoop-connector_2.7.0-2_ppc64el.deb
```

```
gpfs.hadoop-connector-2.7.0-2.ppc64le.rpm
gpfs.msg.en-us_4.1.1-3_all.deb
gpfs.msg.en_US-4.1.1-3.noarch.rpm
manifest
repodata
```

3. Create the respective package list with a combination of commands:

> **Note:** Use the `#NEW_INSTALL_LIST#` directive to perform package installation of Spectrum Scale 4.1.1.0 and 4.1.1.3 in different stages to ensure the update process (from version 4.1.1.0 to version 4.1.1.3) occurs correctly.
>
> For more information and details, see the following xCAT documentation page:
>
> http://xcat-docs.readthedocs.org/en/latest/guides/admin-guides/manage_clusters/common/deployment/additionalpkg/nonubuntu_os_other_pkg.html#file-format-for-otherpkgs-pkglist-file

```
# list=/install/custom/rhels7.2/gpfs-4113.otherpkgs.pkglist

# echo '#NEW_INSTALL_LIST#' > $list
# rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" | grep -v hadoop >>$list

# cat $list
#NEW_INSTALL_LIST#
gpfs-4113/gpfs.base
gpfs-4113/gpfs.docs
gpfs-4113/gpfs.ext
gpfs-4113/gpfs.gpl
gpfs-4113/gpfs.gskit
gpfs-4113/gpfs.msg.en_US
```

4. Include the respective package list in the `otherpkglist` attribute of the `osimage` object with the **`chdef`** command:

```
# chdef -t osimage rh72-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i otherpkglist
Object name: rh72-hpc-diskful
    otherpkglist=/install/osimages/rh72-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh72-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.2/at8.0.otherpkgs.pkglist,/install/custom/rhels7
.2/gpfs-4110.otherpkgs.pkglist,/install/custom/rhels7.2/gpfs-4113.otherpkgs.pkg
list
```

Perform the GPL build

The process to build the GPL requires a working node (for example, a management, login, or compute node) with Spectrum Scale packages (specifically, `gpfs.gpl`) installed.

The node must be capable of building *out-of-tree* kernel modules (that is, with development packages such as `gcc`, `make`, and `kernel-devel` installed, for example), and run the same kernel packages version and processor architecture as the target compute nodes for the GPL produced binaries.

For example, you can build the GPL in the management node or an already installed compute node (if any), if it is a system based on POWER8 running RHEL Server 7.2 for ppc64le, and matches the kernel packages version of the target compute nodes. This section describes the build process with an already installed compute node.

> **Note:** The GPL must be rebuilt and reinstalled in the event of updates to the kernel packages. This process requires the `kernel-devel` package for the respective update version.

To perform the GPL build, complete the following steps:

1. Verify the Spectrum Scale installation (`PATH` environment variable and RPM packages):

```
# xdsh p8r2n2 'echo $PATH' | grep mmfs
p8r2n2: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/lpp/mmfs/bin:
/opt/ibutils/bin

# xdsh p8r2n2 'rpm -qa | grep ^gpfs'
p8r2n2: gpfs.msg.en_US-4.1.1-3.noarch
p8r2n2: gpfs.gpl-4.1.1-3.noarch
p8r2n2: gpfs.gskit-8.0.50-47.ppc64le
p8r2n2: gpfs.docs-4.1.1-3.noarch
p8r2n2: gpfs.ext-4.1.1-3.ppc64le
p8r2n2: gpfs.base-4.1.1-3.ppc64le
```

2. Build the GPL.

   Verify that the build process writes the resulting GPL binary RPM package (`gpfs.gplbin-<kernel version>.<architecture>-<spectrum scale version>.rpm`), and finishes with an *exit code* of *zero* (success).

```
# xdsh p8r2n2 --stream 'cd /usr/lpp/mmfs/src && make Autoconfig && make World
&& make InstallImages && make rpm'
<...>
p8r2n2: Verifying that tools to build the portability layer exist....
p8r2n2: cpp present
p8r2n2: gcc present
p8r2n2: g++ present
p8r2n2: ld present
<...>
p8r2n2: Wrote:
/root/rpmbuild/RPMS/ppc64le/gpfs.gplbin-3.10.0-327.el7.ppc64le-4.1.1-3.ppc64le.
rpm
<...>
p8r2n2: + exit 0
```

3. Copy the package and create the respective package repository with the **`createrepo`** command:

```
# dir=/install/post/otherpkgs/rhels7.2/ppc64le/gpfs-gpl
# mkdir -p $dir

# scp p8r2n2:/root/rpmbuild/RPMS/ppc64le/gpfs.gplbin-*.rpm $dir
gpfs.gplbin-3.10.0-327.el7.ppc64le-4.1.1-3.ppc64le.rpm   <...>

# createrepo $dir
<...>

# ls -1 $dir
```

```
gpfs.gplbin-3.10.0-327.el7.ppc64le-4.1.1-3.ppc64le.rpm
repodata
```

4. Create the respective package list with these commands:

```
# list=/install/custom/rhels7.2/gpfs-gpl.otherpkgs.pkglist

# echo '#NEW_INSTALL_LIST#' >$list
# rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" >>$list

# cat $list
#NEW_INSTALL_LIST#
gpfs-gpl/gpfs.gplbin-3.10.0-327.el7.ppc64le
```

5. Include the respective package list in the `otherpkglist` attribute of the `osimage` object with the **chdef** command.

   This step allows the GPL package to be installed automatically (without rebuild steps) during the provisioning stage of other nodes.

```
# chdef -t osimage rh72-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

# lsdef -t osimage rh72-hpc-diskful -i otherpkglist
Object name: rh72-hpc-diskful
    otherpkglist=/install/osimages/rh72-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh72-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.2/at8.0.otherpkgs.pkglist,/install/custom/rhels7
.2/gpfs-4110.otherpkgs.pkglist,/install/custom/rhels7.2/gpfs-4113.otherpkgs.pkg
list,/install/custom/rhels7.2/gpfs-gpl.otherpkgs.pkglist
```

Note: You can install the GPL in an already installed and running compute node with the **otherpkgs** script of the **updatenode** command:

```
# updatenode p8r3n3 -P otherpkgs
<...>
p8r3n3: pkgsarray:  gpfs-gpl/gpfs.gplbin-3.10.0-327.el7.ppc64le, 2
<...>

# xdsh p8r3n3 'rpm -qa | grep ^gpfs.gpl'
p8r3n3: gpfs.gpl-4.1.1-3.noarch
p8r3n3: gpfs.gplbin-3.10.0-327.el7.ppc64le-4.1.1-3.ppc64le
```

For more information about configuration and usage, see the *Steps to establishing and starting your GPFS cluster* page in the Knowledge Center:

http://www.ibm.com/support/knowledgecenter/STXKQY_4.1.1/com.ibm.spectrum.scale.v4r11.ins.doc/bl1ins_estart.htm

5.6.13 IBM Spectrum LSF

This section provides instructions to install, update, and enable Spectrum LSF (formerly Platform LSF) on the compute nodes, and to configure some features. The Spectrum LSF installation and the user applications (run by using jobs) need to be available on a parallel file system in order for all nodes to concurrently access the same code and data correctly. For the scenario described in this section, a Spectrum Scale file system is available at `/gpfs/gpfs_fs0`.

The process consists of installing and updating Spectrum LSF in one node (that is, only once), and then enabling it on all nodes. The install/update is only required in one node because it is available in the parallel file system accessible by all nodes. The enablement, however, is required on all nodes because it performs configuration steps and enables startup services.

Therefore, the process to install/update Spectrum LSF requires an already provisioned node (for example, a compute or login node) with access to the parallel file system that is available to the other nodes. It is not possible to install and enable Spectrum LSF in a single provisioning stage for one of the nodes, which is used to install Spectrum LSF to the parallel file system. It is possible afterward if it is already installed and available in the parallel file system for access by other nodes, which only need to enable Spectrum LSF and be added to the Spectrum LSF cluster. The option with single provisioning stage is not covered in this section, which still requires enabling Spectrum LSF and adding the nodes to the cluster manually.

To install Spectrum LSF, complete the following steps:

- Install Spectrum LSF
- Update Spectrum LSF
- Enable Spectrum LSF
- Add more nodes
- Configure extra HPC and IBM PE support features
- Configure GPU support features

Install Spectrum LSF

The following steps are performed on the management node, targeting an already provisioned compute node (for example, p8r1n1, defined as `lsf_master`):

1. Create the directories for the installation and distribution directories of Spectrum LSF to be mounted from a Spectrum Scale file system (for example, `/gpfs/gpfs_fs0`):

```
# gpfs_dir='/gpfs/gpfs_fs0/lsf'
# gpfs_top="$gpfs_dir/top"
# gpfs_distrib="$gpfs_dir/distrib"

# lsf_top='/usr/share/lsf'
# lsf_distrib='/usr/share/lsf_distrib'

# lsf_master='p8r1n1'

# xdsh $lsf_master "mkdir -p $lsf_top $gpfs_top && echo '$gpfs_top $lsf_top
none defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_top"
p8r1n1: mount: /gpfs/gpfs_fs0/lsf/top bound on /usr/share/lsf.

# xdsh $lsf_master "mkdir -p $lsf_distrib $gpfs_distrib && echo '$gpfs_distrib
$lsf_distrib none defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_distrib"
p8r1n1: mount: /gpfs/gpfs_fs0/lsf/distrib bound on /usr/share/lsf_distrib.
```

2. Copy the installation tarballs and the entitlement file to the distribution directory:

```
# cd /path/to/lsf-install-files

# ls -1
lsf9.1.3_lnx310-lib217-ppc64le.tar.Z
lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z
lsf.entitlement
```

```
# scp \
    lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z \
    lsf9.1.3_lnx310-lib217-ppc64le.tar.Z \
    lsf.entitlement \
    $lsf_master:$lsf_distrib
lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z
100%  116MB  58.2MB/s   00:02
lsf9.1.3_lnx310-lib217-ppc64le.tar.Z
100%  228MB  76.1MB/s   00:03
lsf.entitlement
100%  167     0.2KB/s   00:00
```

Verify the files are present in the correct directory:

```
# ssh $lsf_master "cd $lsf_distrib; pwd; ls -1"
/usr/share/lsf_distrib
lsf9.1.3_lnx310-lib217-ppc64le.tar.Z
lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z
lsf.entitlement
```

3. Create the administrator user for Spectrum LSF:

```
# lsf_username='lsfadmin'
# lsf_password='<password>'

# xdsh $lsf_master "useradd -m -s /bin/bash $lsf_username && echo
"$lsf_username:$lsf_password" | chpasswd; su -l $lsf_username -c whoami"
p8r1n1: lsfadmin

# xdsh $lsf_master "su -l $lsf_username -c 'cat ~/.profile >> ~/.bash_profile'"
```

4. Create the configuration file for the installation (`install.config`):

   It must be placed in the same directory as the `install_lsf` and `lsf_startup` scripts.

   For the scenario adopted in this chapter:

   – Top directory: `/usr/share/lsf`
   – Distribution directory: `/usr/share/lsf_distrib`
   – Entitlement file: `lsf.entitlement` (in the distribution directory)
   – Administrator username: `lsfadmin`
   – Cluster name: `lsf-cluster`
   – Master/server nodes: `p8r1n1` (that is, `$lsf_master`)
   – Non-master/server nodes: *none at this time*

```
# lsf_cluster='lsf-cluster'
# lsf_entitlement="$lsf_distrib/lsf.entitlement"

# cat <<EOF >/install/postscripts/install.config
LSF_TOP="$lsf_top"
LSF_TARDIR="$lsf_distrib"
LSF_ENTITLEMENT_FILE="$lsf_entitlement"
LSF_ADMINS="$lsf_username"
LSF_CLUSTER_NAME="$lsf_cluster"
LSF_MASTER_LIST="$lsf_master"
LSF_ADD_SERVERS=""
EOF
```

Verify the file contents are correct:

```
# cat /install/postscripts/install.config
LSF_TOP="/usr/share/lsf"
LSF_TARDIR="/usr/share/lsf_distrib"
LSF_ENTITLEMENT_FILE="/usr/share/lsf_distrib/lsf.entitlement"
LSF_ADMINS="lsfadmin"
LSF_CLUSTER_NAME="lsf-cluster"
LSF_MASTER_LIST="p8r1n1"
LSF_ADD_SERVERS=""
```

5. Include the `ed` package in the package list:

```
# lsdef -t osimage rh72-hpc-diskful -i pkglist
Object name: rh72-hpc-diskful
    pkglist=/install/custom/install/rh/rh72-hpc.pkglist

# list=/install/custom/install/rh/rh72-hpc.pkglist

# cat <<EOF >>$list

# Spectrum LSF
ed
EOF
```

Verify the package list is correct:

```
# cat $list
# RHEL Server 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist#

# CUDA Toolkit 7.5 for RHEL 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist#

# Infiniband with Mellanox OFED for RHEL 7.2 (original pkglist)
createrepo
#INCLUDE:/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist#

# PE RTE IVP
ksh

# Spectrum LSF
ed
```

6. Install the packages from the `pkglist` attribute with the `ospkgs` script for the **updatenode** command:

```
# updatenode $lsf_master --scripts ospkgs
p8r1n1: xcatdsklspost: downloaded postscripts successfully
p8r1n1: <...> Running postscript: ospkgs
<...>
p8r1n1: Postscript: ospkgs exited with code 0
p8r1n1: Running of postscripts has completed.
```

Verify the **ed** command is available:

```
# xdsh $lsf_master 'ed --version | head -n1'
p8r1n1: GNU Ed 1.9
```

7. Install Spectrum LSF on the node with the `install_lsf` script for the **updatenode** command.

   Verify that the `exit code` of the script is `zero` (success).

```
# updatenode $lsf_master --scripts install_lsf
p8r1n1: xcatdsklspost: downloaded postscripts successfully
p8r1n1: <...> Running postscript: install_lsf
<...>
p8r1n1: INFO: Installation script DONE.
p8r1n1: INFO: Updating LSF Cluster Configuration Files lsf.conf and lsb.hosts
p8r1n1: Postscript: install_lsf exited with code 0
p8r1n1: Running of postscripts has completed.
```

   Note: If the **ed** command is not available, the following error can happen:

```
# updatenode $lsf_master --scripts install_lsf
<...>
p8r1n1: ERROR: Fail to install LSF. Check Install.log and Install.err in
/usr/share/lsf_distrib/lsf9.1.3_lsfinstall.
<...>

# xdsh $lsf_master "cat /usr/share/lsf_distrib/*/Install.err"
<...>
p8r1n1:     Cannot find UNIX command " ed".
<...>
```

Update Spectrum LSF

Several updates can be available for Spectrum LSF in the form of fixes (or patches).

For more information and details about applying fixes to Spectrum LSF, see the following Knowledge Center page:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_unix_install/lsf_patch_version_manage.dita

To download and apply fixes for Spectrum LSF, complete the following steps.

1. Download the fixes:
   a. Go to the IBM Support page:

      http://ibm.com/support

   b. In **Product Finder**, type **LSF**.
   c. In the results list, select **LSF 9.1.3**.
   d. Under **Downloads**, select **Downloads (fixes & PTFs)**.
   e. The page opens a window inside it (title: **Refine my fix list**; subtitle: **LSF 9.1.3**)
   f. In **Version fix level**, select **9.1.3**.
   g. In **Filter by operating system**, select **Linux Power PC 64 Little Endian**
   h. Click **Continue**.

      The list of fixes is presented, and more details are available in **Show fix details**. For example, these details were shown at the time of writing):

      - interim fix: lsf-9.1.3-build347817

        Abstract: P101215. Fix to enhance Nvidia GPU integration with LSF 9.1.3 in Linux x64 environment. LSF makes use of the cgroup device subsystem to enforce the

GPU in this patch. LSF can disable auto boost for the job with exclusive thread/process multi-GPU requirements. LSF can power off the GPU if it is not in use.

- interim fix: lsf-9.1.3-build368515

  Abstract: P101357. This fix updates the lsb_readjobinfo API to set a job

- fix pack: lsf-9.1.3.3-spk-2015-Jun-build346694

  Abstract: LSF Version 9.1.3 Fix Pack 3. This Fix Pack includes all fixed issues and solutions included in previous LSF Version 9.1.3 Fix Packs and addresses new issues and solutions between 1 February 2015 and 8 June 2015. For detailed descriptions of the issues and solutions in this Fix Pack, refer to the LSF 9.1.3 Fix Pack 3 Fixed Bugs List (lsf9.1.3.3_fixed_bugs.pdf).

- interim fix: lsf-9.1.3-build362511

  Abstract: P101353. Fix to ensure launch job run successfully over 32 nodes.

i. In the results list, select the desired fixes, and click **Continue**. Usually, all the fixes are suggested for the general case.

j. Proceed with the sign-in process.

k. Select a download option (for example, **HTTPS**).

l. Copy the download link of each file (for HTTPS; for other methods, follow the instructions that are provided in the website), or download the files and transfer to the desired node later.

2. Apply the fixes.

   Perform the following commands in one of the nodes with access to the parallel file system with the Spectrum LSF installation (for example, open an SSH connection to `p8r1n1`):

   a. Go to the patch installation directory:

   ```
   # patch_dir=/usr/share/lsf/9.1/install/patches/
   # mkdir -p $patch_dir
   # cd $patch_dir
   ```

   b. Download (or copy) one or more fixes.

   You can use the download links copied in an earlier step.

   ```
   # curl -sOL https://<...>/lsf9.1.3_linux3.10-glibc2.17-ppc64le-347817.tar.Z
   # curl -sOL https://<...>/lsf9.1.3_linux3.10-glibc2.17-ppc64le-368515.tar.Z
   # curl -sOL https://<...>/lsf9.1.3_lnx310-lib217-ppc64le-346694.tar.Z
   # curl -sOL https://<...>/lsf9.1.3_linux3.10-glibc2.17-ppc64le-362511.tar.Z

   # ls -1
   lsf9.1.3_linux3.10-glibc2.17-ppc64le-347817.tar.Z
   lsf9.1.3_linux3.10-glibc2.17-ppc64le-362511.tar.Z
   lsf9.1.3_linux3.10-glibc2.17-ppc64le-368515.tar.Z
   lsf9.1.3_lnx310-lib217-ppc64le-346694.tar.Z
   ```

   c. Run the **`patchinstall`** command on the fixes:

   ```
   # ../patchinstall *-362511.* *-346694.* *-368515.* *-347817.*
   <...>
   Installing package
   "/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-362
   511.tar.Z"...
   <...>
   Are you sure you want to update your cluster with this patch? (y/n) [y]
   ```

```
<...>
Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-3625
11.tar.Z.
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_lnx310-lib217-ppc64le-346694.ta
r.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_lnx310-lib217-ppc64le-346694.tar
.Z.
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-368
515.tar.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-3685
15.tar.Z.
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-347
817.tar.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
   Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-3478
17.tar.Z.

This patch has updated binaries or library files that affect running
daemons.
To make the changes take effect, you must restart your cluster.

Exiting...
```

d. Verify the installed fixes with the **pversions** command:

```
# /usr/share/lsf/9.1/install/pversions

IBM Platform LSF 9.1.3
----------------------
  binary type: linux3.10-glibc2.17-ppc64le, Apr 01 2015, Build 335772
  installed: Nov 30 2015
  patched:  Fix P101353, build 362511, installed Dec 01 2015
            Fix , build 346694, installed Dec 01 2015
            Fix P101357, build 368515, installed Dec 01 2015
            Fix P101215, build 347817, installed Dec 01 2015
```

e. Restart several services in the master node:

```
# lsadmin limrestart all
<...>
```

```
Do you really want to restart LIMs on all hosts? [y/n] y
<...>

# lsadmin resrestart all
Do you really want to restart RES on all hosts? [y/n] y
<...>

# badmin hrestart all
<...>
Restart slave batch daemon on all the hosts? [y/n] y
<...>

# badmin mbdrestart
<...>
Do you want to restart MBD? [y/n] y
<...>
```

Enable Spectrum LSF

The following steps are performed on the management node and target all compute nodes.

1. Modify the `lsf_startup` script to correctly find the Spectrum LSF version based on its installation path.

   This is done only once, in the management node.

   This issue is expected to be resolved in a future xCAT version (xCAT issue #495).

   a. Create a copy of the original script

   ```
   # cp -a /install/postscripts/lsf_startup
   /install/postscripts/lsf_startup.bkp
   ```

   b. Modify the script with the following expression to the **sed** command. The expression is a single, long line (without line breaks).

   ```
   # sed \
   '/^LSF_VERSION=/ a LSF_VERSION="$(find /$LSF_TOP -path "*/install/hostsetup"
   | grep -o "/[^/]\\+/install/hostsetup" | cut -d/ -f2)"' \
   -i /install/postscripts/lsf_startup
   ```

   c. Verify the differences between the original and the modified scripts:

   ```
   # diff /install/postscripts/lsf_startup.bkp /install/postscripts/lsf_startup
   34a35
   > LSF_VERSION="$(find /$LSF_TOP -path "*/install/hostsetup" | grep -o
   "/[^/]\+/install/hostsetup" | cut -d/ -f2)"
   ```

2. Run the `lsf_startup` script in the nodes with the **updatenode** command (Example 5-24).

   You can use a node group (for example, `s822lc`) instead of the `$lsf_master` variable after the nodes in the group are online.

   Verify that the `exit code` of the script is `zero` (success).

   Example 5-24 Running the lsf_startup script with the updatenode command

   ```
   # updatenode $lsf_master --scripts lsf_startup
   p8r1n1: xcatdsklspost: downloaded postscripts successfully
   p8r1n1: <...> Running postscript: lsf_startup
   p8r1n1: INFO: Run hostsetup on each node.
   p8r1n1: Logging installation sequence in /usr/share/lsf/log/Install.log
   p8r1n1:
   p8r1n1: ------------------------------------------------------------
   ```

```
p8r1n1:     L S F    H O S T S E T U P    U T I L I T Y
p8r1n1: ------------------------------------------------------------
p8r1n1: This script sets up local host (LSF server, client or slave)
environment.
p8r1n1: Setting up LSF server host "p8r1n1" ...
p8r1n1: Checking LSF installation for host "p8r1n1.xcat-cluster" ... Done
p8r1n1: Installing LSF RC scripts on host "p8r1n1.xcat-cluster" ... Done
p8r1n1: LSF service ports are defined in /usr/share/lsf/conf/lsf.conf.
p8r1n1: Checking LSF service ports definition on host "p8r1n1.xcat-cluster" ...
Done
p8r1n1: You are installing IBM Platform LSF -  Standard Edition.
p8r1n1:
p8r1n1: ... Setting up LSF server host "p8r1n1" is done
p8r1n1: ... LSF host setup is done.
p8r1n1: INFO: Set LSF environment for root and LSF_ADMINS
p8r1n1: INFO: Start LSF Cluster.
p8r1n1: Starting up LIM on <p8r1n1> ...... done
p8r1n1: Starting up RES on <p8r1n1> ...... done
p8r1n1: Starting up slave batch daemon on <p8r1n1> ...... done
p8r1n1: Postscript: lsf_startup exited with code 0
p8r1n1: Running of postscripts has completed.
```

3. Verify that the Spectrum LSF commands are available for the LSF administrator user, and that the cluster is listed with the **lsclusters** command:

```
# xdsh $lsf_master "su -l $lsf_username -c lsclusters"
p8r1n1: CLUSTER_NAME   STATUS   MASTER_HOST    ADMIN    HOSTS  SERVERS
p8r1n1: lsf-cluster    ok       p8r1n1         lsfadmin     1        1
```

Add more nodes

To add more nodes to the cluster, complete the following steps.

The following steps are performed on the management node, targeting an already provisioned compute node (for example, p8r3n2, defined as `lsf_node`).

For more information and details, see the following Knowledge Center page:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/host_add_lsf.dita

1. Create the directories for the installation and distribution directories of Spectrum LSF to be mounted from a Spectrum Scale file system (for example, `/gpfs/gpfs_fs0`):

```
# gpfs_dir='/gpfs/gpfs_fs0/lsf' # then /top and /distrib
# gpfs_top="$gpfs_dir/top"
# gpfs_distrib="$gpfs_dir/distrib"

# lsf_top='/usr/share/lsf'
# lsf_distrib='/usr/share/lsf_distrib'

# lsf_node='p8r3n2'

# xdsh $lsf_node "mkdir -p $lsf_top $gpfs_top && echo '$gpfs_top $lsf_top none
defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_top"
p8r3n2: mount: /gpfs/gpfs_fs0/lsf/top bound on /usr/share/lsf.
```

```
# xdsh $lsf_node "mkdir -p $lsf_distrib $gpfs_distrib && echo '$gpfs_distrib
$lsf_distrib none defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_distrib"
p8r3n2: mount: /gpfs/gpfs_fs0/lsf/distrib bound on /usr/share/lsf_distrib.
```

2. Add the node to the `lsf.cluster.<cluster-name>` file.

   You can edit the file from any node that can access the parallel file system with the Spectrum LSF installation directory.

   You can add a line for the new node (`p8r3n2`) based on an existing node (`p8r2n2`), for example.

```
# vi /usr/share/lsf/conf/lsf.cluster.lsf-cluster
<...>
Begin   Host
HOSTNAME  model    type        server r1m  mem  swp  RESOURCES    #Keywords
<...>
p8r1n1   !   !   1   3.5   ()   ()   (mg)
p8r3n2   !   !   1   3.5   ()   ()   (mg)
End     Host
```

3. Restart the services on a master node (for example, `p8r1n1`):

```
<management-node> # lsf_master=p8r1n1
<management-node> # ssh $lsf_master

<master-node> # su lsfadmin

<master-node> $ lsadmin reconfig

Checking configuration files ...
No errors found.

Restart only the master candidate hosts? [y/n] y
Restart LIM on <p8r1n1> ...... done

<master-node> $ badmin mbdrestart

Checking configuration files ...

There are warning errors.

Do you want to see detailed messages? [y/n] y
Checking configuration files ...
<...>
-------------------------------------------------------
No fatal errors found.
<...>
Do you want to restart MBD? [y/n] y
MBD restart initiated
```

   Verify the new node is listed by the **lshosts** command (with no hardware details yet):

```
<master-node> $ lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r3n2      UNKNOWN UNKNOWN_   1.0     -      -      -    Yes (mg)
```

4. Create the administrator user for Spectrum LSF:

```
# lsf_username='lsfadmin'
# lsf_password='<password>'

# xdsh $lsf_node "useradd -m -s /bin/bash $lsf_username && echo
"$lsf_username:$lsf_password" | chpasswd; su -l $lsf_username -c whoami"
p8r3n2: lsfadmin

# xdsh $lsf_node "su -l $lsf_username -c 'cat ~/.profile >> ~/.bash_profile'"
```

5. Run the `lsf_startup` script in the nodes with the **updatenode** command.

```
# updatenode $lsf_node --scripts lsf_startup
```

6. Start the services on the node.

   You can reboot the node, instead.

```
# xdsh $lsf_node "su -l $lsf_username -c 'lsadmin limstartup' "
p8r3n2: Starting up LIM on <p8r3n2> ...... done


# xdsh $lsf_node "su -l $lsf_username -c 'lsadmin resstartup' "
p8r3n2: Starting up RES on <p8r3n2> ...... done


# xdsh $lsf_node "su -l $lsf_username -c 'badmin hstartup' "
p8r3n2: Starting up slave batch daemon on <p8r3n2> ...... done
```

7. Verify that the new node is listed by issuing the **lshosts** command (with hardware details now):

```
# lsf_master=p8r1n1

# xdsh $lsf_master "su -l $lsf_username -c lshosts"
p8r1n1: HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1: p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r1n1: p8r3n2      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
```

Configure extra HPC and IBM PE support features

When installing Spectrum LSF you can set the `CONFIGURATION_TEMPLATE` property on `install.config` file to use one of the following configuration templates:

- `DEFAULT`: Use for mixed type of work loads. It provides good performance overall but it is not tuned for any specific type of cluster
- `PARALLEL`: Adds extra support to large parallel jobs, including specific configurations to IBM Parallel Environment (PE)
- `HIGH_THROUGHPUT`: Tunes for high throughput (high rate of mainly short jobs)

As an alternative, you can use `DEFAULT` template but afterwards configure manually according to your needs. The following configuration tasks, for instance, enables the cluster just like it was installed with a `PARALLEL` template. As the Spectrum LSF administrator user, do:

1. Edit the `lsf.shared` configuration file to add following resources in the `Resource` section:

```
ibmmpi     Boolean ()       ()          (IBM POE MPI)
adapter_windows Numeric    30    N    (free adapter windows on css0 on IBM SP)
nrt_windows     Numeric    30    N    (The number of free nrt windows on IBM
systems)
poe             Numeric    30    N    (poe availability)
css0            Numeric    30    N    (free adapter windows on css0 on IBM SP)
```

```
csss            Numeric    30    N    (free adapter windows on csss on IBM SP)
dedicated_tasks Numeric    ()    Y    (running dedicated tasks)
ip_tasks        Numeric    ()    Y    (running IP tasks)
us_tasks        Numeric    ()    Y    (running US tasks)
```

2. Map out new resources on the `lsf.cluster.<cluster_name>` configuration file, in the `ResourceMap` section:

```
poe                 [default]
adapter_windows     [default]
nrt_windows         [default]
dedicated_tasks     (0@[default])
ip_tasks            (0@[default])
us_tasks            (0@[default])
```

3. Reconfigure and restart LIM daemons:

```
$ lsadmin reconfig
```

4. Configure reservation usage of adapter and nrt windows in the `lsb.resources` configuration file. In the `ReservationUsage` section:

```
Begin ReservationUsage
RESOURCE           METHOD
adapter_windows    PER_TASK
nrt_windows        PER_TASK
End ReservationUsage
```

5. Optionally, create a new queue for PE jobs in the `lsb.queues` configuration file. The following sample includes the `hpc_ibm` and `hpc_ibm_tv` queues:

```
Begin Queue
QUEUE_NAME   = hpc_ibm
PRIORITY     = 30
NICE         = 20
#RUN_WINDOW   = 5:19:00-1:8:30 20:00-8:30
#r1m         = 0.7/2.0    # loadSched/loadStop
#r15m         = 1.0/2.5
#pg           = 4.0/8
#ut           = 0.2
#io           = 50/240
#CPULIMIT     = 180/hostA  # 3 hours of host hostA
#FILELIMIT    = 20000
#DATALIMIT    = 20000      # jobs data segment limit
#CORELIMIT    = 20000
#TASKLIMIT    = 5          # job processor limitEnd of change
#USERS        = all        # users who can submit jobs to this queue
#HOSTS        = all        # hosts on which jobs in this queue can run
#PRE_EXEC     = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
#POST_EXEC    = /usr/local/lsf/misc/testq_post |grep -v Hey
RES_REQ = select[ poe > 0 ]
EXCLUSIVE = Y
REQUEUE_EXIT_VALUES = 133 134 135
DESCRIPTION  = IBM Platform LSF 9.1 for IBM. This queue is to run POE jobs
ONLY.
End Queue

Begin Queue
QUEUE_NAME   = hpc_ibm_tv
PRIORITY     = 30
```

```
NICE          = 20
#RUN_WINDOW    = 5:19:00-1:8:30 20:00-8:30
#r1m          = 0.7/2.0    # loadSched/loadStop
#r15m          = 1.0/2.5
#pg            = 4.0/8
#ut            = 0.2
#io            = 50/240
#CPULIMIT      = 180/hostA  # 3 hours of host hostA
#FILELIMIT     = 20000
#DATALIMIT     = 20000      # jobs data segment limit
#CORELIMIT     = 20000
#TASKLIMIT     = 5          # job processor limitEnd of change
#USERS         = all        # users who can submit jobs to this queue
#HOSTS         = all        # hosts on which jobs in this queue can run
#PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
#POST_EXEC     = /usr/local/lsf/misc/testq_post |grep -v Hey
RES_REQ = select[ poe > 0 ]
REQUEUE_EXIT_VALUES = 133 134 135
TERMINATE_WHEN = LOAD PREEMPT WINDOW
RERUNNABLE = NO
INTERACTIVE = NO
DESCRIPTION  = IBM Platform LSF 9.1 for IBM debug queue. This queue is to run
POE jobs ONLY.
End Queue
```

6. Reconfigure and restart batch daemons:

```
$ badmin reconfig
```

You can see further details about the above configuration in the Spectrum LSF manuals at following website:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_unix_install/lsfinstall_about.dita

Configure GPU support features

Spectrum LSF can be configured to manage graphics orocessing unit (GPU) resources so that they can be used on areas like monitoring, requirement expressions, and usage reservation on job submission.

The following resources are exposed as external load indices by the **elim.gpu** External Load Information Manager (ELIM) program:

- `ngpus`: Total number of GPUs
- `ngpus_shared`: Number of GPUs in share mode
- `ngpus_excl_t`: Number of GPUs in exclusive thread mode
- `ngpus_excl_p`: Number of GPUs in exclusive process mode

Before you proceed with configuration to export those resources, verify the **elim.gpu** program is deployed on the directory pointed to by the `LSF_SERVERDIR` environment variable. This is automatically started by the LIM daemon, and can be checked with following command:

```
$ ps -aux | grep elim.gpu
```

Configure Spectrum LSF with the following steps:

1. Edit the `lsf.shared` configuration file to add following resources in the `Resource` section:

```
ngpus      Numeric 60   N          (Number of GPUs)
ngpus_shared  Numeric 60   N          (Number of GPUs in Shared Mode)
```

```
ngpus_excl_t  Numeric 60   N          (Number of GPUs in Exclusive Thread Mode)
ngpus_excl_p  Numeric 60   N          (Number of GPUs in Exclusive Process
Mode)
```

2. Map out new resources in the `lsf.cluster.<cluster_name>` configuration file, in the `ResourceMap` section:

```
ngpus             ([default])
ngpus_shared      ([default])
ngpus_excl_t      ([default])
ngpus_excl_p      ([default])
```

3. Enable reservation usage of those resources in the `lsb.resources` configuration file and in the `ReservationUsage` section:

```
ngpus_shared     PER_HOST      N
ngpus_excl_t     PER_HOST      N
ngpus_excl_p     PER_HOST      N
```

4. Reconfigure and restart LIM and the batch daemons:

```
$ lsadmin reconfig
$ badmin reconfig
```

5. Check if LIM now collects the information about GPUs:

```
$ lshosts -l
```

Other GPU-specific resources exposed by the **elim.gpu.ext** and **elim.gpu.topology** elim programs are not configured in this section. See the following website to read how to configure those too:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/define_gpu_resources.dita

5.6.14 Node provisioning

To start provisioning a compute node (or node group), and install the operating system and software stack according to the `osimage` and node group objects, complete the following steps:

1. Define the `osimage` attribute of a node (or node group) with the **nodeset** command:

```
# nodeset p8r1n1 osimage=rh72-hpc-diskful
p8r1n1: install rhels7.2-ppc64le-compute
```

2. You can verify the changes to the nodes attributes with the **lsdef** command:

```
# lsdef p8r1n1
Object name: p8r1n1
    <...>
    initrd=xcat/osimage/rh72-hpc-diskful/initrd.img
    <...>
    kcmdline=quiet inst.repo=http://!myipfn!:80/install/rhels7.2/ppc64le
inst.ks=http://!myipfn!:80/install/autoinst/p8r1n1 BOOTIF=98:be:94:59:f0:f2
ip=dhcp
    kernel=xcat/osimage/rh72-hpc-diskful/vmlinuz
    <...>
    os=rhels7.2
    <...>
    profile=compute
```

```
    provmethod=rh72-hpc-diskful
    <...>
```

3. Set the boot method to network with the **rsetboot** command (optional):

```
# rsetboot p8r1n1 net
```

> **Note:** This is performed automatically by the **nodeset** command, and is not actually required if the bootloader configuration for automatic boot is correct. For more information, see 5.2.3, “Boot order configuration” on page 57.

4. Reboot the node (or node group) with the **rpower** command (optional):

```
# rpower p8r1n1 reset
```

> **Note:** This is performed automatically (within some time) if the Genesis image for node discovery is still running in the compute node (waiting for instructions from the Management Node).

5. You can watch the node’s console with the **rcons** command:

```
# rcons p8r1n1
```

6. You can monitor the node provisioning progress in the node object and the `/var/log/messages` log file.

   After the OS installation finishes and the node reboots, it performs additional package installations and postscripts execution for some time.

```
# lsdef p8r1n1 -i status
Object name: p8r1n1
    status=booting

# lsdef p8r1n1 -i status
Object name: p8r1n1
    status=booted

# tail -f /var/log/messages
<...>
<...> p8r1n1 xcat: ready
<...> p8r1n1 xcat: done
<...> p8r1n1 xcat: /xcatpost/mypostscript.post return
<...>
```

5.6.15 Post-installation verification

To verify that the software stack is correctly provisioned, complete the following steps:

Verify the CUDA Toolkit

Complete these steps to verify that the CUDA Toolkit is installed:

1. Verify all GPUs are listed with the **nvidia-smi** command:

```
# xdsh p8r1n1 'nvidia-smi --list-gpus'
p8r1n1: GPU 0: Tesla K80 (UUID: GPU-7f8d1ae1-14ed-147a-7596-c93305614055)
p8r1n1: GPU 1: Tesla K80 (UUID: GPU-f2d61b72-f838-c880-c5d5-53e73c8bb21c)
p8r1n1: GPU 2: Tesla K80 (UUID: GPU-592d8b70-88b0-4157-0960-20d806ccdd0e)
p8r1n1: GPU 3: Tesla K80 (UUID: GPU-416ff17c-745e-bddf-2888-3963de3511bc)
```

2. Verify that the persistence mode is enabled, and power limit value is correct (according to the script in 5.6.3, “CUDA Toolkit” on page 113) with the **nvidia-smi** command:

```
# xdsh p8r1n1 'nvidia-smi --query-gpu=persistence_mode,power.limit
--format=csv'
p8r1n1: persistence_mode, power.limit [W]
p8r1n1: Enabled, 175.00 W
p8r1n1: Enabled, 175.00 W
p8r1n1: Enabled, 175.00 W
p8r1n1: Enabled, 175.00 W
```

Verify the Mellanox OFED

To verify the installation of Mellanox OFED, complete these steps:

1. Verify that the `openibd` service is correctly loaded and active with the **systemctl** command:

```
# xdsh p8r1n1 'systemctl status openibd'
p8r1n1: * openibd.service - openibd - configure Mellanox devices
p8r1n1:    Loaded: loaded (/usr/lib/systemd/system/openibd.service; enabled;
vendor preset: disabled)
p8r1n1:    Active: active (exited) since <...>
<...>
p8r1n1: <...> systemd[1]: Starting openibd - configure Mellanox devices...
p8r1n1: <...> openibd[3991]: Unloading HCA driver:[  OK  ]
p8r1n1: <...> openibd[3991]: Loading HCA driver and Access Layer:[  OK  ]
p8r1n1: <...> systemd[1]: Started openibd - configure Mellanox devices.
```

Note: The `openibd` service might fail on the first boot due to unloading problems with the non-Mellanox OFED kernel modules, for example:

```
# xdsh p8r1n1 'systemctl status openibd'
<...>
p8r1n1:    Active: inactive (dead)
<...>
p8r1n1: <...> openibd[131953]: rmmod: ERROR: Module rdma_cm is in use by:
xprtrdma
<...>
```

To resolve this issue, you can use either one of these methods:

- Reboot the node.
- Try to unload the problematic modules and restart the `openibd` service, for example:

```
# xdsh p8r1n1 'modprobe -r ib_isert xprtrdma ib_srpt; systemctl restart
openibd'
```

2. Verify the information and status of the InfiniBand adapter/ports with the **ibstat** command:

```
# xdsh p8r1n1 ibstat
p8r1n1: CA 'mlx5_0'
p8r1n1:         CA type: MT4115
p8r1n1:         Number of ports: 1
p8r1n1:         Firmware version: 12.100.6440
p8r1n1:         Hardware version: 0
p8r1n1:         Node GUID: 0xe41d2d0300ff4910
p8r1n1:         System image GUID: 0xe41d2d0300ff4910
p8r1n1:         Port 1:
p8r1n1:                 State: Initializing
```

```
p8r1n1:                 Physical state: LinkUp
p8r1n1:                 Rate: 40
p8r1n1:                 Base lid: 65535
p8r1n1:                 LMC: 0
p8r1n1:                 SM lid: 0
p8r1n1:                 Capability mask: 0x2651e848
p8r1n1:                 Port GUID: 0xe41d2d0300ff4910
p8r1n1:                 Link layer: InfiniBand
p8r1n1: CA 'mlx5_1'
p8r1n1:         CA type: MT4115
p8r1n1:         Number of ports: 1
p8r1n1:         Firmware version: 12.100.6440
p8r1n1:         Hardware version: 0
p8r1n1:         Node GUID: 0xe41d2d0300ff4911
p8r1n1:         System image GUID: 0xe41d2d0300ff4910
p8r1n1:         Port 1:
p8r1n1:                 State: Initializing
p8r1n1:                 Physical state: LinkUp
p8r1n1:                 Rate: 40
p8r1n1:                 Base lid: 65535
p8r1n1:                 LMC: 0
p8r1n1:                 SM lid: 0
p8r1n1:                 Capability mask: 0x2651e848
p8r1n1:                 Port GUID: 0xe41d2d0300ff4911
p8r1n1:                 Link layer: InfiniBand
```

Verify the XL C/C++ Compiler

Verify the installed version with the **xlc** command:

```
# xdsh p8r1n1 'xlc -qversion'
p8r1n1: IBM XL C/C++ for Linux, V13.1.2 (5725-C73, 5765-J08)
p8r1n1: Version: 13.01.0002.0000
```

Verify the XL Fortran Compiler

Verify the installed version with the **xlf** command:

```
# xdsh p8r1n1 'xlf -qversion'
p8r1n1: IBM XL Fortran for Linux, V15.1.2 (5725-C75, 5765-J10)
p8r1n1: Version: 15.01.0002.0000
```

Verify Advance Toolchain

To verify the installation of Advance Toolchain, complete these steps:

1. Verify the installed version with the Advance Toolchain **gcc** command:

   ```
   # xdsh p8r1n1 '/opt/at8.0/bin/gcc --version | head -n1'
   p8r1n1: gcc (GCC) 4.9.4 20150824 (Advance-Toolchain-at8.0) [ibm/gcc-4_9-branch,
   revision: 227153 merged from gcc-4_9-branch, revision 227151]
   ```

2. Verify the integration with the XL C/C++ Compiler with the **xlc_at** command:

   ```
   # xdsh p8r1n1 'xlc_at -qversion'
   p8r1n1: IBM XL C/C++ for Linux, V13.1.2 (5725-C73, 5765-J08)
   p8r1n1: Version: 13.01.0002.0000
   ```

Verify the GNU Compiler Collection

Verify the installed version of the Linux distribution with the **gcc** command:

```
# xdsh p8r1n1 'gcc --version | head -n1'
p8r1n1: gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-4)
```

Verify the PE RTE

Complete these steps to verify the installation of PE RTE:

1. Verify the installed version of the Korn shell with the **ksh** command:

   ```
   # xdsh p8r1n1 'ksh --version'
   p8r1n1:   version         sh (AT&T Research) 93u+ 2012-08-01
   ```

2. Create a user account with SSH-based authentication:

   ```
   # xdsh p8r1n1 'useradd pe-user && su pe-user -c "ssh-keygen -t rsa -f
   ~/.ssh/id_rsa -N \"\" && cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys"'
   <...>
   p8r1n1: Your identification has been saved in /home/pe-user/.ssh/id_rsa.
   p8r1n1: Your public key has been saved in /home/pe-user/.ssh/id_rsa.pub.
   <...>
   ```

3. Verify the PE RTE Installation Verification Program (IVP).

   ```
   # xdsh p8r1n1 'su pe-user -c "cd &&
   /opt/ibmhpc/pecurrent/ppe.samples/ivp/ivp.linux.script"'
   p8r1n1: Verifying the existence of the Binaries
   p8r1n1: Partition Manager daemon /etc/pmdv23 is executable
   p8r1n1: PE RTE files seem to be in order
   <...>
   p8r1n1: POE IVP: running as task 1 on node p8r1n1
   p8r1n1: POE IVP: running as task 0 on node p8r1n1
   p8r1n1: POE IVP: there are 2 tasks running
   <...>
   p8r1n1: Parallel program ivp.out return code was 0
   p8r1n1:
   p8r1n1:
   p8r1n1: If the test returns a return code of 0, POE IVP
   p8r1n1:  is successful.  To test message passing,
   <...>
   ```

Verify ESSL and PESSL

To verify the installation of ESSL and PESSL, complete these steps:

1. Verify the PESSL IVP (exercises ESSL as well) on an individual compute node, for the permutations of available compilers/programming languages and ESSL SMP libraries:

   ```
   # xdsh p8r1n1 --stream 'IVP="/opt/ibmmath/pessl/5.2/ivps/pesslivp64";
   NUM_NODES="2"; HOST_LIST="/tmp/host.list"; rm -f $HOST_LIST; for i in $(seq 1
   $NUM_NODES); do hostname --long >> $HOST_LIST; done; grep . $HOST_LIST; for
   language in fortran c C++ gcc g++; do for library in esslsmp esslsmpcuda; do
   echo; echo PESSL IVP $language $library; su pe-user -c "cd && $IVP $language
   $NUM_NODES $HOST_LIST $library"; done; done'
   p8r1n1: p8r1n1.xcat-cluster
   p8r1n1: p8r1n1.xcat-cluster

   p8r1n1: PESSL IVP fortran esslsmp
   ```

```
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP fortran esslsmpcuda
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP c esslsmp
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP c esslsmpcuda
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP C++ esslsmp
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP C++ esslsmpcuda
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP gcc esslsmp
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP gcc esslsmpcuda
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP g++ esslsmp
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.

p8r1n1: PESSL IVP g++ esslsmpcuda
p8r1n1:  PARALLEL ESSL installed successfully
p8r1n1: /opt/ibmmath/pessl/5.2/ivps/pesslivp64: Parallel ESSL installation
verification program was successful.
```

Verify Spectrum Scale

1. Verify the Spectrum Scale packages are installed with the **rpm** command:

```
# xdsh p8r1n1 'rpm -qa | grep ^gpfs'
p8r1n1: gpfs.gskit-8.0.50-47.ppc64le
p8r1n1: gpfs.gpl-4.1.1-3.noarch
p8r1n1: gpfs.ext-4.1.1-3.ppc64le
p8r1n1: gpfs.docs-4.1.1-3.noarch
```

```
p8r1n1: gpfs.msg.en_US-4.1.1-3.noarch
p8r1n1: gpfs.base-4.1.1-3.ppc64le
```

2. Verify the node does not yet belong to any cluster with the **mmlscluster** command:

```
# xdsh p8r1n1 'mmlscluster'
p8r1n1: mmlscluster: This node does not belong to a GPFS cluster.
p8r1n1: mmlscluster: Command failed. Examine previous error messages to
determine cause.
```

Check public and site network connectivity (optional)

Verify the connectivity to external and non-cluster nodes with the **ping** command:

```
# xdsh p8r1n1 'ping -c1 example.com'
p8r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p8r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=3.94 ms
<...>
```

5.7 xCAT Login Nodes

The deployment of an xCAT Login Node is procedurally the same as that of an xCAT Compute Node. The main difference is the particular software stack components used on each node type.

Accordingly, this section does not cover the deployment instructions explicitly for login nodes. Rather, it describes some examples of the differences in the software stack components between login and compute nodes.

The following differences are typically considered for the login nodes:

- Device drivers and related software (for example, CUDA Toolkit, and Mellanox OFED for Linux): Usually not required because the additional compute-related hardware is not present on login nodes.
- Compilers and runtime libraries (XL C/C++ and Fortran Compilers): The compilers use the login nodes to compile applications. The compiler-related runtime libraries are not usually required because the applications are not executed on login nodes, but rather on compute nodes.
- HPC software: The kits for PE RTE, PE DE, ESSL, and PESSL provide kit components specifically for login nodes, for example:

```
Object name: pperte-login-2.3.0.0-1547a-rhels-7.2-ppc64le
    description=PE RTE for login nodes

Object name: ppedev.login-2.2.0-0-rhels-7.2-ppc64le
    description=Parallel Environment Developer Edition for login nodes

Object name: essl-loginnode-5.4.0-0-rhels-7.2-ppc64le
    description=essl for login nodes

Object name: pessl-loginnode-5.2.0-0-rhels-7.2-ppc64le
    description=pessl for login nodes
```

- Parallel file system (Spectrum Scale): This system can allow the login node to access the data provided to or produced by the applications running in the compute nodes.
- Job scheduler (Spectrum LSF): Typically required to submit jobs from the login nodes to the compute nodes.

6

Application development and tuning

This chapter provides information about software and tools, which can be used for application development and tuning on the IBM Power System S822LC. In addition, a few development models are described.

The following topics are presented in this chapter:

- Compiler options
- Engineering and Scientific Subroutine Library
- Parallel ESSL
- Using POWER8 vectorization
- Development models
- GPU tuning
- Tools for development and tuning of applications

6.1 Compiler options

Compiler options are one of the main tools to debug and optimize performance of your code during development. Several compilers, including the IBM XL compilers and the GNU Compiler Collection (GCC), support the latest IBM POWER8 processor features.

6.1.1 XL compiler options

XL C/C++ for Linux, v13.1.2, and XL Fortran for Linux, v15.1.2 support POWER8 processors with new features and enhancements, including compiler options for POWER8 processors and built-in functions for POWER8 processors.

By default, these compilers generate code that runs on various Power Systems. Options can be added to exclude older processor chips that are not supported by the target application. Two major XL compiler options control this support:

- `-qarch`: Specifies the processor architecture for which code is generated.
- `-qtune`: Indicates the processor chip generation of most interest for performance. It tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run optimally on a specific hardware architecture.

The `-qarch=pwr8` suboption produces object code that contains instructions that will run on the POWER8 hardware platforms. With the `-qtune=pwr8` suboption, optimizations are tuned for the POWER8 hardware platforms. This configuration can enable better code generation because the compiler takes advantage of capabilities that were not available on those older systems.

For all production codes, it is imperative to enable a minimum level of compiler optimization by adding the `-O` option for the XL compilers. Without optimization, the focus of the compiler is on faster compilation and debug ability, and it generates code that performs poorly at run time.

For projects with increased focus on runtime performance, take advantage of the more advanced compiler optimization. For numerical or compute-intensive codes, the XL compiler options `-O3` or `-qhot -O3` enable loop transformations, which improve program performance by restructuring loops to make their execution more efficient by the target system. These options perform aggressive transformations that can sometimes cause minor differences in the precision of floating point computations. If the minor differences are a concern, the original program semantics can be fully recovered with the `-qstrict` option.

For more information about XL C/C++ support for POWER8 processor, see the following website:

http://www.ibm.com/support/knowledgecenter/?lang=en#!/SSXVZZ_13.1.2/com.ibm.compilers.linux.doc/welcome.html

For more information about XL Fortran, see the following website:

http://www.ibm.com/support/knowledgecenter/?lang=en#!/SSAT4T_15.1.2/com.ibm.compilers.linux.doc/welcome.html

Optimization parameters

The strength of the XL compilers is in their optimization and ability to improve code generation. Optimized code executes with greater speed, uses less machine resource, and increases your productivity.

For XL C/C++ v13.1.2, the available compiler options to maximize application development performance are described in Table 6-1.

Table 6-1 Optimization levels and options

Based optimization level	Additional options that are implied by level	Additional suggested options
-O0	-qsimd=auto	-qarch
-O2	-qmaxmem=8192 -qsimd=auto	-qarch -qtune
-O3	-qnostrict -qmaxmem=-1 -qhot=level=0 -qsimd=auto -qinline=auto	-qarch -qtune
-O4	All of -O3 plus: -qhot -qipa -qarch=auto -qtune=auto -qcache=auto	-qarch -qtune -qcache
-O5	All of -O4 plus: -qipa=level=2	-qarch -qtune -qcache

Several options are used to control the optimization and tuning process, so users can improve the performance of their application at run time.

When you compile programs with any of the following sets of options, the compiler automatically attempts to vectorize calls to system math functions. It does so by calling the equivalent vector functions in the Mathematical Acceleration Subsystem (MASS) libraries, with the exceptions of functions `vdnint`, `vdint`, `vcosisin`, `vscosisin`, `vqdrt`, `vsqdrt`, `vrqdrt`, `vsrqdrt`, `vpopcnt4`, and `vpopcnt8`:

- `-qhot -qignerrno -qnostrict`
- `-qhot -O3`
- `-O4`
- `-O5`

If the compiler cannot vectorize, it automatically tries to call the equivalent MASS scalar functions. For automatic vectorization or scalarization, the compiler uses versions of the MASS functions that are contained in the system library `libxlopt.a`.

In addition to any of the preceding sets of options, when the `-qipa` option is in effect, if the compiler cannot vectorize, it tries to inline the MASS scalar functions before it decides to call them.

Not all options benefit all applications. Trade-offs sometimes occur between an increase in compile time, a reduction in debugging capability, and the improvements that optimization can provide. In addition to the options that are listed in Table 6-2, consult the *Optimization and Programming Guide - XL C/C++ for Linux, V13.1, for big endian distributions,* SC27-4251-01, for details about the optimization and tuning process and writing optimization-friendly source code.

Table 6-2 Optimization and tuning options

Option name	Description
-qarch	Specifies the processor architecture for which the code (instructions) can be generated.
-qtune	Tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run best on a specific hardware architecture.
-qcache	Specifies the cache configuration for a specific execution machine.
-qhot	Performs high-order loop analysis and transformations during optimization.
-qipa	Enables or customizes a class of optimizations that is known as interprocedural analysis (IPA).
-qmaxmem	Limits the amount of memory that the compiler allocates while it performs specific, memory-intensive optimizations to the specified number of kilobytes.
-qignerrno	Allows the compiler to perform optimizations as though system calls will not modify `errno`.
-qpdf1, -qpdf2	Tunes optimizations through profile-directed feedback (PDF), where results from sample program execution are used to improve optimization near conditional branches and in frequently executed code sections.
-p, -pg, -qprofile	Prepares the object files that are produced by the compiler for profiling.
-qinline	Attempts to inline functions instead of generating calls to those functions, for improved performance.
-qstrict	Ensures that optimizations that are performed by default at the `-O3` and higher optimization levels, and, optionally at `-O2`, and do not alter the semantics of a program.
-qsimd	Controls whether the compiler can automatically take advantage of vector instructions for processors that support them.
-qsmp	Enables parallelization of program code.
-qunroll	Controls loop unrolling for improved performance.

For more information about the XL compiler options, see the following websites:

http://www.ibm.com/support/knowledgecenter/?lang=en#!/SSXVZZ_13.1.2/com.ibm.xlcpp1312.lelinux.doc/compiler_ref/fcat_optzn.html

http://www.ibm.com/support/knowledgecenter/?lang=en#!/SSAT4T_15.1.2/com.ibm.xlf1512.lelinux.doc/compiler_ref/fcat_optzn.html

6.1.2 GCC compiler options

For GCC, a minimum level of compiler optimization is `-O2`, and the suggested level of optimization is `-O3`. The GCC default is a strict mode, but the `-ffast-math` option disables strict mode. The `-O` fast option combines `-O3` with `-ffast-math` in a single option. Other important options include `-fpeel-loops`, `-funroll-loops`, `-ftree-vectorize`, `-fvect-cost-model`, and `-mcmodel=medium`.

Support for the POWER8 processor is now available on GCC-4.8.5 through the `-mcpu=power8` and `-mtune=power8` options.

Specifying the `-mveclibabi=mass` option and linking to the MASS libraries enable more loops for `-ftree-vectorize`. The MASS libraries support only static archives for linking. Therefore, they require explicit naming and library search order for each platform/mode:

- POWER8 32 bit: `-L<MASS-dir>/lib -lmassvp8 -lmass_simdp8 -lmass -lm`
- POWER8 64 bit: `-L<MASS-dir>/lib64 -lmassvp8_64 -lmass_simdp8_64 -lmass_64 -lm`

For more information about GCC support on POWER8, see the following GCC website:

https://gcc.gnu.org/gcc-4.8/

Optimization parameters

The commonly used optimization options are shown in Table 6-3.

Table 6-3 Optimization options for GCC

Option name	Description
-O, -O1	With the `-O` option, the compiler tries to reduce code size and execution time without performing any optimizations that significant compilation time.
-O2	The `-O2` option turns on all optional optimizations except for loop unrolling, function inlining, and register renaming. It also turns on the `-fforce-mem` option on all machines and frame pointer elimination on machines where frame pointer elimination does not interfere with debugging.
-O3	The `-O3` option turns on all optimizations that are specified by `-O2` and also turns on the `-finline-functions` and `-frename-registers` options.
-O0	Do not optimize.
-Os	Optimize for size. The `-Os` option enables all `-O2` optimizations that do not typically increase code size. It also performs further optimizations that are designed to reduce code size.
-ffast-math	Sets `-fno-math-errno`, `-funsafe-math-optimizations`, and `-fno-trapping-math`. This option causes the preprocessor macro `__FAST_MATH__` to be defined. This option must never be turned on by any `-O` option because it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules/specifications for math functions.

Option name	Description
-funroll-loops	Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. The `-funroll-loops` option implies both the `-fstrength-reduce` and `-frerun-cse-after-loop` option.
-fforce-mem	Force memory operands to be copied into registers before arithmetic is performed on them.
-fno-inline	Do not pay attention to the inline keyword. Normally, this option is used to keep the compiler from expanding any functions inline.
-fno-math-errno	Do not set `ERRNO` after math functions are called that are executed with a single instruction.
-finline-functions	Integrate all simple functions into their callers. The compiler heuristically decides the functions that are simple enough to be worth integrating in this way.

6.2 Engineering and Scientific Subroutine Library

IBM Engineering and Scientific Subroutine Library (ESSL) includes these runtime libraries:

- ESSL Serial Libraries and ESSL SMP Libraries
- ESSL SMP CUDA Library

The mathematical subroutines, in nine computational areas, are tuned for performance:

- Linear Algebra Subprograms
- Matrix Operations
- Linear Algebraic Equations
- Eigensystem Analysis
- Fourier Transforms, Convolutions and Correlations, and Related Computations v Sorting and Searching
- Interpolation
- Numerical Quadrature
- Random Number Generation

6.2.1 Compilation and run

The ESSL subroutines are callable from programs, which are written in Fortran, C, and C++. Table 6-4, Table 6-5 on page 161 and Table 6-6 on page 161 show how compilation on Linux depends on type of ESSL library (serial, SMP, SMP CUDA), environment (32-bit integer, 64-bit pointer or 64-bit integer, or 64-bit pointer) and compiler (XLF, XLC/C++, gcc/g++).

Table 6-4 Fortran compilation commands for ESSL

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	xlf_r -O -qnosave program.f -lessl
	64-bit integer, 64-bit pointer	xlf_r -O -qnosave program.f -lessl6464

Type of ESSL library	Environment	Compilation command
SMP	32-bit integer, 64-bit pointer	xlf_r -O -qnosave -qsmp program.f -lesslsmp
	64-bit integer, 64-bit pointer	xlf_r -O -qnosave -qsmp program.f -lesslsmp6464
SMP CUDA	32-bit integer, 64-bit pointer	xlf_r -O -qnosave -qsmp program.f -lesslsmpcuda -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64

Table 6-5 shows the compilation commands for XLC/C++.

Table 6-5 XLC/C++ compilation commands (cc_r for XLC, xlC_r for XLC++) for ESSL

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	cc_r(xlC_r) -O program.c -lessl -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib
SMP	32-bit integer, 64-bit pointer	cc_r(xlC_r) -O program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib
SMP CUDA	32-bit integer, 64-bit pointer	cc_r(xlC_r) -O program.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib

Table 6-6 shows the compilation commands for gcc/g++.

Table 6-6 gcc/g++ compilation commands for ESSL

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	gcc(g++) program.c -lessl -lxlf90_r -lxl -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib
SMP	32-bit integer, 64-bit pointer	gcc(g++) program.c -lesslsmp -lxlf90_r -lxl -lxlsmp -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib
SMP CUDA	32-bit integer, 64-bit pointer	gcc(g++) program.c -lesslsmpcuda -lxlf90_r -lxl -lxlsmp -lxlfmath -lm -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib

To compile C/C++ code for 64-bit integer, 64-bit pointer environments, add **-D_ESV6464** to your compile command.

Example 6-1 is a C sample source code that uses ESSL to multiply and generate 20000 by 20000 element matrixes. The examples use CBLAS interfaces to call ESSL routines. CBLAS interfaces provide additional options to specify the matrix order: column-major or row-major. For matrix multiplication, the examples use DGEMM, which works by using the following formula:

*C = alpha*A*B + beta*C*

Here, *alpha* and *beta* are real scalar values, *A*, *B*, and *C* are matrixes of conforming shape.

The examples calculate only time of execution and performance of the DGEMM routine as shown in Example 6-1.

Example 6-1 ESSL C example source code dgemm_sample.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <essl.h> //ESSL header for C/C++

//Function to calculate time in milleseconds
long timevaldiff(struct timeval *starttime, struct timeval *finishtime)
{
  long msec;
  msec=(finishtime->tv_sec-starttime->tv_sec)*1000;
  msec+=(finishtime->tv_usec-starttime->tv_usec)/1000;
  return msec;
}

int main()
{
  struct timeval start, end;
  double diff;
  long n, m, k;
  double *a, *b, *c;
  double rmax, rmin;
  double seed1, seed2, seed3;
  double flop;

  //Seeds for matrix generation
  seed1 = 5.0;
  seed2 = 7.0;
  seed3 = 9.0;

  //Maximum and minimum value elements of matrices
  rmax = 0.5;
  rmin = -0.5;

  //Size of matrices
  n = 20000; m = n; k =n;

  //Number of additions and multiplications
  flop = (double)(m*n*(2*(k-1)));
```

```
  //Memory allocation
  a = (double*)malloc(n*k*sizeof(double));
  b = (double*)malloc(k*m*sizeof(double));
  c = (double*)malloc(n*m*sizeof(double));

  //Matrix generation
  durand(&seed1,n*k,a); //DURAND are included to ESSL, not to CBLAS
  cblas_dscal(n*k,rmax-rmin,a,1);
  cblas_daxpy(n*k,1.0,&rmin,0,a,1);

  durand(&seed2,k*m,b);
  cblas_dscal(k*m,rmax-rmin,b,1);
  cblas_daxpy(k*m,1.0,&rmin,0,b,1);

  durand(&seed3,n*m,c);
  cblas_dscal(n*m,rmax-rmin,c,1);
  cblas_daxpy(n*m,1.0,&rmin,0,c,1);

  //Matrix multiplication (DGEMM)
  gettimeofday(&start,NULL);
  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                            m,n,k,1.0,a,n,b,k,1.0,c,n);
  gettimeofday(&end,NULL);

  //Print results
  printf("%.3lf seconds, ",(double)timevaldiff(&start,&end)/1000);
  printf("%.3lf MFlops\n",flop/(double)timevaldiff(&start,&end)/1000.0);

  //Memory deallocation
  free(a);
  free(b);
  free(c);

  return 0;
}
```

Example 6-2 shows how to compile and execute Example 6-1 on page 162 using the serial version of ESSL and the XLC compiler.

Example 6-2 Compilation and execution of dgemm_sample.c for serial ESSL

```
cc_r -O3 dgemm_sample.c -lessl -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/4.1.2/lib
-L/opt/ibm/xlf/15.1.2/lib -R/opt/ibm/lib -o dgemm_serial

./dgemm_serial
634.451 seconds, 25217.393 MFlops
```

Example 6-3 describes how to compile and execute Example 6-1 on page 162 using the SMP version of ESSL and the XLC compiler. To set the number of SMP threads, XLSMPOPTS are used.

Example 6-3 Compilation and execution of dgemm_sample.c for SMP ESSL

```
export XLSMPOPTS=parthds=20
cc_r -O3 dgemm_sample.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath
-L/opt/ibm/xlsmp/4.1.2/lib -L/opt/ibm/xlf/15.1.2/lib -R/opt/ibm/lib -o dgemm_smp
```

```
./dgemm_smp
56.542 seconds, 282961.338 MFlops
```

You can see that performance improves by 11 times for SMP runs compared with serial runs.

You can try to improve performance more by using the SMP Compute Unified Device Architecture (CUDA) version of ESSL, which enable usage of 4 GPU processors in the machine as shown in Example 6-4.

Example 6-4 Compilation and execution of dgemm_sample.c for SMP CUDA ESSL

```
export XLSMPOPTS=parthds=20
cc_r -O3 dgemm_sample.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas
-lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64
-L/opt/ibm/xlsmp/4.1.2/lib -L/opt/ibm/xlf/15.1.2/lib -R/opt/ibm/lib -o dgemm_cuda

./dgemm_cuda
16.554 seconds, 966485.442 MFlops
```

Additionally, the time required for execution drops by a factor of 3.5 when you start to use graphics processing unit (GPUs).

6.6.1, “Power Cap Limit” on page 196 describes the technique to adjust the Power Cap Limit, which helps to improve the performance of GPU cards. Example 6-5 is the SMP CUDA run after this workaround.

Example 6-5 SMP CUDA ESSL run after increase of Power Cap Limit

```
./dgemm_cuda
6.050 seconds, 2644495.868 MFlops
```

This technique is useful for your runs and increases performance by 2.5 times compared with runs with the default Power Cap Limit.

For more information about how to use other ESSL options or routines such as FFTW, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSFHY8/essl_welcome.html

6.2.2 Run different SMT modes

Example 6-3 on page 163 shows how the operating system chooses CPUs, which are used for the job run, by itself. However, you can use the environment variable *XLSMPOPTS* to control it.

The example system has 20 physical POWER8 cores and 160 logical CPUs. Each 8 CPUs depend on one physical core. By default the system is set to SMT-8 mode, which means that all 8 CPUs per core can be used. In the first run of Example 6-6, the example uses CPUs from 0 to 19. In the second run, only even numbers of CPUs are used, which simulates SMT-4 mode. The third and fourth runs work like SMT-2 and SMT-1 modes respectively.

Example 6-6 Different CPU binding for run with 20 SMP threads

```
export
XLSMPOPTS=parthds=20:PROCS=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
./dgemm_smp
```

```
479.098 seconds, 33394.420 MFlops

export
XLSMPOPTS=parthds=20:PROCS=0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38
./dgemm_smp
195.006 seconds, 82044.655 MFlops

export
XLSMPOPTS=parthds=20:PROCS=0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,7
6
./dgemm_smp
62.399 seconds, 256401.545 MFlops

export
XLSMPOPTS=parthds=20:PROCS=0,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120,128,13
6,144,152
./dgemm_smp
32.362 seconds, 494382.300 MFlops
```

For the fourth run with SMT-1 mode, running one thread on each POWER8 core helps to double performance compared with runs without CPU binding.

6.2.3 ESSL SMP CUDA library options

The ESSL SMP CUDA library has the following options that are controlled by the user:

- **Control which GPUs ESSL uses.** By default, ESSL uses all available devices, but you can change it using the environment variable *CUDA_VISIBLE_DEVICES* or the *SETGPUS* subroutine. GPUs have numeration from 0 in operation system. For example, if you want to use only second and third GPUs in your run, set the environment variable as follows:

  ```
  export CUDA_VISIBLE_DEVICES=1,2
  ```

  or place the following call into the code:

  ```
  int ids[2] = {1, 2}; //GPUs IDs
  int ngpus = 2; //Number of GPUs
  ...
  setgpus(ngpus, ids);
  /*your ESSL SMP CUDA call*/
  ```

  You can also specify a different order of devices. It can be useful in cases when you want to reduce latency between specific CPUs and GPUs.

- **Disable or enable hybrid mode.** By default, ESSL runs in hybrid mode. It means that ESSL routines will use both POWER8 CPUs and NVIDIA GPUs. To disable this mode and start using only GPUs, you need to specify the following environment variable:

  ```
  export ESSL_CUDA_HYBRID=no
  ```

  To enable it back, unset this variable or set it to yes.

- **Pin host memory buffers.** Different options are provided by ESSL:
  - Not pin host memory buffers (default).
  - Allow ESSL to pin buffers by itself. To do this, set the following environment variable:

    export ESSL_CUDA_PIN=yes
  - Provide information to ESSL that you will pin the buffers by yourself before the ESSL routines calls:

    export ESSL_CUDA_PIN=pinned

Example 6-7 shows runs of source code from Example 6-1 on page 162 with different ESSL SMP CUDA library options. Note that the examples use the adjusted Power Cap Limit.

Example 6-7 dgemm_sample.c runs with different SMP CUDA options

```
export XLSMPOPTS=parthds=20
cc_r -O3 dgemm_sample.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas
-lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64
-L/opt/ibm/xlsmp/4.1.2/lib -L/opt/ibm/xlf/15.1.2/lib -R/opt/ibm/lib -o dgemm_cuda
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 4 GPUs hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 4 GPUs non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1,2
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no

VAR=`./dgemm_cuda`
echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=1,2
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0
```

```
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (1st) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (1st) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=3
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (4th) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (4th) non-hybrid mode: $VAR"
```

The results of the runs are shown in Example 6-8.

Example 6-8 Result of different ESSL SMP CUDA runs

```
SMP CUDA run with 4 GPUs hybrid mode: 6.050 seconds, 2644495.868 MFlops
SMP CUDA run with 4 GPUs non-hybrid mode: 6.799 seconds, 2353169.584 MFlops
SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode: 7.322 seconds, 2185086.042
MFlops
SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode: 7.762 seconds,
2061221.335 MFlops
SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode: 10.869 seconds, 1472002.944
MFlops
SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode: 11.107 seconds, 1440460.971
MFlops
SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode: 10.910 seconds, 1466471.127
MFlops
SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode: 11.524 seconds, 1388337.383
MFlops
SMP CUDA run with 1 GPU (1st) hybrid mode: 14.954 seconds, 1069894.343 MFlops
SMP CUDA run with 1 GPU (1st) non-hybrid mode: 19.319 seconds, 828158.807 MFlops
SMP CUDA run with 1 GPU (4th) hybrid mode: 15.179 seconds, 1054035.180 MFlops
SMP CUDA run with 1 GPU (4th) non-hybrid mode: 19.833 seconds, 806695.911 MFlops
```

Hybrid mode gives a performance increase of about 25% for runs with one GPU. For other numbers of GPUs, it gives no so visible improvement. However, it can be useful for cases with significant problems or large number of ESSL runs, where improvement of performance is about 5% and gives many advantages.

For more information about the ESSL SMP CUDA library, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSFHY8_5.4.0/com.ibm.cluster.essl.v5r4.essl100.doc/am5gr_cuda.htm

6.3 Parallel ESSL

Parallel ESSL is highly optimized mathematical subroutine library for clusters of POWER8 processor nodes. Parallel ESSL supports the single program, multiple data (SPMD) programming model using the Message Passing Interface (MPI) library. It also assumes that

your program is using the SPMD programming model. This configuration means that all parallel tasks are identical and work on different sets of data.

Parallel ESSL supports only 32-bit integer, 64-bit pointer environment libraries, which need to be used with IBM Parallel Environment (PE) Runtime Edition MPICH library.

Parallel ESSL subroutines cover following computational areas:

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- Level 3 PBLAS
- Linear Algebraic Equations
- Eigensystem Analysis and Singular Value Analysis
- Fourier Transforms
- Random Number Generation

Parallel ESSL uses calls of the ESSL subroutines for computational purposes.

For communication, Basic Linear Algebra Communications Subprograms (BLACS) is included, which based on MPI.

6.3.1 Program development

During development of your program, the BLACS subroutines need to be used. To include Parallel ESSL calls into the code of the program, you can use the following steps:

1. Initialize the process grid using the BLACS subroutines (*BLACS_GET* call and *BLACS_GRIDINIT* or *BLACS_GRIDMAP* after it).
2. Distribute data across process grid. Try to use different block sizes to improve performance of program. For example, some Parallel ESSL subroutines start to use GPU for large sizes of blocks (about 2000 by 2000).
3. Call the Parallel ESSL subroutine on all processes of the BLACS process grid.
4. Aggregate results of Parallel ESSL runs from all processes.
5. Call the BLACS subroutines to clean the process grid and exit, such as *BLACS_GRIDEXIT*, *BLACS_EXIT*,

Example 6-9 shows a sample Fortran program that uses the Parallel ESSL version of PDGEMM subroutine for a 20000 by 20000 matrix size. PDGEMM works by using the following formula:

*C = alpha*A*B + beta*C*

Here, *alpha* and *beta* are real scalar values, *A*, *B*, and *C* are matrices of conforming shape.

Example 6-9 PESSL Fortran example source code pdgemm_sample.f

```
      program pdgemm_sample
      implicit none
      real*8, allocatable, dimension(:) :: a,b,c
      integer, dimension(9) :: desca,descb,descc
      integer m,n,k
      integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
      integer acol, bcol, ccol
      real*8, parameter :: alpha = 2.d0
      real*8, parameter :: beta  = 3.d0
      integer, parameter :: ia = 1, ib = 1, ic = 1
      integer, parameter :: ja = 1, jb = 1, jc = 1
```

```
      integer, parameter :: nb = 200
! Bigger size for CUDA runs
!     integer, parameter :: nb = 3000

! Initialization of process grid
      call blacs_pinfo(iam,np)
      if (np.ne.20) then
        print *, 'Test expects 20 nodes'
        stop 1
      else
        nr = 5
        nc = 4
      endif
      call blacs_get(0,0,icontxt)
      call blacs_gridinit(icontxt,'r',nr,nc)
      call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Size of matrices
      m = 20000
      n = 20000
      k = 20000

! Fill parameters for PDGEMM call
      desca(1) = 1
      desca(2) = icontxt
      desca(3) = m
      desca(4) = k
      desca(5:6) = nb
      desca(7:8) = 0
      desca(9) = numroc(m,nb,mr,0,nr)
      acol = numroc(k,nb,mc,0,nc)

      descb(1) = 1
      descb(2) = icontxt
      descb(3) = k
      descb(4) = n
      descb(5:6) = nb
      descb(7:8) = 0
      descb(9) = numroc(k,nb,mr,0,nr)
      bcol = numroc(n,nb,mc,0,nc)

      descc(1) = 1
      descc(2) = icontxt
      descc(3) = m
      descc(4) = n
      descc(5:6) = nb
      descc(7:8) = 0
      descc(9) = numroc(m,nb,mr,0,nr)
      ccol = numroc(n,nb,mc,0,nc)

      allocate(a(desca(9)*acol))
      allocate(b(descb(9)*bcol))
      allocate(c(descc(9)*ccol))
```

```
! PDGEMM call
      a = 1.d0
      b = 2.d0
      c = 3.d0
      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
     &            beta,c,ic,jc,descc)

! Deallocation of arrays and exit from BLACS
      deallocate(a)
      deallocate(b)
      deallocate(c)
      call blacs_gridexit(icontxt)
      call blacs_exit(0)
      end
```

For more information about usage of BLACS with the Parallel ESSL library, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSNR5K_5.2.0/com.ibm.cluster.pessl.v5r2.pssl100.doc/am6gr_dapc.htm

For more information about the concept of development using the Parallel ESSL library, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSNR5K_5.2.0/com.ibm.cluster.pessl.v5r2.pssl100.doc/am6gr_dlaspro.htm

6.3.2 Using GPUs with Parallel ESSL

GPUs can be used by the local MPI tasks in two ways within the Parallel ESSL programs:

- **GPUs are not shared.** This setting means that each MPI task on a node uses unique GPUs. For this case, the local rank of MPI tasks can be used.

  Example 6-10 shows how to work with local rank by using the *MP_COMM_WORLD_LOCAL_RANK* variable. It is created from Example 6-9 on page 168 with an additional section that gets the local rank of the MPI task and assigns this task to a respective GPU by rank. Also, change size of process grid to 4 by 2 to fit your cluster, which has two nodes with 4 GPUs on each node, and uses a block size of 3000 by 3000.

  Example 6-10 PESSL Fortran example source code for non-shared GPUs

```
      program pdgemm_sample_local_rank
      implicit none
      real*8, allocatable, dimension(:) :: a,b,c
      integer, dimension(9) :: desca,descb,descc
      integer m,n,k
      integer ids(1)
      integer ngpus
      integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
      integer acol, bcol, ccol
      real*8, parameter :: alpha = 2.d0
      real*8, parameter :: beta  = 3.d0
      integer, parameter :: ia = 1, ib = 1, ic = 1
      integer, parameter :: ja = 1, jb = 1, jc = 1
```

```
      integer, parameter :: nb = 3000

      character*8 rank
      integer lrank,istat

! Initialization of process grid
      call blacs_pinfo(iam,np)
      if (np.ne.8) then
        print *, 'Test expects 8 nodes'
        stop 1
      else
        nr = 4
        nc = 2
      endif
      call blacs_get(0,0,icontxt)
      call blacs_gridinit(icontxt,'r',nr,nc)
      call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Get local rank and assign respective GPU
      call getenv('MP_COMM_WORLD_LOCAL_RANK',value=rank)
      read(rank,*,iostat=istat) lrank
      ngpus = 1
      ids(1) = lrank
      call setgpus(1,ids)

! Size of matrices
      m = 20000
      n = 20000
      k = 20000

! Fill parameters for PDGEMM call
      desca(1) = 1
      desca(2) = icontxt
      desca(3) = m
      desca(4) = k
      desca(5:6) = nb
      desca(7:8) = 0
      desca(9) = numroc(m,nb,mr,0,nr)
      acol = numroc(k,nb,mc,0,nc)

      descb(1) = 1
      descb(2) = icontxt
      descb(3) = k
      descb(4) = n
      descb(5:6) = nb
      descb(7:8) = 0
      descb(9) = numroc(k,nb,mr,0,nr)
      bcol = numroc(n,nb,mc,0,nc)

      descc(1) = 1
      descc(2) = icontxt
      descc(3) = m
      descc(4) = n
      descc(5:6) = nb
      descc(7:8) = 0
```

```
      descc(9) = numroc(m,nb,mr,0,nr)
      ccol = numroc(n,nb,mc,0,nc)

      allocate(a(desca(9)*acol))
      allocate(b(descb(9)*bcol))
      allocate(c(descc(9)*ccol))

! PDGEMM call
      a = 1.d0
      b = 2.d0
      c = 3.d0
      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
    &            beta,c,ic,jc,descc)

! Deallocation of arrays and exit from BLACS
      deallocate(a)
      deallocate(b)
      deallocate(c)
      call blacs_gridexit(icontxt)
      call blacs_exit(0)
      end
```

- **GPUs are shared.** This is the case when the number of MPI tasks per node oversubscribe the GPUs. Parallel ESSL recommends using NVIDIA MPS, described in 6.6.2, “CUDA Multi-Process Service” on page 197, for this case. The process allows you to use multiple MPI tasks concurrently by using GPUs.

  Note: It is possible that Parallel ESSL will be unable to allocate memory on the GPU. In this case, you can reduce the number of MPI tasks per node.

  To inform Parallel ESSL which GPUs to use for MPI tasks, use the SETGPUS subroutine. Example 6-11 shows the updated version of Example 6-9 on page 168, where Parallel ESSL uses only one GPU for each MPI task, and the GPU is assigned in round-robin order.

  Example 6-11 Call of SETGPUS subroutine for 1 GPU usage

```
   program pdgemm_sample
   implicit none
   real*8, allocatable, dimension(:) :: a,b,c
   integer, dimension(9) :: desca,descb,descc
   integer m,n,k
   integer ids(1)
   integer ngpus
   integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
   integer acol, bcol, ccol
   real*8, parameter :: alpha = 2.d0
   real*8, parameter :: beta  = 3.d0
   integer, parameter :: ia = 1, ib = 1, ic = 1
   integer, parameter :: ja = 1, jb = 1, jc = 1

   integer, parameter :: nb = 3000

! Initialization of process grid
   call blacs_pinfo(iam,np)
```

```
      if (np.ne.20) then
        print *, 'Test expects 20 nodes'
        stop 1
      else
        nr = 5
        nc = 4
      endif
      ngpus = 1
      ids(1) = mod(iam,4)
      call setgpus(ngpus,ids)
      call blacs_get(0,0,icontxt)
      call blacs_gridinit(icontxt,'r',nr,nc)
      call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Size of matrices
      m = 20000
      n = 20000
      k = 20000

! Fill parameters for PDGEMM call
      desca(1) = 1
      desca(2) = icontxt
      desca(3) = m
      desca(4) = k
      desca(5:6) = nb
      desca(7:8) = 0
      desca(9) = numroc(m,nb,mr,0,nr)
      acol = numroc(k,nb,mc,0,nc)

      descb(1) = 1
      descb(2) = icontxt
      descb(3) = k
      descb(4) = n
      descb(5:6) = nb
      descb(7:8) = 0
      descb(9) = numroc(k,nb,mr,0,nr)
      bcol = numroc(n,nb,mc,0,nc)

      descc(1) = 1
      descc(2) = icontxt
      descc(3) = m
      descc(4) = n
      descc(5:6) = nb
      descc(7:8) = 0
      descc(9) = numroc(m,nb,mr,0,nr)
      ccol = numroc(n,nb,mc,0,nc)

      allocate(a(desca(9)*acol))
      allocate(b(descb(9)*bcol))
      allocate(c(descc(9)*ccol))

! PDGEMM call
      a = 1.d0
      b = 2.d0
      c = 3.d0
```

```
      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
     &            beta,c,ic,jc,descc)

! Deallocation of arrays and exit from BLACS
      deallocate(a)
      deallocate(b)
      deallocate(c)
      call blacs_gridexit(icontxt)
      call blacs_exit(0)
      end
```

Example 6-11 on page 172 explicitly states to use two GPUs per MPI task. To change to this configuration, change the following lines:

```
ngpus = 1
ids(1) = mod(iam,4)
```

to these:

```
ngpus = 2
ids(1) = mod(iam,2)*2
ids(2) = mod(iam,2)*2 + 1
```

6.3.3 Compilation

The Parallel ESSL subroutines can be called from 64-bit-environment application programs, which are written in Fortran, C, and C++. Compilation commands of source code for different type of Parallel ESSL library (SMP and SMP CUDA) using MPICH libraries are described in Table 6-7, Table 6-8 and Table 6-9 on page 175.

Table 6-7 Fortran compilation commands for PESSL using MPICH

Type of PESSL library	Compilation command
64-bit SMP	mpfort -O program.f -leeslsmp -lpesslsmpich -lblacsmpich -lxlsmp
64-bit SMP CUDA	mpfort -O program.f -leeslsmpcuda -lpesslsmpich -lblacsmpich -lxlsmp -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64

Table 6-8 shows commands for XLC/XLC++.

Table 6-8 XLC/XLC++ compilation commands (mpcc for XLC, mpCC for XLC++)

Type of PESSL library	Compilation command
64-bit SMP	mpcc(mpCC) -O program.c -leeslsmp -lpesslsmpich -lblacsmpich -lxlf90_r -lxlsmp -lxlfmath -lmpigf -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib
64-bit SMP CUDA	mpcc(mpCC) -O program.c -leeslsmpcuda -lpesslsmpich -lblacsmpich -lxlf90_r -lxlsmp -lxlfmath -lmpigf -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib

Table 6-9 shows commands for gcc/g++.

Table 6-9 gcc/g++ compilation commands for PESSL using MPICH

Type of PESSL library	Compilation command
64-bit SMP	mpcc(mpCC) -compiler gnu -O program.c -leeslsmp -lpesslsmpich -lblacsmpich -lxlf90_r -lxl -lxlsmp -lxlfmath -lm-lmpigf -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib
64-bit SMP CUDA	mpcc(mpCC) -compiler gnu -O program.c -leeslsmpcuda -lpesslsmpich -lblacsmpich -lxlf90_r -lxl -lxlsmp -lxlfmath -lm -lmpigf -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version>/lib -L/opt/ibm/xlf/<xlf_version>/lib -R/opt/ibm/lib

For more information about compilation of programs with the Parallel ESSL library, see the following website:

http://www.ibm.com/support/knowledgecenter/#!/SSNR5K_5.2.0/com.ibm.cluster.pessl.v5r2.pssl100.doc/am6gr_det6000l.htm

6.4 Using POWER8 vectorization

The single-instruction, multiple-data (SIMD) instructions are building blocks that are used to exploit parallelism at CPU level. The Power architecture implements SIMD through the VMX and VSX technologies, which are specified in the Power Instruction Set Architecture (Power ISA) version 2.07 for POWER8 generation processor.

Techniques to exploit data parallelism through SIMD instructions are often called *vectorization*. They can be employed by the compiler in the form of auto-vectorization transformations or exposed to applications as application programming interface (API).

Both GNU GCC and IBM XL compilers provide vector APIs based on the Altivec specification for C, C++, and Fortran. Their API is composed of built-in functions (also known as intrinsics), defined vector types, and extensions to the programming language semantics. The specifics of each compiler vector implementation are explained in the next sections.

The auto-vectorization features of the GCC and XL compilers are discussed in 6.1, “Compiler options” on page 156.

6.4.1 Implementation with GNU GCC

On GNU GCC C, the AltiVec API specification is implemented with modifications that are listed in the “PowerPC AltiVec Built-in Functions” section of *Using the GNU Compiler Collection (GCC)* at the following website:

https://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc.pdf

The application must be compiled at least with the `-maltivec` option. Or they must be compiled with `-mvsx`, which enables `-maltivec` with additional features that use VSX instructions.

The GNU GCC AltiVec high-level interface and C language extensions are specified in the `altivec.h` header.

The following features are implemented:

- Add the keywords `__vector`, `vector`, `__pixel`, `pixel`, `__bool`, and `bool`. The `vector`, `pixel`, and `bool` keywords are implemented as context-sensitive, predefined macros that are recognized only when used in C-type declaration statements. In C++ applications, they can be undefined for compatibility.
- Unlike the AltiVec specification, the GNU/GCC implementation does not allow a `typedef` name as a type identifier. You must use the actual `__vector` keyword, for instance, `typedef signed short int16; __vector int16 myvector`.
- Vector data types are aligned on a 16-byte boundary.
- Aggregates (structures and arrays) and unions that contain vector types must be aligned on 16-byte boundaries.
- Load or store to unaligned memory must be carried out explicitly by one of the `vec_ld`, `vec_ldl`, `vec_st`, or `vec_stl` operations. However, the load of an array of components does not need to be aligned, but it must be accessed with attention to its alignment, which is usually carried out with a combination of `vec_lvsr`, `vec_lvsl`, and `vec_perm` operations.
- Using `sizeof()` for vector data types (or pointers) returns 16, for 16 bytes.
- Assignment operation (`a = b`) is allowed only if both sides have the same vector types.
- Address operation `&p` is valid if `a` is `p` vector type.
- The usual pointer arithmetic can be performed on vector type pointer `p`, in particular:
  - `p+1` is a pointer to the next vector after `p`.
  - Pointer dereference (`*p`) implies either a 128-bit vector load from or store to the address that is obtained by clearing the low-order bits of `p`.

C arithmetic and logical operators (+, -, *, /, unary minus, ^, |, &, ~, and %), shifting operators (<<, >>), and comparison operators (==, !=, <, <=, >, >=) can be used on these types. The compiler will generate the correct SIMD instructions for the hardware.

Table 6-10 shows vector data type extensions as implemented by GCC. Vector types are signed by default, unless an otherwise `unsigned` keyword is specified. The only exception is `vector char`, which is unsigned by default. The hardware does not have instructions for supporting `vector long long` and `vector bool long long` types, but they can be used for float-point/integer conversions.

Table 6-10 Vector types as implemented by GCC

Vector types	Description
vector char	Vector of sixteen 8-bit char
vector bool	Vector of sixteen 8-bit unsigned char
vector short	Vector of eight 16-bit short
vector bool short	Vector of eight 16-bit unsigned short
vector pixel	Vector of eight 16-bit unsigned short

Vector types	Description
vector int	Vector of four 32-bit integer
vector bool int	Vector of four 32-bit integer
vector float	Vector of four 32-bit float
vector double	Vector of two 64-bit double. Requires compile with `-mvsx`
vector long	Vector of two 64-bit signed integer. It is implemented in 64-bit mode only. Requires compile with `-mvsx`
vector long long	Vector of two 64-bit signed integer
vector bool long	Vector of two 64-bit signed integer

In addition to vector operations, GCC has a built-in function for cryptographic instructions that operate in vector types. For more information about the implementation and a comprehensive list of built-in functions, see the *PowerPC AltiVec* section in GNU GCC:

https://gcc.gnu.org/onlinedocs/gcc-4.8.5/gcc/PowerPC-AltiVec_002fVSX-Built-in-Functions.html#PowerPC-AltiVec_002fVSX-Built-in-Functions

6.4.2 Implementation with IBM XL

The IBM XL compiler family provides an implementation of AltiVec APIs through feature extensions for both C and C++. Fortran extensions for vector support are also available.

XL C/C++

To use vector extensions, the application must be compiled with `-mcpu=pwr8`, and `-qaltivec` must be in effect.

The XL C/C++ implementation defines the `vector` (or alternatively, `__vector`) keywords that are used in the declaration of vector types.

Similar to GCC implementation, XL C/C++ allows unary, binary, and relational operators to be applied to vector types. It implements all data types that are shown in Table 6-10 on page 176.

The indirection operator, asterisk (`*`), is extended to handle pointer to vector types. Pointer arithmetic is also defined so that a pointer to the following vector can be expressed as `v+ 1`.

Vector types can be cast to other vector types (but not allowed to a scalar). The casting does not represent a conversion and so it is subject to changes in element value. Casting between vector and scalar pointers is also allowed if memory is maintained on 16-byte alignment.

For more information about XL C/C++ 13.1.3 vector support , vector initialization, and the `vec_step` operator, see the manual at the following website:

http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.3/com.ibm.xlcpp1313.lelinux.doc/language_ref/altivec_exts_both.html

A comprehensive list of built-in functions for vector operations is available at XL C and C++ 1.3.1.3 manual at the following website:

http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.3/com.ibm.xlcpp1313.lelinux.doc/compiler_ref/vec_intrin_cpp.html

XL Fortran

To use vector extensions, the application must comply with `-qarch=pwr8`.

The XL Fortran language extension defines the `VECTOR` keyword, which is used to declare 16-byte vector entities that can hold `PIXEL`, `UNSIGNED`, `INTEGER`, and `REAL` type elements. `PIXEL` (2 bytes) and `UNSIGNED` (unsigned integer) types are extensions to the language as well. They must be only used within vectors.

Vectors are automatically aligned to 16 bytes, but exceptions apply. For more information about vector types on XL Fortran 15.1.3, see the manual at the following website:

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.3/com.ibm.xlf1513.lelinux.doc/language_ref/vectordatatype.html

For the list of vector intrinsic procedures available with XL Fortran 15.1.3, see the manual at the following website:

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.3/com.ibm.xlf1513.lelinux.doc/language_ref/vmxintrinsics.html

Example 6-12 uses vectors to calculate the C = alpha * A + B formula where alpha, A, B, and C are real numbers. The lines 11-14 show declaration of vectors with 16 elements of 8 bytes real types. Called methods vec_xld2, `vec_permi`, `and vec_madd` are, respectively, load, permuting, and fused multiply-add SIMD operations that are applied to the vector types.

Example 6-12 Fortran program that demonstrates use of XL compiler vectors

```
 1      SUBROUTINE VSX_TEST
 2      implicit none
 3
 4      real*8, allocatable :: A(:), B(:), C(:), CT(:)
 5      real*8 alpha
 6      integer*8 max_size, ierr
 7      integer*8 i, j, it
 8      integer*8 ia, ialign
 9      integer   n, nalign
10
11      vector(real*8) va1, va2, va3
12      vector(real*8) vb1, vb2
13      vector(real*8) vc1, vc2
14      vector(real*8) valpha
15
16      max_size = 2000
17      alpha = 2.0d0
18
19      ierr = 0
20      allocate(A(max_size),stat=ierr)
21      allocate(B(max_size),stat=ierr)
22      allocate(C(max_size),stat=ierr)
23      allocate(CT(max_size),stat=ierr)
24      if (ierr .ne. 0) then
25        write(*,*) 'Allocation failed'
26        stop 1
27      endif
28
29      do i = 1, max_size
30        a(i) = 1.0d0*i
31        b(i) = -1.0d0*i
```

```
32        ct(i) = alpha*a(i) + b(i)
33      enddo
34
35      ia     = LOC(A)
36      ialign = IAND(ia, 15_8)
37      nalign = MOD(RSHIFT(16_8-ialign,3_8),7_8) + 2
38
39 !     Compute Head
40      j = 1
41      do i = 1, nalign
42        C(j) = B(j) + alpha * A(j)
43        j = j + 1
44      enddo
45
46      n = max_size - nalign - 4
47      it = rshift(n, 2)
48
49      va1 = vec_xld2( -8, A(j))
50      va2 = vec_xld2(  8, A(j))
51      va3 = vec_xld2( 24, A(j))
52
53      va1 = vec_permi( va1, va2, 2)
54      va2 = vec_permi( va2, va3, 2)
55
56      vb1 = vec_xld2(  0, B(j))
57      vb2 = vec_xld2( 16, B(j))
58
59      do i = 1, it-1
60        vc1 = vec_madd( valpha, va1, vb1)
61        vc2 = vec_madd( valpha, va2, vb2)
62
63        va1 = va3
64        va2 = vec_xld2( 40, A(j))
65        va3 = vec_xld2( 56, A(j))
66
67        va1 = vec_permi( va1, va2, 2)
68        va2 = vec_permi( va2, va3, 2)
69
70        call vec_xstd2( va1,  0, C(j))
71        call vec_xstd2( va2, 16, C(j))
72
73        vb1 = vec_xld2( 32, B(j))
74        vb1 = vec_xld2( 48, B(j))
75
76        j = j + 4
77      enddo
78
79      vc1 = vec_madd( valpha, va1, vb1)
80      vc2 = vec_madd( valpha, va2, vb2)
81
82      call vec_xstd2( va1,  0, C(j))
83      call vec_xstd2( va2, 16, C(j))
84
85 !     Compute Tail
86      do i = j, max_size
```

```
87        C(i) = B(i) + alpha * A(i)
88      enddo
89
90      do i = 1, 10
91        write(*,*) C(i), CT(i)
92      enddo
93
94      END SUBROUTINE VSX_TEST
```

6.5 Development models

The High Performance Computing (HPC) solution proposed in this book contains a software stack that allows development of C, C++, and Fortran applications by using different parallel programming models. In this context, applications can be implemented using *pure models* as MPI, OpenMP, CUDA, PAMI, or OpenSHMEM, or by using some combinations of these (also known as *hybrid models*).

This section discusses aspects of the IBM PE, compilers (GNU and IBM XL families), libraries, and toolkits that developers can use to implement applications on either pure or hybrid parallel programming models. It does not describe how those applications can be run with IBM PE. That topic is covered in 7.2, “Using the IBM Parallel Environment runtime” on page 233.

6.5.1 MPI programs with IBM Parallel Environment

The MPI development and runtime environment that is provided by the IBM PE version 2.3 has the following general characteristics:

- Provides the implementation of MPI version 3.0 standard, based on open source MPICH project.
- The MPI library uses PAMI protocol as a common transport layer.
- Supports MPI application in C, C++, and Fortran.
- Supports 64-bit applications only.
- Supports GNU and IBM XL compilers.
- MPI operations can be carried out on both main or user-space threads.
- The I/O component (also known as MPI-IO) is an implementation of ROMIO provided by MPICH 3.1.2.
- Provides a CUDA-aware MPI implementation.
- Employs a shared memory mechanism for message transport between tasks on the same compute node. In contrast, the User Space (US) communication subsystem, which provides direct access to a high-performance communication network by way of an InfiniBand adapter, is used for internode tasks.
- Allows message stripping, failover, and recovery on multiple or single (with some limitations) network configurations.
- Allows for dynamic process management.

This section introduces some MPI implementation aspects of IBM PE Runtime and general guidance on how to build parallel applications. For more detailed information, see the *Parallel Environment Runtime Edition for Linux: MPI Programming Guide* that is found at the following website:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.3.0/com.ibm.cluster.pe.v2r3.pe400.doc/am106_about.htm?cp=SSFK3V_2.3.0%2F0-0-2

The MPI API

The MPI implementation of PE is based on MPICH. Therefore, to get more information, see its man pages for further information about the MPI API at the following website:

http://www.mpich.org

The provided compilers

The compilers provide a set of compilation scripts to build parallel applications that support both GNU and IBM XL family compilers for C, C++ and Fortran.

C, C++ and Fortran applications built, respectively, with **mpcc**, **mpCC**, and **mpfort** legacy compilation scripts are linked with the threaded version of MPI and poe libraries by default. They also apply some instrumentation on binary file so that poe is indirectly evoked to manage the parallel execution.

The **mpicc**, **mpicxx**, **mpif77**, and **mpif90** compilation scripts for C, C++, Fortran77, and Fortran90, respectively, are designed to build MPICH-based parallel applications. Notice that a program that is compiled with those scripts can be executed through poe.

Example 6-13 shows the **mpicc** command compiling of an MPI C application using the GNU GCC compiler.

Example 6-13 IBM PE Runtime mpicc command to compile an MPI C program

```
$ export PATH=/opt/ibmhpc/pecurrent/base/bin:$PATH
$ mpicc -compiler gnu -O3 -mcpu=power8 -mtune=power8 -o myApp main.c
```

Use the `-show` option to display the command that will be executed to compile the application. In Example 6-13, the **mpicc -show** command produces this output:

```
$ mpicc -show -compiler gnu -O3 -mcpu=power8 -mtune=power8 -o myApp main.c
/usr/bin/gcc -Xlinker --no-as-needed  -O3  -mcpu=power8  -mtune=power8  -o  myApp
main.c  -m64 -D__64BIT__ -Xlinker --allow-shlib-undefined -Xlinker
--enable-new-dtags -Xlinker -rpath -Xlinker /opt/ibmhpc/pecurrent/mpich/gnu/lib64
-I/opt/ibmhpc/pecurrent/mpich/gnu/include64 -I/opt/ibmhpc/pecurrent/base/include
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64 -lmpi
```

All of these compilation scripts use the XL compilers unless the MP_COMPILER variable or `-compiler` option is set, which instructs them to use another compiler. You can use `gnu` or `xl` option values to evoke GNU or XL compilers. For third-party compilers, use the fully qualified path (for example, `MP_COMPILER=/opt/at9.0/bin/gcc`).

Note: The compilation scripts that are provided by the latest version of the IBM PE Runtime are installed in `/opt/ibmhpc/pecurrent/base/bin`.

Details of MPI-IO implementation

The ROMIO implementation of IBM PE Runtime is configured to exploit the IBM Spectrum scale file system, delivering high-performance I/O operations. Some environment variables are also introduced to allow users to control the behavior of some operations, such as collective aggregations.

Although the configuration also supports NFS and POSIX compliance file systems, some limitations might apply.

The file system detection mechanism uses system calls, unless the parallel file system is set with the `ROMIO_FSTYPE_FORCE` environment variable. Many changes in the default configuration of MPI-IO by passing hints to ROMIO are allowed, setting them in the `ROMIO_HINTS` environment variable.

The local rank property

The Parallel Environment provided MPI implementation comes with a mechanism to determine the rank of a task among others tasks running in the same machine, which is also called the *task local rank*.

The read-only MP_COMM_WORLD_LOCAL_RANK variable can be used to obtain the local rank. Each task can read its local rank attributed and made available by the runtime environment.

Example 6-14 shows an MPI C program that reads the MP_COMM_WORLD_LOCAL_RANK and print its value to standard output.

Example 6-14 Simple MPI C program that prints task local rank

```
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>

int main(int argc, char* argv[]) {
  int world_rank, world_size, local_rank;
  char hostname[255];

  MPI_Init(&argc, &argv);

  gethostname(hostname, 255);
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  local_rank = atoi(getenv("MP_COMM_WORLD_LOCAL_RANK"));
  printf("Task %d: running on node %s and local rank %d\n", world_rank, hostname,
local_rank);

  MPI_Finalize();
  return EXIT_SUCCESS;
}
```

The output of that program is shown in Example 6-15.

Example 6-15 Show output of simple MPI C program that prints the local rank

```
$ mpcc main.c
$ MP_RESD=poe MP_PROCS=5 ./a.out
```

```
Task 0: running on node xcat-mn.xcat-cluster and local rank 0
Task 1: running on node xcat-mn.xcat-cluster and local rank 1
Task 4: running on node xcat-mn.xcat-cluster and local rank 4
Task 2: running on node xcat-mn.xcat-cluster and local rank 2
Task 3: running on node xcat-mn.xcat-cluster and local rank 3
```

Switch MPI configurations with environment modules

The process of compiling and executing a parallel application with PE requires setting a number of environment variables. Some predefined profiles that use environment modules[1] to change the system's variables are available to ease this task. As of PE 2.3, the following development profiles for MPI applications are provided:

- `perf`: Compile the MPI application with XL and set to run in development mode with minimal error checking
- `debug`: Compile the MPI application with XL and set to run in development mode with the debug versions of the libraries
- `trace`: Compile the MPI application with XL and set to run in development mode with the trace libraries

The environment module command (module) can be installed in the system. For RHEL 7.2, it can be installed with the following commands:

```
# yum install environment-modules
```

After the module command is installed in the system, you can list all available module, add the ones provided by PE, and load them as shown in Example 6-16.

Example 6-16 Show the use of modules to set environment to build and run MPI applications

```
$ module avail

-------------------------------------------------------------------------
/usr/share/Modules/modulefiles
-------------------------------------------------------------------------
dot         module-git  module-info modules     null        use.own
$ module use -a /opt/ibmhpc/pecurrent/base/module
$ echo $MODULEPATH
/usr/share/Modules/modulefiles:/etc/modulefiles:/opt/ibmhpc/pecurrent/base/module
$ module avail

-------------------------------------------------------------------------
/usr/share/Modules/modulefiles
-------------------------------------------------------------------------
dot         module-git  module-info modules     null        use.own

-----------------------------------------------------------------------
/opt/ibmhpc/pecurrent/base/module
-----------------------------------------------------------------------
pe2300.xl.debug pe2300.xl.perf  pe2300.xl.trace
$ module whatis pe2300.xl.debug
pe2300.xl.debug      : Adds PE environment variables for xl compiler and debug
develop mode to user environment.

$ module whatis pe2300.xl.perf
```

[1] Learn about Linux environment modules at the project website: http://modules.sourceforge.net

```
pe2300.xl.perf       : Adds PE environment variables for xl compiler and
performance develop mode to user environment.

$ module whatis pe2300.xl.trace
pe2300.xl.trace      : Adds PE environment variables for xl compiler and trace
develop mode to user environment.

$ env | grep MP
$ module load pe2300.xl.perf
  Adds these PE settings into your environment:

  MP_COMPILER=xl
  MP_EUIDEVELOP=min
  MP_MPILIB=mpich
  MP_MSG_API=mpi
  MP_CONFIG=2300
$ env | grep MP
MP_EUIDEVELOP=min
MP_CONFIG=2300
MP_MSG_API=mpi
MP_MPILIB=mpich
MP_COMPILER=xl
$ module load pe2300.xl.debug
  Adds these PE settings into your environment:

  MP_COMPILER=xl
  MP_EUIDEVELOP=debug
  MP_MPILIB=mpich
  MP_MSG_API=mpi
  MP_CONFIG=2300
$ env | grep MP
MP_EUIDEVELOP=debug
MP_CONFIG=2300
MP_MSG_API=mpi
MP_MPILIB=mpich
MP_COMPILER=xl
```

Simulate different SMT modes

The example cluster contains two nodes (S822LC), each of which has two sockets. One NUMA node corresponds to one socket and has 10 physical POWER8 cores inside it. The systems are configured with SMT-8, which means that a physical core can split a job between eight logical CPUs (threads). The structure of the S822LC is provided in Figure 6-1.

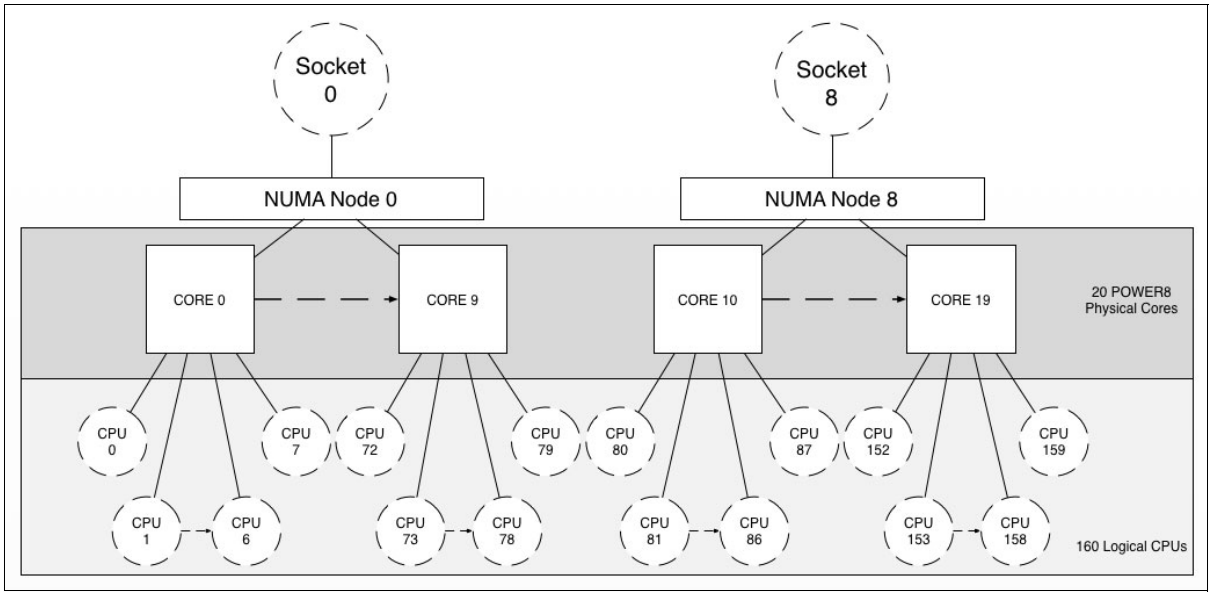

Figure 6-1 Structure of node in cluster

PE MPI provides run options or environment variables, which helps to simulate runs with different SMT modes. The following cases describe two ways when a job uses only 1 logical CPU from each POWER8 core (SMT-1 mode), and fully leverages all 160 logical CPUs using Example 6-9 on page 168 with Parallel ESSL calls:

1. To configure the job to use only one logical CPU per POWER8 core, the following run command can be used:

   ```
   MP_TASK_AFFINITY=core:1 MP_RESD=poe MP_PROCS=40 ./test_pdgemm
   ```

   MP_TASK_AFFINITY=core:1 says to PE MPI that each MPI task can use only one POWER8 core. Overall the full cluster has 40 POWER8 cores (each node contains 20 cores), that means that a maximum of 40 MPI (*MP_PROCS=40*) tasks can be used for such run.

   Another possible solution to simulate SMT-1 mode is to take 20 MPI tasks with two POWER8 cores for each of them using the following run command:

   ```
   MP_TASK_AFFINITY=core:2 MP_RESD=poe MP_PROCS=20 ./test_pdgemm
   ```

2. The following run command shows how to take advantage of the cluster and use all 160 logical CPUs per node:

   ```
   MP_TASK_AFFINITY=cpu:16 MP_RESD=poe MP_PROCS=20 ./test_pdgemm
   ```

   MP_TASK_AFFINITY=core:1 means for PE MPI that each MPI task can use only 16 logical CPUs. Each node in the cluster contains 160 logical CPUs, so 10 MPI tasks per node and 20 for the overall cluster can be used in this run.

If you want to change the CPU affinity variable and still want to use all logical CPUs, the number of MPI tasks need to be changed respectively. The following commands describe the alternative calls:

```
MP_TASK_AFFINITY=cpu:160 MP_RESD=poe MP_PROCS=2 ./test_pdgemm
MP_TASK_AFFINITY=cpu:8 MP_RESD=poe MP_PROCS=40 ./test_pdgemm
MP_TASK_AFFINITY=cpu:2 MP_RESD=poe MP_PROCS=160 ./test_pdgemm
```

6.5.2 CUDA C programs with the NVIDIA CUDA Toolkit

The development of parallel programs using the General Purpose GPU (GPGPU) model is provided in the IBM Power S822LC systems with support of the NIVIDIA CUDA Toolkit 7.5 for Linux on POWER8.

This section highlights some relevant aspects to the CUDA development in the proposed solution. This section also discusses characteristics of the NVIDIA K80 GPU, integration between compilers, and availability of libraries. For a deeper and platform-agnostic understanding of the NVIDIA CUDA development model and resources, see the following website:

http://developer.nvidia.com/cuda-zone

Understanding the NVIDIA K80 GPU CUDA capabilities

The NVIDIA Tesla K80 is a dual-GPU product with two attached GPUs each implementing the Kepler GK210 CUDA compute architecture. As such, the host operating system recognizes four GPUs in an IBM S822LC system that is fully populated with two Tesla K80 cards.

The Kepler GK210 architecture delivers CUDA computability version 3.7.

Table 6-11 summarizes the available resources (per GPU) and the limitations for the CUDA C++ applications.

Table 6-11 CUDA available resources per GPU

GPU resources	Value
Total of global memory	12 GB
Total amount of constant memory	64 KB
Shared memory per block	48 KB
Stream Multiprocessors (SMX)	13
Maximum warps per SMX	64
Threads per Warps	32
Maximum threads per SMX	2048
Maximum thread blocks per SMX	16
Maximum threads per block	1024
Maximum grid size	(x,y,z) = (2147483647, 65535, 65535)
Maximum thread block size	(x,y,z) = (1024, 1024, 64)

GPU resources	Value
Maximum texture dimension size	1D = (65536), 2D = (65536, 65536), 3D = (4096, 4096, 4096)
Maximum layered texture Size and number of layers	1D = ((16384), 2048 layers), 2D = (16384, 16384), 2048 layers)

The following features are supported by the Tesla K80 GPU:

- Dynamic parallelism
- Hyper-Q (also known as CUDA streams)
- Host pinned memory
- Supports Unified Virtual Addressing (UVA) space

Note: GPUDirect is not supported in the IBM Power Systems servers at the time of writing. Even though, the IBM Parallel Environment provides an CUDA-Aware API that supports GPU to GPU buffer transfers. For more information, see 6.5.3, “Hybrid MPI and CUDA programs with IBM Parallel Environment” on page 190.

The NVIDIA nvcc compiler

The NVIDIA **nvcc** compiler driver is responsible for generating the final executable, which is a combination of host (CPU) and device (GPU) codes.

Both IBM XL and GNU GCC compilers can be used to generate the host code and even their flags. As described in 6.1, “Compiler options” on page 156, this process produces an optimized code to run in the IBM POWER8 processor. By default, **nvcc** is going to use the GNU GCC, unless the **-ccbin** flag is passed to set another back end compiler.

Note: The **nvcc** compiler uses the GNU GCC C++ compiler by default to generate the host code. If it is desired, the IBM XL C++ compiler instead uses the **-ccbin xlC** option.

The NVIDIA Tesla K80 GPUs are built on Kepler GK210 architecture, which provides CUDA computability 3.7. Use the `-gencode` compiler option to set the virtual architecture and binary compatibility for example, `-gencode arch=compute_37,code=sm_37 will generate code compatible to the Kepler GK210 architecure (virtual architecture 3.7).`

Example 6-17 shows a simple CUDA program that prints to standard output the index values for 2D thread blocks. The executable is built with **nvcc** that uses xlC as the host compiler (`-ccbin xlC` option). Parameters to xlC are passed with `-Xcompiler` option.

Example 6-17 Simple CUDA C program and nvcc compilation

```
$ cat hello.cu
#include<cuda_runtime.h>
#include<stdio.h>

  __global__ void helloKernel() {
      int tidx = threadIdx.x;
      int tidy = threadIdx.y;
      int blockidx = blockIdx.x;
      int blockidy = blockIdx.y;

      printf("I am CUDA thread (%d, %d) in block (%d, %d)\n",
        tidx, tidy, blockidx, blockidy);
  };
```

```
int main(int argc, char* argv[]) {
  dim3 block(16,16);
  dim3 grid(2,2);
  helloKernel<<<grid, block>>>();
  cudaDeviceSynchronize();
  return 0;
}
$ nvcc -ccbin xlC -Xcompiler="-qarch=pwr8 -qtune=pwr8 -qhot -O3" -gencode
arch=compute_37,code=sm_37 hello.cu -o helloCUDA
$ ./helloCUDA
I am CUDA thread (0, 0) in block (1, 0)
I am CUDA thread (1, 0) in block (1, 0)
I am CUDA thread (2, 0) in block (1, 0)
I am CUDA thread (3, 0) in block (1, 0)
I am CUDA thread (4, 0) in block (1, 0)
I am CUDA thread (5, 0) in block (1, 0)
<... Output omitted ...>
```

The **nvcc** also supports cross-compilation of CUDA C and C++ code to PowerPC 64-bit Little-Endian (ppc64le).

> **Note:** The support for cross-compilation for POWER8 Little-Endian was introduced in CUDA Toolkit 7.5.

For more information about nvcc compilation process and options, see the following website:

http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html

CUDA libraries

Accompanying the CUDA Toolkit for Linux are some mathematical utility and scientific libraries that uses the GPU to improve performance and scalability:

- `cuBLAS`: Basic Linear Algebra Subroutines
- `cuFFT`: Fast Fourier Transforms
- `cuRAND`: Random Number Generation
- `cuSOLVER`: Dense and Sparse Direct Linear Solvers and Eigen Solvers
- `cuSPARSE`: Sparse matrix routines
- `Thrust`: Parallel Algorithm and data structures

Example 6-18 illustrates the use of the cuBLAS API to calculate the DOT product of vectors. It uses IBM XLC++ (xlC) to generate the host code and compile with `-lcublas` flag that links dynamically with the `cuBLAS` library.

Example 6-18 Sample code for CUDA C using cuBLAS library

```
#include<cublas_v2.h>
#include<cuda_runtime.h>
#include<cuda_runtime_api.h>
#include<stdlib.h>
#include<stdio.h>

#define N 10000

int main(int argc, char* argv[]) {
        int const VEC_SIZE = N*sizeof(double);
```

```
        double* h_vec_A = (double*) malloc(VEC_SIZE);
        double* h_vec_B = (double*) malloc(VEC_SIZE);

        double *d_vec_A, *d_vec_B, result;
        cublasStatus_t status;
        cublasHandle_t handler;
        cudaError_t error;
        // Initialize with random numbers between 0-1
        int i;
        for(i=0; i<N; i++) {
                h_vec_A[i] = (double) (rand() % 100000)/100000.0;
                h_vec_B[i] = (double) (rand() % 100000)/100000.0;
        }
cudaMalloc((void **)&d_vec_A, VEC_SIZE);
        cudaMalloc((void **)&d_vec_B, VEC_SIZE);
        cudaMemcpy(d_vec_A, h_vec_A, VEC_SIZE ,cudaMemcpyHostToDevice);
        cudaMemcpy(d_vec_B, h_vec_B, VEC_SIZE ,cudaMemcpyHostToDevice);

        // Initialize cuBLAS
        status = cublasCreate(&handler);
        // Calculate DOT product
        status = cublasDdot(handler, N , d_vec_A, 1, d_vec_B, 1, &result);
        if(status != CUBLAS_STATUS_SUCCESS) {
                printf("Program failed to calculate DOT product\n");
                return EXIT_FAILURE;
        }
        printf("The DOT product is: %G\n", result);

        // Tear down cuBLAS
        status = cublasDestroy(handler);
        return EXIT_SUCCESS;
}
```

The source code as shown in Example 6-18 on page 188 is built and executed as shown in Example 6-19.

Example 6-19 Build and execute source code an cuBLAS sample code

```
$ make
Building file: ../main.c
Invoking: NVCC Compiler
/usr/local/cuda-7.5/bin/nvcc -I/usr/local/cuda-7.5/include/ -G -g -O0 -ccbin xlC
-Xcompiler -qtune=pwr8 -Xcompiler -qhot -Xcompiler -O3 -Xcompiler -qarch=pwr8
-gencode arch=compute_37,code=sm_37 -m64 -odir "." -M -o "main.d" "../main.c"
/usr/local/cuda-7.5/bin/nvcc -I/usr/local/cuda-7.5/include/ -G -g -O0 -ccbin xlC
-Xcompiler -qtune=pwr8 -Xcompiler -qhot -Xcompiler -O3 -Xcompiler -qarch=pwr8
--compile -m64  -x c -o  "main.o" "../main.c"
Finished building: ../main.c

Building target: cudaBLAS
Invoking: NVCC Linker
/usr/local/cuda-7.5/bin/nvcc --cudart static -ccbin xlC
--relocatable-device-code=false -gencode arch=compute_37,code=compute_37 -gencode
arch=compute_37,code=sm_37 -m64 -link -o  "cudaBLAS"  ./main.o   -lcublas
Finished building target: cudaBLAS
```

```
$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-7.5/lib64/
$ ./cudaBLAS
The DOT product is: 2500.22
```

6.5.3 Hybrid MPI and CUDA programs with IBM Parallel Environment

The IBM Parallel Environment compilers and runtime environment support build and execution of hybrid of MPI and CUDA programs.

Building the program

The common case is to organize sources on separate files for MPI and CUDA codes, and therefore use different compilers to build the objects and then link them using the MPI compiler.

Example 6-20 illustrates this procedure. The MPI source is built with the **mpCC** script, whereas the **nvcc** compiler is used for the CUDA code. Finally, the object files are linked into the executable, along with the CUDA runtime library. Implicitly `mpCC` links the executable to `libmpi` (MPI), `libpami` (PAMI), and `libpoe` (POE).

Example 6-20 Show how hybrid MPI and CUDA programs can be built

```
$ mpCC -o helloMPICuda_mpi.o -c helloMPICuda.cpp
$ nvcc -ccbin g++ -m64 -gencode arch=compute_37,code=sm_37 -o helloMPICuda.o -c
helloMPICuda.cu
$ mpCC -o helloMPICuda helloMPICuda_mpi.o helloMPICuda.o
-L/usr/local/cuda-7.5/lib64 -lcudart
```

It is possible to compile source files that contain MPI and CUDA code mixed (often called spaghetti programming style), although it is not recognized as a good programming practice. In this case, you can compile it invoking the **nvcc** (CUDA C compiler) and setting the MPI library and the headers as follows:

```
$ nvcc -I/opt/ibmhpc/pecurrent/mpich/gnu/include64/
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64/ -L/opt/ibmhpc/pecurrent/base/gnu/lib64/
-lmpi -lpami main.cu
```

The CUDA-aware MPI support

The cuda-aware MPI feature of PE allows direct access of tasks to the GPU memory's buffers on operations of message passing, which can improve the application performance significantly.

Note: The CUDA-aware MPI API was introduced in IBM Parallel Environment version 2.3.

The CUDA development model can be generalized with following steps:

1. Allocate data on the host (CPU) memory.
2. Allocate data on the device (GPU) memory.
3. Move the data from host to device memory.
4. Perform on that data some computation (kernel) on device.
5. Move processed data from device back to the host memory.

In a context of send/receive communication and without the CUDA-aware MPI capability, the task do not have access to the GPU memory. Thus step 5 is required because the data must be in the host memory before it is sent. However, with CUDA-aware MPI, the task will access

to the portion of memory that is allocated on step 2, meaning that data cannot be staged into host memory (step 5 is optional).

> **Note:** By default, CUDA-aware MPI is disabled on IBM PE run time. That behavior can be changed by exporting the MP_CUDA_AWARE environment variable with `yes` (enable) or `no` (disable).

The code in Example 6-21 shows how the CUDA-aware feature can be used within a hybrid of CUDA and MPI programs. It is a simple MPI program meant to run two jobs where task 0 initialize an array, increment its values by one utilizing the GPU computation, then sends the result to task 1. In line 56, the call to the MPI_Send function utilizes the device buffer (allocated on line 40) directly.

Example 6-21 A simple CUDA-aware MPI program

```
 1 #include<cuda_runtime.h>
 2 #include<mpi.h>
 3 #include<stdio.h>
 4 #include<stdlib.h>
 5 #include<assert.h>
 6
 7  __global__ void vecIncKernel(int* vec, int size) {
 8     int tid = blockDim.x * blockIdx.x + threadIdx.x;
 9     if(tid < size) {
10        vec[tid] += 1;
11     }
12  }
13
14 #define ARRAY_ELEM 1024
15 #define ARRAY_ELEM_INIT 555
16
17 int main(int argc, char* argv[]) {
18   int taskid, numtasks, tag=0;
19   MPI_Status status;
20   int array_size = sizeof(int) * ARRAY_ELEM;
21   int threadsPerBlock = 32;
22   int blockSize = ceil(ARRAY_ELEM/threadsPerBlock);
23
24   MPI_Init(&argc, &argv);
25   MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
26   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
27
28   if(numtasks != 2) {
29     printf("This program must run only 2 tasks\n");
30   }
31   /*
32    * Task 0: initialize an array, increment its values by 1,
33    *  and send it to Task 1.
34    */
35   if(taskid == 0) {
36       int *vSend = (int*) malloc(array_size);
37       for(int i=0; i < ARRAY_ELEM; i++)
38         vSend[i]=ARRAY_ELEM_INIT;
39       int *vDev;
40       if(cudaMalloc((void **)&vDev, array_size) == cudaSuccess) {
```

```
41          cudaMemcpy(vDev,vSend, array_size,cudaMemcpyHostToDevice);
42          vecIncKernel<<< blockSize, threadsPerBlock>>>(vDev, ARRAY_ELEM);
43          cudaDeviceSynchronize();
44        } else {
45          printf("Failed to allocate memory on GPU device\n");
46          MPI_Abort(MPI_COMM_WORLD, MPI_ERR_OTHER);
47          exit(0);
48        }
49        if(strcmp(getenv("MP_CUDA_AWARE"),"yes") != 0) {
50            printf("Cuda-aware MPI is disabled, MPI_Send will fail.\n");
51        }
52        /*
53         * CUDA-AWARE MPI Send: using the buffer allocated in GPU device.
54         *  Do not need to transfer data back to host memory.
55         */
56        MPI_Send(vDev, ARRAY_ELEM, MPI_INT, 1, tag, MPI_COMM_WORLD);
57    } else {
58        /*
59         * Task 1: receive array from Task 0 and verify its values
60         *  are incremented by 1.
61         */
62        int *vRecv = (int*) malloc(array_size);
63        MPI_Recv(vRecv, ARRAY_ELEM, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
64        int expected = ARRAY_ELEM_INIT+1;
65        for(int i=0; i < ARRAY_ELEM_INIT; i++) {
66          assert(vRecv[i]==expected);
67        }
68    }
69
70    MPI_Finalize();
71    return 0;
72  }
```

The program in Example 6-21 on page 191 was compiled and executed twice as shown in Example 6-22. The first time enables the CUDA-aware MPI (see the line starting with `MP_CUDA_AWARE=yes`). Afterwards, it executes with the feature disabled (the line starting with `MP_CUDA_AWARE=no`), which makes it exit with a segmentation fault. In this situation, a remediation can be implemented which consists of copying the buffer back to the host memory and use it in the MPI message pass call.

Example 6-22 Compile and run the simple CUDA-aware MPI program

```
$
LD_LIBRARY_PATH=/opt/ibmhpc/pecurrent/mpich/gnu/lib64/:/opt/ibmhpc/pecurrent/base/
gnu/lib64/:$LD_LIBRARY_PATH
$ nvcc -I/opt/ibmhpc/pecurrent/mpich/gnu/include64/
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64/ -L/opt/ibmhpc/pecurrent/base/gnu/lib64/
-lmpi -lpami main.cu
$ MP_CUDA_AWARE=yes MP_RESD=poe MP_PROCS=2 poe ./a.out
$ MP_CUDA_AWARE=no MP_RESD=poe MP_PROCS=2 poe ./a.out
  5 Cuda-aware MPI is disabled, MPI_Send will fail.
  6 ERROR: 0031-250  task 0: Segmentation fault
```

Note: At the time of writing, the NVIDIA GPUDirect technology is not supported on IBM Power Systems. However, the CUDA-aware MPI API of IBM PE implement transparent means to send/receive data to/from the GPU buffers.

As of Parallel Environment 2.3, the CUDA-aware MPI is implemented on these features:

- All two-sided (point-to-point) communication (either blocking and non-blocking and intra- and inter-communicator)
- All blocking intra-communicator collectives

However, some features are not supported on the context of CUDA-aware MPI:

- One-sided communications
- One-sided synchronization calls
- Fabric Collective Accelerator (FCA)
- Non-blocking collective API calls
- Inter-communicator collective API
- The collective selection with the **`pami_tune`** command

Using MPI local rank to balance the use of GPU devices

A policy to grant shared access and load balance to GPU device among local tasks (running on same compute node) can be implemented by using the local rank feature available in the Parallel Environment MPI. For example, one algorithm can limit the number of tasks that use GPUs to avoid oversubscription.

For more infromation about the Parallel Environment local rank, see 6.5.1, “MPI programs with IBM Parallel Environment” on page 180.

Launching concurrent CUDA kernels from multiple tasks

There are some considerations about shared use of GPUs with multiple MPI tasks. This topic is covered in 6.6.2, “CUDA Multi-Process Service” on page 197.

6.5.4 OpenMP programs with the IBM Parallel Environment

The OpenMP applies a shared memory parallel programming model of development. It is a multi-platform directive-based API available to many languages, including C, C++, and Fortran.

Development and execution of OpenMP applications are fully supported by the IBM PE Runtime.

From a compiler perspective, XL C/C++ 13.1.3, and Fortran 15.1.3 provide full support to OpenMP API version 3.1, and partial support to versions 4.0 and 4.5. Similarly, the GNU GCC compiler that is provided by the Linux distribution is based on OpenMP API version 3.1.

6.5.5 OpenSHMEM programs with the IBM Parallel Environment

The OpenSHMEM[2] provides a specification API and reference implementation for the Partitioned Global Address Space (PGAS) parallel programming model, which abstracts the concept of global shared memory on processes distributed across different address spaces. Its implementation involves the use of a communication library that often leverages Remote Direct Memory Access (RDMA) techniques.

[2] Learn more about OpenSHMEM at `http://openshmem.org`

Parallel Environment provides an OpenSHMEM runtime library (libshmem.so), compiler scripts, and tools that enable development and execution of OpenSHEM programs. The PAMI library is used for communication among processing elements (PEs) of the PGAS program, and it is able to take advantage of RDMA capabilities of the InfiniBand interconnect to improve performance.

As of Parallel Environment version 2.3, there is only support for parallel programs that are written in C and C++. Its implementation is based on the OpenSHEM API specification version 1.2[3] with some minor deviations. For a complete list of supported and unsupported routines, see the OpenSHEM section at the following website:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.3.0/com.ibm.cluster.pe.v2r3.pe100.doc/am102_openshmem.htm

The OpenSHMEM tools are installed at `/opt/ibmhpc/pecurrent/base/bin`. The compile scripts for C and C++ are, respectively, **oshcc** and **oshCC**. The **oshrun** script runs programs, although **poe** can be used as well.

Example 6-23 illustrates the OpenSHMEM program. The API can only be used after a shared memory section is initialized (line 8). A symmetric memory area (the basket variable declared on line 6) is written by all processing elements (call to `shmem_int_put` on line 14) on PE zero. Then, all of the elements read the variable from PE zero (call to `shmem_int_get` on line 15) to print its value.

Example 6-23 A simple OpenSHMEM program

```
 1 #include<shmem.h>
 2 #include<stdio.h>
 3 #include<stdlib.h>
 4
 5   int main() {
 6     static int basket; // Global shared (symetric) variable
 7     int cents_p, cents_g; // Processing element's local variable
 8     shmem_init(); // Initialize SHMEM
 9     int my_pe = shmem_my_pe(); // Processing element's number
10     //printf("Hello, I am PE %d\n", my_pe);
11     cents_p = rand()%10;
12     shmem_barrier_all();
13     if(my_pe != 0)
14       shmem_int_put(&basket, &cents_p, 1, 0);
15     shmem_int_get(&cents_g, &basket, 1, 0);
16     printf("Hello, I am PE %d. I put %d cents but I get %d\n", my_pe, cents_p,
cents_g);
17     shmem_finalize(); // Finalize SHMEM
18
19     return 0;
20   }
```

[3] Download the OpenSHEM specification version 1.2 document at http://openshmem.org/site/sites/default/site_files/openshmem-specification-1.2.pdf

The source code in Example 6-23 on page 194 can be compiled with the **oshcc** script as shown in Example 6-24. It is a common practice to name an OpenSHMEM executable with the suffix *.x*.

Example 6-24 Show how to compile an OpenSHMEM program

```
$ oshcc -O3 hellosh.c -o hellosh.x
$ ls
hellosh.c  hellosh.x
```

By default **oshcc** (or **oshCC**) compiles the application with **xlc** (or **xlC**) unless it is not installed into the system or the MP_COMPILER environment variable is set to use **gcc** (or **g++**).

The program is launched with the **oshrun** script. Alternatively, it can evoke **poe** directly. For more information about execution of OpenSHMEM programs, see 7.2.3, “Running OpenSHMEM programs” on page 239.

6.5.6 Parallel Active Messaging Interface programs

Parallel Active Messaging Interface (PAMI) is a low-level protocol that is the foundation of communications on MPI and OpenSHMEM implementations of the IBM PE Runtime. It provides an API for programs to access PAMI capabilities, such as collective communications and RDMA using InfiniBand Host Channel Adapters (HCAs).

Many sample codes to demonstrate PAMI subroutines are included with the IBM PE Runtime and are available in the `/opt/ibmhpc/pecurrent/ppe.samples/pami` folder. Example 6-25 shows how to build the PAMI samples and run a program (`alltoall.exe`) that uses the all to all communication functions.

Example 6-25 Show how to build and run PAMI sample codes

```
$ cp -r /opt/ibmhpc/pecurrent/ppe.samples/pami .
$ cd pami/
$ make
$ cd pami_samples
$ vi host.list
$ MP_RESD=poe MP_PROCS=4 ./coll/alltoall.exe
# Context: 0
# Alltoall Bandwidth Test(size:4) 0x1000e400, protocol: I1:Alltoall:P2P:P2P
# Size(bytes)  iterations    bytes/sec      usec
# -----------      -----------    -----------    ---------
            1              100       413223.1         2.42
            2              100       840336.1         2.38
            4              100      1659751.0         2.41
            8              100      3347280.3         2.39
           16              100      6722689.1         2.38
           32              100     13502109.7         2.37
           64              100     26778242.7         2.39
          128              100     53112033.2         2.41
          256              100     81012658.2         3.16
          512              100    171812080.5         2.98
         1024              100    320000000.0         3.20
         2048              100    552021563.3         3.71
         4096              100    869639065.8         4.71
```

Check out the *Parallel Environment Runtime Edition for Linux: PAMI Programming Guide* available at following website for more details about this topic:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.3.0/com.ibm.cluster.protocols.v2r3.pp400.doc/bl510_about.htm?cp=SSFK3V_2.3.0%2F0-0-5

6.6 GPU tuning

This section explains how to overcome performance degradation due to the Power Cap Limit featured by the NVIDIA GPU. Also, it shows how to manage shared access to GPUs by multi-process applications through Multi-Process Service (MPS).

6.6.1 Power Cap Limit

While running your applications, the power cap limit can exceed the Software Power Cap Limit of the NVIDIA GPU cards. If this happens, performance will be degraded because the frequency of the GPU clock will be reduced because the GPU is consuming too much power.

To adjust the Power Cap Limit and check how performance looks after this adjustment, you need to make following checks:

1. Check the current, default and maximum power limits:

   ```
   nvidia-smi -q | grep 'Power Limit'
   ```

   Example 6-26 contains output for the example NVIDIA Tesla K80 after execution of this command.

   Example 6-26 NVIDIA Tesla K80 Power limits by default

   ```
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   ```

2. Set persistence to the following settings:

   ```
   nvidia-smi -pm 1
   ```

3. Increase the Power Cap limit to, for example, to 175 watts:

   ```
   nvidia-smi -pl 175
   ```

You can check again the limits after these three steps have been completed by using the command from the first step as shown in Example 6-27.

Example 6-27 NVIDIA Tesla K80 Power limits after changes

```
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
```

Example 6-4 on page 164 and Example 6-5 on page 164 show runs of sample ESSL SMP CUDA program with the default and the adjusted Power Cap Limit respectively. This technique gives performance improvements of about 2.5 times for this case.

Note: You need to set the Power Cap Limit and persistence after each system reboot.

6.6.2 CUDA Multi-Process Service

MPS is client-server runtime implementation of the CUDA API that helps multiple CUDA processes to share GPUs. MPS takes advantage of parallelism between MPI tasks to improve GPU utilization.

MPS contains following components:

- Control Daemon Process: Coordinates connections between servers and clients, starts and stops server.
- Client Runtime: Any CUDA application can use the MPS client runtime from the CUDA driver library.
- Server Process: This is shared clients’ shared connection to the GPU and provides concurrency between clients.

MPS is recommended to use for jobs that do not generate enough work for GPUs. If you have multiple runs at the same time, MPS helps to balance the workload of the GPU and increases its utilization.

Also MPI programs that have different MPI-tasks but share the same GPU can get improvement of performance by using MPS to control access to the GPU between tasks.

Note: The MPS Server supports up to 16 client CUDA contexts, and these contexts can be distributed over up to 16 processes.

A recommendation is to use MPS with the EXCLUSIVE_PROCESS mode to be sure that only the MPS server runs on GPUs. Other compute modes, described in 8.2.3, “Compute modes” on page 261, are not recommended or unsupported.

The CUDA program works using MPS if the MPS control daemon already runs in the system. The program at startup tries to connect to the MPS control daemon, which creates an MPS server or reuses an existing one if it has the same user ID as the user who launches the job. Therefore, each user has its own MPS server. The MPS server creates the shared GPU context for the different jobs in the system, manages its clients, and calls to GPU on behalf of them. Figure 6-2 shows the overview of this process.

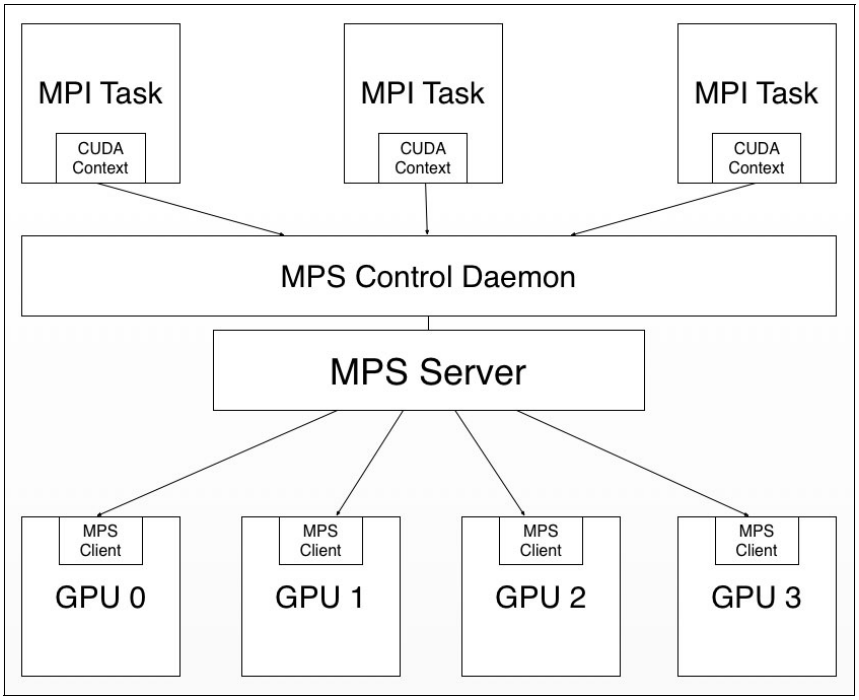

Figure 6-2 NVIDIA Multi-Process Service

To run MPS, you need to run the following commands, as root:

1. Set environment variable *CUDA_VISIBLE_DEVICES* to inform the system which GPUs will be used. This step is optional. To use all GPUs start from the second step, issue this command:

   ```
   export CUDA_VISIBLE_DEVICES=0,1 #Use only first and second GPUs
   ```

2. Change the compute mode for all GPUs or to specific ones, chosen in the first step:

   ```
   nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
   nvidia-smi -i 1 -c EXCLUSIVE_PROCESS
   ```

3. Start the daemon:

   ```
   nvidia-cuda-mps-control -d
   ```

To stop the MPS daemon, execute following command as root:

```
echo quit | nvidia-cuda-mps-control
```

Example 6-28 shows the output after the execution starts and stops steps in the example system.

Example 6-28 Start and stop commands of the MPS for 4 GPUs

```
$ nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0000:03:00.0.
All done.
$ nvidia-smi -i 1 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0000:04:00.0.
All done.
$ nvidia-smi -i 2 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0002:03:00.0.
All done.
$ nvidia-smi -i 3 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0002:04:00.0.
All done.
$ nvidia-cuda-mps-control -d
$ echo quit | nvidia-cuda-mps-control
```

If the system has running jobs, the **nvidia-smi** command provides output similar to that shown in Example 6-29.

Example 6-29 nvidia-smi output for system with MPS

```
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0     23715    C   nvidia-cuda-mps-server                         89MiB |
|    1     23715    C   nvidia-cuda-mps-server                         89MiB |
|    2     23715    C   nvidia-cuda-mps-server                         89MiB |
|    3     23715    C   nvidia-cuda-mps-server                         89MiB |
+-----------------------------------------------------------------------------+
```

For more information about the CUDA MPS, see the following website:

https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

6.7 Tools for development and tuning of applications

Many tools for parallel program development and tuning are available for the cluster environment described in this book. Because different development models are allowed, more than one tool is usually needed to develop the application.

This section covers some tools that have been developed by IBM, and others that are useful tools but are maintained by third-party companies or open source communities.

Table 6-12 gives a brief description of the tools and development models.

Table 6-12 Summary of tools for development and tuning of parallel applications

Development model	Provider	Tool
MPI	Parallel Environment (PE) Developer Edition (DE)	Call graph analysis
		I/O Profiling (MIO)
		MPI Profiling
		Hardware Performance Monitor (HPM)
		PDB
Hybrid of MPI and OpenMP	Parallel Environment (PE) Developer Edition (DE)	OpenMP profiling
CUDA	CUDA Toolkit	CUDA-memcheck
		CUDA Debugger
		nvprof
		Nsight Eclipse Edition
Hybrid of MPI and CUDA	Parallel Environment (PE) Developer Edition (DE)	GPU Performance Monitor (GPU)
MPI, OpenMP, PAMI, OpenSHMEM	Eclipse Parallel Tools Platform (PTP)	Integrated Development Environment (IDE)

6.7.1 The Parallel Environment Developer Edition

The PE DE provides a set of tools (see Table 6-12) and libraries to help during the development of applications written in C, C++ or Fortran by using pure MPI or hybrid with OpenMP and CUDA. This encompassed two main components:

- HPC Toolkit: Provides back-end runtime and development libraries and command-line tools
- hpctView: Provides a graphical user interface (GUI) front end to the HPC Toolkit. Additionally, hpctView plug-ins to Eclipse for Parallel Application Developers (PTP) are available.

Note: Since version 2.2, PE DE does not provide the workbench component. Instead, there is the hpctView plug-ins to the Eclipse for Parallel Application Developers. For more information, see 6.7.3, “Eclipse for Parallel Application Developers” on page 218.

The fluxogram as shown in Figure 6-3 on page 201 describes the analysis cycle of a parallel application by using the IBM PE DE. As general rule, it evolves in three phases:

- Instrument: An application executable is designed to use a certain tool of the HPC Toolkit.
- Profile: Run the application to record execution data.
- Visualize: Visualize the resulting profile by using either command-line or GUI (hpctView) tool.

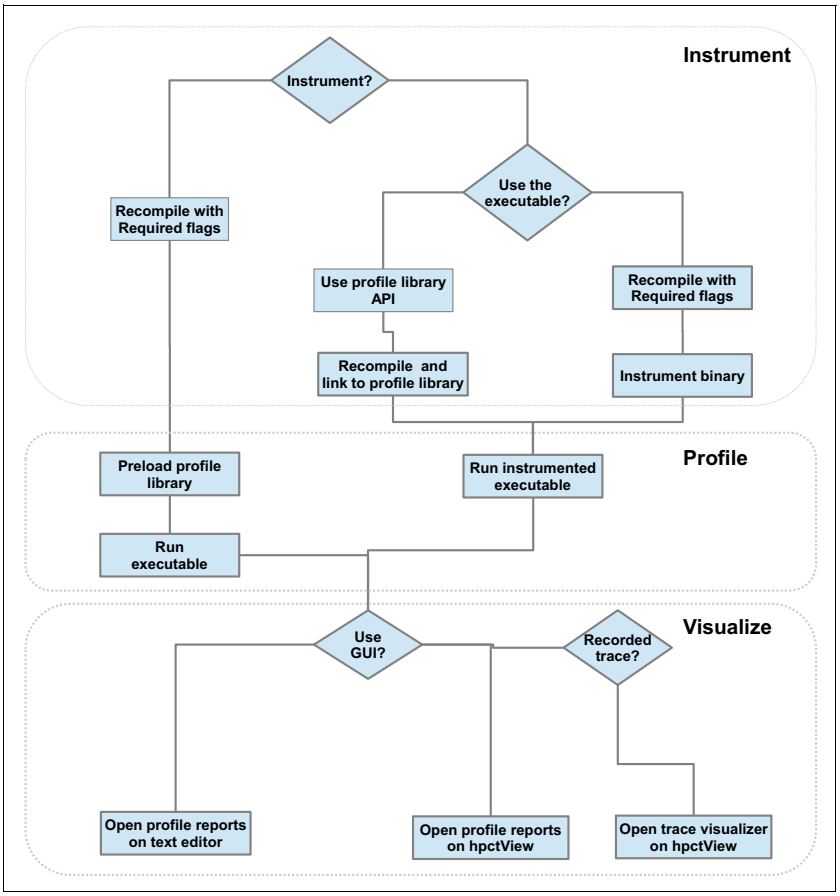

Figure 6-3 Fluxogram shows how to use IBM Parallel Environment (PE) Developer Edition (DE)

Instrumentation is an optional step for some tools. If you need fine-grained collection of runtime information, use any of the instrument mechanisms provided:

- Binary instrumentation: Uses the **`hpctInst`** command or hpcView (GUI) to place probes on the executable file. It does not require you to rebuild the application
- Profiling library API: Places calls to library methods to control profiling at the source level. It does require you to recompile the application and link it to the profiling library

The profile phase is simply to run an instrumented executable with a relevant work load so that useful data is collected. For situations where the binary is not iinstrumented, it requires you to preload at run time the corresponding profiling tool library and use of environment variables to control the behavior of the library. The library then takes over some method calls to produce the profile data.

In order to visualize the data produced by the profiling tools, you can use either hpctView for the Graphical visualization or Linux commands like **`cat`** to inspect the report files.

Using hpctView to carry on the instrument-profile-visualize analysis cycle is a convenient method because it permits fine-grained instrumentation at file, function, MPI call, or region levels. It also provides other means to profile the parallel application, and easy visualization of profile data and graphical tracing.

The PE DE tools are briefly introduced in the next sections, which show the many ways to profile and trace a parallel application. For more information about those topics, see the guide to getting started at the following website and select **Cluster software** → **Parallel Environment Developer Edition** → **Parallel Environment Developer Edition 2.2.0**:

http://www.ibm.com/support/knowledgecenter

Call graph analysis tool

The call graph analysis tool in the hpctView is used to visualize GNU `gprof` files of MPI applications. When `gprof` profiling data is collected with parallel SPMD programs, one file per task is created, which can make it difficult to visualize the information.

To use the call graph tool, you need to compile the application with the **`-pg`** compiler flag, then run the application to produce the profile data. It is important to use a workload that is as close as possible to the production scenario so that good profile data can be generated.

The following shows an example on how to compile and link a Fortran MPI application:

```
$ mpfort -g -c  -O bcast.f
$ mpfort -o bcast -pg -g bcast.o
```

One `gmon.out` profile file is build per task.

> **Note:** A bug in the GNU Library C (`glibc`) that is installed by default with Red Hat Enterprise Linux 7.2 prevents `gmon.out` files from being generated. This problem can be solved by installing the latest `glibc` package version by way of the Red Hat Extended Update Support (EUS), also known as *z-stream*.
>
> For more information about the problem and solution, see the following website:
>
> http://bugzilla.redhat.com/show_bug.cgi?id=1249102

Along with the call graph methods, the tool provides information about these statistics:

- Total execution time (and percentage) spent on each method
- Number of times each method was called
- Average time per call that is spent on each method

This information comes handy to identify hotspots so you can target performance optimizations on specific methods.

Follow these steps to load the `gmon.out` files into hpctView:

1. Import the executable by clicking **File** → **Open Executable**.
2. On the menu bar, select **File** → **Load Call Graph (gmon.out).**
3. In the Select gmon.out Files window, browse the remote file system to select the `gmon.out` files that you want to load, and click **OK**.

The hpctView callgraph visualizer opens as shown in Figure 6-4.

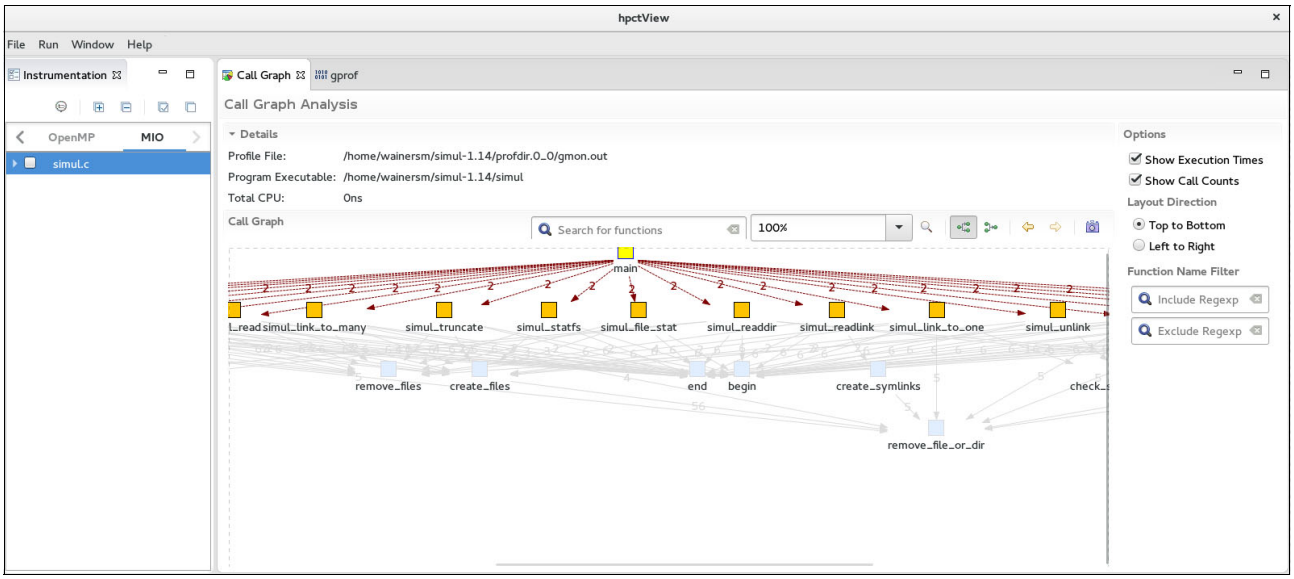

Figure 6-4 hpctView with example gprof data loaded on callgraph tool

In hpctView, you can choose to display the data from the gprof files in many ways in the information recorder as shown in Figure 6-5.

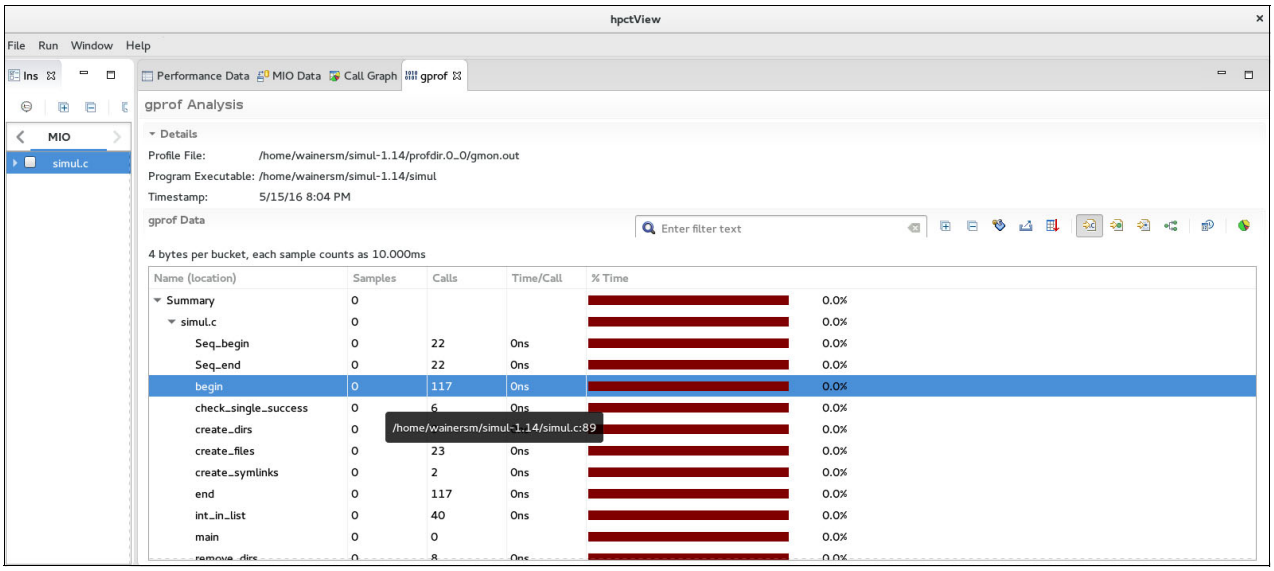

Figure 6-5 hpctView with an example of gprof loaded

MPI profiling and tracing

The MPI profiling and tracing tool obtains data about the usage of the MPI routines in parallel programs. The following are the profiles and reporting information (per tasks):

- Number of times that MPI routines are executed
- Total wall time that is spent in each MPI routine
- Average transferred bytes on message passing routines
- Point out the tasks with minimal, median, and maximum communication time. This information is provided only in the report of task 0.

You can use the profile and trace data to do many kinds of analysis, such as communication patterns, hotspots, and data movements.

Example 6-30 shows the textual report for task 0 of an MPI program. The last section of the report (**Communication summary for all tasks**) shows that the minimum, median, and maximum communication time are spent on tasks 7, 4, and 5, respectively.

Example 6-30 HPC Toolkit - MPI profiling report for task 0

```
$ cat hpct_0_0.mpi.txt
-----------------------------------------------------------------
MPI Routine                     #calls   avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                        1          0.0          0.000
MPI_Comm_rank                        1          0.0          0.000
MPI_Bcast                            1          4.0          0.000
MPI_Barrier                          1          0.0          0.000
MPI_Allreduce                        4         26.0          0.000
-----------------------------------------------------------------
total communication time = 0.000 seconds.
total elapsed time       = 2.725 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Bcast                       #calls   avg. bytes      time(sec)
                                     1          4.0          0.000

MPI_Allreduce                   #calls   avg. bytes      time(sec)
                                     3          8.0          0.000
                                     1         80.0          0.000

-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 0.000 sec for task 7
  median  communication time = 0.001 sec for task 4
  maximum communication time = 0.002 sec for task 5
```

Example 6-31 lists the files that are generated by the tool. Notice that although the parallel job used in this example has eight tasks in total, only the reports of tasks 0, 4, 5, and 7 are available. The MPI profiling tool generates by default reports for the four most significant tasks according to the communication time criteria: Task 0 (an aggregate of all tasks) and tasks with minimum, median, and maximum communication time.

Example 6-31 Listing the files generated by the tool

```
$ ls
hpct_0_0.mpi.mpt  hpct_0_0.mpi.txt  hpct_0_0.mpi.viz  hpct_0_4.mpi.txt
hpct_0_4.mpi.viz  hpct_0_5.mpi.txt  hpct_0_5.mpi.viz  hpct_0_7.mpi.txt
hpct_0_7.mpi.viz
```

Still taking as example the same parallel job, the task with maximum communication time is 5. The content of the profiling of task 5 is displayed in Example 6-32.

Example 6-32 Contents of the profiling tasks

```
$ cat hpct_0_5.mpi.txt
-----------------------------------------------------------------
MPI Routine                     #calls   avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                        1          0.0          0.000
MPI_Comm_rank                        1          0.0          0.000
MPI_Bcast                            1          4.0          0.000
MPI_Barrier                          1          0.0          0.000
MPI_Allreduce                        4         26.0          0.001
-----------------------------------------------------------------
total communication time = 0.002 seconds.
total elapsed time       = 2.725 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Bcast                       #calls   avg. bytes      time(sec)
                                     1          4.0          0.000

MPI_Allreduce                   #calls   avg. bytes      time(sec)
                                     3          8.0          0.001
                                     1         80.0          0.000
```

Along with the profile data, the tool records MPI routines calls over time that can be used within hpctView trace visualizer. This information is useful to analyze the communication patterns within the parallel program.

The easiest way to profile your MPI application is to load the trace library before the execution by exporting the LD_PRELOAD environment variable. However, it can produce too much data from large programs. Using the hpctView instrumentation-run-visualize analysis cycle from within hpctView is also a convenient way to profile the application as it permits fine-grained instrumentation at levels of files, functions, MPI routines, or just code regions.

The examples showed in this section were generated by using the script shown in Example 6-33. To use the MPI profile preload library, the parallel program can be compiled with the **-g -Wl**,**--hash-style=sysv -emit-stub-syms** flags.

Example 6-33 Script to profile MPI with HPC Toolkit

```
01 #!/bin/bash
02 #
03 # Use it as: $ MP_PROCS=<num> poe ./hpct_mpiprofile.sh <app> <args>
04 #
05
06 . /opt/ibmhpc/ppedev.hpct/env_sh
07
08 #
09 # HPCT MPI Trace control variables
10 #
11
12 ## Uncomment to set maximum limit of events traced.
13 ## Default is 30000.
```

```
14 #MAX_TRACE_EVENTS=
15
16 ## Uncomment to generate traces for all ranks.
17 ## Default are 4 tasks: task 0 and tasks with maximum, minimum and median
communication time.
18 #OUTPUT_ALL_RANKS=yes
19
20 ## Uncomment to enable tracing of all tasks.
21 ## Default are tasks from 0 to 255 ranks.
22 #TRACE_ALL_TASKS=yes
23
24 ## Uncomment to set maximum limit of rank traced.
25 ## Default are 0-255 ranks or all if TRACE_ALL_TASKS is set.
26 #MAX_TRACE_RANK=
27
28 ## Uncomment to set desired trace back level. Zero is level where MPI function
is called.
29 ## Default is the immediate MPI function's caller.
30 #TRACEBACK_LEVEL=
31
32 # Preload MPI Profile library
33 LD_PRELOAD=/opt/ibmhpc/ppedev.hpct/lib64/preload/libmpitrace.so
34
35 $@
```

MPI I/O profiling

The MPI I/O (MIO) tool records information about Linux I/O system calls (syscalls) carried out by the parallel program while accessing and manipulating files. The following statistics are calculated per file:

- Number of times each I/O syscall
- Total time spent on each I/O syscall

Depending on the I/O syscall, it also profiles these statistics:

- Total of bytes requested
- Total of bytes delivered
- Minimum requested size in bytes
- Maximum requested size in bytes
- Rate in bytes
- Suspend wait count
- Suspend wait time
- Forward seeks average
- Backward seeks average

For your MIO tool, you can compile the application with the **-g -Wl,--hash-style=sysv -Wl,--emit-stub-syms** flags that prepare the binary for instrumentation.

Example 6-34 shows the **hpctInst** command used to instrument all I/O calls of an application called simul. Next, the application is executed by using the MIO_DEFAULTS and MIO_FILES environment variables to set the MIO trace modules. As a result, MIO generates trace and statistics data on a set of files withthe *mio* pattern in their names on a per-task basis.

Example 6-34 Instrument and profile an application with HPC Toolkit MIO tool

```
$ . /opt/ibmhpc/ppedev.hpct/env_sh
$hpctInst -dmio simul
```

```
$ MP_RESD=poe MP_PROCS=4 MIO_FILES="*[trace/xml/events]"
MIO_DEFAULTS="trace/stats=miostats/mbytes" ./simul.inst -d ./iodir
$ ls hpct*
hpct_0_0.mio.iot           hpct_0_0.mio.viz           hpct_0_1.mio.txt
hpct_0_2.mio.miostats.txt  hpct_0_3.mio.iot           hpct_0_3.mio.viz
hpct_0_0.mio.miostats.txt  hpct_0_1.mio.iot           hpct_0_1.mio.viz
hpct_0_2.mio.txt           hpct_0_3.mio.miostats.txt
hpct_0_0.mio.txt           hpct_0_1.mio.miostats.txt  hpct_0_2.mio.iot
hpct_0_2.mio.viz           hpct_0_3.mio.txt
```

Example 6-34 shows the `MIO_DEFAULTS` and `MIO_FILES` environment variables that are used to control the behavior of the profiling tool. Those are the minimum required variables to manage MIO tool, but others are available.

Follow these steps to load MIO files into hpctView:

1. Import the executable by clicking **File** → **Open Executable**.
2. On menu bar, click **File** → **Load MIO Trace (*.iot)**.
3. In the Select Files window, browse the remote file system and select files with `*.iot` pattern in their names that you want to load. Click **OK**.
4. Likewise, go to the menu bar and click **File** → **Load Viz Files (*.viz)** to open the visualization files that are generated by the MIO tool.

Figure 6-6 displays the hpctView visualization mode for the data generated with the MIO tool. Detailed information about each I/O system call is provided on a per-task basis.

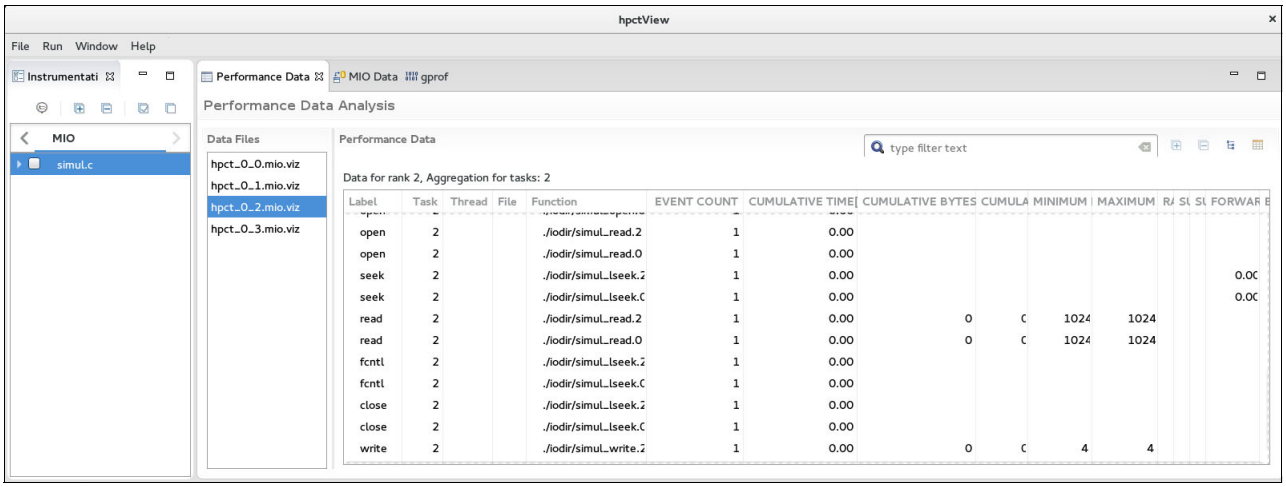

Figure 6-6 hpctView: MPI I/O tool profiling visualization

Another view of the hpctView that displays the I/O trace data is shown in Figure 6-7.

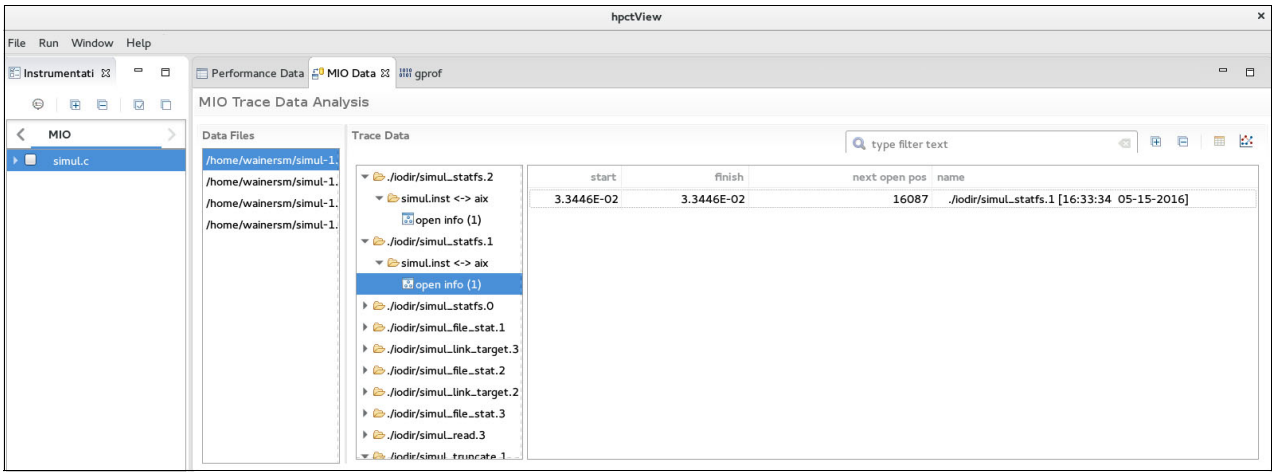

Figure 6-7 hpctView: MPI I/O tool trace visualization

Hardware Performance Monitor

The Processor Monitor Unit (PMU) of POWER8 is a part of the processor dedicated to record hardware events. It has six Performance Monitor Counter (PMC) registers. Registers 0-3 can be programmed to count any of the more than one thousand of events available, 4 can count run instructions completed, and 5 can count run cycles.

These events are important because they can reveal performance issues like pipeline bubbles, inefficient use of caches, and high ratio of branches misprediction from the perspective of the processor. Metrics for performance measurements can also be calculated from hardware events, such as instructions per cycle, million of instructions per second (MIPS), and memory bandwidth.

For single programs, tools such as Oprofile and Perf provide access to either system-wide or application profiling of those hardware events on POWER8. Parallel SPMD programs are often difficult to profile with these traditional tools because they are designed to deal with a single process (whether multi-threaded or not).

For more information about Oprofile, see the Oprofile tool website at:

http://oprofile.sourceforge.net

For more information about Oprofile, see the Perf tool wiki page at:

http://perf.wiki.kernel.org

The Hardware Performance Monitor (HPM) tool is an analog tool, but provides easy SPMD programs profiling on Linux on Power Systems.With HPM, you can profile MPI programs regarding any of the hardware events available or obtain any of the 31 predefined metrics that are most commonly used in performance analysis. The complete list of predefine metrics can be found at the following website:

http://www.ibm.com/support/knowledgecenter/SSFK5S_2.2.0/com.ibm.cluster.pedev.v2r2.pedev100.doc/bl7ug_derivedmetrics.htm

Figure 6-8 shows an HPM report opened with hpctView.

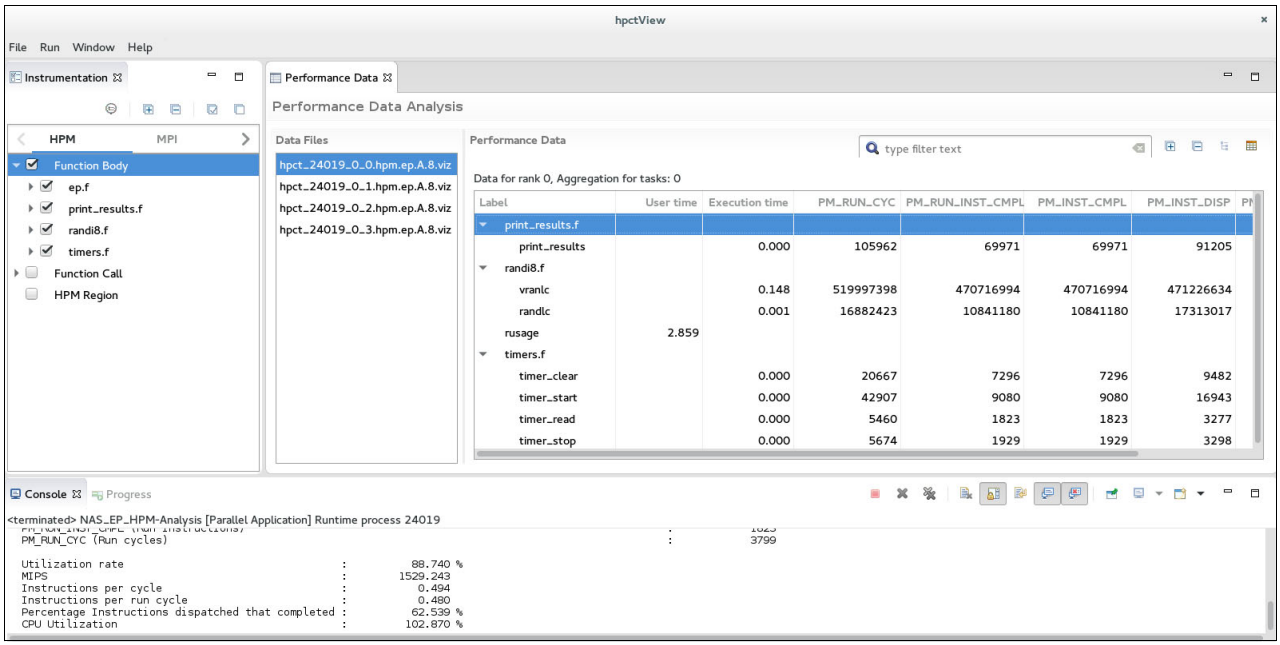

Figure 6-8 hpctView: HPC tool visualization

GPU Performance Monitoring

The GPU Performance Monitoring (GPM) tool is designed to profile and trace hardware events that are produced by the GPU on hybrid MPI with CUDA C programs.

> **Note:** GPU was introduced in PE DE 2.2.

The HPM tool can profile raw hardware events and obtain metrics from a predefined list. The **`gpmlist`** command is the interface to fetch the list of events and metrics supported. Example 6-35 shows a sample of the list of events available for the NVIDIA K80 GPU.

Example 6-35 HPC Toolkit - how to list events available to GPM profile tool

```
$ /opt/ibmhpc/ppedev.hpct/bin/gpmlist -d k80 -e -l
  Domain 0
    2082 tex0_cache_sector_queries - tex0 cache sector queries
    2083 tex1_cache_sector_queries - tex1 cache sector queries
    2084 tex2_cache_sector_queries - tex2 cache sector queries
    2085 tex3_cache_sector_queries - tex3 cache sector queries
    2086 tex0_cache_sector_misses - tex0 cache sector misses
    2087 tex1_cache_sector_misses - tex1 cache sector misses
    2088 tex2_cache_sector_misses - tex2 cache sector misses
<... Output Omitted ...>
Domain 2
    2401 gld_inst_8bit - gld inst 8bit
    2402 gld_inst_16bit - gld inst 16bit
    2403 gld_inst_32bit - gld inst 32bit
    2404 gld_inst_64bit - gld inst 64bit
    2405 gld_inst_128bit - gld inst 128bit
<... Output Omitted ...>
  Domain 3
    2601 prof_trigger_00 - prof trigger 00
```

```
    2602 prof_trigger_01 - prof trigger 01
    2603 prof_trigger_02 - prof trigger 02
    2604 prof_trigger_03 - prof trigger 03
    2605 prof_trigger_04 - prof trigger 04
    2606 prof_trigger_05 - prof trigger 05
    2607 prof_trigger_06 - prof trigger 06
    2608 prof_trigger_07 - prof trigger 07
    2617 warps_launched - warps launched
    2618 threads_launched - threads launched
<... Output Omitted ...>
```

Example 6-36 shows a sample of the metrics available for the NVIDIA K80 GPU.

Example 6-36 HPC Toolkit - how to list metrics available to GPM profile tool

```
$ /opt/ibmhpc/ppedev.hpct/bin/gpmlist -d k80 -m -l
    1201 l1_cache_global_hit_rate - L1 Global Hit Rate
        2645 l1_global_load_hit - l1 global load hit
        2646 l1_global_load_miss - l1 global load miss
    1202 l1_cache_local_hit_rate - L1 Local Hit Rate
        2641 l1_local_load_hit - l1 local load hit
        2643 l1_local_store_hit - l1 local store hit
        2642 l1_local_load_miss - l1 local load miss
        2644 l1_local_store_miss - l1 local store miss
    1203 sm_efficiency - Multiprocessor Activity
        2629 active_cycles - active cycles
        2193 elapsed_cycles_sm - elapsed_cycles_sm
<... Output Omitted ...>
```

As with other HPC Toolkit tools, GPM is also flexible regarding the instrument-profile-visualize cycle possibilities. It can also be used together with the HPM tool.

The following environment variables are used to configure the profiler:

- `GPM_METRIC_SET`: Use to set the metrics to be profiled.
- `GPM_EVENT_SET`: Use to set the hardware events to be profiled. Cannot be exported together with `GPM_METRIC_SET`.
- `GPM_VIZ_OUTPUT=y`: Enables creation of visualization files that are used by hpctView visualizer. By default it is disabled.
- `GPM_STDOUT=n`: Suppresses the profiler messages to standard output (stdout). By default it sends messages to `stdout`.
- `GPM_ENABLE_TRACE=y`: Turns on trace mode. By default, it is off.

Example 6-37 demonstrates a profiling session of a hybrid MPI and CUDA C application named `a.out`. The lines ranging from 1-26 are the content of the `gpm.sh` script, which exports the GPM control variables (lines 13 -15) which instruct the system to calculate the `sm_efficiency` metric, then evoke the wrapper called `gpm_wrap.sh`. In turn the wrapper (lines 29-35) script preloads (line 33) the GPM library. The remained lines are messages that are printed by GPM on standard output.

Example 6-37 HPC Toolkit - profiling with GPU Performance Monitor tool

```
  1 $ cat gpm.sh
  2
  3 #!/bin/bash
```

```
  4
  5
  6
  7 export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-7.5/lib64:/opt/ibmhpc/pecurrent/m
pich/gnu/lib64:/opt/ibmhpc/pecurrent/gnu/lib64/
  8
  9
 10
 11 # GPU Performance Monitor - variables
 12
 13 export GPM_METRIC_SET=sm_efficiency
 14
 15 export GPM_VIZ_OUTPUT=y
 16
 17
 18
 19 export MP_CUDA_AWARE=yes
 20
 21 export MP_RESD=poe
 22
 23 export MP_PROCS=2
 24
 25 poe ./gpm_wrap.sh
 26
 27 $ cat gpm_wrap.sh
 28
 29 #!/bin/bash
 30
 31
 32
 33 export LD_PRELOAD=/opt/ibmhpc/ppedev.hpct/lib64/preload/libgpm.so:$LD_PRELOAD
 34
 35 ./a.out
 36
 37 $ ./gpm.sh
 38
 39
 40
 41  GPM (IBM HPC Toolkit for PE Developer Edition) results
 42
 43
 44
 45 Device 0 (Tesla K80):
 46
 47 ------------------------------------------------
 48
 49   Symbol Name: cudaMalloc
 50
 51   ----------------------------------------------
 52
 53              Memory Kind: cudaMalloc
 54
 55             Bytes Copied: 4096
 56
```

```
 57
 58
 59   Symbol Name: cudaMemcpy
 60
 61   ----------------------------------------------
 62
 63              Memory Kind: cudaMemcpyHostToDevice
 64
 65             Bytes Copied: 4096
 66
 67
 68
 69   Kernel Name: _Z12vecIncKernelPii
 70
 71   ---------------------------------------------
 72
 73          GPU Kernel Time: 0.000005 seconds
 74
 75             W/clock Time: 0.012338 seconds
 76
 77            sm_efficiency: 31.155739%
 78
 79
 80
 81   Symbol Name: cudaMemcpy
 82
 83   ----------------------------------------------
 84
 85              Memory Kind: cudaMemcpyDeviceToHost
 86
 87             Bytes Copied: 4096
 88
 89
 90
 91   Totals for device 0
 92
 93   ----------------------------------------------
 94
 95           Total Memory Copied: 8192 bytes
 96
 97         Total GPU Kernel Time: 0.000005 seconds
 98
 99            Total W/clock Time: 0.012338 seconds
100
101                sm_efficiency: 31.155739%
102
```

As a result, the tool reports the calculated metric or counted event on per-task files. Example 6-38 shows the report that is generated for execution in Example 6-37 on page 210.

Example 6-38 HPC Toolkit - report generate by GPU Performance Monitoring tool

```
$ cat hpct_0_0.gpm.a.out.txt

 GPM (IBM HPC Toolkit for PE Developer Edition) results
```

```
Totals for device 0
---------------------------------------------
        Total Memory Copied: 8192 bytes
      Total GPU Kernel Time: 0.000005 seconds
         Total W/clock Time: 0.012338 seconds
              sm_efficiency: 31.155739%
```

For more details about GPM tool, see the following website:

http://www.ibm.com/support/knowledgecenter/SSFK5S_2.2.0/com.ibm.cluster.pedev.v2r2.pedev100.doc/bl7ug_gpuhwcounters.htm

IBM HPC Toolkit hpctView

The IBM HPCView is a front end for the HPC Toolkit runtime and command-line instrumentation tools. It provides a GUI to instrument the parallel executable file,which you can run to collect data and visualize information from the developer workstation.

The tool can be run from the developer workstation. Version 2.2 is supported on Mac OS 10.9 (Mavericks) or later, Microsoft Windows 64 bit, and any Linux distribution.

Find the hpctView as a tarball file distributed together with PE DE. Then proceed as follows to install it and run it:

```
$ tar xvzf hpctView-2.2.0-0-linux-gtk-x86_64.tar.gz
$ cd hpctView
$ ./hpctView &
```

Profile and trace files that are generated by using the HPC Toolkit command-line tools and libraries can be loaded into hpctView for visualization. Or you can execute a complete cycle of instrument-run-visualize from within it.

To instrument a compiled parallel executable, complete these steps:

1. In the **Instrumentation** pane (left side pane), select the profiler that you want to instrument the binary for. The options are **HPM**, **MPI**, **OpenMP**, and **MIO.**
2. Click **File** → **Open Executable**. This action opens a window with a connection to the remote machine. Select the file to be instrumented as shown Figure 6-9.

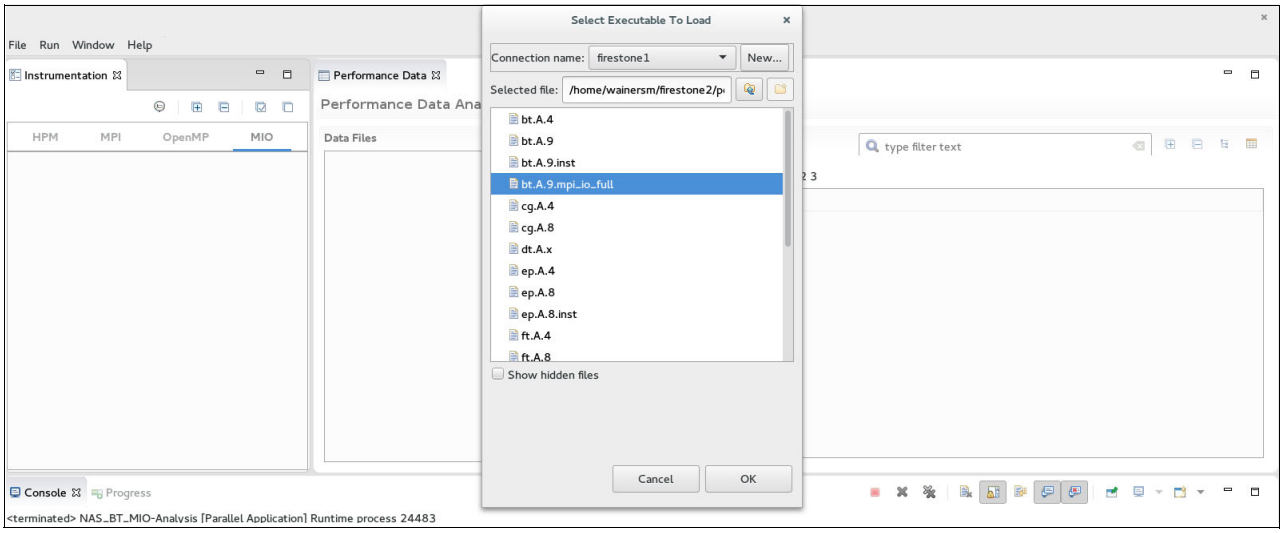

Figure 6-9 hpctView: Select binary for instrumentation

3. Click the **New** button to create a connection to the remote system where the binary is hosted if it does not already exist.
4. Select the binary, then click **OK.**
5. The binary content will be analyzed and sections corresponding to the instrumentation portions of the application code are listed in tree format in the Instrumentation pane. For each type of profiling tool, different instrumentation options are displayed. Figure 6-10, for example, shows the options (**Function body**, **Function Call**, **HPM Region**) for the instrumentation of a binary to be used with the HPM tool. Select the instrumentation points according to your needs.

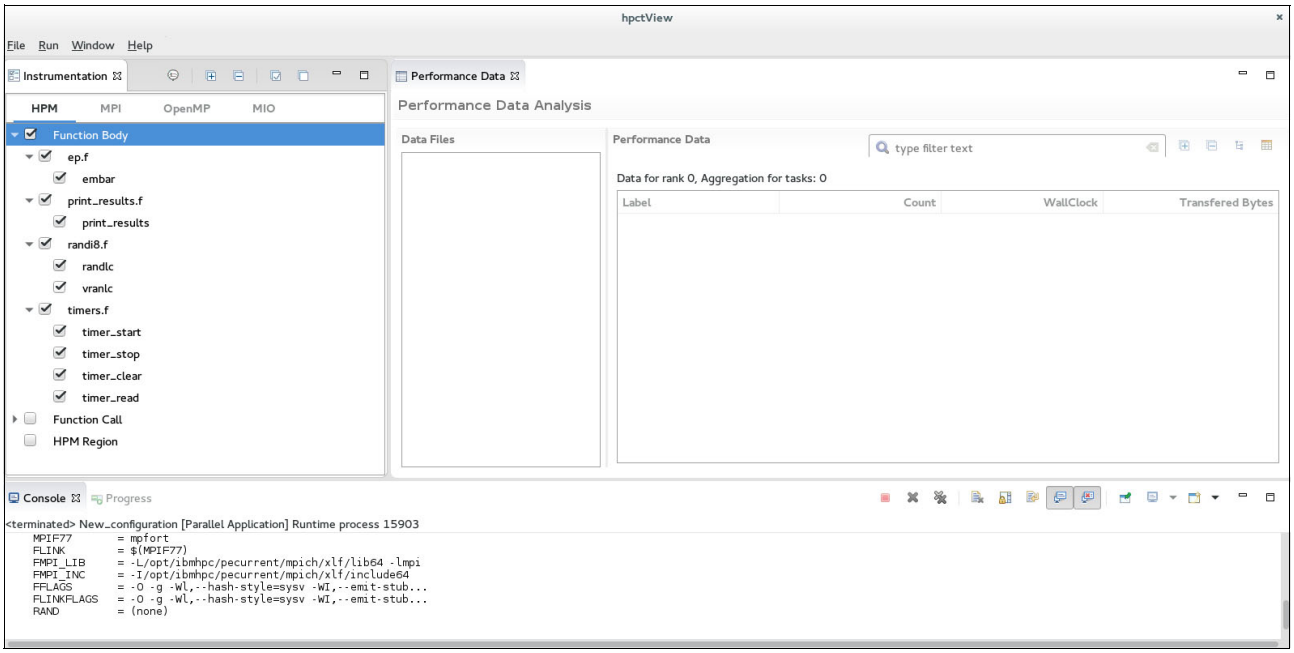

Figure 6-10 hpctView: Binary instrumentation

6. Click the instrument executable icon in the Instrumentation pane. If the operation succeeds, then a file called *<binary>.inst* is saved in the same folder as the original binary.

From within hpctView, you can launch an instrumented binary using either IBM Parallel Environment (by way of POE) or IBM Spectrum Load Sharing Facility (LSF). The view of the profiling tool is usually automatically opened. The binary file cannot necessarily be instrumented with the `hpctView`, but it can use one of the built-in HPC Toolkit command-line tools.

To run an instrumented binary, complete these steps:

1. Click **Run** → **Profile Configurations** to open the Profile Configurations window.
2. Double-click the **Parallel Application** list on the left side pane. A new configuration entry is created and a pane is displayed for it as shown in Figure 6-11.

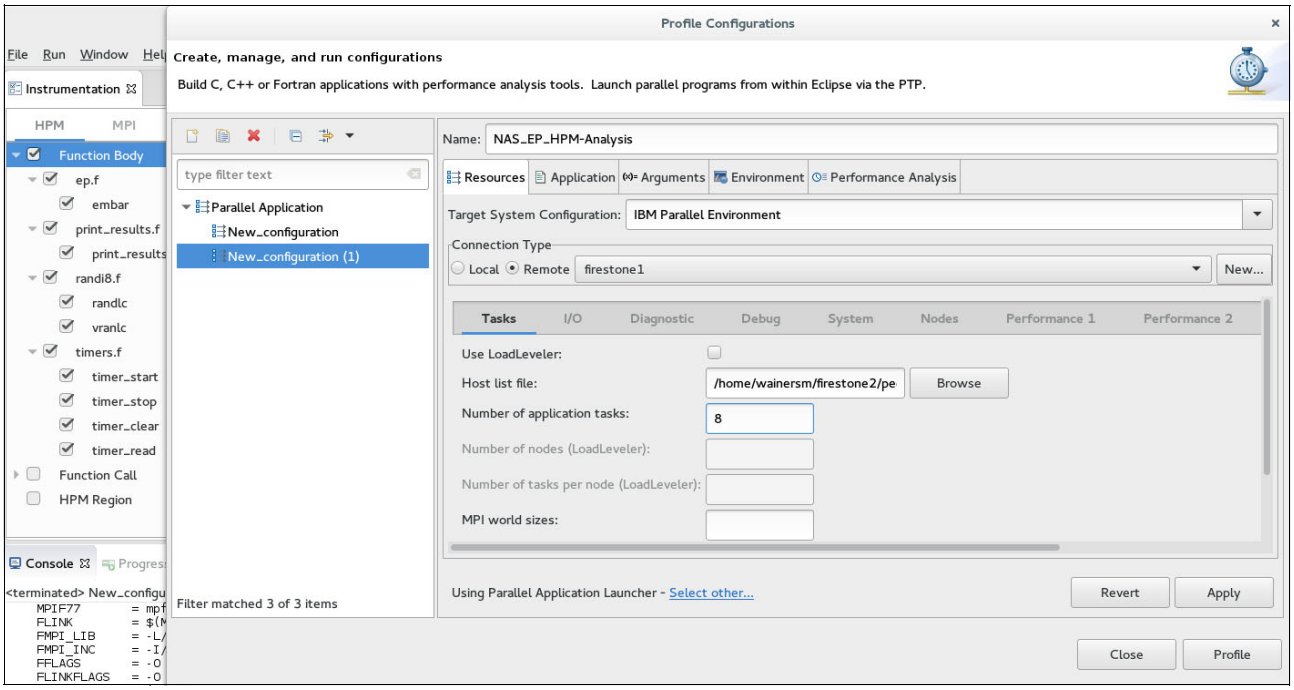

Figure 6-11 Profile configurations

3. Click the Performance Analysis tab. This tab contains the options of the profiling tools that can be entered as shown in Figure 6-12.

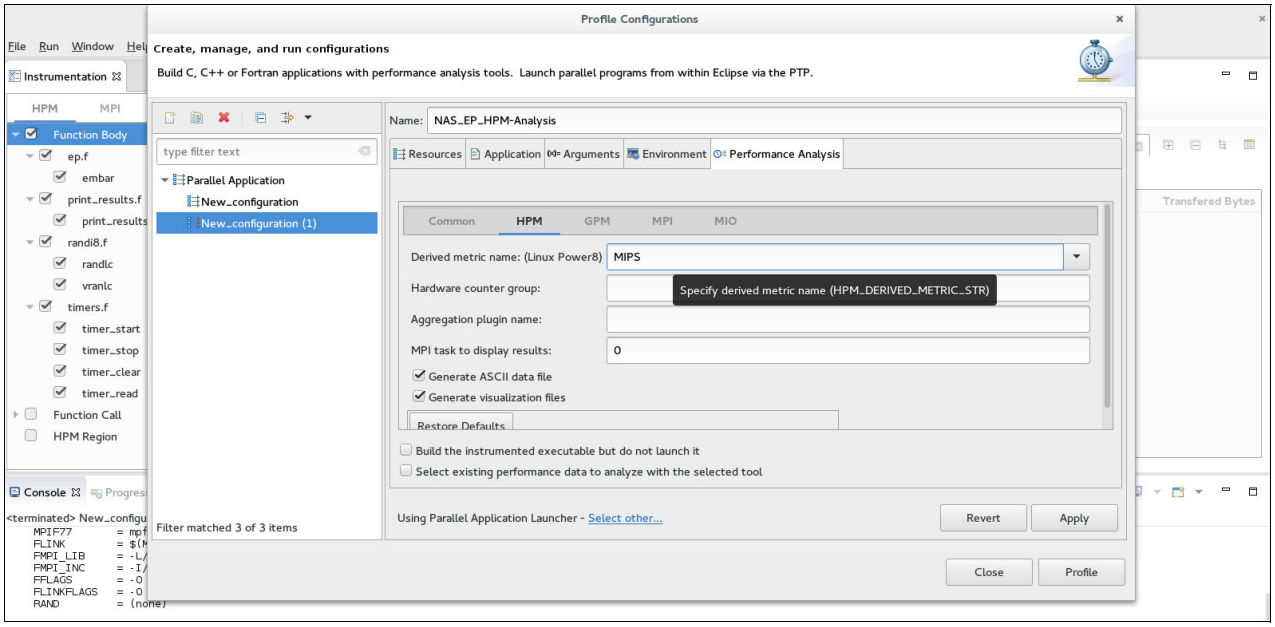

Figure 6-12 HPM tool

4. After this is all set, click **Profile.**

6.7.2 IBM PE Parallel Debugger

PDB and parallel environment poe communicate by using a portable, scalable, and high-performance protocol, the Scalable Communication Infrastructure (SCI).

The Parallel Debugger (PDB) for the IBM Parallel Environment is a command-line debugger that comes with IBM PE and operates in a similar manner to many instances of GBD debugging multiple programs.

PDB comes with subcommands that manage and switch between the many instances of gdb clients, often one per program running. These subcommands allow for multiple consoles, message control, thread stack information for each task, and controls to reduce the task information that is generated.

The debugging session can be started by either launching the parallel application with the **pdb** command or attaching it to a running job.

Example 6-39 launches a program called `threaded_ring` to debug with the **pdb** command. Any parameters to Parallel Operating Environment (POE) are passed with **--poe** option. It is set breakpoint (`break threaded_ring.c:send_thread`) at `send_thread` function, where you want to check the value of a variable called `dest`. The following debug steps are carried out:

1. Run the program to stop at breakpoint.
2. Create the groups task0 (`group task0=0`) and task1 (`group task1=1`) that map to gbd instances attached to tasks 0 and 1, respectively.
3. Enter gbd client mapped to task 0 (`on task0`), then use **gdb** subcommands **next** and **print** to execute the next line of code and display the value of the `dest` variable
4. Enter the gdb client mapped to task 1 (`on task1`), then proceed with the above steps

Example 6-39 IBM PE pdb command to debug parallel programs

```
$ pdb ./threaded_ring --debugger gdb --poe "-procs 2 -resd poe"
PDB -- Parallel Debugger for IBM Parallel Environment on Linux

Internal Debugger: gdb(Provided by "--debugger" value)

PDB warning: lxc-cr-attach/lxc-attach is not found, checkpointable job debug will
be impacted
Current PDB debug session number is 92668. Be aware, only one console
will connect to this session after startup, type 'pdb -c 92668' to
connect more consoles to this session.

At the prompt, enter any gdb command.

Enter 'help' for more usage.

Info: Connecting to poe domain socket with timeout 60 seconds...Done.
Info: Getting tasks' information from poe domain socket...Done.
Info: Deploy all debugger instances within maximum 600 seconds...
Process: 100%(use 1 seconds totally)
Info: All the debugger instances have been deployed successfully.
(all) break threaded_ring.c:send_thread
0:1 | Breakpoint 1 at 0x10000fd4: file threaded_ring.c, line 100.
(all) run
0:1 | Starting program: /home/wainersm/threads/./threaded_ring
0:1 | [Thread debugging using libthread_db enabled]
```

```
0:1 | Using host libthread_db library "/lib64/power8/libthread_db.so.1".
1 | [New Thread 0x100007b4ed90 (LWP 92784)]
0 | [New Thread 0x100007b4ed90 (LWP 92783)]
1 | [New Thread 0x100007f4ed90 (LWP 92785)]
0 | [New Thread 0x100007f4ed90 (LWP 92786)]
1 | [New Thread 0x10000834ed90 (LWP 92788)]
0 | [New Thread 0x10000834ed90 (LWP 92787)]
0 | [New Thread 0x100009cfed90 (LWP 92791)]
1 | [New Thread 0x100009cfed90 (LWP 92789)]
0 | [New Thread 0x10000a37ed90 (LWP 92792)]
1 | [New Thread 0x10000a37ed90 (LWP 92790)]
0 | [New Thread 0x10000a77ed90 (LWP 92793)]
1 | [New Thread 0x10000a77ed90 (LWP 92794)]
0 | [New Thread 0x10000ab7ed90 (LWP 92797)]
1 | [New Thread 0x10000ab7ed90 (LWP 92798)]
0 | [Switching to Thread 0x10000ab7ed90 (LWP 92797)]
1 | [Switching to Thread 0x10000ab7ed90 (LWP 92798)]
0:1 |
0:1 | Breakpoint 1, send_thread (dummy=0x0) at threaded_ring.c:100
0:1 | 100          dest = (me==tasks-1) ? 0 : me+1;
0:1 | Missing separate debuginfos, use: debuginfo-install
glibc-2.17-105.el7.ppc64le libgcc-4.8.5-4.el7.ppc64le
libstdc++-4.8.5-4.el7.ppc64le
(all) group task0=0
(all) group task1=1
(all) on task0
(task0) next
0 | 101    rc = MPI_Isend(out,ARRAY_SIZE,MPI_INT,dest,TAG,MPI_COMM_WORLD,&msgid);
(task0) print dest
0 | $1 = 1
(task0) on task1
(task1) next
1 | 101    rc = MPI_Isend(out,ARRAY_SIZE,MPI_INT,dest,TAG,MPI_COMM_WORLD,&msgid);
(task1) print dest
1 | $1 = 0
(task1) on all
(all) c
0:1 | Continuing.
0 | TEST COMPLETE
1 | [New Thread 0x10000af7ed90 (LWP 92800)]
0 | [New Thread 0x10000af7ed90 (LWP 92799)]
1 | [Thread 0x10000ab7ed90 (LWP 92798) exited]
1 | [Thread 0x10000af7ed90 (LWP 92800) exited]
0 | [Thread 0x10000ab7ed90 (LWP 92797) exited]
1 | [Thread 0x10000a37ed90 (LWP 92790) exited]
0 | [Thread 0x10000af7ed90 (LWP 92799) exited]
1 | [Thread 0x100009cfed90 (LWP 92789) exited]
0 | [Thread 0x10000a37ed90 (LWP 92792) exited]
1 | [Thread 0x10000a77ed90 (LWP 92794) exited]
0 | [Thread 0x100009cfed90 (LWP 92791) exited]
1 | [Thread 0x10000834ed90 (LWP 92788) exited]
0 | [Thread 0x10000a77ed90 (LWP 92793) exited]
1 | [Thread 0x100007f4ed90 (LWP 92785) exited]
0 | [Thread 0x10000834ed90 (LWP 92787) exited]
1 | [Thread 0x100007b4ed90 (LWP 92784) exited]
```

```
0 | [Thread 0x100007f4ed90 (LWP 92786) exited]
0 | [Thread 0x100007b4ed90 (LWP 92783) exited]
1 | [Inferior 1 (process 92778) exited normally]
0 | [Inferior 1 (process 92777) exited normally]
(all)
```

For more information about pdb options and subcommands, see the following website:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.3.0/com.ibm.cluster.pe.v2r3.pe100.doc/am102_dishandpdb.htm

6.7.3 Eclipse for Parallel Application Developers

Eclipse for Parallel Application Developers provides an entire IDE. It bundles some open source projects under the Eclipse umbrella that includes C/C++ Development Tools (CDT) and PTP. It provides a complete environment for coding, compilation, launch, analysis, and debug of applications written in C, C++, and Fortran. It uses MPI, OpenMP, PAMI, and OpenSHMEM.

The specialized editors feature syntax highlighting, code assistant, place markers for compiler errors, auto-completion, and many other helpers to improve productivity. They also come with static analysis tools that are used to identify common coding mistakes such as MPI barriers mismatch.

You can build programs by using Eclipse managed makefile (integrated with IBM XL and GNU compilers), Makefile, Autotools, and custom scripts and commands. The integration with XL and GNU compilers also includes build output parsers that can correlate errors to source code files.

Remotely launched applications on either IBM Parallel Environment (by way of POE) or IBM Spectrum LSF are supported.

Its parallel debugger provides specific debugging features for parallel applications that distinguish it from the Eclipse debugger for serial applications. Its parallel debugger is also able to launch and remotely debug parallel applications through POE and LSF.

Most of the topics that are introduced in this section are detailed in the Eclipse PTP user guide at the following website. Select the **Parallel Tools Platform (PTP) User Guide**.

http://help.eclipse.org

To install Eclipse, download its tarball file and then uncompress it on your development workstation. Visit the Eclipse download website (http://www.eclipse.org/downloads/) and select **Eclipse for Parallel Application Developers**. Download the Eclipse tarball file for your workstation operating system and architecture. At the time of writing, the latest Eclipse version was 4.5 SR1 (also known as Eclipse Mars.1).

Remote synchronized development model

The Eclipse for Parallel Application developers implements a remote development working model where the code editing is carried out locally within a workbench (GUI). However, other tasks that are usually required to be performed on the server side, such as build, launch, debug, and profile, are carried out remotely.

The model uses a synchronized project type to keep local and remote copies of the working directory updated so that code programming is minimally affected by network latency or slow response time in the editor. The C, C++, and Fortran editors appear as though you are

developing locally on the Power Systems machine even though all of the resources are on the server side. With this approach, you do not need to rely on a cross-compilation environment, which is often difficult to set up. This approach provides an efficient method for remote development.

The parallel debugger

The Eclipse PTP parallel debugger extends the default C/C++ Eclipse debugger for serial application to scale parallel programs.

It combines into a single viewer the information obtained from the many processes and threads that a parallel job is composed of. Therefore, you can browse the entities separately if needed.

Debugging operations can be carried out on any arbitrary subset of processes within a parallel job, as for example, step in and out, and stop at breakpoint. A different type of breakpoint, which is called a *parallel breakpoint*, can be applied in these collections of processes.

The debugger is able to launch and remote debug parallel programs by using several types of workload managers, including the IBM Parallel Environment and Spectrum LSF.

Using IBM PE Developer Edition plug-ins

The hpctView plug-ins can be installed on top of Eclipse in addition to the PE DE profiling and tracing tools. First, uncompress the Eclipse update that bundles the hpctView plug-ins, which releases the PE DE 2.2 files:

```
$ mkdir ppedev_update-2.2.0
$ unzip ppedev_update-2.2.0-0.zip -d ppedev_update-2.2.0
```

Install the plug-ins by using the Eclipse install software using these steps:

1. At tool bar menu, select **Help** → **Install New Software**.
2. Click **Add** to create a new repository location from where Eclipse can find plug-ins to be installed.
3. Enter the location where hpctView update site files were uncompressed, and click **OK** to add the repository configuration.
4. In the selection tree appears, select its root element (IBM PE Developer Edition Components) to install all PE DE plug-ins. Click **Next** → **Next**.
5. The license agreement text comes up. Read and accept it to continue the installation.
6. Click **Finish** to install the plug-ins.

See the Eclipse workbench user guide if you find it difficult to install the plug-ins. It can be accessed from `http://help.eclipse.org` by selecting **Workbench User Guide** → **Tasks** → **Updating and installing software**.

To use hpctView, open its perspective within Eclipse by completing these steps:

1. Click **Window** → **Perspective** → **Open Perspective** → **Other**.
2. In the perspective window that opens, select **HPCT** and click **OK**.

The PE DE embedded user guide gives detailed instructions on how to use the hpctView resources and HPC Toolkit from inside Eclipse. To read its user guide, on the tool bar menu click **Help** → **Help Contents**, then open the **IBM PE Developer Edition** content page.

6.7.4 NVIDIA Nsight Eclipse Edition for CUDA C/C++

The NVIDIA Nsight Eclipse Edition is a complete IDE for development of CUDA C and C++ applications that run on NVIDIA GPUs.

It provides developers with an Eclipse-based GUI that includes tools for a complete cycle of development: Code writing, compilation, debug, and profile and tuning.

Two development models are allowed as follows:

- Local: The complete development cycle is carried out at the machine where Nsight is running. The program produced is targeted for the local host architecture and GPU.
- Remote: The complete development cycle is carried out at a remote host where Nsight is running.

The next sections show basic use of Nsight for remote development of CUDA C applications for IBM Power Systems. The examples illustrated here use an x86_64 workstation with Linux Fedora 21 and running the NVIDIA Nsight Eclipse Edition that comes with the CUDA Toolkit 7.5.

Because that Nsight has many functionalities, they are not all described in this book. Instead, it shows how to create compile and debug projects. See the following website to learn more about usage instructions:

http://developer.nvidia.com/nsight-eclipse-edition

Running NVIDIA Nsight Eclipse Edition

The Nsight GUI can be started as follows:

```
$ export PATH=/usr/local/cuda-7.5/bin:$PATH
$ nsight &
```

Creating a project for remote development

Before you begin, check the requirements and carry out the following tasks:

1. Ensure that the machine (usually the cluster login node) you are going to use for remote development has Git installed. Nsight uses **`git`** commands to synchronize local and remote copies of the project files.
2. Connect to the remote machine to configure user name and email used by Git, for example:

   ```
   $ git config --global user.name "Wainer dos Santos Moschetta"
   $ git config --global user.email "wainersm@br.ibm.com"
   ```

3. Ensure that the directory that is going to hold the project files in the remote machines is already created, for example:

   ```
   $ mkdir -p ~/wainer/cudaBLAS
   ```

In the Nsight GUI, complete these steps to create a CUDA C/C++ project:

1. On the main menu tool bar, click **File** → **New** → **CUDA C/C++ Project** to open the window shown in Figure 6-13.

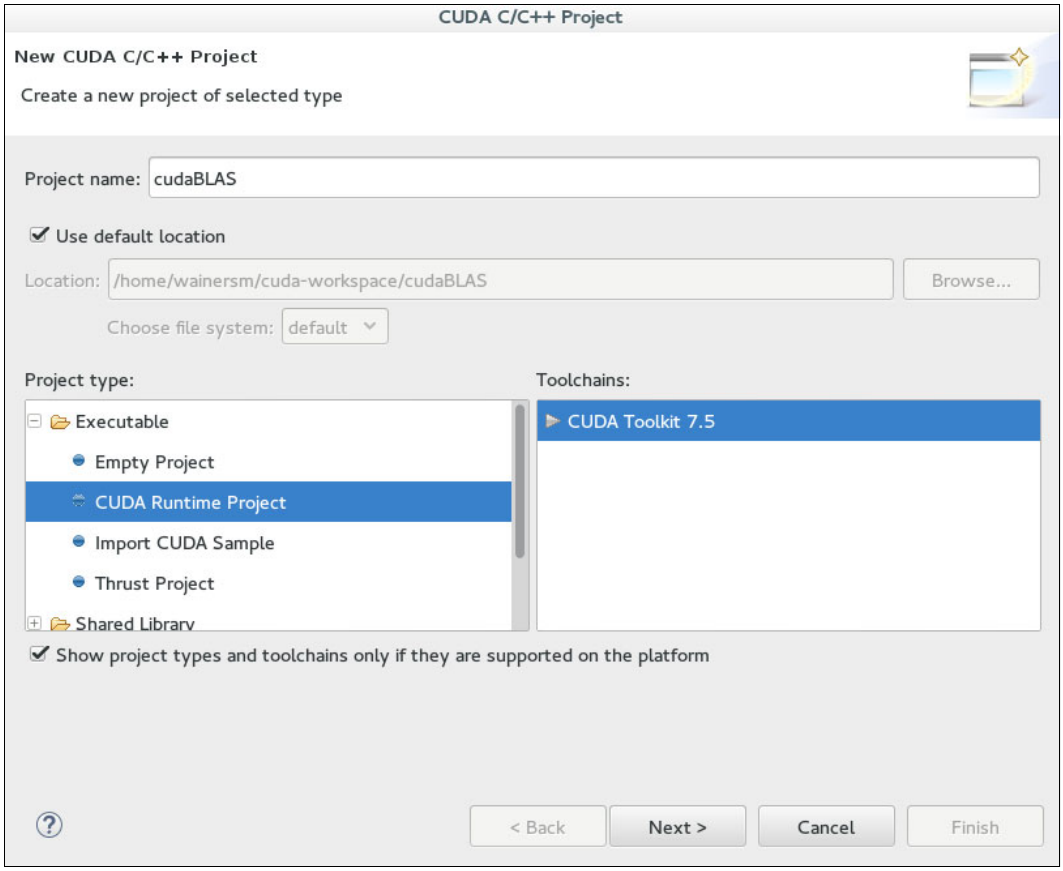

Figure 6-13 Nsight: New CUDA C/C++ project wizard

2. Click **Next** → **Next**.

3. Select **3.7** on both **Generate PTX code** and **Generate GPU code** fields as shown in Figure 6-14, and click **Next.**

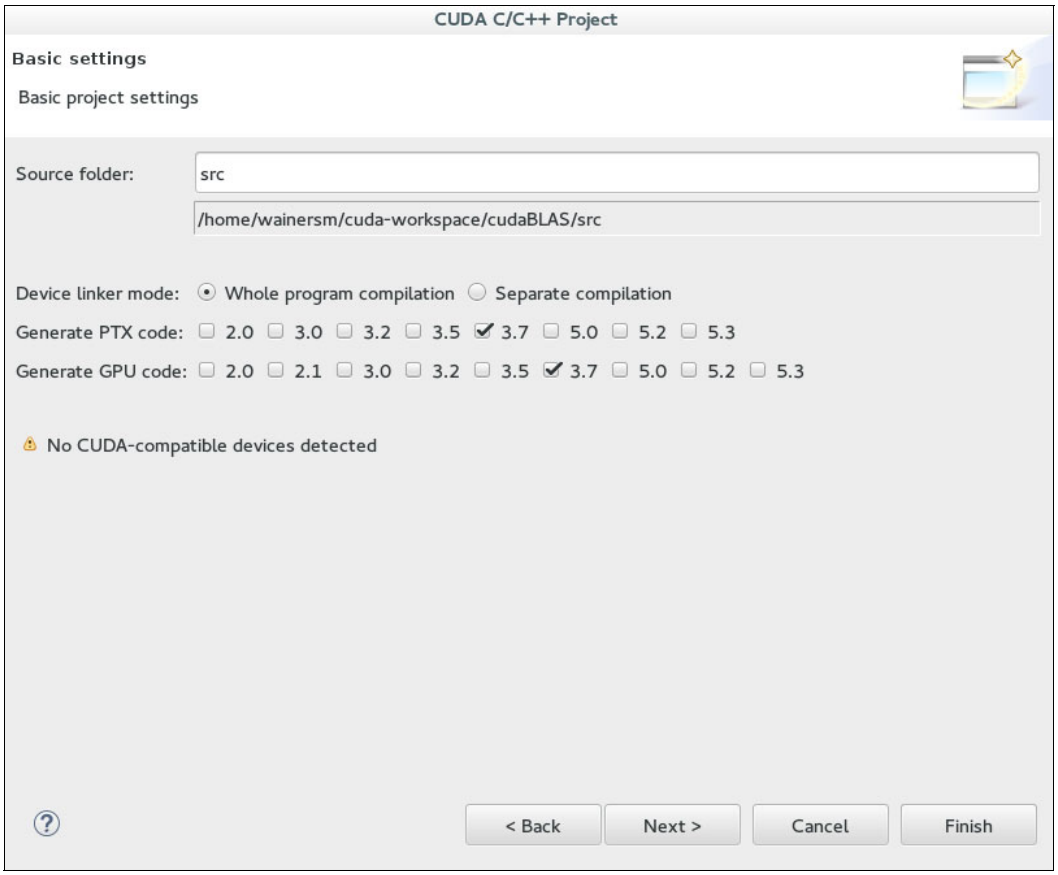

Figure 6-14 Nsight: New CUDA C/C++ project basic settings

4. Click **Manage** (as pointed out by the arrow in Figure 6-15) to configure a new connection to the remote machine.

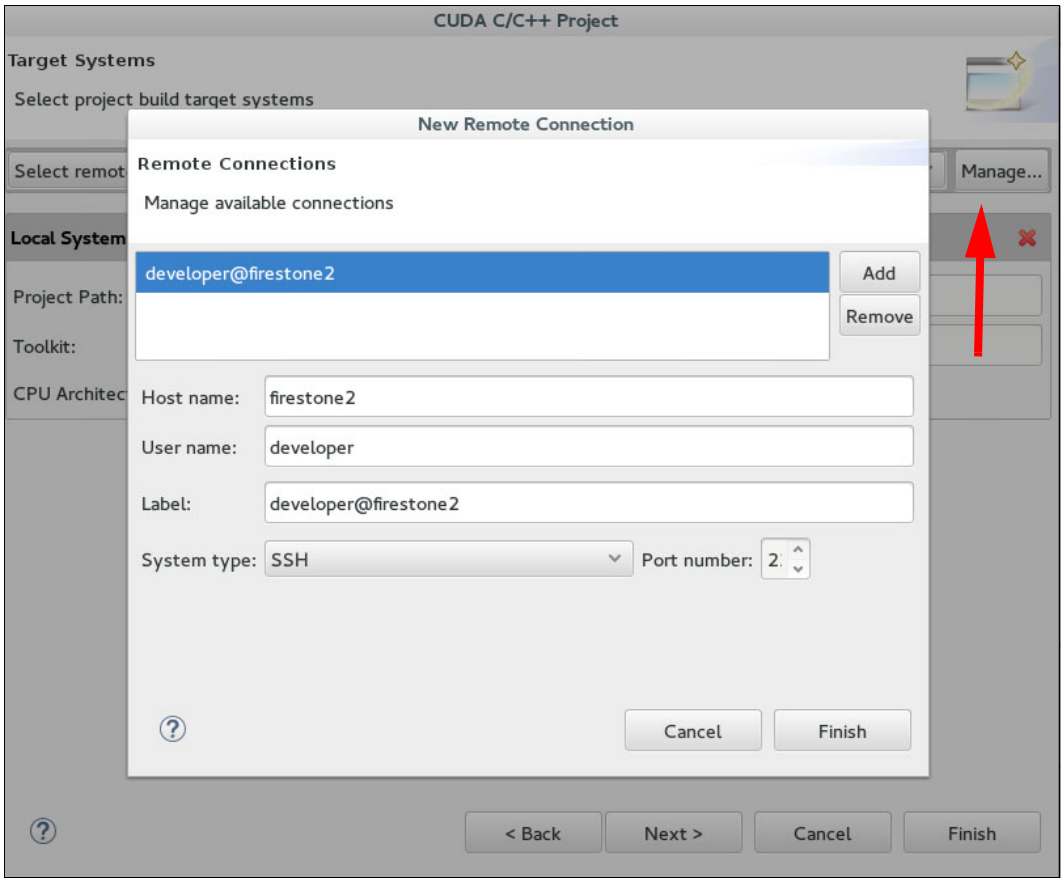

Figure 6-15 Nsight: New CUDA C/C++ project remote connection setup

5. As shown in Figure 6-15, fill in the connection information fields (host name, user name). Click **OK.**

6. Fill out the **Project path**, **Toolkit**, and set the **CPU Architecture** fields for the newly created connection (see Figure 6-16). You must remove the **Local System** configuration. Click **Next** → **Finish.**

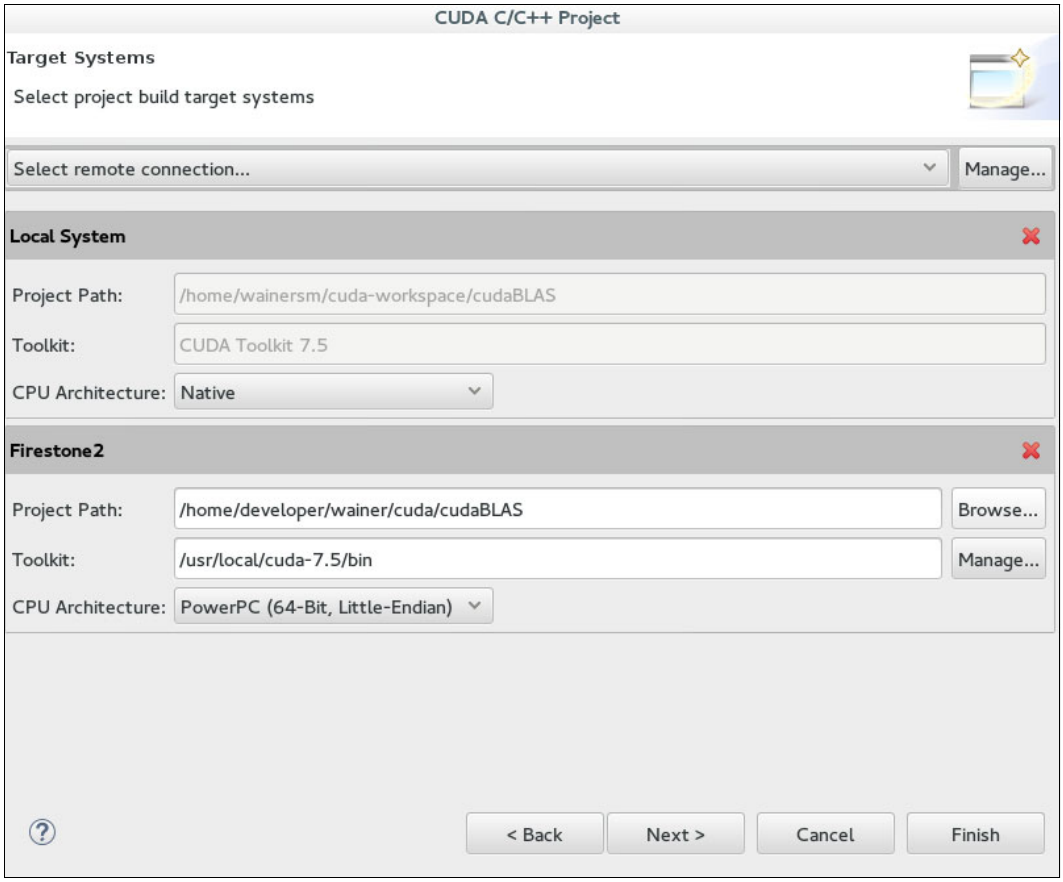

Figure 6-16 Nsight - new CUDA C/C++ project further remote configuration setup

After you successfully created the project, perform a few more tasks to configure the Nsight to automatically synchronize the local and remote project files. Complete these steps:

1. Create a source code file (for example, `main.c`).
2. Right-click the project in the Project Explorer pane and select **Synchronize** → **Set active**. Then choose the option that matches the name of the connection configuration to the remote machine.
3. Right-click the project again and select **Synchronize** → **Sync Active now** to perform the first synchronization between local and remote folders. That operation can be performed at any time to force a synchronization. Remember that Nsight is already configured to do so after or before some tasks (for example, before compiling the project).

4. As an optional step, i set the host compiler that is evoked by **nvcc**. Right-click the project name and select **Properties**. Expand **Build**, and then choose **Settings**. Click the Tool Settings tab, then select **Build Stages** to fill out the **Compiler Path** and **Preprocessor options** fields (see Figure 6-17). Change the content of the **Compiler Path** under the Miscellaneous section of the NVCC Linker window.

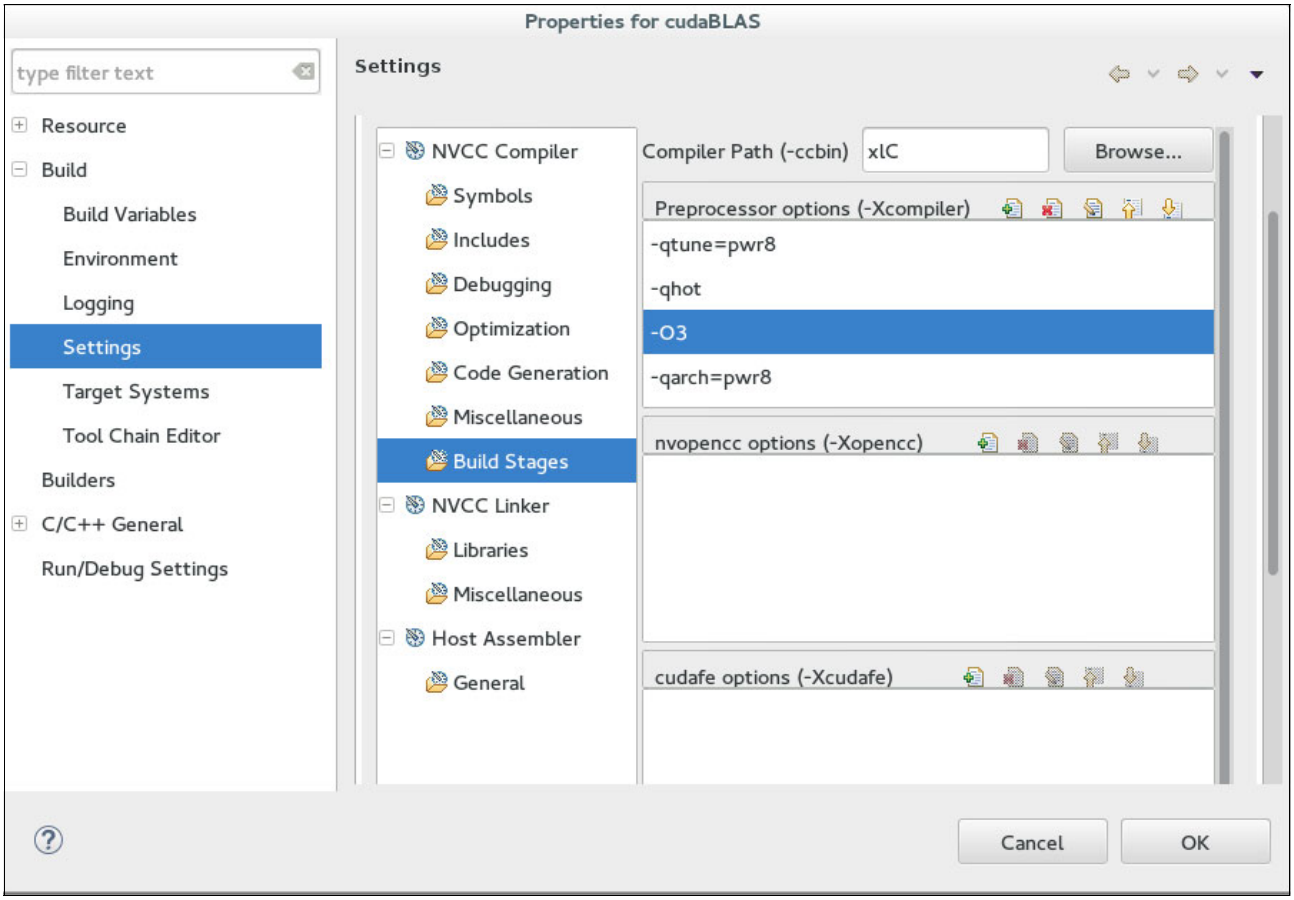

Figure 6-17 Nsight - new CUDA C/C++ project build settings

After completing these steps, the new created project is ready for CUDA C/C++ programming. You can take advantage of many of the features that are provided by the Nsight C and C++ code editor, such as syntax highlighting, code completion, static analysis, and error markers.

Compiling the application

To build the application, complete these steps:

1. Right-click the project in the Project Explorer pane and select **Run As** → **Remote C/C++ Application**.
2. Check that all the information is correct, and change it if needed. Then click **Run**.

Debugging the application

The steps to launch the Nsight debugger are similar to those of running the application. However, the executable is started by the **cuda-gdb** command, which in turn is backed by the GNU GDB debugger tool.

See 6.7.5, “Command-line tools for CUDA C/C++” on page 226 for more information about **cuda-gdb** command.

To debug the application, complete these steps:

1. Right-click project name in the Project Explorer pane and select **Debug As** → **Remote C/C++ Application**.
2. A window allowing permission to open the Eclipse Debug perspective comes up. Click **Yes**.

By default, the debugger stops at the first instruction on the main method. Figure 6-18 shows the Nsight debugger.

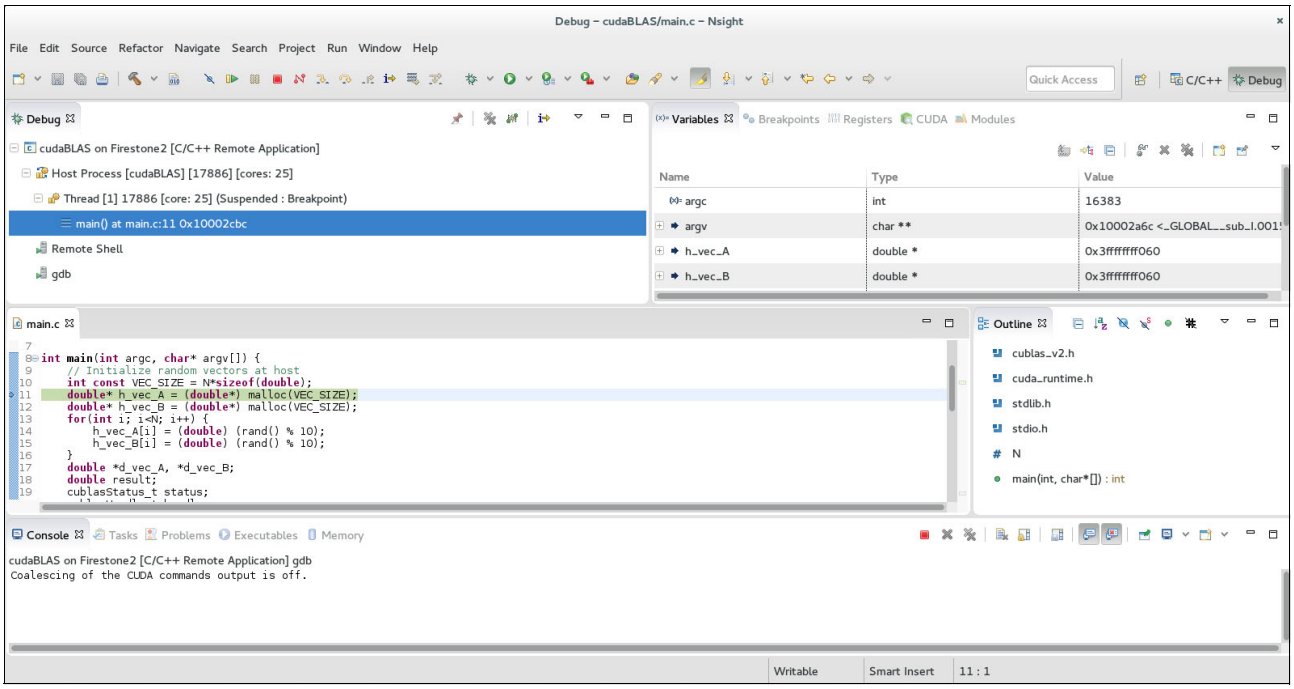

Figure 6-18 Nsight: CUDA C/C++ debugger

6.7.5 Command-line tools for CUDA C/C++

The NVIDIA Toolkit 7.5 provides the following tools for debugging problems on CUDA C and C++ applications.

CUDA memcheck

The memcheck tool (cuda-memcheck) detects memory-related flaws on programs. It dynamically instruments the program executable on run time and is able to check allocation/deallocation and accesses on global and shared memories.

Other than memory checking, cuda-memcheck is also able to inspect the application to catch these types of errors:

- Race condition: Check for race condition on access of shared and local variables. Uses the **`--tool racecheck`** option.
- initcheck: Check for use of non-initialized memory. Uses the **`--tool initcheck`** option

Notice that for complete visualization of source code file and line number where detected problems occur, it is needed to compile the binary with **`nvcc -g -lineinfo`** options.

For more information about the CUDA memcheck tool, check its user manual at the following website:

http://docs.nvidia.com/cuda/cuda-memcheck/index.html

CUDA GDB

CUDA application portions of the code runs on CPU (host) and GPUs (devices), and also program can often have thousands of CUDA threads that are spread across many GPUs. Although traditional debuggers are effective to work with CPU code, they are not properly built to deal with both.

The cuda-gdb tool is an implemented debugger extension of GNU GDB designed to deal with the scenario that CUDA hybrid model imposes.

nvprof

The nvprof is a profiling tool able to count and report the occurrence of GPU events while the CUDA application executes performance measurements.

The tool can report several metrics that are calculated upon the information of hardware events collected.

nvprof is flexible because it allows to profile all processes in the entire system, only a given application or just some elements of it (for example, kernels, streams, and contexts). Also, it allows you to select which GPU device must be profiled (default is all on the system).

Another useful function of **nvprof** is to generate traces of the application execution.

For more information about the nvprof tool, check its user manual at the following website:

http://docs.nvidia.com/cuda/profiler-users-guide/index.html

7

Running applications

This chapter describes how to run applications.

This chapter contains the following sections:

- Controlling the execution of multithreaded applications
- Using the IBM Parallel Environment runtime
- Using the IBM Spectrum LSF

7.1 Controlling the execution of multithreaded applications

The technical computing applications that run on a computing node typically use multiple threads. The runtime environment has several options to support the runtime fine-tuning of multithreaded programs. First, this chapter shows how to control the OpenMP[1] workload by setting certain environment variables. Then, it explains the tools that a user can use to retrieve or set affinity of a process at run time. It also shows how to control the nonuniform memory access (NUMA) policy for processes and shared memory[2].

7.1.1 Running OpenMP applications

A user can control the execution of OpenMP applications by setting environment variables. This section covers only a subset of the environment variables that are especially important for technical computing workloads.

Note: The environment variables that are prefixed with OMP_ are defined by the OpenMP standard. Other environment variables mentioned in this section are specific to a particular compiler.

Distribution of a workload

The OMP_SCHEDULE environment variable controls the distribution of a workload among threads. If the workload items have uniform computational complexity, the static distribution fits well in most cases. If an application does not specify the scheduling policy internally, a user can set it to `static` at run time by exporting the environment variable:

```
export OMP_SCHEDULE="static"
```

Specifying the number of OpenMP threads

OpenMP applications are often written so that they can spawn an arbitrary number of threads. In this case, to specify the number of threads to create, the user sets the OMP_NUM_THREADS environment variable. For example, to run the application with `20` threads, use this setting:

```
export OMP_NUM_THREADS=20
```

Showing OpenMP data

The OpenMP 4.0 standard introduced the OMP_DISPLAY_ENV environment variable to show the OpenMP version and list the internal control variables (ICVs). The OpenMP run time prints data to the `stderr` output stream. If the user sets the value to TRUE, the OpenMP version number and initial values of ICVs are printed. The VERBOSE value instructs the OpenMP run time to augment the output with the values of vendor-specific variables. If the value of this environment variable is set to FALSE or undefined, no information is printed. This variable is useful when you need to be certain that the runtime environment is configured as expected at the moment that the program loads.

Placement of OpenMP threads

The POWER8 microprocessor can handle multiple hardware threads simultaneously. With the increasing number of logical processors, the operating system kernel scheduler has more

[1] OpenMP (Open Multi-Processing) is an application programming interface that facilitates the development of parallel applications for shared memory systems.

[2] This section is based on the content that originally appeared in Chapter 7 of *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263.

possibilities for automatic load balancing. But for technical computing workloads, the fixed position of threads within the server is typically preferred.

The IDs of the logical processors are zero based. Each logical processor has the same index regardless of the simultaneous multithreading (SMT) mode. Logical processors are counted starting from the first core of the first socket. The first logical processor of the second core has the index 8. Start numbering the logical processors of the second socket only after you finish the numbering of the logical processors of the first socket.

IBM XL compilers

For the OpenMP application that is compiled with IBM XL compilers, you need to use the XLSMPOPTS environment variable to control thread placement. This environment variable has many suboptions, and only a few of them control thread binding. You can use either the combination of `startproc` and `stride` suboptions, or the `procs` suboption as described in the following section:

- The `startproc` suboption is used to specify the starting logical processor number for binding the first thread of an application. The `stride` suboption specifies the increment for the subsequent threads. For example, the following value of XLSMPOPTS instructs the OpenMP runtime environment to bind OpenMP threads to logical processors 80, 84, 88, and so on, up to the last available processor:

  ```
  export XLSMPOPTS=startproc=80:stride=4
  ```

- A user can also explicitly specify a list of logical processors to use for thread binding with the `procs` suboption. For example, to use only even-numbered logical processors of a processor's second core, specify the following value of XLSMPOPTS:

  ```
  export XLSMPOPTS=procs=8,10,12,14
  ```

> **Note:** The `startproc`, `stride`, and `procs` suboptions have been deprecated in favor of the OMP_PLACES environment variable. These subooptions might be removed in future releases of the IBM XL compiler runtime environment.

For more information about the **`XLSMPOPTS`** environment variable, see the XLSMPOPTS section of the online manuals at the following websites:

- XL C/C++ for Linux:

  http://www.ibm.com/support/knowledgecenter/SSXVZZ/welcome

- XL Fortran for Linux:

  http://www.ibm.com/support/knowledgecenter/SSAT4T/welcome

GCC compilers

For the OpenMP application that is compiled with GNU Compiler Collection (GCC) compilers, use the GOMP_CPU_AFFINITY environment variable. Assign a list of the logical processors that you want to utilize to the GOMP_CPU_AFFINITY environment variable. The syntax and semantics are the same as with the `procs` suboption of the IBM XL compilers XLSMPOPTS environment variable. For more information about the GOMP_CPU_AFFINITY environment variable, see the corresponding section of the GCC manual at the following website:

http://bit.ly/1thwOaq

Support for thread binding in the recent versions of the OpenMP standard

The OpenMP 3.1 revision introduced the OMP_PROC_BIND environment variable. The Open MP 4.0 revision introduced the OMP_PLACES environment variable. These variables control thread binding and affinity in a similar manner to XLSMPOPTS and GOMP_CPU_AFFINITY, although their syntax slightly differs.

Performance impact

For the discussion of thread binding impact on the performance, see “Effects of basic performance tuning techniques” on page 278. This section also provides an easy-to-use code that you can use to generate your own binding map. For more information, see “Sample code for the construction of thread affinity strings” on page 309.

7.1.2 Setting and retrieving process affinity at run time

With the Linux **taskset** command, you can manipulate the affinity of any multithreaded program, even if you do not have access to the source code. You can use the **taskset** command to launch a new application with a certain affinity by specifying a mask or a list of logical processors. The Linux scheduler restricts the application threads to a certain set of logical processors only.

You can also use the **taskset** command when an application creates many threads and you want to set the affinity for highly loaded threads only. In this circumstance, identify the process identifiers (PIDs) of highly loaded running threads and perform binding only on those threads. You can discover these threads by examining the output of the **top -H** command.

Knowing the PID, you can also use the **taskset** command to retrieve the affinity of the corresponding entity (a thread or a process).

7.1.3 Controlling NUMA policy for processes and shared memory

With the **numactl** command, you can specify a set of nodes and logical processors that you want your application to run on. In the current context, you can assume that this tool defines a *node* as a group of logical processors that are associated with a particular memory controller. For POWER8, such node is a whole processor. To discover the indexes of nodes and estimate the memory access penalty, run **numactl** with the **-H** argument. Example 7-1 shows the corresponding output for a 20-core IBM Power System S822LC (Model 8335-GTA) server that is running Red Hat Enterprise Linux Server release 7.2 little endian.

Example 7-1 numactl -H command (truncated) output in a 20-core IBM Power System S822LC

```
$ numactl -H
available: 2 nodes (0,8)
< ... output omitted ... >
node distances:
node   0   8
  0:  10  40
  8:  40  10
```

You can pass these indexes of the nodes to **numactl** as an argument for the **-N** option to bind the process to specific *nodes.* To bind the process to specific *logical processors*, use the **-C** option. In the latter case, the indexes follow the same conventions as the OpenMP environment variables and a **taskset** command.

The memory placement policy significantly affects the performance of technical computing applications. You can enforce a certain policy by the **numactl** command. The **-l** option instructs the operating system to always allocate memory pages on the current node. Use the **-m** option to specify a list of nodes that the operating system can use for memory allocation. You need to use the **-i** option to ask the operating system for a round-robin allocation policy on specified nodes.

7.2 Using the IBM Parallel Environment runtime

This section describes the execution of a parallel application through the IBM Parallel Environment Runtime.

7.2.1 Running applications

The IBM Parallel Environment provides an environment to manage the execution of parallel applications, and the Parallel Operating Environment (POE) is started with a call to the **poe** command line.

In a nutshell, an execution of an application with POE is going to spread processes across the cluster nodes as follows:

- Parallel tasks are spawned on compute nodes.
- One instance of the partition manager daemon (PMD) per compute node that has tasks of the application running.
- The *poe* process resides on the submitting node (home node) machine where it was evoked.

The PMD is in charge of controlling communication between the home **poe** process and the spawned tasks in a given compute node. The PMD is also used to pass the standard input, output, and error streams of the home **poe** to the tasks.

However, if the PMD process exits abnormally, such as with *kill* signals or the **bkill** command of the IBM Spectrum LSF, then the shared memory used for intra-node message passing cannot get cleaned up properly. In this case, use the **ipcrm** command to reclaim that memory.

The environment works in two modes: interactive and batch. It also allows single program, multiple data (SPMD) and multiple program multiple data (MPMD) programs.

Many environment variables are in place to control the behavior of **poe**. Some of them are discussed throughout this section.

The number of tasks in the parallel application is specified with MP_PROCS variable that is equivalent to `-procs` option of **poe** command. The following example sets the application with 10 tasks:

```
$ MP_PROCS=10 poe ./myApp
```

poe manages the execution of applications that are implemented with different models and eventually mixed. To set the messaging API, set the MP_MSG_API variable (equivalent to the `-msg_api` option). The accepted values are MPI, PAMI, and `shmem`, and it does not need to be set for MPI programs because it is the default. In following command, **poe** is called to execute a 10 task OpenSHMEM program:

```
$ MP_PROCS=10 MP_MSG_API=shmem poe ./myApp
```

Compute nodes allocation

The partition manager can connect to an external resources manager that will determine allocation of compute nodes. For example, the configuration described in 4.11, “IBM Spectrum LSF (formerly IBM Platform LSF)” on page 50 can be configured to integrate seamlessly with IBM PE so that hosts are selected by the Spectrum LSF **bsub** command for batch jobs submission.

By default, the allocation uses any resources manager that is configured. However, it can be disabled by setting MP_RESD (or `-resd` option). Also, native partition manager nodes allocation mechanism are enabled with `MP_RESD=poe` and require a hosts list file.

Example 7-2 displays what a host list file looks like when resource manager is not used with **poe**. Notice `MP_RESD=poe` is exported to enable the internal host allocation mechanism.

Example 7-2 Run parallel application without resource manager through IBM PE

```
$ cat host.list
! Host list - One host per line
! Tasks 0 and 2 run on p8r1n1 host
! Tasks 1 and 3 run on p8r2n2 host
p8r1n1
p8r2n2
p8r1n1
p8r2n2
$ MP_PROCS=4 MP_RESD=poe poe ./myApp
```

The format and content of the hosts list file changes whether using a resource manager or not. See the following website on IBM PE documentation for further information:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.3.0/com.ibm.cluster.pe.v2r3.pe100.doc/am102_chlf.htm

If it needs to point out the host file location, use the MP_HOSTFILE variable (same as `-hostfile` option of **poe**). If it is not set, then poe looks for a file named `host.list` in the local directory.

When a resource manager is in place, the system administrators often configure pools of hosts. Therefore, consider using the MP_RMPOOL variable (or `-rmpool` option) to determine which of the pools of machines configured (if any) by the administrators to use.

Other variables are available to configure the resource manager behavior. The MP_NODES (`-nodes` option) and MP_TASKS_PER_NODE (`-tasks_per_node` option) variables set the number of physical nodes and tasks per node, respectively.

Considerations about network configuration

The IBM PE provides variables to control and configure the use of network adapters by the parallel applications. Some of them might already be implicitly set depending on the combination of other settings or by the use of a resource manager.

To specify the network adapters to use for message passing, set the MP_EUIDEVICE variable (or `-euidevice` option). It accepts the value `sn_all` (one or more windows are on each network) or `sn_single` (all windows are on a single network). The `sn_all` value is frequently used to enable protocol stripping, failover, and recovery.

Network adapters can be shared or dedicated. That behavior is defined with the MP_ADAPTER_USE variable (or `-adapter_use` option). It accepts the `shared` and `dedicated` values.

Considerations about Remote Direct Memory Access

The IBM PE implements message passing by using Remote Direct Memory Access (RDMA) through the InfiniBand (IB) interconnect. In such a mechanism, memory pages are automatically pinned and buffer transferences are handled directly by the InfiniBand adapter without the host CPU involvement.

RDMA on messaging passing is disabled by default. Export `MP_USE_BULK_XFER=yes` to enable bulk data transfer mechanism. Also, use the MP_BULK_MIN_MSG_SIZE variable to set the minimum message length for bulk transfer.

Considerations about affinity

Several levels of affinity are available for parallel applications through **poe**. These levels are also controlled by environment variables (or **poe** options). Because resource managers usually employ their own affinity mechanisms, those variables can be overwritten or ignored.

The primary variable to control placement of message passing interface (MPI) tasks is MP_TASK_AFFINITY (or `-task_affinity` option), when a resource manager is not employed. You can bind tasks at the physical processor (`core` value), logical CPU (`cpu` value), and multi-chip module (`mcm` value) levels.

For example, the following command allocates one core per task:

```
$ poe -task_affinity core -procs 2 ./myApp
```

More examples of MP_TASK_AFFINITY being used to control task affinity are shown in 6.5.1, “MPI programs with IBM Parallel Environment” on page 180.

The MP_CPU_BIND_LIST (or `-cpu_bind_list` option) and MP_BIND_MODE (or `-bind_mode spread` option) environment variables can be used together with MP_TASK_AFFINITY to further control the placement of tasks. MP_CPU_BIND_LIST specifies a list of processor units for establishing task affinity. In the following command, affinity is restricted to only the second core of each socket:

```
$ poe -task_affinity core -cpu_bind_list 0/16,8/1040 -procs 2
```

When using a resource manager, the MP_PE_AFFINITY variable can be set to `yes` so that **poe** will assume control over affinity. However, if IBM Spectrum Load Sharing Facility (LSF) is in use and it has already set affinity, **poe** honors the allocated CPUs. If `MP_PE_AFFINITY=yes` is enabled in Spectrum LSF batch jobs, it will enable the InfiniBand adapter affinity.

To assist on affinity definition, the IBM PE runtime provides the **cpuset_query** command that displays information about current assignments of a running program, and also provides the topology of any given compute node. As shown in n Example 7-3, **cpuset_query -t** displays the topology of an IBM S822LC System running on SMT-8 mode with two sockets with 10 cores each with eight hardware threads.

Example 7-3 cpuset_query command to show the node topology

```
$ cpuset_query -t
MCM(Socket): 0
         CORE: 8
                 CPU: 0
                 CPU: 1
                 CPU: 2
                 CPU: 3
                 CPU: 4
                 CPU: 5
                 CPU: 6
                 CPU: 7
         CORE: 16
                 CPU: 8
                 CPU: 9
                 CPU: 10
                 CPU: 11
```

```
                CPU: 12
                CPU: 13
                CPU: 14
                CPU: 15
<... Output Omitted ...>
MCM(Socket): 8
        CORE: 1032
                CPU: 80
                CPU: 81
                CPU: 82
                CPU: 83
                CPU: 84
                CPU: 85
                CPU: 86
                CPU: 87
        CORE: 1040
                CPU: 88
                CPU: 89
                CPU: 90
                CPU: 91
                CPU: 92
                CPU: 93
                CPU: 94
                CPU: 95
        CORE: 1048
<... Output Omitted ...>
```

A way to check the affinity is to run **cpuset_query** command through **poe** as shown in Example 7-4. The command shows a two tasks program that is launched with affinity at a core level. Each task is allocated (see CPUs with value 1) with a full core that has eight hardware threads on SMT-8 mode node.

Example 7-4 cpuset_query command to show task affinity

```
MCM/QUAD(0) contains:
cpu0, cpu1, cpu2, cpu3, cpu4,
cpu5, cpu6, cpu7, cpu8, cpu9,
cpu10, cpu11, cpu12, cpu13, cpu14,
cpu15, cpu16, cpu17, cpu18, cpu19,
cpu20, cpu21, cpu22, cpu23, cpu24,
cpu25, cpu26, cpu27, cpu28, cpu29,
cpu30, cpu31, cpu32, cpu33, cpu34,
cpu35, cpu36, cpu37, cpu38, cpu39,
cpu40, cpu41, cpu42, cpu43, cpu44,
cpu45, cpu46, cpu47, cpu48, cpu49,
cpu50, cpu51, cpu52, cpu53, cpu54,
cpu55, cpu56, cpu57, cpu58, cpu59,
cpu60, cpu61, cpu62, cpu63, cpu64,
cpu65, cpu66, cpu67, cpu68, cpu69,
cpu70, cpu71, cpu72, cpu73, cpu74,
cpu75, cpu76, cpu77, cpu78, cpu79,
[Total cpus for MCM/QUAD(0)=80]
MCM/QUAD(8) contains:
cpu80, cpu81, cpu82, cpu83, cpu84,
cpu85, cpu86, cpu87, cpu88, cpu89,
cpu90, cpu91, cpu92, cpu93, cpu94,
```

```
cpu95, cpu96, cpu97, cpu98, cpu99,
cpu100, cpu101, cpu102, cpu103, cpu104,
cpu105, cpu106, cpu107, cpu108, cpu109,
cpu110, cpu111, cpu112, cpu113, cpu114,
cpu115, cpu116, cpu117, cpu118, cpu119,
cpu120, cpu121, cpu122, cpu123, cpu124,
cpu125, cpu126, cpu127, cpu128, cpu129,
cpu130, cpu131, cpu132, cpu133, cpu134,
cpu135, cpu136, cpu137, cpu138, cpu139,
cpu140, cpu141, cpu142, cpu143, cpu144,
cpu145, cpu146, cpu147, cpu148, cpu149,
cpu150, cpu151, cpu152, cpu153, cpu154,
cpu155, cpu156, cpu157, cpu158, cpu159,
[Total cpus for MCM/QUAD(8)=80]

Total number of MCMs/QUADs found  = 2
Total number of COREs found       = 20
Total number of CPUs found        = 160
cpuset for process 95014 (1 = in the set, 0 = not included)
cpu0  = 1
cpu1  = 1
cpu2  = 1
cpu3  = 1
cpu4  = 1
cpu5  = 1
cpu6  = 1
cpu7  = 1
cpu8  = 0
cpu9  = 0
cpu10 = 0
cpu11 = 0
cpu12 = 0
cpu13 = 0
cpu14 = 0
cpu15 = 0
<... Output Omitted ...>
MCM/QUAD(0) contains:
cpu0, cpu1, cpu2, cpu3, cpu4,
cpu5, cpu6, cpu7, cpu8, cpu9,
cpu10, cpu11, cpu12, cpu13, cpu14,
cpu15, cpu16, cpu17, cpu18, cpu19,
cpu20, cpu21, cpu22, cpu23, cpu24,
cpu25, cpu26, cpu27, cpu28, cpu29,
cpu30, cpu31, cpu32, cpu33, cpu34,
cpu35, cpu36, cpu37, cpu38, cpu39,
cpu40, cpu41, cpu42, cpu43, cpu44,
cpu45, cpu46, cpu47, cpu48, cpu49,
cpu50, cpu51, cpu52, cpu53, cpu54,
cpu55, cpu56, cpu57, cpu58, cpu59,
cpu60, cpu61, cpu62, cpu63, cpu64,
cpu65, cpu66, cpu67, cpu68, cpu69,
cpu70, cpu71, cpu72, cpu73, cpu74,
cpu75, cpu76, cpu77, cpu78, cpu79,
[Total cpus for MCM/QUAD(0)=80]
MCM/QUAD(8) contains:
```

```
cpu80, cpu81, cpu82, cpu83, cpu84,
cpu85, cpu86, cpu87, cpu88, cpu89,
cpu90, cpu91, cpu92, cpu93, cpu94,
cpu95, cpu96, cpu97, cpu98, cpu99,
cpu100, cpu101, cpu102, cpu103, cpu104,
cpu105, cpu106, cpu107, cpu108, cpu109,
cpu110, cpu111, cpu112, cpu113, cpu114,
cpu115, cpu116, cpu117, cpu118, cpu119,
cpu120, cpu121, cpu122, cpu123, cpu124,
cpu125, cpu126, cpu127, cpu128, cpu129,
cpu130, cpu131, cpu132, cpu133, cpu134,
cpu135, cpu136, cpu137, cpu138, cpu139,
cpu140, cpu141, cpu142, cpu143, cpu144,
cpu145, cpu146, cpu147, cpu148, cpu149,
cpu150, cpu151, cpu152, cpu153, cpu154,
cpu155, cpu156, cpu157, cpu158, cpu159,
[Total cpus for MCM/QUAD(8)=80]

Total number of MCMs/QUADs found  = 2
Total number of COREs found       = 20
Total number of CPUs found        = 160
cpuset for process 95015 (1 = in the set, 0 = not included)
cpu0  = 0
cpu1  = 0
cpu2  = 0
cpu3  = 0
cpu4  = 0
cpu5  = 0
cpu6  = 0
cpu7  = 0
cpu8  = 1
cpu9  = 1
cpu10 = 1
cpu11 = 1
cpu12 = 1
cpu13 = 1
cpu14 = 1
cpu15 = 1
cpu16 = 0
cpu17 = 0
cpu18 = 0
cpu19 = 0
cpu20 = 0
cpu21 = 0
cpu22 = 0
cpu23 = 0
<... Output Omitted ...>
```

Considerations about CUDA-aware MPI

The IBM PE runtime implements a CUDA-aware MPI mechanism, but it is disabled by default. Use the `MP_CUDA_AWARE=yes` variable to enable it. This topic is further described in 6.5.3, “Hybrid MPI and CUDA programs with IBM Parallel Environment” on page 190.

7.2.2 Managing application

IBM PE Runtime provides a set of commands to manage stand-alone **poe** jobs. This section introduces the most common commands.

Canceling an application

Because **poe** command handles signal of all tasks in the partition, sending an interrupt (SIGINT) or terminate (SIGTERM) signal will trigger it to all remote processes. If **poe** runs with 100413 pid, you can kill the program with the following command:

```
$ kill -s SIGINT 100413
```

However, if some remote processes are orphan, use the **poekill** *program_name* to kill all remaining tasks. In fact, **poekill** is capable of sending any signal to all the remote processes.

Suspend and resume a job

Likewise, to cancel a **poe** process, suspend and resume applications by way of signals. Use a **poekill** or **kill** command to send a `SIGTSTP` to suspend the poe process (which will trigger the signal to all tasks).

The application can be resumed by sending a `SIGCONT` to continue **poe**. Use a **poekill**, **kill**, **fg**, or **bg** command to deliver the signal.

7.2.3 Running OpenSHMEM programs

Some environment variables can be set to run an *OpenSHMEM* program with the IBM PE runtime through a **poe** command:

- *MP_MSG_API=shmem*

  Instructs **poe** to use *openSHMEM* message passing API.

- *MP_USE_BULK_XFER=yes*

  Enables exploitation of RDMA in Parallel Active Messaging Interface (PAMI).

MP_PROCS=<num> can be used to set the number of Processing Elements (PE).

NAS Parallel Benchmarks with OpenSHMEM

The NAS Parallel Benchmarks[3] (NPB) is a well-known suite of parallel applications that are often used to evaluate high performance computers. The benchmark provides programs, kernels, and problem solvers that simulate aspects such as computation, data movement, and I/O of real scientific applications. You can select the size of the workload each benchmark is going to process among a list of classes (A, B, C, and so on).

A version of NPB rewritten by using *OpenSHMEM* in C and Fortran has been released by the openshmem.org. NPB3.2-SHMEM is the implementation of NPB 3.2 and provides some benchmarks in Fortran and only one in C as shown in Example 7-5.

Example 7-5 openshmen-npbs implementation

```
$ git clone https://github.com/openshmem-org/openshmem-npbs
$ cd openshmem-npbs/C
$ cd config
$ cp suite.def.template suite.def
$ cp make.def.template make.def
```

[3] Learn more about NAS Parallel Benchmarks at http://www.nas.nasa.gov/publications/npb.html.

```
# Set CC in make.def
$ cd ../
$ make is NPROCS=2 CLASS=A
make suite
$ cd bin
$ ls
host.list  is.A.2
[developer@redbook01 bin]$ MP_RESD=poe oshrun -np 2 ./is.A.2
```

This section uses the Integer Sort kernel implementation of NPB3.2-SHMEM to demonstrate the use of OpenSHMEM with IBM PE, and the impact of some configurations on the performance of the application.

The Integer Sort benchmark was compiled with the **oshcc** compiler script of the IBM PE and the IBM XL C compiler.

The workload class C of the Integer Sort benchmark was executed as shown in Example 7-5 on page 239.

7.3 Using the IBM Spectrum LSF

The IBM Spectrum LSF is a load and a resources manager that allows shared access to cluster hosts while maximizing occupancy and efficiency on use of resources.

Spectrum LSF provides a queue-based and policy-driven scheduling system for a user's batch jobs that employs mechanisms to optimize resource selection and allocation based on the requirements of the application.

All development models that are described in Chapter 6, "Application development and tuning" on page 155 are fully supported by Spectrum LSF. The preferred way to run applications in a production cluster is by using the Spectrum LSF job submission mechanism. Also, you can manage any job. This section shows how to submit and manage jobs by using Spectrum LSF commands.

7.3.1 Submit jobs

This section describes the job submission process. Tools to monitor jobs and queues are introduced in Chapter 8, "Cluster monitoring" on page 247.

To submit a job to Spectrum LSF, use the **bsub** command. Spectrum LSF allows you to submit by using command-line options, interactive command-line mode, or a control file. The tool provides a rich set of option that allows fine-grained job management:

- Control input and output parameters
- Define limits
- Specify submission properties
- Notify users
- Control scheduling and dispatch
- Specify resources and requirements

The simplest way is to use the command-line options as shown in Example 7-6.

Example 7-6 Spectrum LSF bsub command to submit a job by using command-line options

```
$ bsub -o %J.out -e %J.err -J 'omp_affinity' -q short './affinity'
Job <212> is submitted to queue <short>.
```

Example 7-6 shows some basic options of the **bsub** command. The `-o`, `-i`, and `-e` flags specify standard output, input, and error files, respectively. In the example, `-J` sets the job name, but it can also be used to submit multiple jobs (also know as an array of jobs). and `-q` sets the queue that it can be part of. If the `-q` flag is not specified, the default queue is used (usually the `normal` queue). The last option in Example 7-6 on page 241 is the application to be executed.

The use of the shell script is convenient when you have to submit jobs regularly or that require many parameters. The file content shown in Example 7-7 is a regular shell script that embodies special comments (lines starts with `#BSUB`) to control the behavior of **bsub** command, and executes the **noop** application by using the `/bin/sh` interpreter.

Example 7-7 Shell script to run Spectrum LSF batch job

```
#!/bin/sh

#BSUB -o %J.out -e %J.err
#BSUB -J serial

./noop
```

Either **bsub myscript** or **bsub < myscript** commands can be issued to submit a script to Spectrum LSF. In the first case, `myscript` is not spooled, which means that changes on the script will take effect as long as the job is still executing. On the other side, **bsub < myscrypt** (see Example 7-8) is going to spool the script.

Example 7-8 Spectrum LSF bsub command to submit a script job

```
$ bsub < noop_lsf.sh
Job <220> is submitted to default queue <normal>.
```

Considerations for OpenMP programs

You can use environment variables to control the execution of OpenMP applications as explained in 7.1.1, “Running OpenMP applications” on page 230. By default, the **bsub** command propagates all variables on the submitting host to the environment on the target machine.

Example 7-9 shows some OpenMP control variables (OMP_DISPLAY_ENV, OMP_NUM_THREADS, and OMP_SCHEDULE) which are exported on the environment before a job is scheduled to run on the `p8r2n2` host. The content of `230.err` file, where errors were logged, shows that those variables are indeed propagated on the remote host.

Example 7-9 Exporting OpenMP variables to Spectrum LSF bsub command

```
$ export OMP_DISPLAY_ENV=true
$ export OMP_NUM_THREADS=20
$ export OMP_SCHEDULE="static"
$ bsub -m "p8r2n2" -o %J.out -e %J.err ./affinity
Job <230> is submitted to default queue <normal>.
$ cat 230.err
```

```
OPENMP DISPLAY ENVIRONMENT BEGIN
  _OPENMP = '201307'
  OMP_DYNAMIC = 'FALSE'
  OMP_NESTED = 'FALSE'
  OMP_NUM_THREADS = '20'
  OMP_SCHEDULE = 'STATIC'
  OMP_PROC_BIND = 'FALSE'
  OMP_PLACES = ''
  OMP_STACKSIZE = '70368222510890'
  OMP_WAIT_POLICY = 'PASSIVE'
  OMP_THREAD_LIMIT = '4294967295'
  OMP_MAX_ACTIVE_LEVELS = '2147483647'
  OMP_CANCELLATION = 'FALSE'
  OMP_DEFAULT_DEVICE = '0'
OPENMP DISPLAY ENVIRONMENT END
```

If you do not want to export OpenMP environment variables, then the **-env** option can be used to control how **bsub** propagates them. As an example, that same results in Example 7-9 on page 241 can be achieved with the following command, but without exporting any variable:

```
$ bsub -m "p8r2n2" -o %J.out -e %J.err -env "all, OMP_DISPLAY_ENV=true,
OMP_NUM_THREADS=20,  OMP_SCHEDULE='static'" ./affinity
```

As a last example, Example 7-10 shows how the environment variables can be set in a job script to control the OpenMP behavior.

Example 7-10 Exporting OpenMP variables to Spectrum LSF job script

```
#!/bin/bash

#BSUB -J "openMP example"
#BSUB -o job_%J.out -e job_%J.err
#BSUB -q short
#BSUB -m p8r2n2

export OMP_NUM_THREADS=20
export OMP_SCHEDULE=static
export OMP_DISPLAY_ENV=true

./affinity
```

Considerations for MPI programs

Use the **-n** option of the **bsub** command to allocate the number of tasks (or job slots) for the parallel application. Note that depending on the configuration of Spectrum LSF, job slots can be set in terms of CPUs in the cluster. As an example, the following command submits an MPI job with six tasks:

```
$ bsub -n 6 -o %J.out -e %J.err poe ./helloMPI
```

You can select a set of hosts for a parallel job by using **bsub** command options:

- Use the **-m** option to namely select hosts or groups of host.
- Resources-based selection with requirements expressions (-R option).
- Indicate a host file by using the -hostfile option. Do not use with **-m** or **-R** options.

The following examples show the usage of **-m** and **-R**, respectively, to allocate hosts:

```
$ bsub -n 2 -m "p8r1n1! p8r2n2" -o %J.out -e %J.err poe ./myAPP
```

Run two tasks MPI application on hosts p8r1n1 and p8r2n2. The symbol *!* indicates that **poe** will be first executed on `p8r1n1`.

```
$ bsub -n 4 -R "select[ncores==20] same[cpuf]" -o %J.out -e %J.err poe ./myApp
```

Run four tasks MPI application on hosts with 20 CPU cores (`select[ncores==20]`) as long as they have same CPU factor (`same[cpuf]`).

The job locality can be specified with the `span` string in a resources requirement expression (`-R` option). You can use either one of the following formats to specify the locality:

- `span[hosts=1]`: Set to run all tasks on same host
- `span[ptile=n]`: Where *n* is an integer that sets the number of tasks per host
- `span[block=size]`: Where *size* is an integer that sets the block size

Job task affinity is enabled with the use of the `affinity` string in the resources requirement expression (`-R` option). The affinity applies to either CPU and memory, and is defined in terms of processor units assigned per task, being either `core`, `numa`, `socket`, and `task`. The following example has a 10 task MPI application, allocated five per host, and each with four designated coreswith binding by threads:

```
$ bsub -n 10 -R "select[ncores >= 20] span[ptile=5]
affinity[core(4):cpubind=thread]"  -o %J.out -e %J.err
```

Further processor unit specification makes the affinity expression powerful. See the following website for a detailed explanation of affinity expressions in Spectrum LSF:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/affinity_res_req_string.dita

Spectrum LSF integrates nicely with the IBM Parallel Environment (PE) Runtime Edition and supports execution of parallel applications through POE. Some configurations in Spectrum LSF are required. For more information, see 5.6.13, “IBM Spectrum LSF” on page 134.

Example 7-11 has a Spectrum LSF job script to execute an MPI program with **poe**. Notice that the IBM PE environment variables are going to take effect.

Example 7-11 Spectrum LSF job script to submit a simple IBM PE application

```
#!/bin/bash

#BSUB -J "MPILocalRank"    # Set job name
#BSUB -o lsf_job-%J.out    # Set output file
#BSUB -e lsf_job-%J.err    # Set error file
#BSUB -q short             # Set queue
#BSUB -n 5                 # Set number of tasks

export MP_INFOLEVEL=1
export LANG=en_US
export MP_RESD=POE
poe ./a.out
```

Spectrum LSF provides a native network-aware scheduling of the IBM PE parallel application through the `-network` option of the **bsub** command. That option encloses the attributes that are described in Table 7-1.

Table 7-1 Spectrum LSF bsub network attributes for IBM PE

Attribute	Description	Values
`type`	Manage network windows reservation	► `sn_single` (reserves windows from one network for each task) ► `sn_all` (reserve windows from all networks for each task)
`protocol`	Set the messaging API in use	► `mpi` ► `shmem` ► `pami`
`mode`	The network type	► `US` ► `IP`
`usage`	The network adapter usage among processes	► `shared` ► `dedicated`
`instance`	Number of instances for reservation window	Positive integer number

The following command submits a two tasks (`-n 2`) MPI (`protocol=mpi`) job. It shares the network adapters (`usage=shared`) and reserves windows on all of them.

```
$ bsub -n 2 -R "span[ptile=1]" -network "protocol=mpi:type=sn_all:
instances=2:usage=shared" poe ./myApp
```

Considerations for CUDA programs

You can take advantage of graphics processing unit (GPU) resources mapping when either using requirements expressions to allocate hosts and express usage reservation. If Spectrum LSF is configured as shown in 5.6.13, “IBM Spectrum LSF” on page 134, then the following resource fields are available:

- `ngpus`: Total number of GPUs
- `ngpus_shared`: Number of GPUs in share mode
- `ngpus_excl_t`: Number of GPUs in exclusive thread mode
- `ngpus_excl_p`: Number of GPUs in exclusive process mode

The `ngpus` resource field can be used in requirement expressions (`-R` option) for demanding the number of GPUs needed to execute a CUDA application. The following command submits a job to any host with one or more GPU:

```
$ bsub -R "select [ngpus > 0]" ./cudaApp
```

In terms of usage, it can reserve GPUs by setting *ngpus_shared* (number of shared), *ngpus_excl_t* (number of GPUs on exclusive thread mode), *ngpus_excl_p* (number of GPUs on exclusive process mode) resources. Use the `rusage` with `-R` option of the **bsub** command to reserve GPU resources.

In Example 7-12 the job script sets one GPU in shared mode, and use by a `cudaCUDA` application.

Example 7-12 Script to set one GPU in shared mode to be used by a cudaCUDA application

```
#!/bin/bash

#BSUB -J "HelloCUDA"
#BSUB -o helloCUDA_%J.out -e helloCUDA_%J.err
#BSUB -R "select [ngpus > 0] rusage [ngpus_shared=1]"

./helloCUDA
```

When it comes to exclusive use of GPUs by parallel applications, set the value of `ngpus_excl_t` or `ngpus_excl_p` to change run mode properly. The following example executes a parallel two tasks (`-n 2`) application in one host (`span[hosts=1]`), where each host reserves two GPUs on exclusive process mode (`rusage[ngpus_excl_p=2]`):

```
$ bsub -n 2 -R "select[ngpus > 0] rusage[ngpus_excl_p=2] span[hosts=1]" poe
./mpi-GPU_app
```

Considerations for OpenSHMEM

The `-network` option of the **bsub** command can set the parallel communication protocol of the application. For an OpenSHMEM application, it can be set using this command:

```
$ bsub -n 10 -network "protocol=shmem" poe ./shmemApp
```

7.3.2 Manage jobs

Spectrum LSF provides a set of commands to manage batch jobs. This section introduces the most common commands.

Modifying a job

The batch job can assume several statuses throughout its lifecycle, of which these are the most common:

- PEND: Waiting to be scheduled status
- RUN: Running status
- PSUSP, USUSP or SSUSP: Suspended status
- DONE or EXIT: Terminated status

It is possible to modify submission options of a job either in pending or running status. To change it, use the **bmod** command.

Most of submission options can be changed with **bmod**. It is allowed to operate in an option to cancel, reset to its default value or override it.

To override a submission parameter, use the same option as of **bsub**. In the following example, -o "`%J_noop.out`" changes the output file of the job with identifier 209:

```
$ bmod -o "%J_noop.out" 209
Parameters of job <209> are being changed
```

To cancel a submission parameter, append *n* to the option. For example, **-Mn** removes the memory limits.

Because the **bmod** command is flexible and has many options, read its manual, which is located in the following website:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_command_ref/bmod.1.dita

Canceling a job

To kill one or more jobs, use the **bkill** command. This command by default on Linux sends the SIGINT, SIGTERM, and SIGKILL signals in sequence. The time interval can be configured in the lsb.params configuration file. In reality, **bkill -s** <*signal*> sends the <*signal*> signal to the job.

The user can only kill their own jobs. The root and Spectrum LSF administrators can terminate any job.

In the following example the job with identifier 217 is terminated:

```
$ bkill 217
Job <217> is being terminated
```

Suspend and resume a job

To suspend one or more unfinished jobs, use the **bstop** command. In Linux, it sends the SIGSTOP and SIGTSTP signal to, respectively, serial and parallel jobs. As an alternative, the **bkill -s SIGSTOP** or the **bkill -s SIGTSTP** commands send the stop signal to a job.

In Example 7-13, the job with identifier 220 had not finished (status RUN) when it was stopped (bstop 220). As a result, bjobs 220 shows it is now in a suspended status (USUSP).

Example 7-13 Spectrum LSF bstop command to suspend a job

```
$ bjobs 220
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
220     wainers RUN   normal     p8r1n1      p8r2n2      serial     Apr 17 16:06
$ bstop 220
Job <220> is being stopped
$ bjobs 220
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
220     wainers USUSP normal     p8r1n1      p8r2n2      serial     Apr 17 16:06
```

To resume a job, use the **bresume** command. In Linux, it sends a SIGCONT signal to the suspended job. To stop or kill a job, **bkill -s** can be used to send the continue signal **bkill -s CONT**. The following command shows how to resume the job stopped in Example 7-13:

```
$ bresume 220
Job <220> is being resumed
```

8

Cluster monitoring

Monitoring is an important activity to assist in the maintenance of cluster resources, and to ensure its serviceability to end users. It is composed of resource control and health check tasks on the various nodes (compute, login, services), networking devices, and storage systems.

This topic is vast and deserves an entire book itself. Instead of covering this subject in detail, this chapter introduces some tools and resources that can be employed to support monitoring of a high performance computing (HPC) cluster.

This chapter includes the following topics:

- IBM Spectrum LSF tools for monitoring
- nvidia-smi tool for monitoring GPU

8.1 IBM Spectrum LSF tools for monitoring

IBM Spectrum Load Sharing Facility (LSF) provides a comprehensive set of tools that can be employed for monitoring the cluster. This section shows how to use some of these tools to carry out common tasks. The following list summarizes the commands that are used in this section:

- **lsclusters**: Lists all configured clusters
- **lsid**: Displays the current LSF version number, the cluster name, and master host name
- **lshosts**: Displays hosts information
- **lsload**: Displays load-per-host information
- **bhosts**: Displays information about batches
- **bjobs**: Displays information about jobs
- **bqueues**: Displays information about batch queues
- **bparams**: Displays batch parameters
- **badmin**: Provides a set of administrative subcommands for batches
- **lsadmin**: Provides a set of administrative subcommands for hosts

For a complete list of commands and usage guides, see the Spectrum LSF manual:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_kc_cmd_ref.dita

8.1.1 General information about clusters

The **lsclusters** command shows information about the clusters managed by the Spectrum LSF master host server as shown in Example 8-1.

Example 8-1 Spectrum LSF lsclusters command to gather information about the clusters

```
$ lsclusters
CLUSTER_NAME   STATUS   MASTER_HOST                ADMIN    HOSTS  SERVERS
lsf-cluster    ok       p8r1n1                  lsfadmin        2        2
```

As shown in Example 8-2, the **lsid** command is used to display the cluster name, master host server, and version of Spectrum LSF.

Example 8-2 Spectrum LSF lsid command to display cluster name and master host

```
[wainersm@p8r2n2 ~]$ lsid
IBM Platform LSF Standard 9.1.3.0, Mar 31 2015
Copyright IBM Corp. 1992, 2014. All rights reserved.
US Government Users Restricted Rights - Use, duplication or disclosure restricted
by GSA ADP Schedule Contract with IBM Corp.

My cluster name is lsf-cluster
My master name is p8r1n1
```

Use the **badmin** command with the **showstatus** subcommand as shown in Example 8-3 to get an overview of servers, users, groups, and jobs in the cluster. Notice that **badmin** is a suite of commands to manage batch-related configuration and daemons. More of its features are shown later in this chapter.

Example 8-3 Spectrum LSF badmin command to show master host batch daemon status

```
$ badmin showstatus
LSF runtime mbatchd information
```

```
Available local hosts (current/peak):
    Clients:                0/0
    Servers:                2/2
       CPUs:                2/2
      Cores:                40/40
      Slots:                unlimited/unlimited

Number of servers:          2
    Ok:                     2
    Closed:                 0
    Unreachable:            0
    Unavailable:            0

Number of jobs:             1
    Running:                1
    Suspended:              0
    Pending:                0
    Finished:               0

Number of users:            4
Number of user groups:      1
Number of active users:     1

Latest mbatchd start:       Wed Apr 13 21:21:33 2016
Active mbatchd PID:         4617

Latest mbatchd reconfig:    -
```

8.1.2 Getting information about hosts

The Spectrum LSF master host obtains static and dynamic information about the slave hosts of the cluster. Static information means properties that are hard to change such as system memory, disk capacity, topology, and number of CPUs. In contrast, dynamic information depends on current workload on the system, such as the amount of memory utilized, amount of swap memory, and jobs scheduled to run.

By default the commands that are highlighted in this section display information about all hosts of the cluster, unless a host name is specified. Also, a subset of hosts can be selected by using *resources requirement expressions*. Example 8-4 shows an expression (specified as an argument of the **-R** option) to select hosts whose model matches `POWER8`.

Example 8-4 Spectrum LSF lshosts command to show slave hosts selected by the expression

```
$ lshosts -R "select[model==POWER8]"
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r2n2      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
```

Resource requirements can also be used in scheduling of jobs and with other Spectrum LSF commands. The language to express requirements is covered in the Spectrum LSF manual, which can be found at the following address:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/res_req_strings_about.dita

By default the **lshost** command displays static information about all hosts that are configured in the local cluster as shown in Example 8-5. The columns `cpuf`, `ncpus`, `maxmem`, and `maxswp` display the CPU performance factor, number of CPUs, maximum memory, and maximum swap memory.

Example 8-5 Spectrum LSF lshosts command shows static information of hosts

```
$ lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r2n2      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
```

More information about individual hosts can be queried by passing the **-l** option to **lshost** command as shown in Example 8-6. Notice that besides the static information, the **-l** option reports the current load status of the host.

Example 8-6 Spectrum LSF lshosts command displays static information about a specified host

```
$ lshosts -l p8r2n2

HOST_NAME:  p8r2n2
type         model  cpuf ncpus ndisks maxmem maxswp maxtmp  rexpri server nprocs ncores nthreads
LINUXPPC64  POWER8 250.0   160      1   256G   3.9G 949020M      0    Yes      1     20        8

RESOURCES: (mg)
RUN_WINDOWS:  (always open)

LOAD_THRESHOLDS:
  r15s   r1m  r15m   ut    pg    io   ls   it   tmp   swp   mem   pnsd pe_network
     -   3.5     -    -     -     -    -    -     -     -     -      -          -
```

The **lshosts** command with **-T** option shows the nonuniform memory access (NUMA) topology of the hosts. In Example 8-7, the **lshosts** command reports a two-node (`0` and `8`) S822LC host (`p8r2n2`), with 128 GB of memory available per node (for a total of 256 GB). It also shows 10 processor cores per NUMA node, each with 8 CPUs (SMT8 mode).

Example 8-7 Spectrum LSF lshosts command shows NUMA topology of a specified host

```
[wainersm@p8r2n2 ~]$ lshosts -T p8r2n2
Host[256G] p8r2n2
    NUMA[0: 128G]
        core(0 1 2 3 4 5 6 7)
        core(8 9 10 11 12 13 14 15)
        core(16 17 18 19 20 21 22 23)
        core(24 25 26 27 28 29 30 31)
        core(32 33 34 35 36 37 38 39)
        core(40 41 42 43 44 45 46 47)
        core(48 49 50 51 52 53 54 55)
        core(56 57 58 59 60 61 62 63)
        core(64 65 66 67 68 69 70 71)
        core(72 73 74 75 76 77 78 79)
    NUMA[8: 128G]
        core(80 81 82 83 84 85 86 87)
        core(88 89 90 91 92 93 94 95)
        core(96 97 98 99 100 101 102 103)
        core(104 105 106 107 108 109 110 111)
        core(112 113 114 115 116 117 118 119)
```

```
        core(120 121 122 123 124 125 126 127)
        core(128 129 130 131 132 133 134 135)
        core(136 137 138 139 140 141 142 143)
        core(144 145 146 147 148 149 150 151)
        core(152 153 154 155 156 157 158 159)
```

In contrast to the **lshost** command that provides only static information, the **lsload** command gives dynamic information for the hosts. Example 8-8 shows load information for all hosts in the cluster.

Example 8-8 Spectrum LSF lsload command reports dynamic information for the hosts

```
[wainersm@p8r2n2 ~]$ lsload
HOST_NAME       status  r15s   r1m  r15m   ut    pg  ls    it   tmp   swp   mem
p8r2n2              ok   0.1   0.0   0.0   0%   0.0   1     0  920G  3.9G  246G
p8r1n1              ok   1.0   0.0   0.1   0%   0.0   0  8518  920G  3.9G  245G
```

The following are the means of each column header in the report:

- `status` reports the current status of the host. This columen has these possible values:
  - `ok`: The host is ready to accept remote jobs.
  - `-ok`: The LIM daemon is running, but the RES daemon is unreachable.
  - `busy`: The host is overloaded.
  - `lockW`: The host is locked by its run window.
  - `lockU`: The host is locked by the LSF administrator or root.
  - `unavail`: The host is down or the LIM daemon is not running.
- `r15s`, `r1m`, and `r15m` indicate that the CPU load averaged exponentially over the last 15 seconds, 1 minute, and 15 minutes, respectively.
- `ut` is the CPU utilization time averaged exponentially over the last minute.
- `pg` is the memory paging rate averaged exponentially over the last minute.
- `ls` is the number of current login users.
- `it` shows the idle time of the host.
- `tmp` is the amount of free space in the `/tmp` directory.
- `swp` is the amount of free swap space.
- `mem` is the amount of memory available on the host.

Other options can be used with the **lsload** command to display different information. For example, the **-l** option displays network resources information for scheduling IBM Parallel Environment (PE) jobs and also the disk I/O rate.

8.1.3 Getting information about jobs and queues

Some commands are available to monitor the workload in the cluster, and give information on jobs and queue status. The **bhosts** command reports jobs statistics per-host, which is useful to see overall cluster workload as shown in Example 8-9.

Example 8-9 Spectrum LSF bhosts to report on hosts batch jobs status

```
$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
```

```
p8r1n1            ok              -     -     1     1     0     0     0
p8r2n2            ok              -     -     0     0     0     0     0
```

To show unfinished jobs in the entire cluster, use the **bjobs** command (see Example 8-10). You can use the **-a** option with the **bjobs** command to view all recently finished jobs in addition to those jobs still running.

Example 8-10 Spectrum LSF bjobs command to show unfinished jobs

```
$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
209     wainers RUN   normal     p8r1n1      p8r1n1      serial     Apr 14 17:18
```

The **bjobs** command can also be used to report detailed information about a job as shown in Example 8-11.

Example 8-11 Spectrum LSF bjobs command to show detailed information about a job

```
$ bjobs -l 209

Job <209>, Job Name <serial>, User <wainersm>, Project <default>, Status <RUN>,
                     Queue <normal>, Command <#!/bin/sh; #BSUB -o %J.out -e %J
                    .err;#BSUB -J serial; ./noop>, Share group charged </waine
                    rsm>
Thu Apr 14 17:18:41: Submitted from host <p8r1n1>, CWD <$HOME/lsf/noop>, Output
                     File <209.out>, Error File <209.err>;
Thu Apr 14 17:18:42: Started 1 Task(s) on Host(s) <p8r1n1>, Allocated 1 Slot(s)
                     on Host(s) <p8r1n1>, Execution Home </home/wainersm>, Exe
                    cution CWD </home/wainersm/lsf/noop>;
Thu Apr 14 17:22:41: Resource usage collected.
                    MEM: 22 Mbytes;  SWAP: 0 Mbytes;  NTHREAD: 5
                    PGID: 82953;  PIDs: 82953 82954 82958 82959


 MEMORY USAGE:
 MAX MEM: 22 Mbytes;  AVG MEM: 22 Mbytes

 SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -

 RESOURCE REQUIREMENT DETAILS:
 Combined: select[type == local] order[r15s:pg]
 Effective: select[type == local] order[r15s:pg]
```

The **bqueues** command displays information about the queues of the jobs available in a cluster along with use statistics. The command list queues are sorted by priority (`PRIO`). It also reports their status (`STATUS`), maximum number of job slots (`MAX`), maximum jobs slots per users (`JL/U`), maximum job slots per processors (`JL/P`), maximum job slots per slot (`JL/H`), number of tasks for jobs (`NJOBS`), and number of jobs pending (`PEN`), running (`RUN`) and suspended (`SUSP`).

Example 8-12 shows the ouput of a **bqueues** command.

Example 8-12 Spectrum LSF bqueues command to show available job queues

```
[wainersm@p8r1n1 ~]$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
admin            50  Open:Active       -    -    -    -     0     0     0     0
owners           43  Open:Active       -    -    -    -     0     0     0     0
priority         43  Open:Active       -    -    -    -     0     0     0     0
night            40  Open:Inact        -    -    -    -     0     0     0     0
chkpnt_rerun_qu  40  Open:Active       -    -    -    -     0     0     0     0
short            35  Open:Active       -    -    -    -     0     0     0     0
license          33  Open:Active       -    -    -    -     0     0     0     0
normal           30  Open:Active       -    -    -    -     1     0     1     0
interactive      30  Open:Active       -    -    -    -     0     0     0     0
hpc_linux        30  Open:Active       -    -    -    -     0     0     0     0
hpc_linux_tv     30  Open:Active       -    -    -    -     0     0     0     0
idle             20  Open:Active       -    -    -    -     0     0     0     0
```

8.1.4 Administering the cluster

HPC cluster administration involves many non-trivial tasks, although tools like Spectrum LSF simplify the process. This section shows just a small part of the Spectrum LSF capabilities and tools. For more information, see the administration section of the user guide available at the following website:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_kc_managing.dita

The Spectrum LSF configuration directory has a few files where global and per-cluster basis configurations are defined. Its path depends on the installation performed by the administrator and the `LSF_ENVDIR` environment variable value:

```
$ echo $LSF_ENVDIR
/usr/share/lsf/conf
```

The bare minimum Spectrum LSF configuration directory includes the following files:

- `lsf.shared`: Contains the cluster names and definitions that can be referenced by configuration files
- `lsf.conf`: Contains global configurations that are shared among clusters defined on `lsf.shared`.
- `lsf.cluster.<cluster-name>`: Contain per cluster name-specific configurations such as the list of hosts and their attributes, administrators, and resources mapping.

Other configuration files exist. For example, `lsf.queue` is the file where batch queues are defined.

Typically, any change requires restarting the Spectrum LSF servers and daemons, so they are usually followed by running **badmin reconfig** and **lsadmin reconfig** commands so that the new configurations take effect.

Managing batch services

To check all batch configuration parameters, use the **bparams** command. Without any options, it displays basic information as shown in Example 8-13. `MBD_SLEEP_TIME` is the jobs dispatch interval.

Example 8-13 Spectrum LSF bparams command to show basic batch configuration parameters

```
[wainersm@p8r1n1 ~]$ bparams
Default Queues:  normal interactive
MBD_SLEEP_TIME used for calculations:  10 seconds
Job Checking Interval:  7 seconds
Job Accepting Interval:  0 seconds
```

Example 8-14 lists configured batch parameters with the **bparams -l** command. Because the list of parameters is long, many of them were omitted from the output shown in Example 8-14.

Example 8-14 Spectrum LSF bparams to show detailed batch configuration parameters

```
$ bparams -l

System default queues for automatic queue selection:
    DEFAULT_QUEUE = normal interactive

Amount of time in seconds used for calculating parameter values:
    MBD_SLEEP_TIME = 10 (seconds)

The interval for checking jobs by slave batch daemon:
    SBD_SLEEP_TIME = 7 (seconds)

The interval for a host to accept two batch jobs:
    JOB_ACCEPT_INTERVAL = 0 (* MBD_SLEEP_TIME)

The idle time of a host for resuming pg suspended jobs:
    PG_SUSP_IT = 180 (seconds)

The amount of time during which finished jobs are kept in core memory:
    CLEAN_PERIOD = 3600 (seconds)

The maximum number of retries for reaching a slave batch daemon:
    MAX_SBD_FAIL = 3

<... Omitted output ...>
Resets the job preempted counter once this job is requeued, migrated, or rerun:
    MAX_JOB_PREEMPT_RESET = Y

Disable the adaptive chunking scheduling feature:
    ADAPTIVE_CHUNKING = N
```

Spectrum LSF uses default batch parameters unless the (optional) `lsb.params` file is deployed in `$LSF_ENVDIR/<cluster-name>/configdir` as a per-cluster configuration file. After any change to `lsb.params`, run the **badmin reconfig** command to reconfigure the mbatchd daemon.

The **badmin** administrative tool provides some debugging subcommands that can help determine problems with batch daemons. The following commands can be executed with the Spectrum LSF administrator user:

- sbddebug: Debug the slave batch daemon
- mbddebug: Debug the master batch daemon
- schddebug: Debug the scheduler batch daemon

As a case study, Example 8-15 shows the output of the **bhosts** command, where the p8r2n2 host is marked as unreachable.

Example 8-15 Spectrum LSF bhosts command to determine problem on batch services

```
[lsfadmin@p8r1n1 wainersm]$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
p8r1n1             ok              -      -      0      0      0      0      0
p8r2n2             unreach         -      -      0      0      0      0      0
```

From the master host (p8r1n1), neither **badmin mbddebug** nor **badmin schddebug** reported errors. Now, **badmin sbddebug** shows that the batch daemon is unreachable:

```
$ badmin sbddebug p8r2n2
failed : Slave batch daemon (sbatchd) is unreachable now on host p8r2n2
```

The above issues are fixed by restarting the slave batch daemon as shown in Example 8-16.

Example 8-16 Spectrum LSF badmin command to start up slave batch daemon

```
# badmin hstartup p8r2n2
Start up slave batch daemon on <p8r2n2> ? [y/n] y
Start up slave batch daemon on <p8r2n2> ...... done
```

As a result, the slave batch daemon is back online again:

```
$ bhosts p8r2n2
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
p8r2n2             ok              -      -      0      0      0      0      0
```

Managing the LIM and Remote Execution Server services

Spectrum LSF deploys the Load Information Manager (LIM) and Remote Execution Server servers in each host, where they run as daemons. They play these roles in the Spectrum LSF system:

- LIM collects load and configuration information about the host.
- Remote Execution Server provides execution services. It allows secure and transparent execution of jobs and tasks on the host.

The **lsadmin** tool provides subcommands to manage the LIM and Remote Execution Server in a single host or groups. Operations to start, stop, and restart the daemons are carried out by the limstartup, limshutdown, limrestart, resstartup, resshutdown and resrestart subcommands.

lsadmin provides a useful subcommand to check the correctness of LIM and Remote Execution Server configurations called **chkconfig:**

```
$ lsadmin ckconfig

Checking configuration files ...
No errors found.
```

The **badmin** command comes with **limdebug** and **resdebug** subcommands for debugging problems with LIM and RES daemons, respectively.

8.2 nvidia-smi tool for monitoring GPU

NVIDIA provides a tool to control the status and health of GPUs called the System Management Interface (nvidia-smi). The tool shows different levels of information depending on the generation of your card. Some options can be disabled and enabled when using this tool.

Example 8-17 shows default output for the example system with NVIDIA K80 cards.

Example 8-17 Default nvidia-smi output

```
$ nvidia-smi
+-----------------------------------------------------+
| NVIDIA-SMI 352.59     Driver Version: 352.59         |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 0000:03:00.0     Off |                    0 |
| N/A   31C    P8    27W / 175W |     55MiB / 11519MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K80           On   | 0000:04:00.0     Off |                    0 |
| N/A   26C    P8    26W / 175W |     55MiB / 11519MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla K80           On   | 0002:03:00.0     Off |                    0 |
| N/A   31C    P8    25W / 175W |     55MiB / 11519MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla K80           On   | 0002:04:00.0     Off |                    0 |
| N/A   27C    P8    28W / 175W |     55MiB / 11519MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

8.2.1 Information about jobs on GPU

When the system has jobs that use graphics processing unit (GPU) calls inside them, you can find information (name and PID of tasks, GPU number for each process, and GPU memory usage) about it in the bottom section of the default *nvidia-smi* call. This section is shown in Example 8-18 for a sample MPI run with 10 MPI tasks, where each task uses only one GPU with a number that is assigned in round-robin order.

Example 8-18 Process section of nvidia-smi output during MPI run

```
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
```

```
|=============================================================================|
|    0     23102    C   ./sample_mpi                                    971MiB |
|    1     23103    C   ./sample_mpi                                    834MiB |
|    2     23104    C   ./sample_mpi                                    971MiB |
|    3     23105    C   ./sample_mpi                                    834MiB |
|    0     23106    C   ./sample_mpi                                    971MiB |
|    1     23107    C   ./sample_mpi                                    834MiB |
|    2     23108    C   ./sample_mpi                                    971MiB |
|    3     23109    C   ./sample_mpi                                    834MiB |
|    0     23110    C   ./sample_mpi                                    971MiB |
|    1     23111    C   ./sample_mpi                                    834MiB |
+-----------------------------------------------------------------------------+
```

8.2.2 All GPU details

To show all information about GPUs inside your node, use the `-q` option. To list only data about specific GPU, specify the ID with the `-i` option. Example 8-19 provides details for the second GPU using these options.

Example 8-19 Detailed information about second GPU in system using nvidia-smi

```
$ nvidia-smi -i 1 -q

==============NVSMI LOG==============

Timestamp                           : Tue Dec 15 04:26:08 2015
Driver Version                      : 352.59

Attached GPUs                       : 4
GPU 0000:04:00.0
    Product Name                    : Tesla K80
    Product Brand                   : Tesla
    Display Mode                    : Disabled
    Display Active                  : Disabled
    Persistence Mode                : Enabled
    Accounting Mode                 : Disabled
    Accounting Mode Buffer Size     : 1920
    Driver Model
        Current                     : N/A
        Pending                     : N/A
    Serial Number                   : 0324914053157
    GPU UUID                        : GPU-3602edfc-edcb-9392-9fcc-b619254d8d2f
    Minor Number                    : 1
    VBIOS Version                   : 80.21.1B.00.02
    MultiGPU Board                  : No
    Board ID                        : 0x400
    Inforom Version
        Image Version               : 2080.0200.00.04
        OEM Object                  : 1.1
        ECC Object                  : 3.0
        Power Management Object     : N/A
    GPU Operation Mode
        Current                     : N/A
        Pending                     : N/A
    PCI
```

```
        Bus                         : 0x04
        Device                      : 0x00
        Domain                      : 0x0000
        Device Id                   : 0x102D10DE
        Bus Id                      : 0000:04:00.0
        Sub System Id               : 0x106C10DE
        GPU Link Info
            PCIe Generation
                Max                 : 3
                Current             : 1
            Link Width
                Max                 : 16x
                Current             : 16x
        Bridge Chip
            Type                    : N/A
            Firmware                : N/A
        Replays since reset         : 0
        Tx Throughput               : N/A
        Rx Throughput               : N/A
    Fan Speed                       : N/A
    Performance State               : P8
    Clocks Throttle Reasons
        Idle                        : Active
        Applications Clocks Setting : Not Active
        SW Power Cap                : Not Active
        HW Slowdown                 : Not Active
        Unknown                     : Not Active
    FB Memory Usage
        Total                       : 11519 MiB
        Used                        : 55 MiB
        Free                        : 11464 MiB
    BAR1 Memory Usage
        Total                       : 16384 MiB
        Used                        : 2 MiB
        Free                        : 16382 MiB
    Compute Mode                    : Default
    Utilization
        Gpu                         : 0 %
        Memory                      : 0 %
        Encoder                     : 0 %
        Decoder                     : 0 %
    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled
    ECC Errors
        Volatile
            Single Bit
                Device Memory       : 0
                Register File       : 0
                L1 Cache            : 0
                L2 Cache            : 0
                Texture Memory      : 0
                Total               : 0
            Double Bit
                Device Memory       : 0
```

```
            Register File       : 0
            L1 Cache            : 0
            L2 Cache            : 0
            Texture Memory      : 0
            Total               : 0
        Aggregate
            Single Bit
            Device Memory       : 0
            Register File       : 0
            L1 Cache            : 0
            L2 Cache            : 0
            Texture Memory      : 0
            Total               : 0
            Double Bit
            Device Memory       : 0
            Register File       : 0
            L1 Cache            : 0
            L2 Cache            : 0
            Texture Memory      : 0
            Total               : 0
    Retired Pages
        Single Bit ECC          : 0
        Double Bit ECC          : 0
        Pending                 : No
    Temperature
        GPU Current Temp        : 26 C
        GPU Shutdown Temp       : 93 C
        GPU Slowdown Temp       : 88 C
    Power Readings
        Power Management        : Supported
        Power Draw              : 26.44 W
        Power Limit             : 175.00 W
        Default Power Limit     : 149.00 W
        Enforced Power Limit    : 175.00 W
        Min Power Limit         : 100.00 W
        Max Power Limit         : 175.00 W
    Clocks
        Graphics                : 324 MHz
        SM                      : 324 MHz
        Memory                  : 324 MHz
    Applications Clocks
        Graphics                : 562 MHz
        Memory                  : 2505 MHz
    Default Applications Clocks
        Graphics                : 562 MHz
        Memory                  : 2505 MHz
    Max Clocks
        Graphics                : 875 MHz
        SM                      : 875 MHz
        Memory                  : 2505 MHz
    Clock Policy
        Auto Boost              : On
        Auto Boost Default      : On
    Processes                   : None
```

To print only specific information about the GPUs, use the `-d` option with the names of interesting sections. Example 8-20 provides only the details about temperature, power, and clocks using this option.

Example 8-20 Specific details of GPU using the -d option

```
$ nvidia-smi -i 1 -q -d POWER,TEMPERATURE,CLOCK

==============NVSMI LOG==============

Timestamp                           : Tue Dec 15 04:36:20 2015
Driver Version                      : 352.59

Attached GPUs                       : 4
GPU 0000:04:00.0
    Temperature
        GPU Current Temp            : 26 C
        GPU Shutdown Temp           : 93 C
        GPU Slowdown Temp           : 88 C
    Power Readings
        Power Management            : Supported
        Power Draw                  : 26.20 W
        Power Limit                 : 175.00 W
        Default Power Limit         : 149.00 W
        Enforced Power Limit        : 175.00 W
        Min Power Limit             : 100.00 W
        Max Power Limit             : 175.00 W
    Power Samples
        Duration                    : 118.07 sec
        Number of Samples           : 119
        Max                         : 26.44 W
        Min                         : 26.20 W
        Avg                         : 26.24 W
    Clocks
        Graphics                    : 324 MHz
        SM                          : 324 MHz
        Memory                      : 324 MHz
    Applications Clocks
        Graphics                    : 562 MHz
        Memory                      : 2505 MHz
    Default Applications Clocks
        Graphics                    : 562 MHz
        Memory                      : 2505 MHz
    Max Clocks
        Graphics                    : 875 MHz
        SM                          : 875 MHz
        Memory                      : 2505 MHz
    SM Clock Samples
        Duration                    : 3207.02 sec
        Number of Samples           : 100
        Max                         : 875 MHz
        Min                         : 324 MHz
        Avg                         : 472 MHz
    Memory Clock Samples
        Duration                    : 3207.02 sec
        Number of Samples           : 100
```

```
        Max                           : 2505 MHz
        Min                           : 324 MHz
        Avg                           : 1652 MHz
    Clock Policy
        Auto Boost                    : On
        Auto Boost Default            : On
```

For more information about the options of the **nvidia-smi** command, see the following website:

http://developer.download.nvidia.com/compute/cuda/6_0/rel/gdk/nvidia-smi.331.38.pdf

8.2.3 Compute modes

nvidia-smi can change compute modes of GPUs by using following command:

nvidia-smi -i <GPU_number> -c <compute_mode>

NVIDIA GPUs support the following compute modes:

- PROHIBITED: The GPU cannot compute applications and no contexts are allowed.
- EXCLUSIVE_THREAD: Only one process can be assigned to the GPU at a time and will only perform work from one thread of this process.
- EXCLUSIVE_PROCESS: Only one process can be assigned to the GPU at a time and different process threads can submit jobs to the GPU concurrently.
- DEFAULT: Multiple processes can use the GPU simultaneously and different process threads can submit jobs to the GPU concurrently.

8.2.4 Persistence mode

Persistence mode is a mode of the NVIDIA driver that helps to keep GPU initialized even when no processes are accessing the cards. This mode requires more power, but shortens delays that occur at each start of GPU jobs. It is useful when you have a series of short runs.

To enable persistence mode for all GPUs, use following command:

```
nvidia-smi -pm 1
```

After this command is issued, you get the output for your system like that shown in Example 8-21.

Example 8-21 Enable persistence mode for all GPUs

```
$ nvidia-smi -pm 1
Enabled persistence mode for GPU 0000:03:00.0.
Enabled persistence mode for GPU 0000:04:00.0.
Enabled persistence mode for GPU 0002:03:00.0.
Enabled persistence mode for GPU 0002:04:00.0.
All done.
```

To enable persistence mode only for a specific GPU, use the `-i` option. Example 8-22 shows how to enable this mode for the first GPU.

Example 8-22 Enable persistence mode for specific GPU

```
$ nvidia-smi -i 0 -pm 1
Enabled persistence mode for GPU 0000:03:00.0.
All done.
```

To disable persistence mode for all or a specific GPU, use the `0` value for the `-pm` option. Example 8-23 shows the usage of this command.

Example 8-23 Disable persistence mode for all or specific GPU

```
$ nvidia-smi -pm 0
Disabled persistence mode for GPU 0000:03:00.0.
Disabled persistence mode for GPU 0000:04:00.0.
Disabled persistence mode for GPU 0002:03:00.0.
Disabled persistence mode for GPU 0002:04:00.0.
All done.
$
$ nvidia-smi -i 0 -pm 0
Disabled persistence mode for GPU 0000:03:00.0.
All done.
```

A

Applications and performance

Although this book generally targets system administrators and application developers, this appendix covers several topics that are mostly relevant to the application users of the IBM POWER8 high-performance computing solution.

This appendix includes the following sections:

- Application software

  This section gives examples of application software, such as bioinformatics, computational fluid dynamics, and molecular dynamics packages.

- Effects of basic performance tuning techniques

  This section suggests some practices for tuning applications (an example of the *NAS Parallel Benchmarks suite*) and describes their impact on performance.

- General methodology of performance benchmarking

  This section proposes a sequence of actions to be performed when evaluating performance of parallel applications.

- Sample code for the construction of thread affinity strings

  This section provides sample code that can be used to construct affinity strings for thread binding environment variables.

- ESSL performance results

  This section shows the performance of the DGEMM routine from the ESSL library.

Application software

This section lists examples of software packages from the following application domains:

- Bioinformatics
- Computational fluid dynamics
- Molecular dynamics

This section also provides basic guidance about the compilation and execution of several of these applications.

Bioinformatics

Personal healthcare is a rapidly growing area. Driven by the high demand for low-cost nucleotide sequencing, several new genome sequencing methods were developed recently. These methods are commonly known as next-generation sequencing (NGS) methods. In recent years, these methods were implemented in commercial sequencer apparatus, and sequencing of genetic material has become a routine procedure.

The NGS technology produces vast amounts of raw genome sequence data. As a result, researchers and clinicians need solutions that can solve the problem of large volume NGS data management, processing, and analysis. The underlying computing resources are equipped ideally with multiple fast processors, a large amount of RAM, and an efficient storage system. POWER8 machines that run the Linux operating system are good candidates for the role of NGS data machines.

Trinity

Trinity is a popular tool for the processing and analysis of genomic sequencing data. The paper available at the following link lists a possible choice of compilation options and evaluates the performance of POWER8 processor-based systems in NGS analysis:

http://www.ibm.com/partnerworld/wps/servlet/ContentHandler/stg_ast_sys_wp-performance-of-trinity-rna-seqde-novo-assembly

BioBuilds suite

Major genomics applications on POWER8, including Trinity, are available for download as part of the *BioBuilds* suite. The suite is a collection of open source bioinformatics tools, and is distributed at no additional charge. The package is pre-built and optimized for the IBM Linux on Power platform. It also includes supporting libraries for the tools. Table A-1 lists the set of available tools as of the *BioBuilds* 2015.11 release.

Table A-1 List of bioinformatics tools available in BioBuilds 2015.11 release

Bioinformatics tools from the BioBuilds suite (2015.11 release)			
ALLPATHS-LG	ClustalW	iSAAC	SOAP3-DP
BAMtools	Cufflinks	Mothur	SOAPaligner
Barracuda	EBSEQ	NCBI	SOAPbuilder
bedtools	EMBOSS	Oases/Velvet	SOAPdenovo2
Bfast	FASTA	Picard	STAR
Bioconductor	FastQC	PLINK	tabix
BioPython	HMMER	Pysam	TMAP

Bowtie	HTSeq	RSEM	TopHat
Bowtie2	htslib	Samtools	Trinity
BWA	IGV	SHRiMP	variant_tools

Because genome sequencing is a compute-intensive task, the sequencing can gain a large performance benefit by using an accelerator. The *Barracuda* and *SOAP3-DP* tools listed in Table A-1 on page 264 are examples of open source bioinformatics applications that can offload computations to a GPU.

For more information about the *BioBuilds* collection, see the following website:

http://www.ibm.com/partnerworld/gsd/solutiondetails.do?solution=51837

BALSA

BALSA is another example of an application that uses the computational power of the GPU for the secondary analysis of next generation sequencing data. With two GPUs installed, the tool can analyze two samples in parallel. For more information about BALSA, see the following website:

http://www.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=POS03141USEN

OpenFOAM

The Open Field Operation and Manipulation (OpenFOAM) Computational Fluid Dynamics (CFD) toolbox is an open source CFD software package that is available at no additional charge. It has a large user base across most areas of engineering and science, from both commercial and academic organizations. OpenFOAM has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence, and heat transfer, to solid dynamics and electromagnetic. It includes tools for meshing, notably *snappyHexMesh*, a parallelized mesher for complex computer-aided engineering (CAE) geometries, and for pre- and post-processing. Almost everything (including meshing, and pre- and post-processing) runs in parallel as standard, enabling users to take full advantage of computer hardware at their disposal.

Several versions of OpenFOAM are available. This section provides an example of OpenFOAM 2.4.0, focusing on how to install and run it, and how to get the most out of the POWER8 architecture with this application.

Preparation before installation of OpenFOAM

This example uses the GNU compiler and the OpenMPI for MPI parallelization as shown in Example A-1.

Example A-1 Preparation before installation of OpenFOAM

```
$ export MP_COMPILER=gnu
```

Installation of OpenFOAM

This section shows how you can download and install the OpenFOAM 2.4.0 package. First, download and decompress the source codes of OpenFOAM and the third-party toolkit as shown in Example A-2.

Example A-2 Prepare the required sources of OpenFOAM and third-party pack

```
$ mkdir -p $HOME/OpenFOAM
$ cd $HOME/OpenFOAM
$ wget http://jaist.dl.sourceforge.net/project/foam/foam/2.4.0/OpenFOAM-2.4.0.tgz
$ wget
http://jaist.dl.sourceforge.net/project/foam/foam/2.4.0/ThirdParty-2.4.0.tgz
$ tar zxvf OpenFOAM-2.4.0.tgz
$ tar zxvf ThirdParty-2.4.0.tgz
```

For OpenFOAM 2.4.0, obtain a patch file from the following website:

http://www.openfoam.org/mantisbt/view.php?id=1759

From this site, get the file `enable_ppc64el_patch.patch`, put it on `$HOME/OpenFOAM`, and apply it as shown in Example A-3.

Example A-3 Apply the required patch to OpenFOAM

```
$ cd $HOME/OpenFOAM/OpenFOAM-2.4.0
$ patch -p1 < ../enable_ppc64el_patch.patch
```

Set the environment variables required for OpenFOAM and start the shell script `Allwmake` as shown in Example A-4.

Example A-4 Start OpenFOAM building

```
$ export FOAM_INST_DIR=$HOME/OpenFOAM
$ source $FOAM_INST_DIR/OpenFOAM-2.4.0/etc/bashrc
$ cd $FOAM_INST_DIR/OpenFOAM-2.4.0
$ ./Allwmake
```

Running OpenFOAM

To run OpenFOAM, a series of input data is needed that is a combination of data sets such as boundary conditions, initial conditions, various physical parameters, and the selections from many solvers and methods prepared in OpenFOAM. This input data defines the physical simulation that the user wants to solve. OpenFOAM includes examples of these data sets that are called *tutorials*. This example shows a tutorial named *motorbike*. The motorbike tuitorial simulates a typical CFD that calculates the steady flow around a motorcycle and rider and is one of the major tutorials for the performance benchmark.

Figure A-1 shows the images of the motorcycle and rider. These images are rendered by using the ParaView tool. The ParaView tool binary for Windows 64-bit, Windows 32-bit, Linux 64-bit, and Mac OS X is downloadable from the following website:

http://www.paraview.org/

The source code of the ParaView tool is also included in the third-party toolkit introduced above.

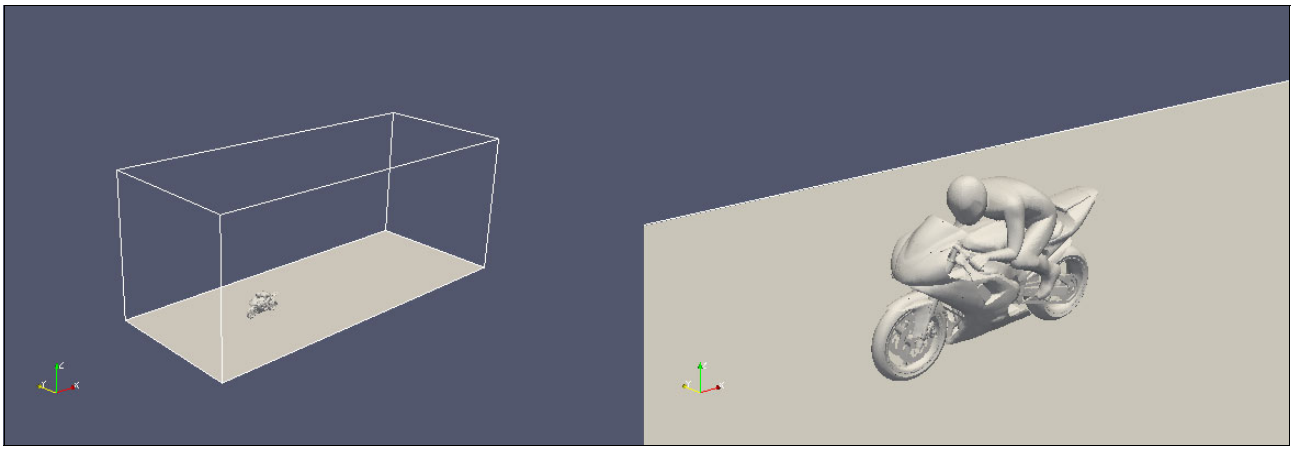

Figure A-1 Motorbike input for OpenFOAM simulation

There are nine programs for completing the motorbike simulation as shown in Example A-5. These programs are implemented in the `Allrun` script file that is located in the following directory:

`$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/`

These nine programs are executed one by one in the `Allrun` script. Among these nine programs, four programs (`snappyHexMesh`, `patchSummary`, `potentialFoam`, and `simpleFoam`) are executed with message passing interface (MPI) parallelization. By default, they are executed with six MPI processes. The other five programs (`surfaceFeatureExtract`, `blockMesh`, `decomposePar`, `reconstructParMesh`, and `reconstructPar`) are not MPI implemented and executed serially using one CPU core.

Example A-5 Nine OpenFOAM programs executed in the motorbike simulation

```
surfaceFeatureExtract
blockMesh
decomposePar
snappyHexMesh
patchSummary
potentialFoam
simpleFoam
reconstructParMesh
reconstructPar
```

Among these programs, `simpleFoam` is the main solver for this motorbike simulation, which solves the velocity and pressure by iterative calculation using the Semi-Implicit Method for Pressure-Linked Equation (SIMPLE) method. In general, `simpleFoam` takes plenty of time to complete and its elapsed time is the majority of the elapsed time of all nine programs.

Simulating a large-size problem with many MPI processes

The POWER8 architecture has high memory bandwidth. Even if you increase the number of grids of the problem and increase the number of MPI processes, you can run your jobs comfortably without seeing the performance degradation caused by memory performance bottleneck. Therefor, you can increase the problem size and the number of MPI processes to take the advantage of the POWER8 architecture.

The following section shows how to increase the problem size (the number of grids) and the number of MPI processes using the motorbike case as an example.

How to increase the problem size

This section describes how to increase the problem size and the number of processes by referring to the previous motorbike case (Figure A-1 on page 267).

To change the problem size of the simulation, modify the parameter settings in the `blockMeshDict` file. This file is located in the following directory:

```
$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/constant/polyMesh/
```

The default is 1280 grids (20 x 8 x 8 grids) as shown in Figure A-2.

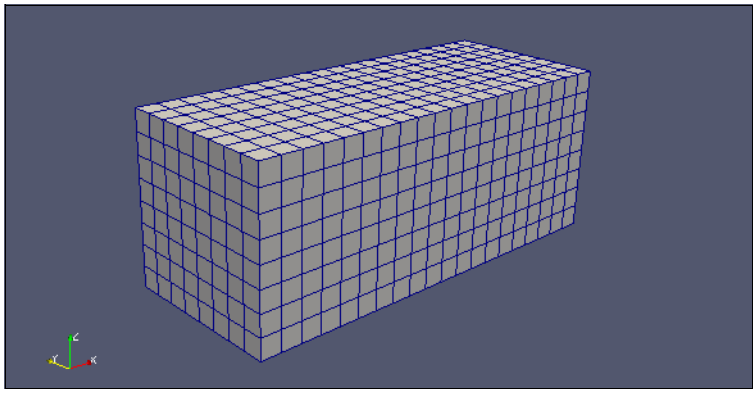

Figure A-2 20 x 8 x 8 grids (default)

For example, if you want to change the default grid into a 40 × 16 × 16 grids (10240 grids) as shown in Figure A-3, you need to modify `blockMeshDict` as shown in Example A-6 on page 269.

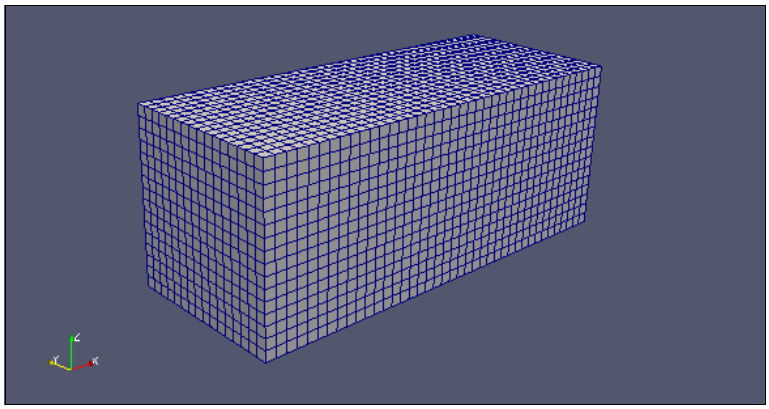

Figure A-3 40 x 16 x 16 grids

Example A-6 shows modifying the `blockMeshDict` file.

Example A-6 Modification of blockMeshDict file

```
$ diff blockMeshDict.org blockMeshDict
34c34
<     hex (0 1 2 3 4 5 6 7) (20 8 8) simpleGrading (1 1 1)
---
>     hex (0 1 2 3 4 5 6 7) (40 16 16) simpleGrading (1 1 1)
```

For more information about `blockMeshDict`, see 4.3 Mesh generation with the blockMesh utility in the following website:

http://www.openfoam.com/documentation/user-guide/blockMesh.php

How to increase the number of MPI processes

The method of parallel computing that is used by OpenFOAM is known as *domain decomposition*. In this method, the geometry and associated fields are broken into pieces and allocated to each CPU core for computation. The flow of parallel computation involves decomposition of mesh and fields, running the application in parallel, and post-processing the decomposed case as described in the following sections. The parallel running uses OpenMPI for the standard MPI.

To change the number of MPI processes, you need to make the following modifications.

Select the method of the domain decomposition

You can select the scotch method rather than default hierarchical method for the domain decomposition. The scotch decomposition requires no geometric input from the user and attempts to minimize the number of processor boundaries.

This section shows the difference between hierarchical and scotch by using simple figures of 1280 grids (20 x 8 x 8 grids).

Figure A-4 shows the hierarchical method, and in this case. The domain is decomposed as (X-direction, Y-direction, Z-direction) = (3, 2, 1) based on the parameter settings that are stated in `decomposeParDict` file.

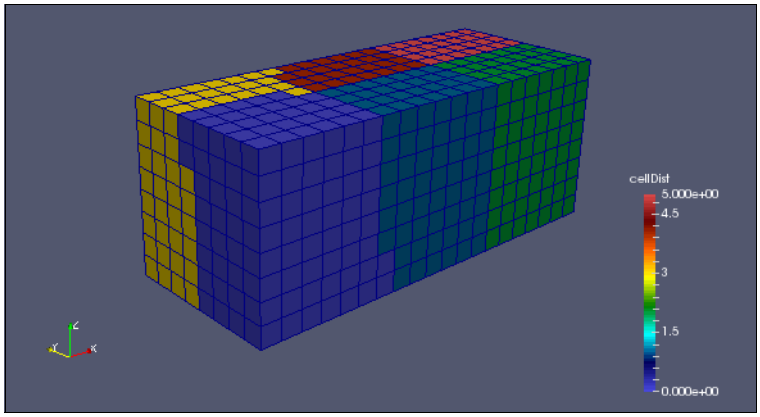

Figure A-4 Domain decomposition by hierarchical method

Figure A-5 shows the scotch method. In this method, the scotch library automatically decides the optimal domain decomposition, so you do not need to set the number of decomposition for each direction of X, Y, and Z.

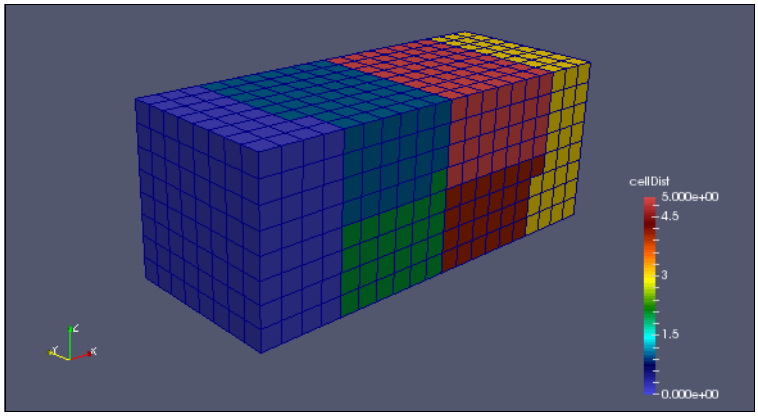

Figure A-5 Domain decomposition by scotch method

To select the scotch method instead of the hierarchical method, modify the parameter settings of the `decomposeParDict` file as shown in Example A-7.

Example A-7 Change the parameter settings in the decomposeParDict file

```
$ cd
$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/system
/
$ diff decomposeParDict.org decomposeParDict
20c20
< method          hierarchical;
---
> method          scotch;
```

Change the number of subdomains

To change the number of subdomains from six to forty as shown in Figure A-6, you need to modify the parameter settings of the `decomposeParDict` file as shown in Example A-8 on page 271.

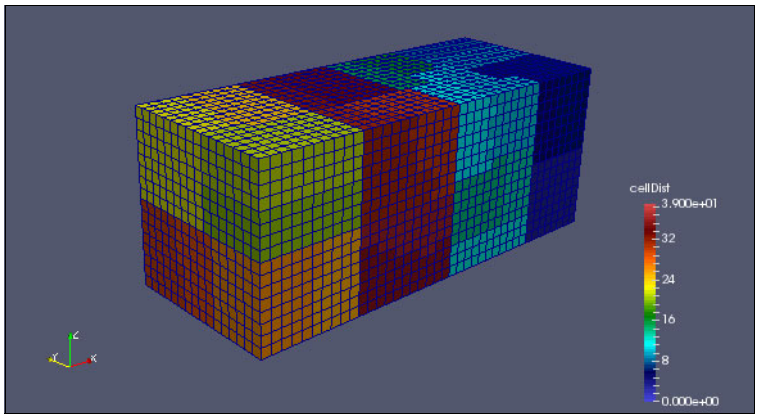

Figure A-6 Forty subdomains for forty MPI processes

Example A-8 shows how to modify the parameters of the `ecomposeParDict` file.

Example A-8 Change the parameter settings in decomposeParDict file

```
$ cd
$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/system
/
$ diff decomposeParDict.org decomposeParDict
18c18
< numberOfSubdomains 6;
---
> numberOfSubdomains 40;
```

Change the number of MPI processes

Change the number of MPI processes in the `Allrun` file as shown in Example A-9. Usually, the number of MPI processes need to be the same as the number of subdomains as described in the previous section.

Example A-9 Change the parameter settings of Allrun

```
$ cd $HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/
$ diff Allrun.org Allrun
14c14
< runParallel snappyHexMesh 6 -overwrite
---
> runParallel snappyHexMesh 40 -overwrite
23,25c23,25
< runParallel patchSummary 6
< runParallel potentialFoam 6
< runParallel $(getApplication) 6
---
> runParallel patchSummary 40
> runParallel potentialFoam 40
> runParallel $(getApplication) 40
```

Note: To check what the domain decomposition looks like, use the following command:

```
[home/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike]$
diff Allrun.org Allrun
13c13
< runApplication decomposePar
---
> runApplication decomposePar -cellDist
```

Running the motorbike simulation

After completing these preparations, run the motorbike simulation by running the `Allrun` script as shown in Example A-10. After the `Allrun` script starts, the binart files listed in Example A-5 on page 267 are automatically executed one by one.

Example A-10 Kick Allrun script

```
$ cd $HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/
$ time ./Allrun
```

After finishing running the series of binary files, a new directory named 500 is created. This new directory includes some simulated values such as *U* (Velocity), *p* (Pressure), and so on. These values show the physical state after 500 iterative calculations.

Figure A-7 shows the images of these simulated values of *U* (Velocity) and *p* (Pressure) around the motorbike visualized by the ParaView tool.

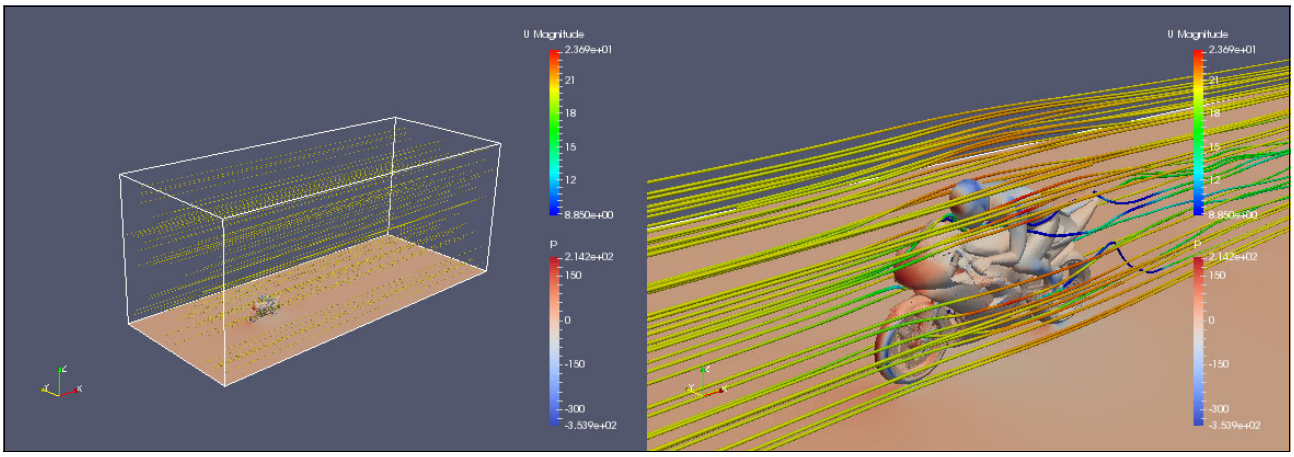

Figure A-7 motorbike simulation result

Tuning techniques

This section introduces these tuning techniques:

- CPU binding
- SMT settings
- Linking tcmalloc

CPU binding

To get higher and stable performance, each MPI process needs to be bound to each CPU core. In OpenMPI, the **`mpirun`** command automatically binds processes at the start of the v1.8 series.

For more information, see the following website:

https://www.open-mpi.org/doc/v1.8/man1/mpirun.1.php

SMT settings

Generally, CAE applications such as OpenFOAM have a hard time using the advantages of hardware threads. However, in some cases, using SMT on POWER8 and assigning some MPI processes into a single CPU core can improve the total throughput. Therefore, trying some combination of SMT modes and the number of MPI processes assigned to a single core is valuable for benchmarking and performance tuning.

Linking tcmalloc

In some cases, linking a *thread-caching malloc (tcmalloc)* library into the OpenFOAM binary files will improve their performance. For the nine example OpenFOAM binary files that are listed in Example A-5 on page 267, the performance of `surfaceFeatureExtract`, `blockMesh`, `decomposePar`, `snappyHexMesh`, `patchSummary`, `reconstructParMesh`, and `reconstructPar` were improved, but the performance of `potentialFoam` and `simpleFoam` were slightly degraded by linking to a tcmalloc library.

Therefore, link the tcmalloc library only to the binary files that are expected to be improved by using the shell-script as shown in Example A-11.

Example A-11 Apply tcmalloc

```
$ cd $HOME/OpenFOAM
$ cat @apply_tcmalloc
#!/bin/sh
for i in surfaceFeatureExtract blockMesh decomposePar snappyHexMesh patchSummary
reconstructParMesh reconstructPar
do
  rm ./OpenFOAM-2.4.0/platforms/linuxPPC64leGccDPOpt/bin/$i
  cd `find ./OpenFOAM-2.4.0/applications/ -name $i -type d`
  wmake |tr '\\\n' ' ' > out.wmake
  echo `cat out.wmake` -L/opt/at9.0/lib64/ -ltcmalloc |sh -x
  rm out.wmake
  cd -
done
$ ./@apply_tcmalloc
```

For more information about tcmalloc, see the following website:

http://goog-perftools.sourceforge.net/doc/tcmalloc.html

NAMD program

Nanoscale Molecular Dynamics (NAMD) is a freeware molecular dynamics simulation package written using the Charm++ parallel programming model.

This section introduces how to install and how to run NAMD 2.11 in an IBM POWER8 server using NVIDIA graphics programming units (GPUs) with CUDA 7.5.

Installation of NAMD

Download the source code `NAMD_2.11_Source.tar.gz` from the following website:

http://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=NAMD

Before downloading the source code, complete the registration process, which requires your name and email address, and that you answer some questions and accept the license agreement.

Put the downloaded file `NAMD_2.11_Source.tar.gz` in your preferred directory, for example $HOME/NAMD. Then build the prerequisite package as shown in Example A-12.

Example A-12 Building prerequisite package for NAMD

```
$ cd $HOME/NAMD
$ tar zxvf NAMD_2.11_Source.tar.gz
$ cd NAMD_2.11_Source
$ tar xvf charm-6.7.0.tar
$ cp -rp charm-6.7.0/src/arch/mpi-linux-ppc charm-6.7.0/src/arch/mpi-linux-ppc64le
$ cd charm-6.7.0
$ ./build charm++ mpi-linux-ppc64le -O -DCMK_OPTIMIZE=1
$ cd ../arch
$ ls -l | grep Linux-POWER
-rw-r----- 1 kame IBM1  190  6? 18  2014 Linux-POWER-g++.arch
-rw-r----- 1 kame IBM1  495  2?  7 2011 Linux-POWER-xlC.arch
```

```
-rw-r----- 1 kame IBM1    0  2?  7 2011 Linux-POWER.base
-rw-r----- 1 kame IBM1  467 12?  2 12:52 Linux-POWER.cuda
-rw-r----- 1 kame IBM1  138  2?  7 2011 Linux-POWER.fftw
-rw-r----- 1 kame IBM1  191  6? 18  2014 Linux-POWER.tcl
```

Before starting to build NAMD, prepare the FFTW package as shown in Example A-13.

Example A-13 Prepare FFTW package

```
[/work/NAMD]$ wget ftp://ftp.fftw.org/pub/fftw/fftw-3.3.4.tar.gz
[/work/NAMD]$ tar zxvf fftw-3.3.4.tar.gz
[/work/NAMD]$ cd fftw-3.3.4
[/work/NAMD/fftw-3.3.4]$ ./configure --enable-float --prefix=`pwd`
[/work/NAMD/fftw-3.3.4]$ make
[/work/NAMD/fftw-3.3.4]$ make install
[/work/NAMD/fftw-3.3.4]$ ls lib
libfftw3f.a  libfftw3f.la pkgconfig
```

Prepare the TCL package as shown in Example A-14.

Example A-14 Prepare TCL package

```
[/work/NAMD]$ wget
http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-ppc64le-threaded.tar
.gz
[/work/NAMD]$ tar zxvf tcl8.5.9-linux-ppc64le-threaded.tar.gz
[/work/NAMD]$ ls tcl8.5.9-linux-ppc64le-threaded/lib
libtcl8.5.a  libtclstub8.5.a  tcl8 tcl8.5  tclConfig.sh
```

Prepare some files for the parameter settings as shown in Example A-15.

Example A-15 Prepare some files for parameter settings

```
[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER-xlC_MPI.arch
NAMD_ARCH = Linux-POWER
CHARMARCH = mpi-linux-ppc64le
CXX = mpCC -w
CXXOPTS = -O3 -q64 -qnohot -qstrict -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8
-qtune=pwr8
CXXNOALIASOPTS = -O4 -q64 -qaggrcopy=nooverlap -qalias=noallptrs -qarch=pwr8
-qtune=pwr8
CXXTHREADOPTS = -O3 -q64 -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8 -qtune=pwr8
CC = xlc -w
COPTS = -O4 -q64 -qarch=pwr8 -qtune=pwr8

[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER.cuda
CUDADIR=/usr/local/cuda-7.5
CUDAINCL=-I$(CUDADIR)/include
CUDALIB=-L$(CUDADIR)/lib64 -lcudart_static -lrt -ldl
CUDASODIR=$(CUDADIR)/lib64
LIBCUDARTSO=
CUDAFLAGS=-DNAMD_CUDA
CUDAOBJS=$(CUDAOBJSRAW)
CUDA=$(CUDAFLAGS) -I. $(CUDAINCL)
CUDACC=$(CUDADIR)/bin/nvcc -O3 --maxrregcount 32 $(CUDAGENCODE) $(CUDA)
```

```
CUDAGENCODE=-gencode arch=compute_20,code=sm_20 -gencode
arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode
arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52

[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER.fftw3
FFTDIR=/work/NAMD/fftw-3.3.4/
FFTINCL=-I$(FFTDIR)/include
FFTLIB=-L$(FFTDIR)/lib -lfftw3f
FFTFLAGS=-DNAMD_FFTW -DNAMD_FFTW_3
FFT=$(FFTINCL) $(FFTFLAGS)

[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER.tcl
TCLDIR=/work/NAMD/tcl8.5.9-linux-ppc64le-threaded/
TCLINCL=-I$(TCLDIR)/include
# TCLLIB=-L$(TCLDIR)/lib -ltcl8.5 -ldl
TCLLIB=-L$(TCLDIR)/lib -ltcl8.5 -ldl -lpthread
TCLFLAGS=-DNAMD_TCL
TCL=$(TCLINCL) $(TCLFLAGS)
```

Modify the `config` file as shown in Example A-16 to avoid errors during configuration.

Example A-16 Modify the config file

```
[/work/NAMD/NAMD_2.11_Source]$ diff config.org config
383,390c383,390
<     if ( $charm_arch_mpi || ! $charm_arch_smp ) then
<       echo ''
<       echo "ERROR: $ERRTYPE builds require non-MPI SMP or multicore Charm++ arch
for reasonable performance."
<       echo ''
<       echo "Consider ibverbs-smp or verbs-smp (InfiniBand), gni-smp (Cray), or
multicore (single node)."
<       echo ''
<       exit 1
<     endif
---
> #    if ( $charm_arch_mpi || ! $charm_arch_smp ) then
> #      echo ''
> #      echo "ERROR: $ERRTYPE builds require non-MPI SMP or multicore Charm++
arch for reasonable performance."
> #      echo ''
> #      echo "Consider ibverbs-smp or verbs-smp (InfiniBand), gni-smp (Cray), or
multicore (single node)."
> #      echo ''
> #      exit 1
> #    endif
```

Configure NAMD as shown in Example A-17.

Example A-17 Configure NAMD

```
[/work/NAMD/NAMD_2.11_Source]$ ./config Linux-POWER-xlC_MPI --with-fftw3
--with-tcl --with-cuda

Selected arch file arch/Linux-POWER-xlC_MPI.arch contains:
```

```
NAMD_ARCH = Linux-POWER
CHARMARCH = mpi-linux-ppc64le
CXX = mpCC_r -w
CXXOPTS = -O3 -q64 -qnohot -qstrict -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8
-qtune=pwr8
CXXNOALIASOPTS = -O4 -q64 -qaggrcopy=nooverlap -qalias=noallptrs -qarch=pwr8
-qtune=pwr8
CXXTHREADOPTS = -O3 -q64 -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8 -qtune=pwr8
CC = xlc_r -w
COPTS = -O4 -q64 -qarch=pwr8 -qtune=pwr8
Creating directory: Linux-POWER-xlC_MPI
Creating link: .. to .rootdir
Writing build options to Linux-POWER-xlC_MPI/Make.config
Using Charm++ 6.7.0 build found in main build directory
Linking Makefile
Linking Make.depends
Linking src directory
Linking plugins directory
Linking psfgen directory

Generated Linux-POWER-xlC_MPI/Make.config contains the following:

CHARMBASE = .rootdir/charm-6.7.0
include .rootdir/arch/Linux-POWER-xlC_MPI.arch
CHARM = $(CHARMBASE)/$(CHARMARCH)
NAMD_PLATFORM = $(NAMD_ARCH)-MPI-CUDA
include .rootdir/arch/$(NAMD_ARCH).base
include .rootdir/arch/$(NAMD_ARCH).tcl
include .rootdir/arch/$(NAMD_ARCH).fftw3
include .rootdir/arch/$(NAMD_ARCH).cuda

You are ready to run make in directory Linux-POWER-xlC_MPI now.
```

Then, use **make** to build NAMD as shown in Example A-18.

Example A-18 Make NAMD

```
[/work/NAMD/NAMD_2.11_Source]$ cd Linux-POWER-xlC_MPI
[/work/NAMD/NAMD_2.11_Source/Linux-POWER-xlC_MPI]$ make

******

xlc -w -Isrc
-I/vol/xcae1/b5p218za/work/NAMD/tcl8.5.9-linux-ppc64le-threaded//include
-DNAMD_TCL -O4 -q64 -qarch=pwr8 -qtune=pwr8 -DNAMD_VERSION=\"2.11\"
-DNAMD_PLATFORM=\"Linux-POWER-MPI-CUDA\"  -DREMOVE_PROXYRESULTMSG_EXTRACOPY
-DNODEAWARE_PROXY_SPANNINGTREE -DUSE_NODEPATCHMGR  -o flipbinpdb src/flipbinpdb.c
|| \
echo "#!/bin/sh\necho unavailable on this platform" > flipbinpdb; \
chmod +x flipbinpdb
cp .rootdir/charm-6.7.0/mpi-linux-ppc64le/bin/charmrun charmrun
```

If you succeed in running **make**, you can find the execution binary file named `namd2` in the directory named `Linux-POWER-xlC_MPI`.

Running NAMD

Sample simulations are provided at the following website:

http://www.ks.uiuc.edu/Research/namd/utilities/

For this example, select ApoA1, which has been the standard NAMD cross-platform benchmark for years as shown in Figure A-8. The official name of this gene is *apolipoprotein A-I*.

The APOA1 gene provides instructions for making a protein called apolipoprotein A-I (apoA-I). ApoA-I is a component of high-density lipoprotein (HDL). HDL is a molecule that transports cholesterol and certain fats called phospholipids through the bloodstream from the body's tissues to the liver. After they are in the liver, cholesterol and phospholipids are redistributed to other tissues or removed from the body. Figure A-8 shows a visualization of this protein.

For more information about ApoA1, see the following website:

https://ghr.nlm.nih.gov/gene/APOA1

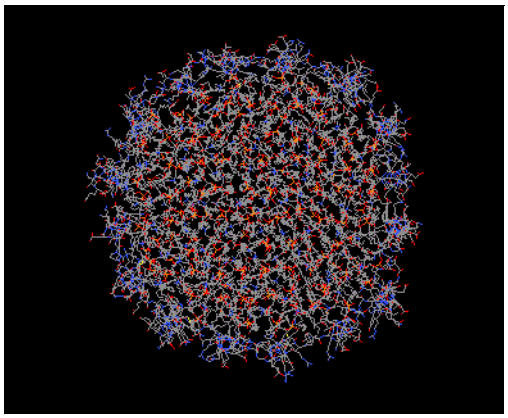

Figure A-8 Visualization of ApoA1 protein

After preparing the data directory of ApoA1 named `apoa1` on the same directory as `namd2`, execute this sample model using 40 MPI processes as shown in Example A-19.

Example A-19 Execution of NAMD with each example

```
$ MP_RESD=poe MP_HOSTFILE=./hf MP_PROCS=40 MP_SHARED_MEMORY=yes
MP_EAGER_LIMIT=65536 MEMORY_AFFINITY=MCM MP_INFOLEVEL=4 MP_BINDPROC=yes
MP_PE_AFFINITY=yes MP_BIND_MODE=spread MP_TASK_AFFINITY=cpu time poe ./namd2
./apoa1/apoa1.namd
```

For more information about the environment variables for the **poe** command, see the following website:

https://www.ibm.com/support/knowledgecenter/SSFK3V_2.3.0/com.ibm.cluster.pe.v2r3.pe100.doc/am102_poemanpage.htm

Note: NAMD seems to scale best using simultaneous multithreading (SMT) of no more than two threads per core. For example, if you have 20 physical cores on a Power System S822LC, running NAMD with 40 threads can get the best performance.

Effects of basic performance tuning techniques

This section evaluates the effects of basic performance tuning techniques. It uses the NAS Parallel Benchmarks (NPB)[1] suite of applications as examples.

The NPB suite was originally used for complex performance evaluation of supercomputers. The developers of the NPB programs distilled the typical computational physics workloads and put the most widespread numerical kernels into their product.

This section uses the OpenMP flavors of NPB benchmarks to achieve the following goals:

- Shows the performance variation for the different SMT modes
- Provides guidance for the choice of compilation parameters
- Demonstrates the importance of binding threads to logical processors

For benchmarking, the example used a 20-core IBM Power System S822LC (model 8335-GTA) based on the POWER8 processor. The cores run at 2.92 GHz. Each memory slot of the system was populated with an 8 GB RAM module for a total of 256 GB. The server was running Red Hat Enterprise Linux operating system version 7.2 (little-endian). The operating system was installed in a non-virtualized mode. The Linux kernel version was 3.10.0-327. Version v15.1.2 of IBM XL Fortran compiler was used to compile the sources.[2]

Note: The performance numbers that are shown in this section must not be treated as the official results. They are provided to demonstrate the possible effect of various performance tuning techniques on application performance. The results that are obtained in different hardware and software environments or with other compilation parameters can vary widely from the numbers shown here.

The size of the problems in the NPB benchmarking suite is predefined by the developers. The example uses the benchmarks of class C. The source code of the NPB suite was not changed. The code generation was controlled by setting compilation parameters in a `make.def` file as shown in Example A-20.

Example A-20 NPB: A sample make.def file for the -O3 parameter set

```
F77 = xlf_r -qsmp=noauto:omp -qnosave
FLINK = $(F77)
FFLAGS = -O3 -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
FLINKFLAGS = $(FFLAGS)
CC = xlc_r -qsmp=noauto:omp
CLINK = $(CC)
C_LIB = -lm
CFLAGS = $(FFLAGS)
CLINKFLAGS = $(CFLAGS)
UCC = xlc
BINDIR = ../O3
RAND = randi8
WTIME = wtime.c
MACHINE = -DIBM
```

[1] The NPB suite was developed by NASA Advanced Supercomputing (NAS) Division. For more information, see "NAS Parallel Benchmarks" at http://www.nas.nasa.gov/publications/npb.html.

[2] At the time of writing, IBM XL Fortran for Linux, V15.1.3 (little-endian distributions) was available.

It is not feasible to cover all compilation parameters, so the tests varied only the level of optimization. The following parameters were common for all builds:

```
-qsmp=noauto:omp -qnosave -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
```

Teh example considers four sets of compilation parameters. In addition to the previous compilation parameters, the remaining parameters are listed in Table A-2. The column **Option set name** lists shortcuts used to reference each set of compilation parameters that are presented in the **Compilation parameters** columns.

Table A-2 NPB: Compilation parameters used to build the NPB suite executable files

Option set name	Compilation parameters	
	Varying	Common
-O2	-O2	-qsmp=noauto:omp -qnosave -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
-O3	-O3	
-O4	-O4	
-O5	-O5	

All of the runs were performed with the following environment variables:

```
export OMP_DYNAMIC="FALSE"
export OMP_SCHEDULE="static"
```

To establish the affinity of threads, the example used a simple program. The source code for this program is listed in “Sample code for the construction of thread affinity strings” on page 309.

Note: In several cases, performance results that are presented deviate significantly from the general behavior within the same plot. The plot bars with deviations can be ignored because the tests can experience unusual conditions (operating system jitters, parasitic external workload, and so on).

The performance impact of a rational choice of an SMT mode

The POWER8 core is able to run instructions from up to eight application threads simultaneously. This capability is known as SMT. The POWER8 architecture supports the following four multithreading levels:[3]

- ST (single-thread)[4]
- SMT2 (two-way multithreading)
- SMT4 (four-way multithreading)
- SMT8 (eight-way multithreading)

The SMT mode, which can be used to obtain the optimal performance, depends on the characteristics of the application. Compilation parameters and mapping between application threads and logical processors can also affect the timing.

[3] B. Sinharoy et al, “IBM POWER8 processor core microarchitecture,” IBM J. Res. & Dev., vol. 59, no. 1, Paper 2, pp. 2:1–2:21, Jan./Feb. 2015, http://dx.doi.org/10.1147/JRD.2014.2376112.

[4] Single-thread mode is referred sometimes as SMT1.

Reason behind a conscious choice of an SMT mode

Execution units of a core are shared by all logical processors of a core (two logical processors in SMT2 mode, four logical processors in SMT4 mode, and eight logical processors in SMT8 mode). There are execution units with multiple instances (for example, load/store units, fixed-point units) and single instances (for example, branch execution unit).

In ideal conditions, application threads do not compete for execution units. This configuration results in each of eight logical processors of a core running in SMT8 mode being almost as fast as a core running in ST mode.

Depending on the application threads instruction flow, some execution units become fully saturated with instructions that come from different threads. As a result, the progress of the depended instructions is postponed. This postponement limits the overall performance of the eight logical processors of a core running in SMT8 to the performance of a core running in ST mode.

It is also possible for even a single thread to fully saturate resources of a whole core. Therefore, adding more threads to a core can result in performance degradation. For example, see the performance of $mg.C$ benchmark as shown in Figure A-15 on page 287.

Note: Generally, the performance of each logical processor of a core running in SMT2, SMT4, or SMT8 mode is not equivalent to the performance of a core running in ST mode. The performance of each logical processor of a core is influenced by all other logical processors in a core. The influence comes from the application threads instruction flow.

Performance impact of SMT mode on NPB benchmarks

The bar charts in Figure A-9 on page 281 through Figure A-17 on page 289 show the performance benefits that come from the rational choice of SMT mode for applications from the NPB suite (`bt.C`, `cg.C`, `ep.C`, `ft.C`, `is.C`, `lu.C`, `mg.C`, `sp.C`, and `ua.C`). The performance of the applications was measured for each combination of the following options:

- Four sets of compiler parameters as shown in Table A-2 on page 279.
- Eleven core layouts:
  - 1 - 10 cores from just one socket
  - Twenty cores from both sockets

The plots are organized according to the following scheme:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a particular set of compiler options.
- The *x*-axis lists core layouts. For example, the third pair (sockets: 1, cores: 3) designates the benchmarking run where application threads were bound to three cores within one socket.
- The *y*-axis shows the performance gain as measured in percentage relative to a baseline. As a baseline, we choose an SMT mode that is less favorable for the particular application.

The results show that the choice of SMT mode affects performance substantially.

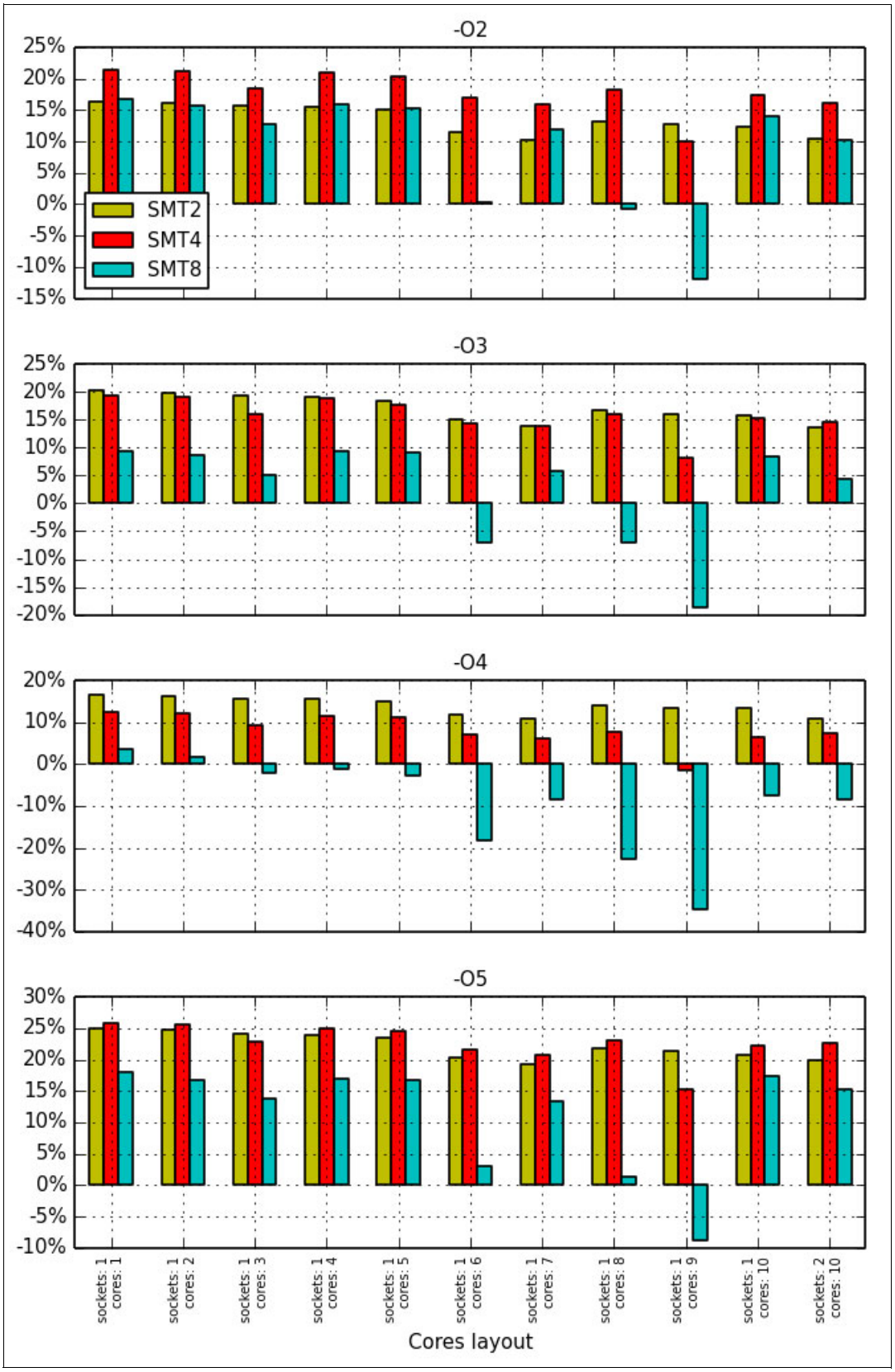

Figure A-9 NPB: Performance benefits from the rational choice of SMT mode for the bt.C benchmark

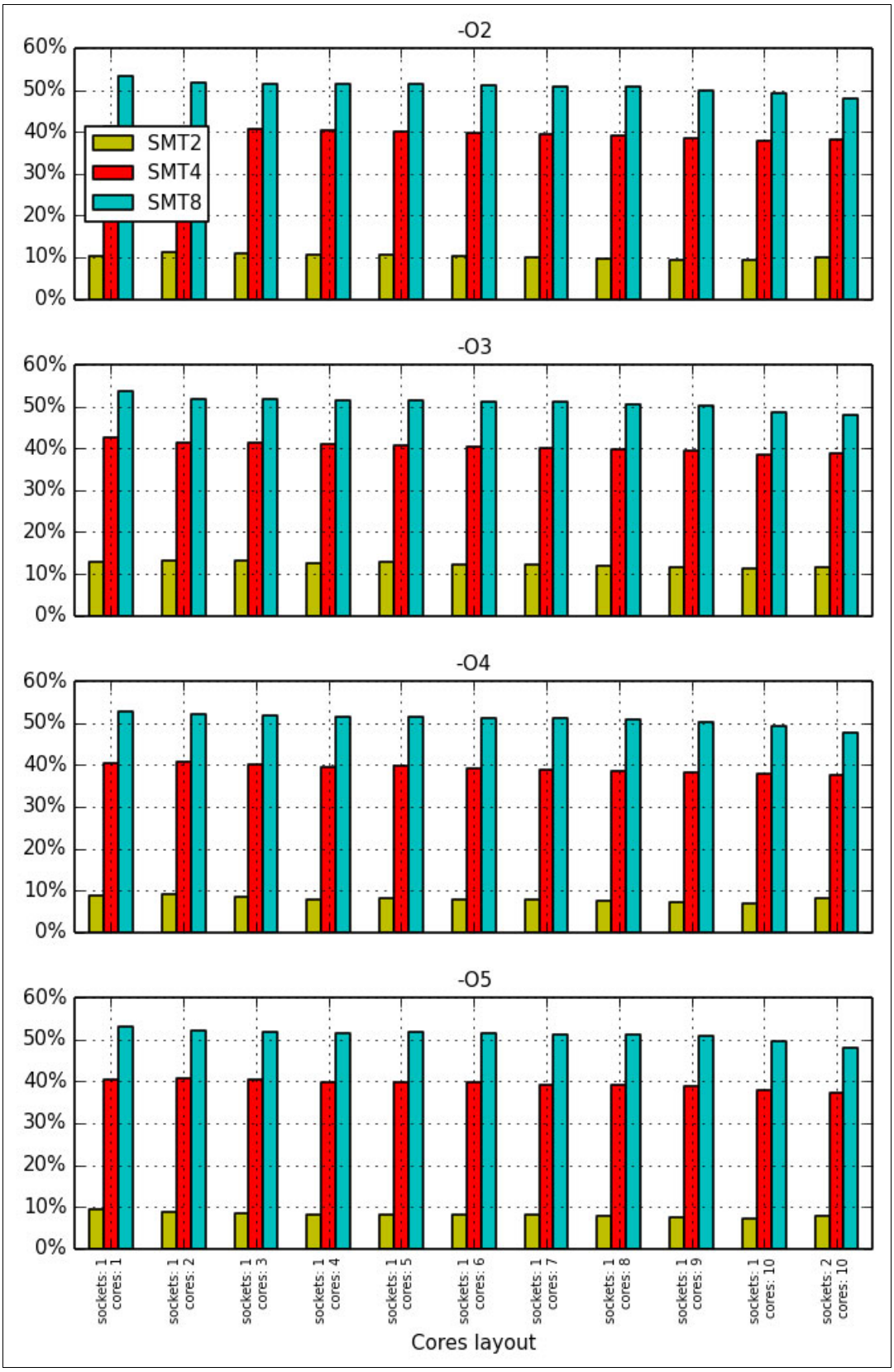

Figure A-10 NPB: Performance benefits from the rational choice of SMT mode for the cg.C benchmark

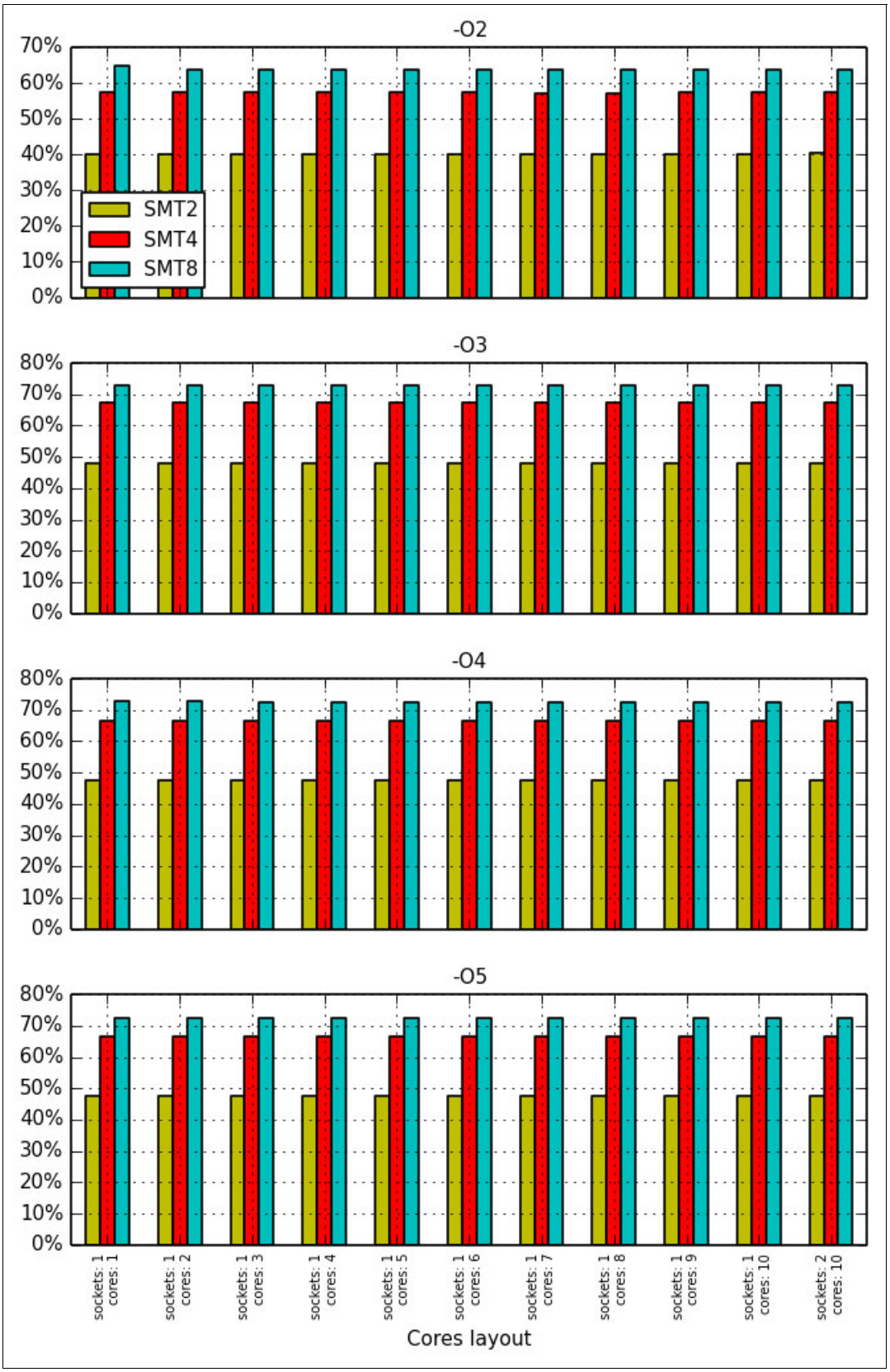

Figure A-11 NPB: Performance benefits from the rational choice of SMT mode for the ep.C benchmark

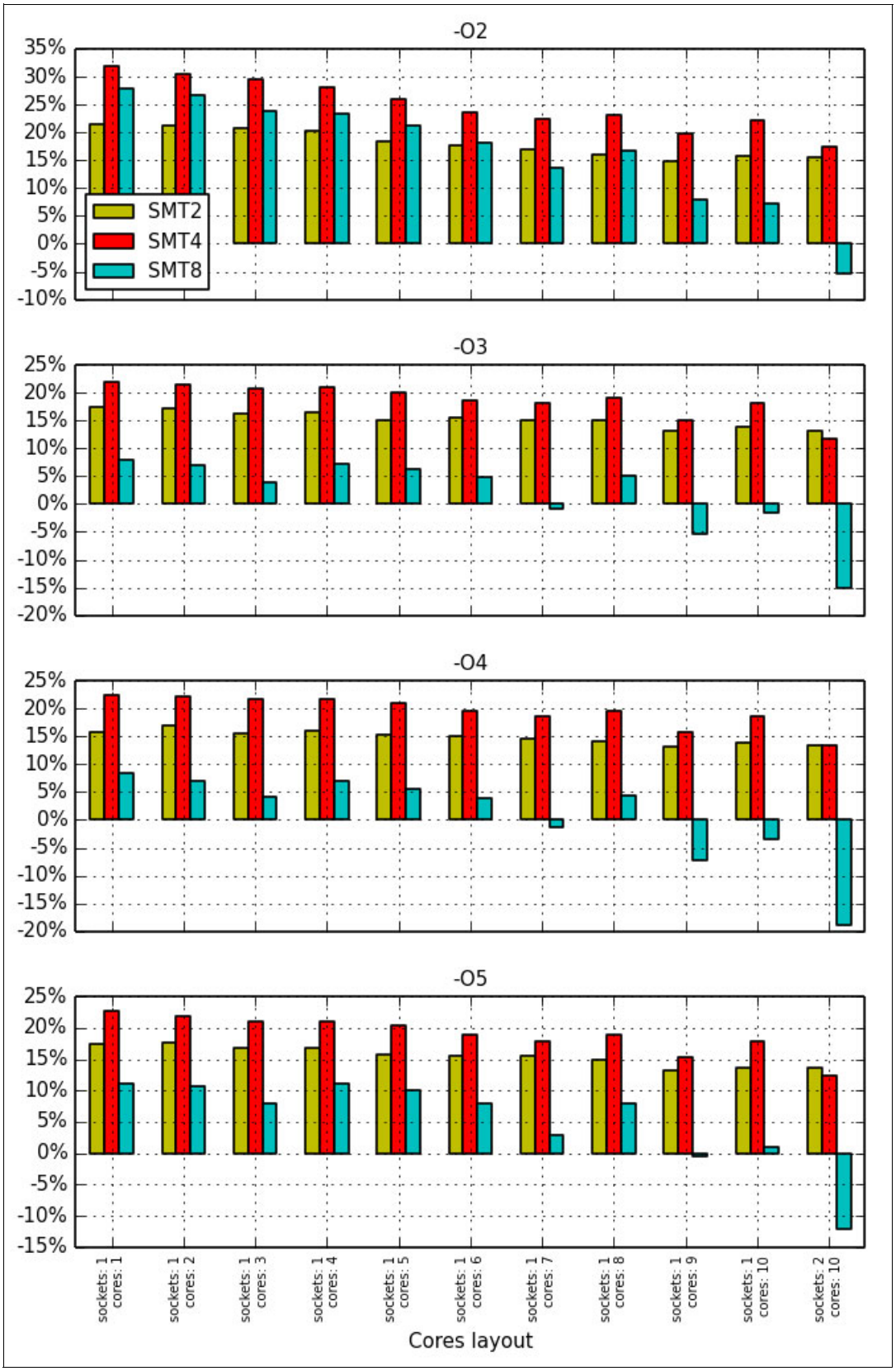

Figure A-12 NPB: Performance benefits from the rational choice of SMT mode for the ft.C benchmark

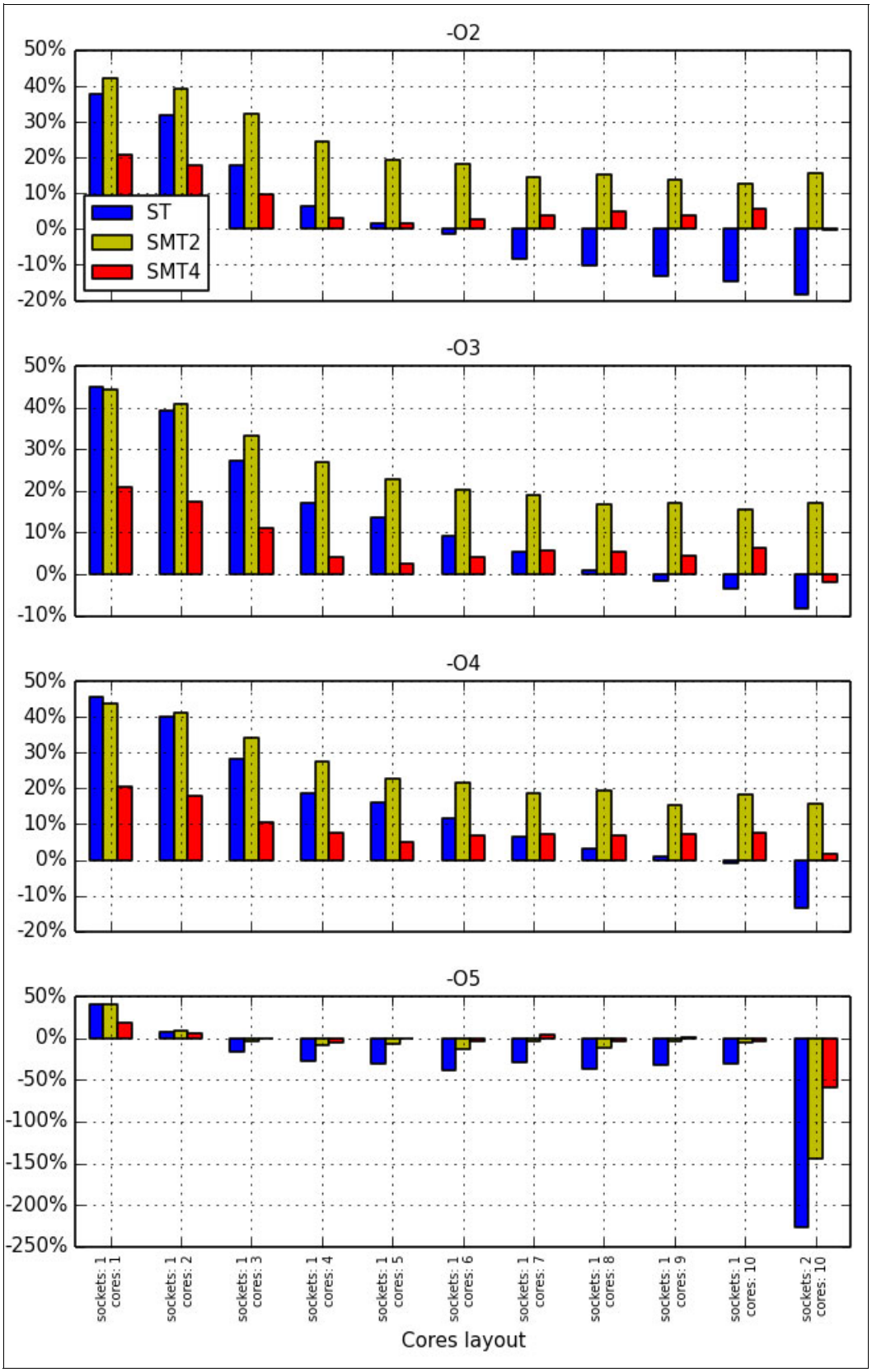

Figure A-13 NPB: Performance benefits from the rational choice of SMT mode for the is.C benchmark

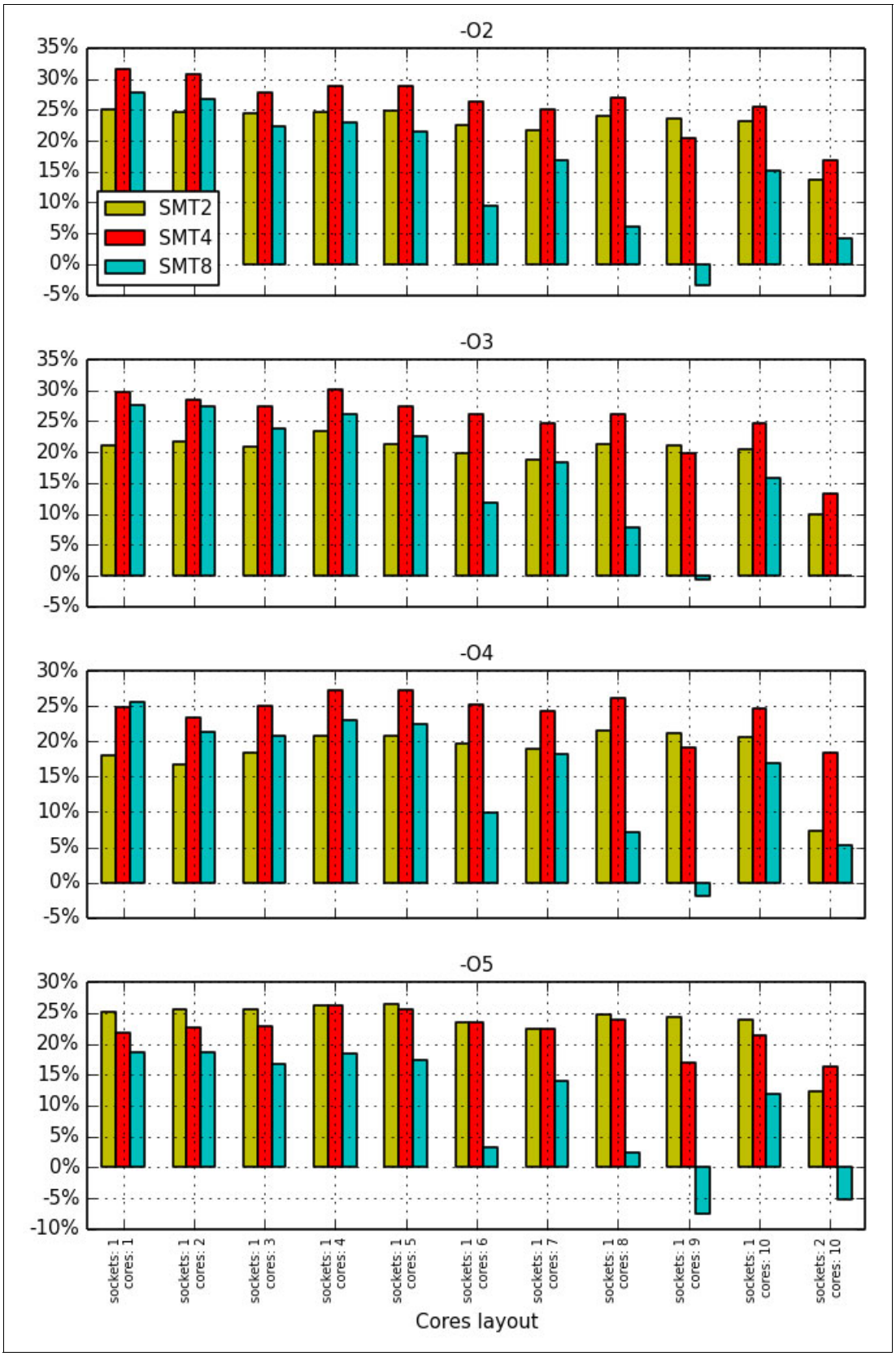

Figure A-14 NPB: Performance benefits from the rational choice of SMT mode for the lu.C benchmark

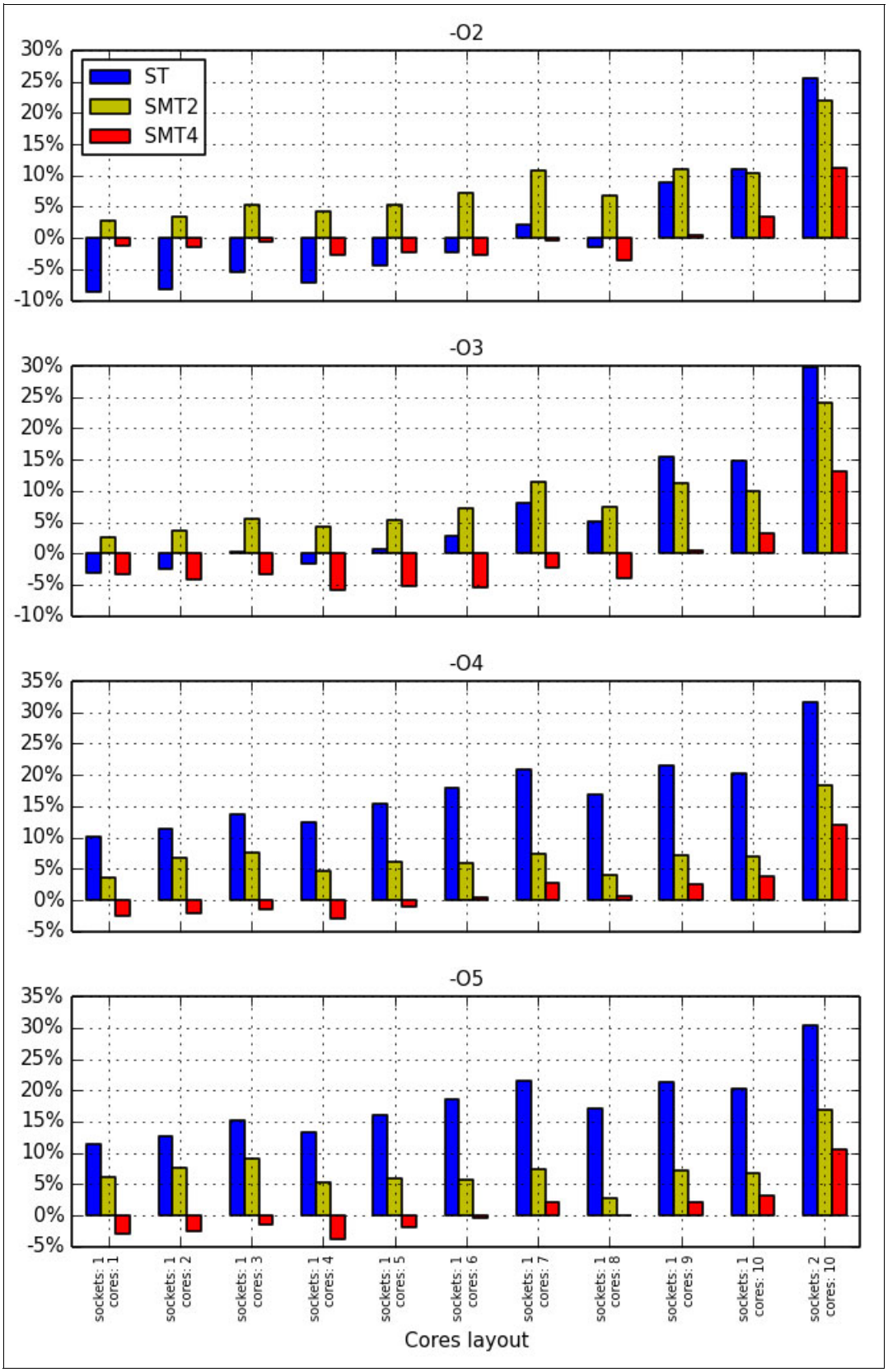

Figure A-15 NPB: Performance benefits from the rational choice of SMT mode for the mg.C benchmark

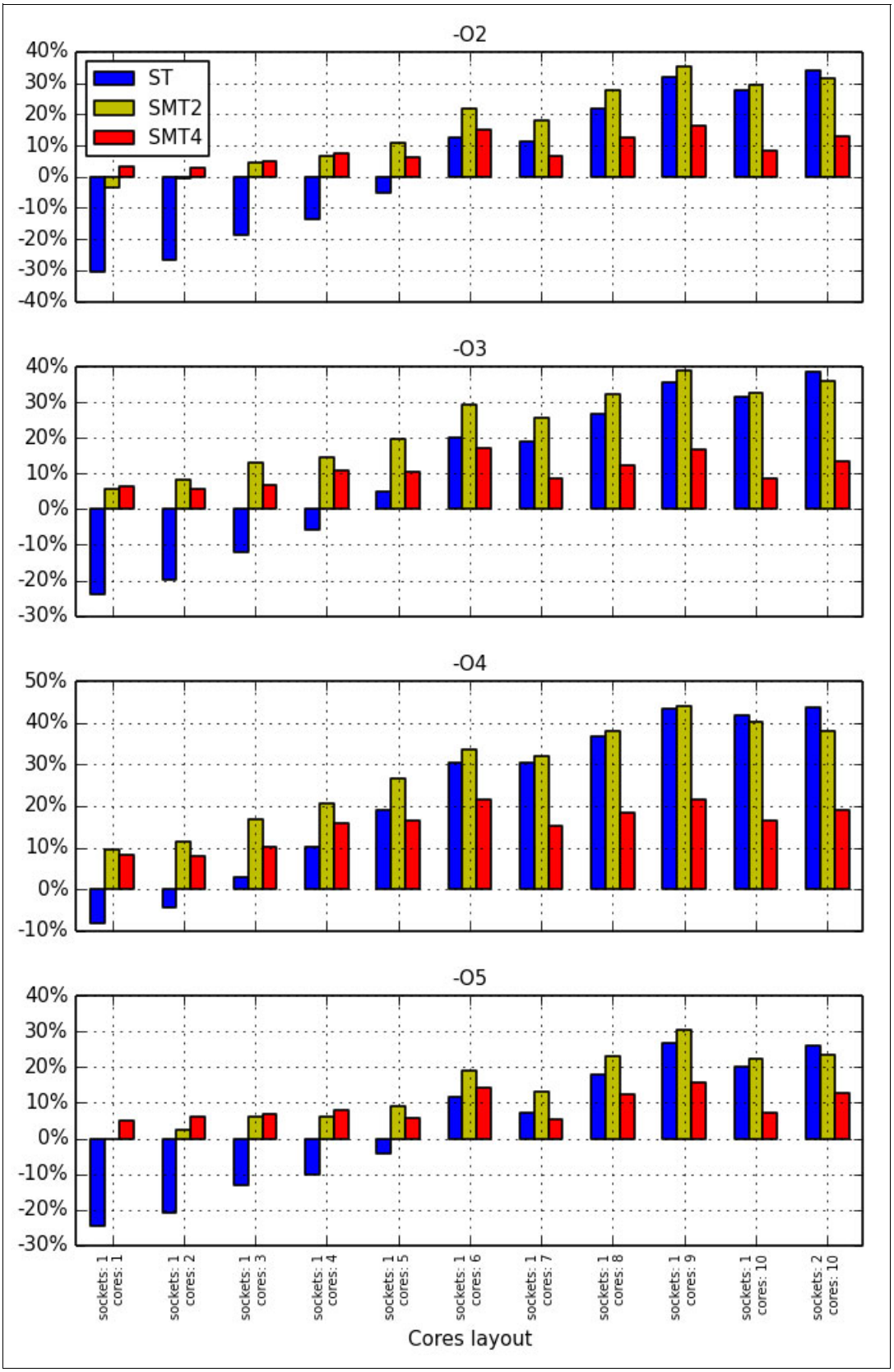

Figure A-16 NPB: Performance benefits from the rational choice of SMT mode for the sp.C benchmark

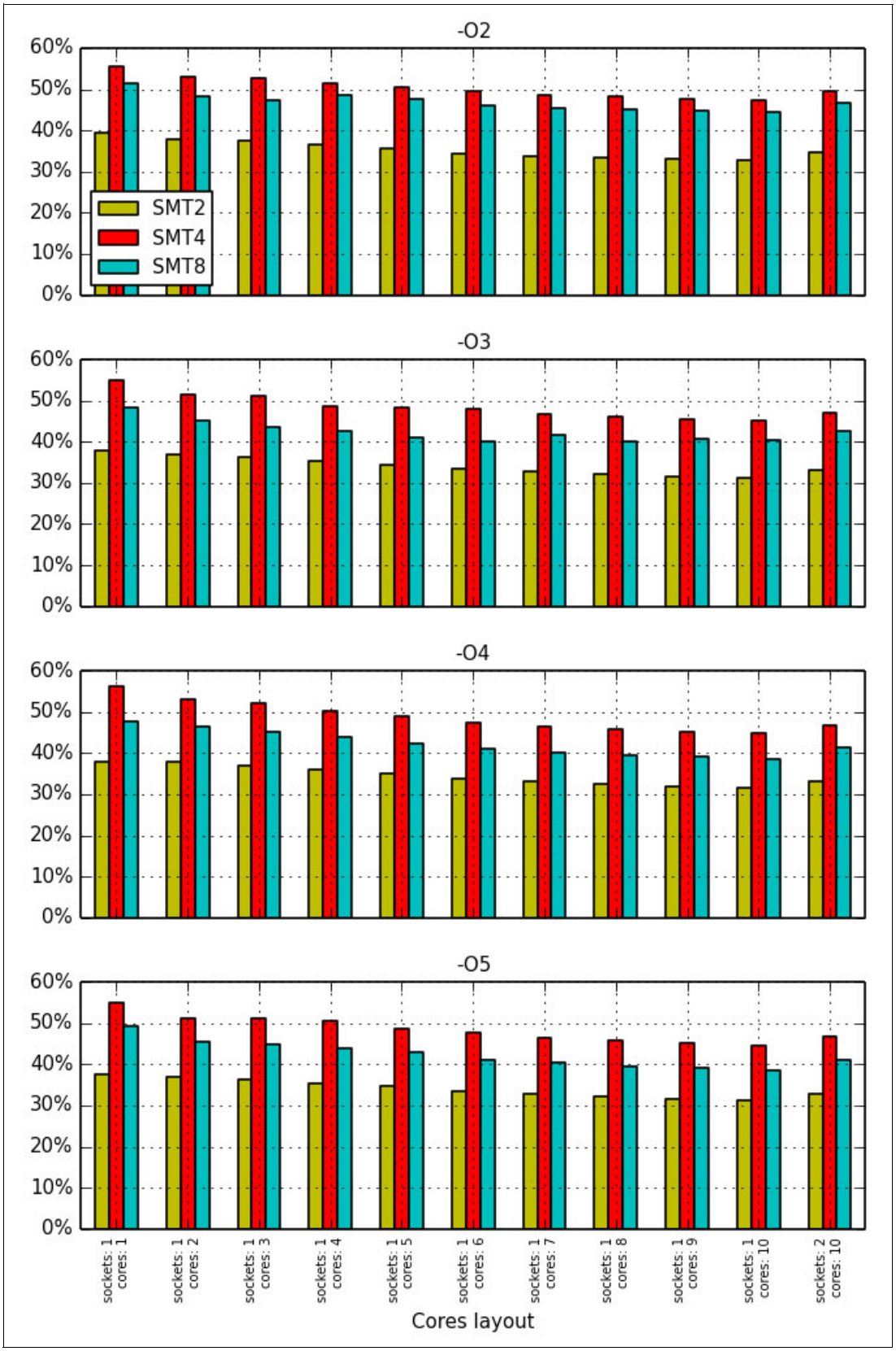

Figure A-17 NPB: Performance benefits from the rational choice of SMT mode for the ua.C benchmark

Choice of SMT mode for computing nodes

Many NPB applications benefit from SMT8 mode. So, from the general system management point of view, it can be unwise to restrict users to lower SMT values. The administrator should consider the following recommendations:

- Run the system in SMT8 mode
- Use a processor core as a unit of hardware resource allocation[5]

By using the physical processor as a unit of hardware resource allocation, you ensure that no other applications use the idle logical processors of a physical core that is assigned to the user. Before the users run a productive workload, they need to execute several benchmarks for an application of their choice. The benchmarks are necessary to determine a favorable number of software threads to run at each core. After the value is identified, that number needs to be taken into account when users arrange productive workloads.

If possible, recompile the application with the *-qtune=pwr8:smtX* option of the IBM XL compiler (where *X* is 1, 2, 4, or 8, depending on the favorable SMT mode), and repeat the benchmark.

The reason for MPI applications

The same logic holds for applications that use MPI. For programs based on OpenMP, seek a favorable number of threads to execute on each core. Similarly, for an application that is created with MPI, find a favorable number of MPI processes to execute on each core.

The impact of optimization options on performance

Various compiler options affect the performance characteristics of produced binary code. However, not all of the optimization options are equally suited for all types of workloads. Compilation parameters that result in good performance for one type of application might not perform equally well for other types of computations. Choose a favorable set of compiler options based on the timing of a particular application.

The bar charts in Figure A-18 on page 291 through Figure A-26 on page 299 compare the performance benefits that come from the rational choice of compiler options for the same applications from the NPB suite (`bt.C`, `cg.C`, `ep.C`, `ft.C`, `is.C`, `lu.C`, `mg.C`, `sp.C`, and `ua.C`) that are examined in “The performance impact of a rational choice of an SMT mode” on page 279. Again, the four sets of compiler options considered are shown in Table A-2 on page 279. The application threads are bound to either 1 - 10 cores of one socket or 20 cores of two sockets.

The organization of the plots is similar to the scheme that was used in “The performance impact of a rational choice of an SMT mode” on page 279:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a particular SMT mode.
- The *x*-axis lists the number of cores that is used.
- The *y*-axis shows the performance gain as measured in percentage relative to a baseline. As a baseline, we chose a compiler option set that is less favorable for the particular application.

The results show that the rational choice of a compiler option set substantially affects the performance for all of the considered NPB applications.

[5] This recommendation is targeted to applications limited by the computing power of a processor only. It does not take into account interaction with the memory subsystem and other devices. At some supercomputing sites and user environments, it can be reasonable to use a socket or even a server as a unit of hardware resource allocation.

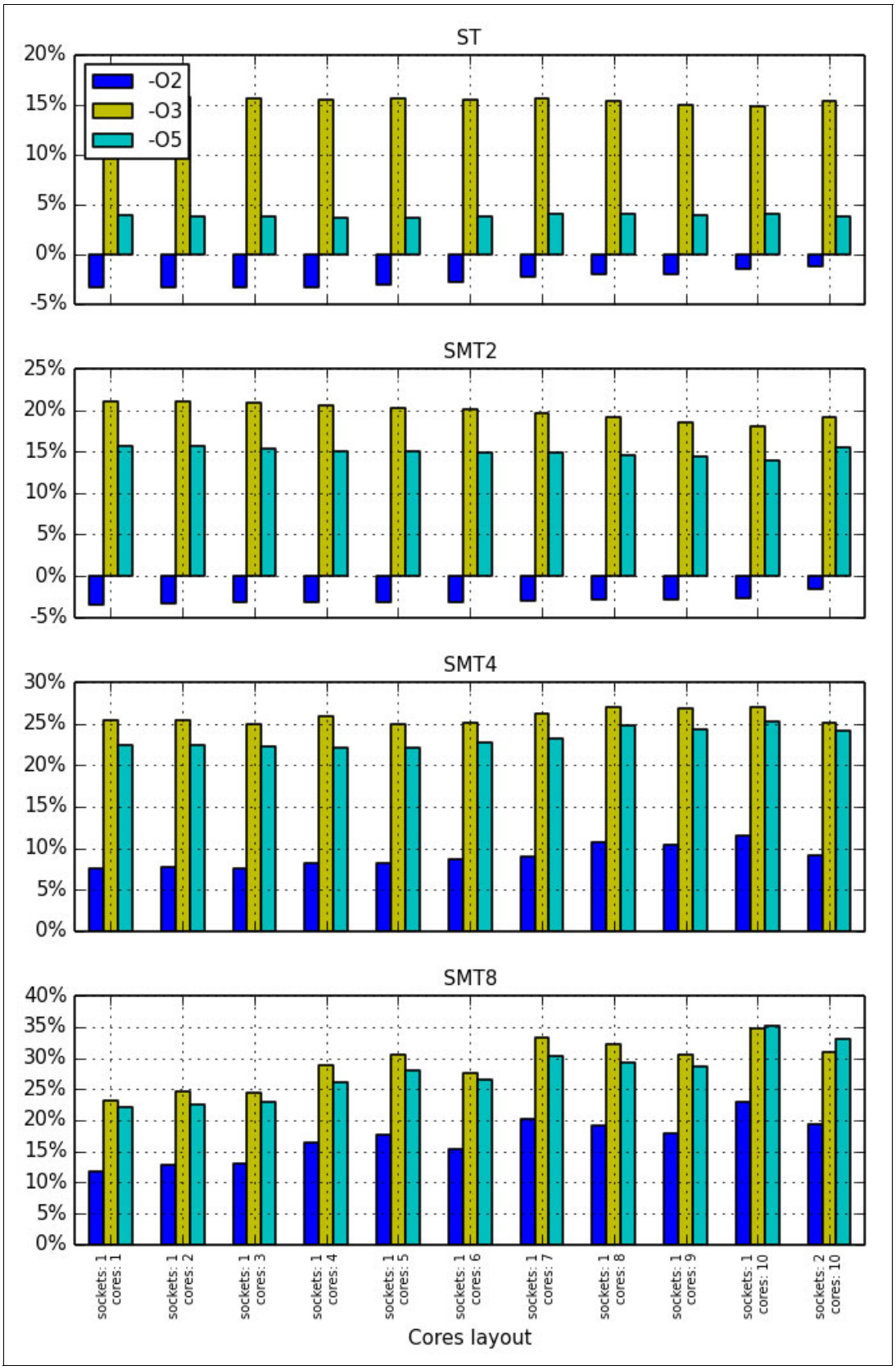

Figure A-18 NPB: Performance gain from the rational choice of compiler options for the bt.C test

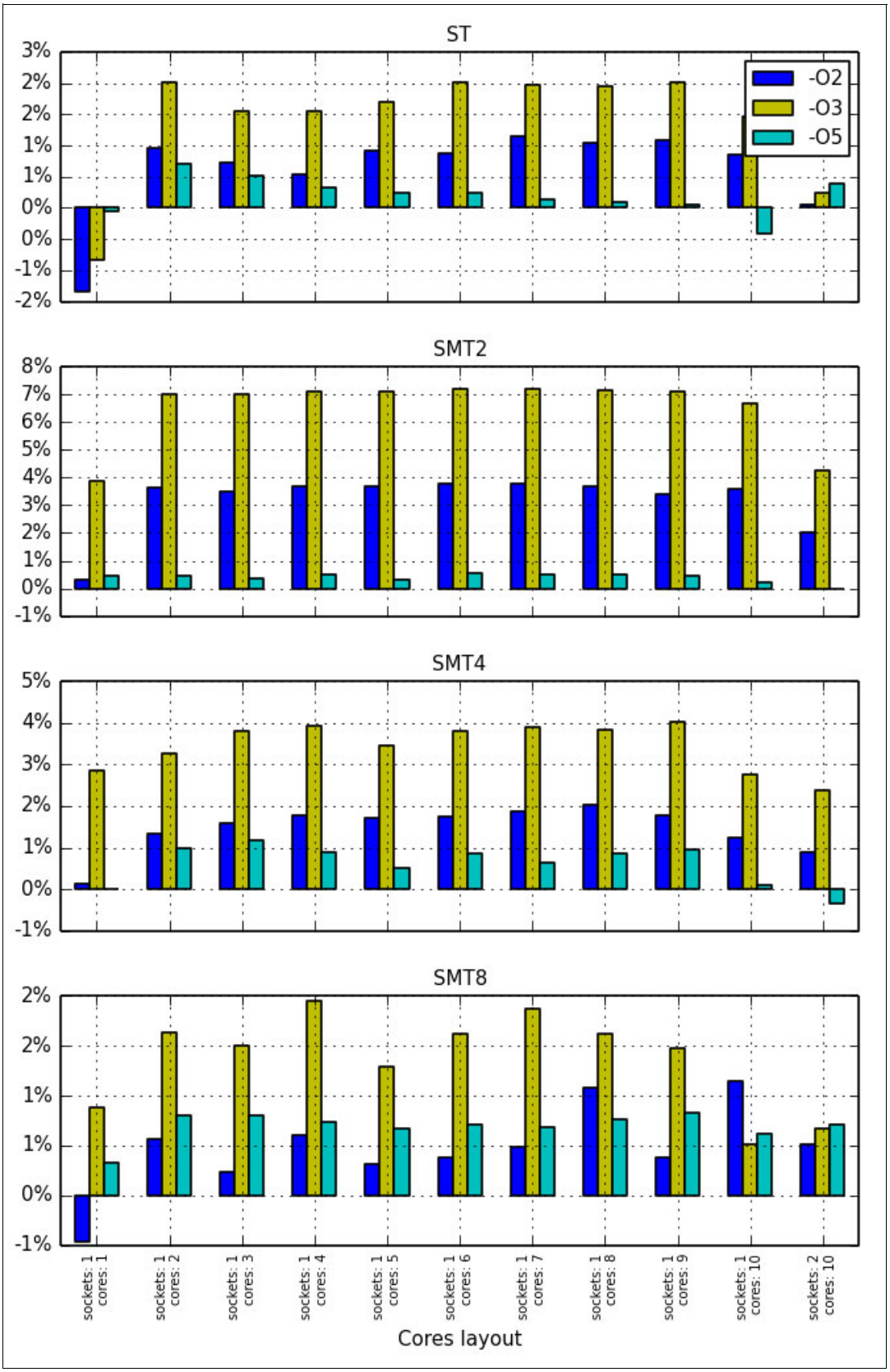

Figure A-19 NPB: Performance gain from the rational choice of compiler options for the cg.C test

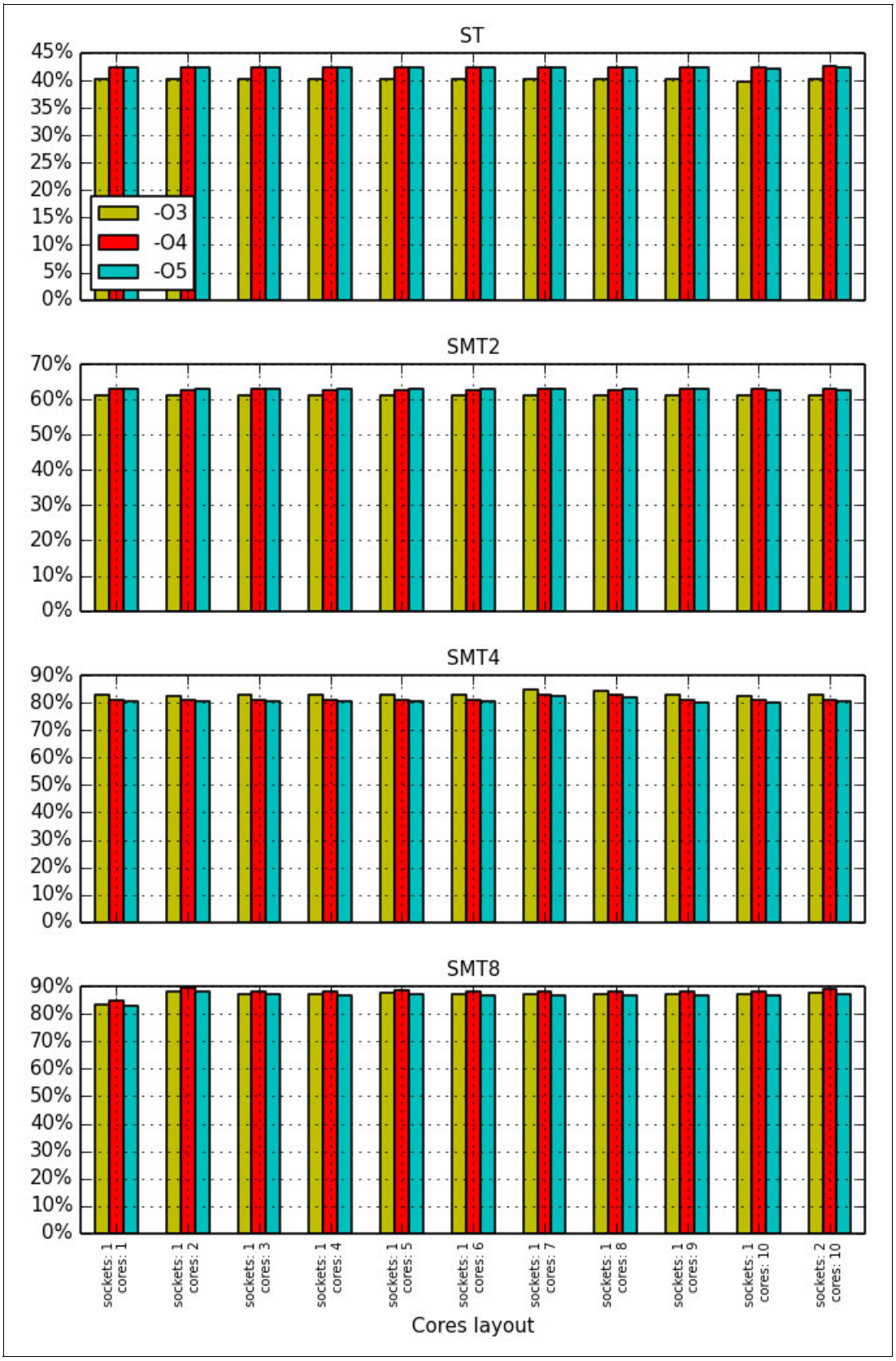

Figure A-20 NPB: Performance gain from the rational choice of compiler options for the ep.C test

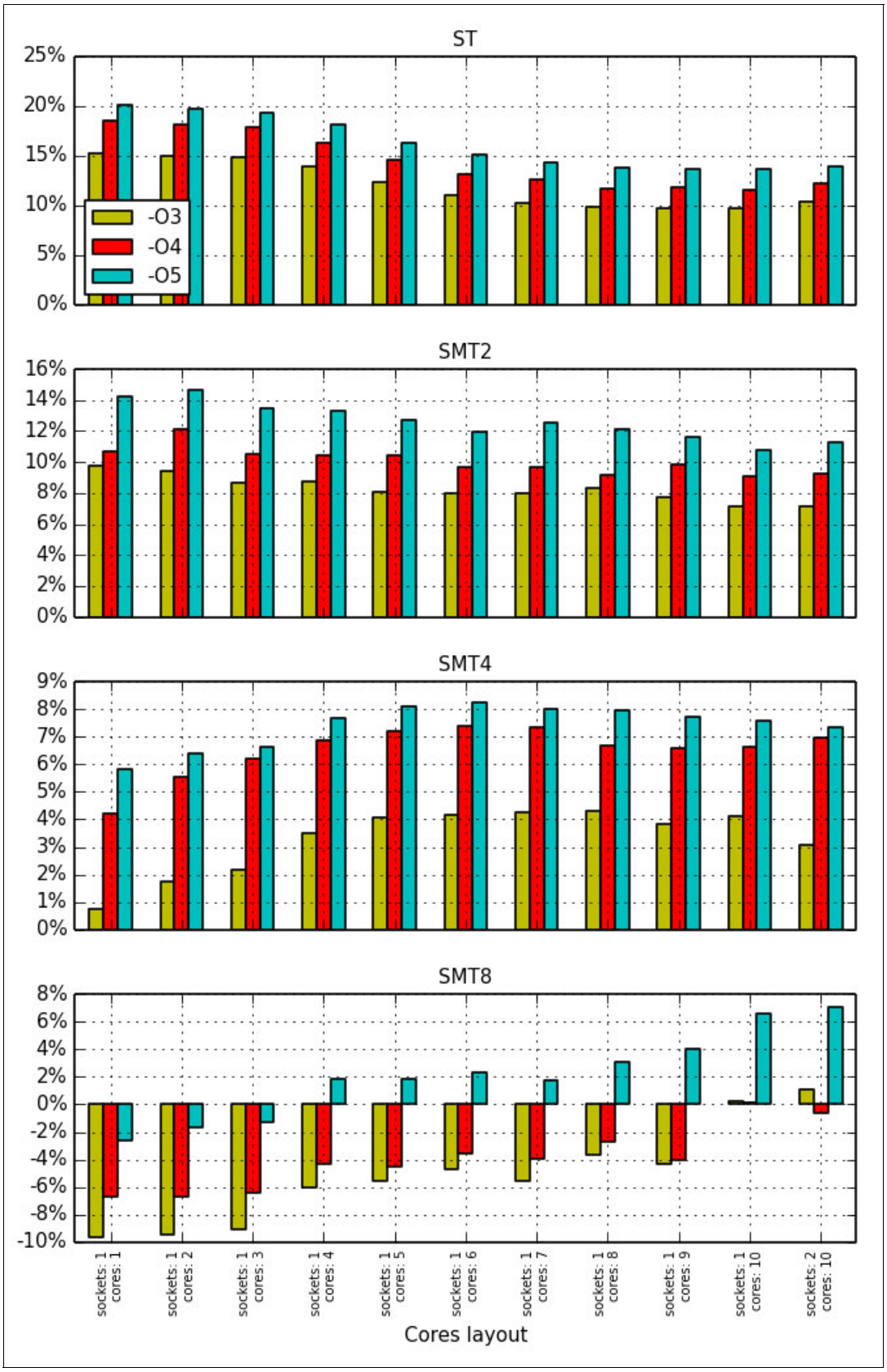

Figure A-21 NPB: Performance gain from the rational choice of compiler options for the ft.C test

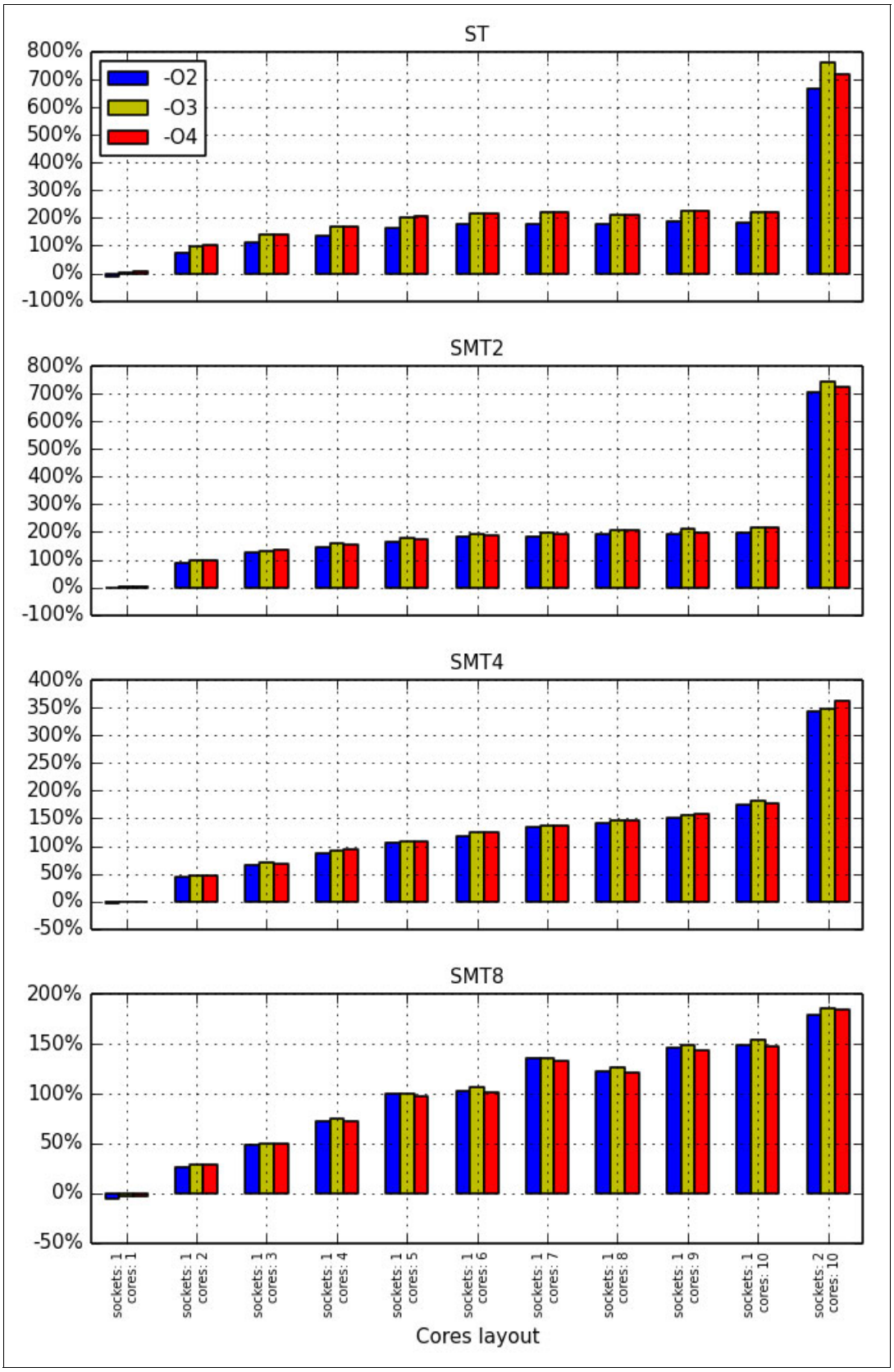

Figure A-22 NPB: Performance gain from the rational choice of compiler options for the is.C test

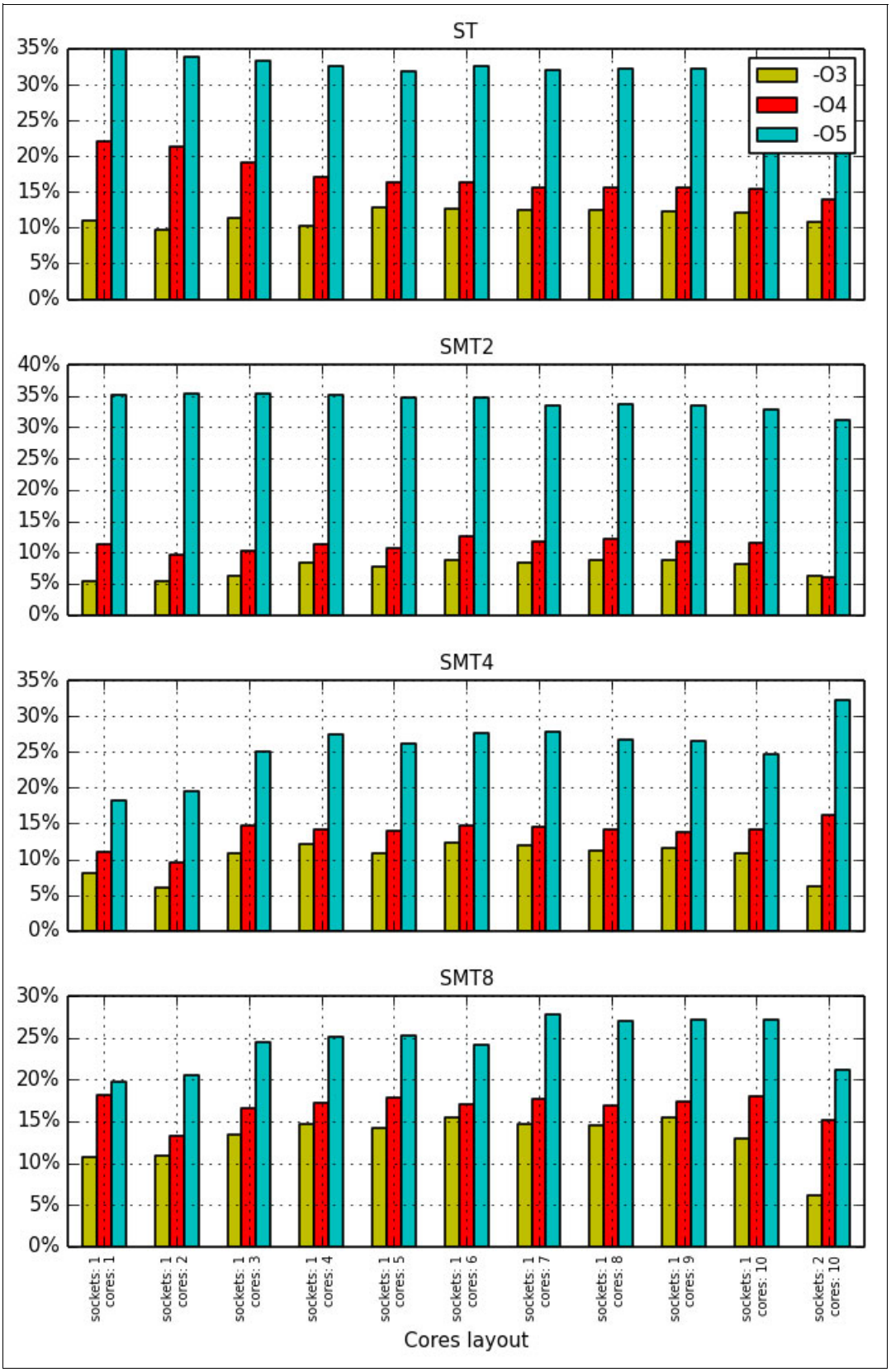

Figure A-23 NPB: Performance gain from the rational choice of compiler options for the lu.C test

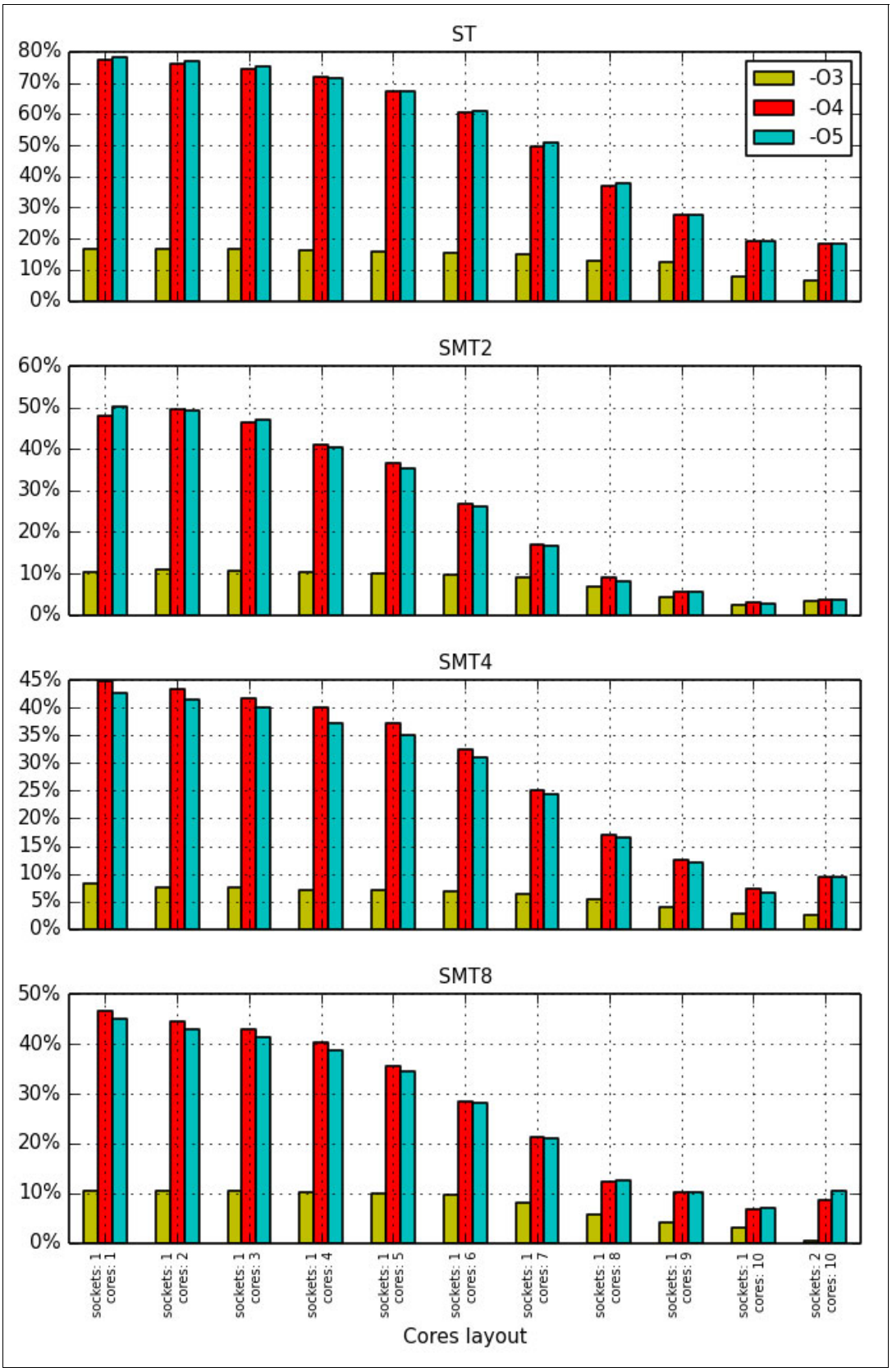

Figure A-24 NPB: Performance gain from the rational choice of compiler options for the mg.C test

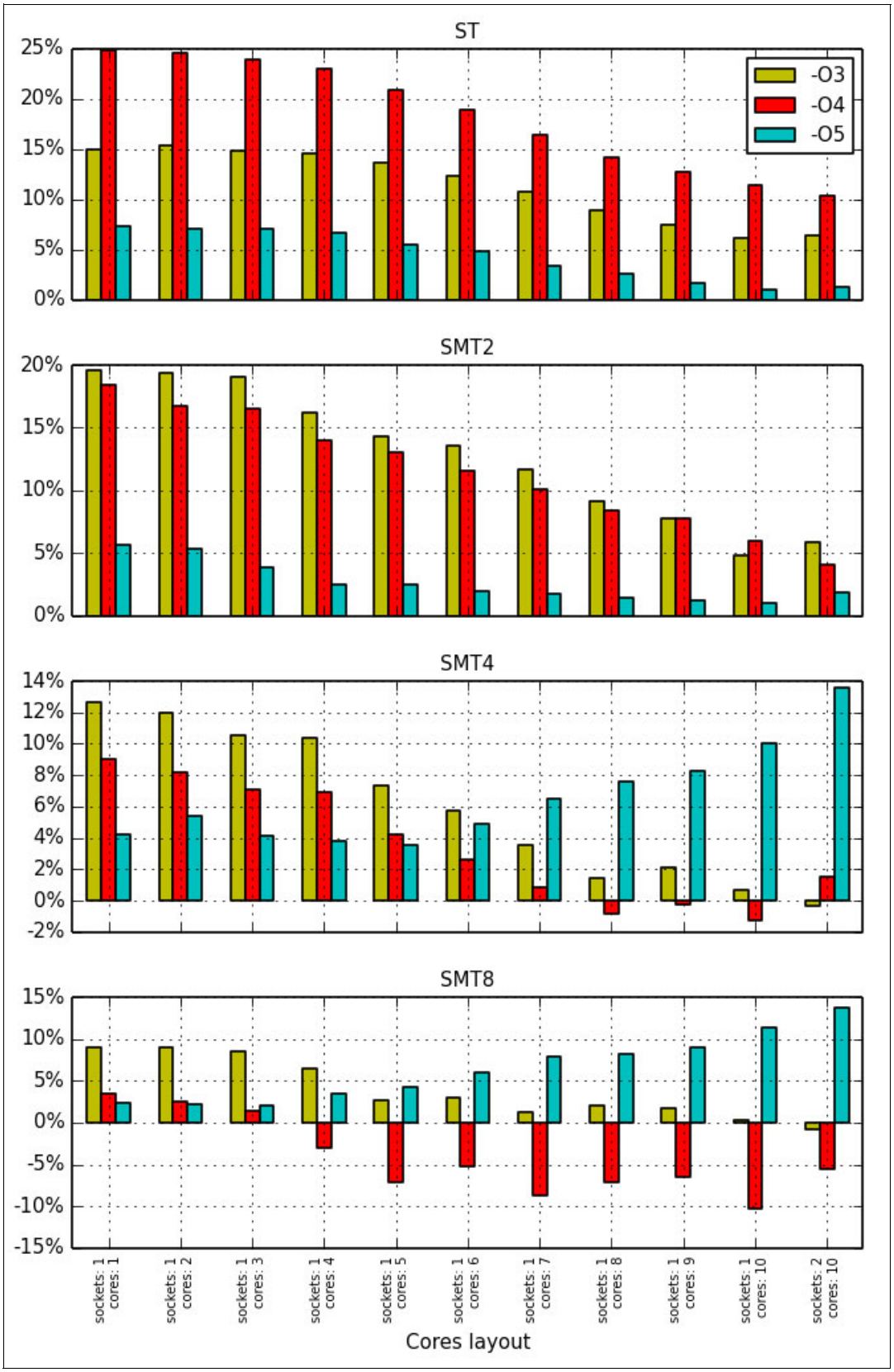

Figure A-25 NPB: Performance gain from the rational choice of compiler options for the sp.C test

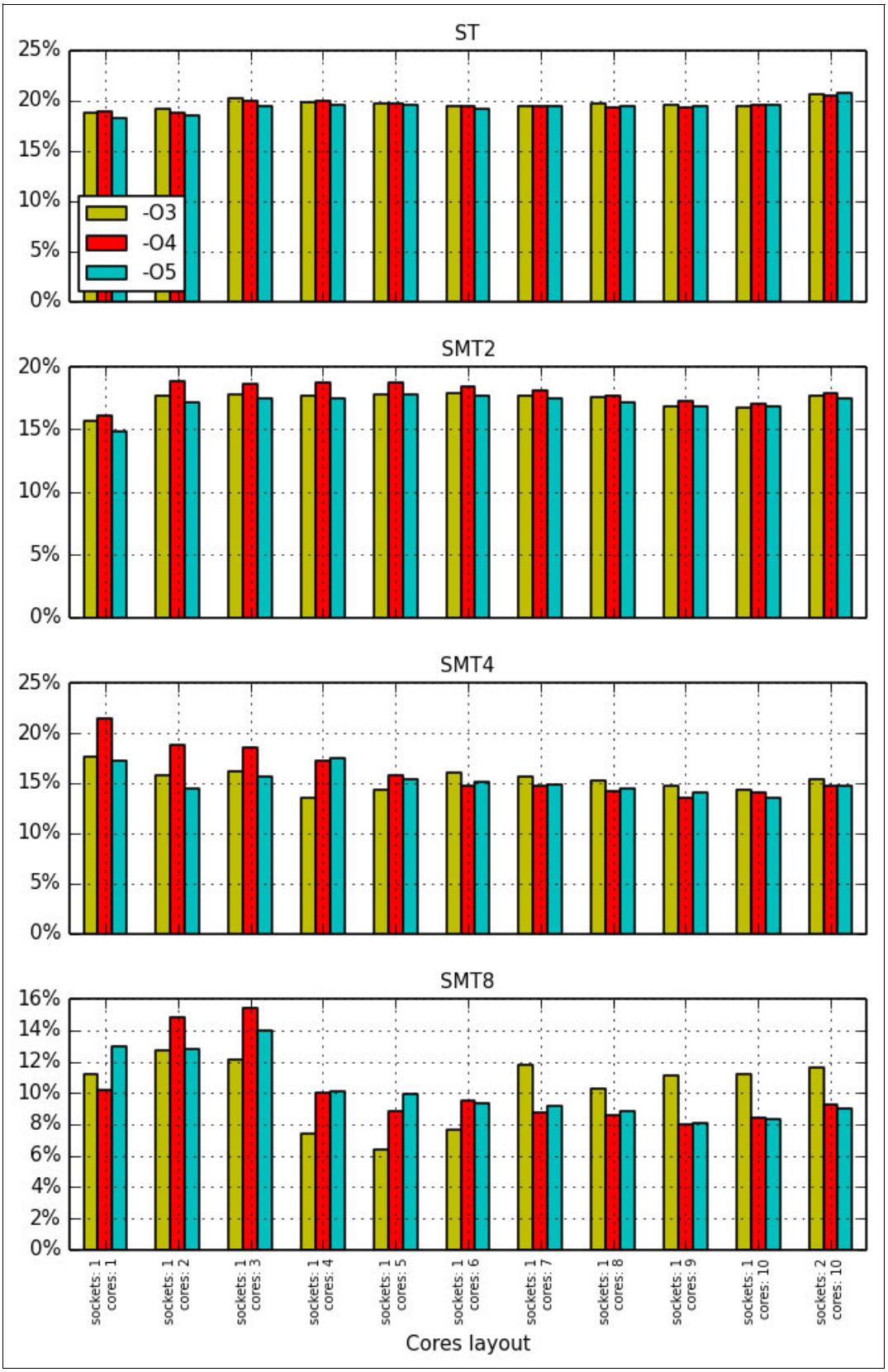

Figure A-26 NPB: Performance gain from the rational choice of compiler options for the ua.C test

Summary of favorable modes and options for applications from the NPB suite

Table A-3 summarizes SMT modes and compiler optimization options that are favorable for most of the runs performed in “The performance impact of a rational choice of an SMT mode” on page 279 and “The impact of optimization options on performance”. The row headers (**ST**, **SMT2**, **SMT4**, and **SMT8**) designate SMT modes, and the column headers (**-O2**, **-O3**, **-O4**, and **-O5**) refer to sets of compiler options that are listed in Table A-2 on page 279.

Table A-3 NPB: Favorable modes and options for applications from the NPB suite

	-O2	**-O3**	**-O4**	**-O5**
ST	—	—	—	mg.C
SMT2	—	bt.C, is.C, sp.C	—	lu.C
SMT4	—	ua.C	—	ft.C
SMT8	—	cg.C	ep.C	—

One can observe that favorable SMT modes can vary from ST to SMT8 and favorable compilation options can vary from **-O3** to **-O5**.

It is hard to know before experimenting which SMT mode and which set of compilation options will be favorable for a user application. Therefore, do not neglect benchmarking before production runs. For more details, see “General methodology of performance benchmarking” on page 301.

The importance of binding threads to logical processors

The operating system can migrate application threads between logical processors if a user does not explicitly specify thread affinity. As shown in 7.1, “Controlling the execution of multithreaded applications” on page 230, a user can specify the binding of application threads to a specific group of logical processors. One option for binding threads is to use system calls in a source code. The other option is to set environment variables that control threads affinity.

For technical computing workloads, you typically want to ensure that application threads are bound to logical processors. Thread affinity often helps to use the POWER architecture memory hierarchy and reduce the processor usage that is related to the migration of threads by an operating system.

To demonstrate the importance of this technique, the `mg.C` test from the NPB suite was chosen as an example. The performance of the `mg.C` application peaks at SMT1 (see Table A-3 on page 300). For this benchmark, you can expect a penalty if an operating system puts more than one thread on each core.

Figure A-27 shows the performance improvement that was obtained by binding application threads to logical processors. The baseline corresponds to runs without affinity. The bars show the relative gain that was obtained after the assignment of software threads to hardware threads. As SMT mode increases, the operating system has more freedom in scheduling threads. As the result, the effect of thread binding becomes more pronounced for higher values of SMT mode.

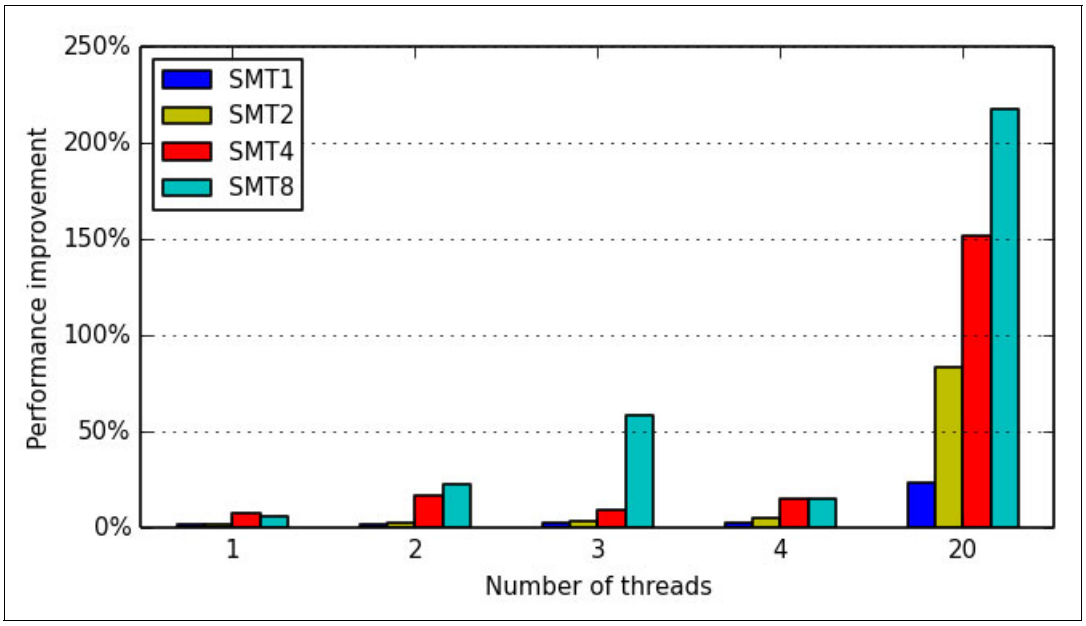

Figure A-27 Performance improvement for an application when thread binding is used

General methodology of performance benchmarking

This section describes how to evaluate the performance of an application on a system with massively multithreaded processors. It also provides some hints on how to take advantage of the performance of an application without access to the source code.

This example assumes the application is thread-parallel and does not use MPI[6] or GPGPU[7]. However, the generalization of the methodology to a broader set of applications is relatively straightforward.

For the simplicity of table structure throughout this section, the examples make the following assumptions about the configuration of a computing server:

- The server is a two-socket system with fully populated memory slots.
- The server has 10 cores per socket (20 cores in total).

The computing server reflects the architecture of the IBM Power Systems S822LC (8335-GTA) system.

This section also describes performance benchmarking summarizes the methodology in a form of a step-by-step instruction. For more informtion, see “Summary” on page 309.

[6] Message Passing Interface (MPI) is a popular message-passing application programmer interface for parallel programming.

[7] General-purpose computing on graphics processing units (GPGPU) is the use of GPUs for the solution of compute intensive data parallel problems.

Defining the purpose of performance benchmarking

Before starting a performance benchmarking, it is important to clarify the purpose of this activity. A clearly defined purpose of the benchmarking is helpful in creating a plan of the benchmarking and defining success criteria.

The purpose of performance benchmarking looks different from each of the following two points of view:

- Performance benchmarking that is carried out by an *application developer*.
- Performance benchmarking that is done by an *application user* or a *system administrator*.

Benchmarking by application developers

From the perspective of an application developer, the performance benchmarking is part of a software development process. Application developer uses performance benchmarking in pursuing the following goals:

- Comparison of a code's performance with the performance model.
- Identification of bottlenecks in a code that limit its performance (this process is typically done by using profiling).
- Comparison of the code's scalability with the scalability model.
- Identification of parts of the code that prevent scaling.
- Tuning the code to specific computer hardware.

Basically, a software developer carries out a performance benchmarking of an application to understand how to improve application performance by changing an algorithm. Performance benchmarking as it is viewed from an application developer perspective has complex methodologies. These methodologies are not covered in this publication.

Benchmarking by application users and system administrators

From the perspective of an *application user* or a *system administrator,* performance benchmarking is a necessary step before using an application for production runs on a computing system that is available for them. Application users and system administrators make the following assumptions:

- An application will be used on a computing system many times in future.
- It makes sense to invest time into performance benchmarking because the time will be made up by faster completion of production runs in future.
- Modification of an application source code is out of their scope.

The last statement implies that application users and system administrators are limited to the following options to tune the performance:

- Choice between different compilers
- Choice of compiler optimization options
- Choice of an SMT mode
- Choice of the number of computing cores
- Choice of runtime system parameters and environment variables
- Choice of operating system parameters

Essentially, applications users and system administrators are interested in how to make an application solve problems as fast as possible without changing the source code.

System administrators also needs performance benchmarking results to help determine hardware allocation resources when configuring a computing system. For more information, see “Choice of SMT mode for computing nodes” on page 290.

This books covers performance benchmarking from the perspective of an application user or a system administrator.

Plan for benchmarking

Perfom these steps for your benchmarking project:

1. A kick-off planning session with the experts
2. Workshop with the experts
3. Benchmarking
4. Session to discuss the intermediate results with the experts
5. Benchmarking
6. Preparation of the benchmarking report
7. Session to discuss the final results with the experts

You also need to prepare the following lists:

- List of applications to be benchmarked
- List of persons who are responsible for specific applications

Planning and discussion sessions are ideally face-to-face meetings between the benchmarking team and POWER architecture professionals. A workshop with the POWER architecture experts is a form of knowledge transfer educational activity with hands-on sessions.

The specific technical steps that you perform when benchmarking individual applications are discussed in the following paragraphs.

For an example of technical part of a plan, refer to the step-by-step instructions in “Summary” on page 309.

Defining the performance metric and constraints

The *performance metric* provides a measurable indicator of application performance. Essentially, performance metric is a number that can be obtained in a fully automated manner. The following are examples of the most commonly used performance metrics:

- Time of application execution
- Number of operations performed by an application in a unit of time

Some applications impose constraints in addition to the performance metric. Typically, the violation of a constraint means that running the application in such conditions is unacceptable. The following are some examples of the constraints:

- Latency of individual operations (for example, execution of a given ratio of operations takes no longer than a given time threshold).
- Quality metric of the results produced by the application does not fall below a specified threshold (for example, a video surveillance system drops no more than a specified number of video frames in a unit of time).

Defining the success criteria

Satisfying a success criteria is a formal reason to finalize performance benchmarking. Usually, success criteria is based on the performance results. Typically, *success criteria* is a numeric threshold that provides a definition of an acceptable performance (see “Defining the performance metric and constraints” on page 303).

The performance benchmarking can result in the following outcomes:

- The success criteria are satisfied. This result means that the process of performance tuning can be finalized.
- The application does not scale as expected (see “Probing the scalability” on page 306). This results indicates that you need to reconsider the success criteria.
- The success criteria are not satisfied. Use the following techniques to solve the problem:
  - Discuss the performance tuning options that you tried over and the results you obtained with the experts. For this purpose, keep a verbose benchmarking log. For more details, see “Keeping the log of benchmarking” on page 305.
  - Engage the experts to perform deep performance analysis.
  - Seek help of software developers to modify the source code.

Performance of a logical processor versus performance of a core

It is important to understand that in the most cases, there is no sense in defining success criteria based on the performance of a logical processor of a core taken in isolation. As mentioned in “The performance impact of a rational choice of an SMT mode” on page 279, the POWER8 core is able to run instructions from multiple application threads simultaneously. Therefore, the whole core is a minimal entity of reasoning about performance. For more information, see “Choice of SMT mode for computing nodes” on page 290.

Aggregated performance statistics for poorly scalable applications

Similarly, some applications are not designed to scale up to a whole computing server. For example, an application can violate constraints under a heavy load (see “Defining the performance metric and constraints” on page 303). In such situation, it makes sense to run several instances of an application on a computing node and collect aggregated performance statistics. This technique allows you to evaluate performance of a whole server running a particular workload.

Correctness and determinacy

Before you start performance benchmarking, check that the application works correctly and produces deterministic results. If the application produces undeterministic results by design (for example, the application implements the Monte-Carlo method or other stochastic approach), you have at least two options:

- Modify the application by making its output deterministic (for example, fix the seed of a random number generator).
- Develop a reliable approach for measuring performance of an undeterministic application (this can require many more runs than for a deterministic application).

During each stage of the performance tuning, verify that the application still works correctly and produces deterministic results (for example, by using regression testing).

Keeping the log of benchmarking

Preserve the history and output of all commands used in the preparation of the benchmark and during the benchmark. This log should include the following files:

- Makefiles
- Files with the output of the **make** command
- Benchmarking scripts
- Files with the output of benchmarking scripts
- Files with the output of individual benchmarking runs

A benchmarking script is a shell script that includes the following commands:

- A series of commands that capture the information about the execution environment
- A sequence of commands that run applications under various conditions

Example A-21 lists several commands that you might want to include in the beginning of a benchmarking script to capture the information about the execution environment.

Example A-21 An example of the beginning of a benchmarking script

```
#!/bin/bash -x
uname -a
tail /proc/cpuinfo
numactl -H
ppc64_cpu --smt
ppc64_cpu --cores-present
ppc64_cpu --cores-on
gcc --version
xlf -qversion
env
export OMP_DISPLAY_ENV="VERBOSE"
```

Typically, a benchmarking script combines multiple benchmarking runs in a form of loops over a number of sockets, cores, and SMT modes. By embedding the commands that you use to execute an application into a script, you keep the list of commands along with their outputs.

Running a benchmarking script

If you do not use a job scheduler to submit a benchmarking script, run a benchmarking script inside a session created by the **screen** command. Doing so gives protection against network connection issues and helps to keep applications running even during a network failure.

One option to run a benchmarking script is to execute the following command:

```
./benchmarking_script_name.sh 2>&1 | tee specific_benchmarking_scenario.log
```

This command combines the standard output and the standard error streams, whereas the **tee** command writes the combined stream both to a terminal and to a specified file.

Choosing names for log files

It is helpful to write the output of individual benchmarking runs to separate files with well-defined descriptive names. For example, we found the following format useful for the purposes of the benchmarking runs presented in “Effects of basic performance tuning techniques” on page 278:

```
$APP_NAME.t$NUM_SMT_THREADS.c$NUM_CORES.s$NUM_SOCKETS.log
```

This format facilitated the postprocessing of the benchmarking logs with the **sed** command.

Probing the scalability

Probing the scalability is a necessary step of the performance benchmarking for the following reasons:

- It is the way to evaluate the behavior of an application when the number of computing resources increases.
- It helps to determine the favorable number of processor cores to use for production runs.

To be concise, the ultimate purpose of probing the scalability is to check whether an application is scalable.

> **Note:** In the performance benchmarking within a one-node system, the scalability is probed only in a *strong* sense, also known as a *speed-up* metric. This metric is obtained by fixing the size of a problem and varying the number of processor cores. For a multi-node system, the complimentary metric is a scalability in a *weak* sense. It is obtained by measuring performance when the amount of computing work per node is fixed.

Before performing runs, the user should answer the following questions:

- Does the application has some specific limits on the number of logical processors it can use?

  (For example, some applications are designed to run with the number of logical processors that is a power of two only: 1, 2, 4, 8, and so on)
- Is the scalability model of the application available?

  The model can be pure analytical (for example, the application vendor can claim a linear scalability) or the model can be based on the results of previous runs.
- What are the scalability criteria for the application? In other words, what is the numeric threshold that defines the “yes” or “no” answer to the following formal question: “Given $N > N_{base}$ logical processors, are you ready to agree that the application is scalable on N logical processors in respect to N_{base} logical processors?”

If available, the scalability model helps you to choose the scalability criteria.

Next step is to compile the application with a basic optimization level. For example, you can use the following options:

- `-O3 -qstrict` (with IBM XL Compilers)
- `-O3` (with GCC[8] and IBM Advance Toolchain)

For more details about IBM XL Compilers, GNU Compiler Collection (GCC), and IBM Advance Toolchain, see 4.7, “IBM XL compilers, GCC, and Advance Toolchain” on page 46 and 6.1, “Compiler options” on page 156.

For scalability probing, choose the size of a problem to be large enough to fit the memory of a server. In contrast, solving a problem of a small size is typically just a waste of computing cores. If you really need to process multiple small problems, run multiple one-core tasks simultaneously on one node.

[8] GNU Compiler Collection (GCC) contains various compilers, including C, C++, and Fortran compilers.

When you decided on a problem size, run the application in ST mode with different number of logical processors. Complete the **Performance**, **Meets the scalability model? (yes/no)**, and **Meets the scalability criteria? (yes/no)** rows as shown in Table A-4.

Table A-4 A skeleton of a scalability probing measurements table

Number of cores	1	2	3	…	10	12	14	16	18	20
Number of sockets	1	1	1	…	1	2	2	2	2	2
Number of cores per socket	1	2	3	…	10	6	7	8	9	10
Performance										
Meets the scalability model? (yes/no)	—									
Meets the scalability criteria? (yes/no)	—									

Note: Remember to bind application threads to logical processors. Failing to do so typically ends up in worse results (see “The importance of binding threads to logical processors” on page 300).

When probing the scalability, vary the number of cores, and run each core in ST mode. Therefore, the total number of threads used can be equal to the number of cores.

Depending on the results of the benchmarking, you end up with one of the following three outcomes:

- The application is scalable up to 20 cores
- The application is scalable within the socket (up to 10 cores)
- The application is not scalable within the socket

If the scalability of the application meets your expectations, proceed to the next step and evaluate the performance by using the maximum number of cores that you determined in this section.

If the application does not meet your scalability expectations, you first clarify the reasons of that behavior and repeat probing the scalability until satisfied. Only after that you proceed to the next step.

Evaluation of performance on a favorable number of cores

The performance of an application heavily depends on the choice of SMT mode and compilation modes. For more information, see “” on page 291. It is hard to know beforehand which set of compiler options and number of hardware threads per core can be a favorable choice for an application. That means, you need to run the application with several combinations of options and SMT modes and select the most appropriate one.

The previous step determined the maximum reasonable number of cores to be used by an application. For more information, see “Probing the scalability” on page 306. The next step is to use that number of cores to run the application with different sets of compiler options and different SMT modes.

You need to try each of four SMT modes for several sets of compiler options. If you are limited on time, try fewer sets of compiler options, but go through all SMT modes. Put the results of your performance measurements in Table A-5 (the table is similar to the Table A-3 on page 300). Your actual version of a table can have another number of columns and different headings of the columns depending on the sets of compiler options you choose.

Table A-5 A skeleton of a performance measurements results table

	-O2	**-O3**	**-O4**	**-O5**
ST				
SMT2				
SMT4				
SMT8				

Optionally, you can repeat the previous step (see “Probing the scalability” on page 306) with the compiler options that give the highest performance.

Evaluation of scalability

The core of the IBM POWER8 chip is a multithreaded processor. Therefore, there is no sense in measuring performance of a single application thread. There is more application performance meaning when a whole core is used. The application performance depends on the SMT mode of a core as described in “Reason behind a conscious choice of an SMT mode” on page 280.

Before the evaluation of the scalability of an application, select a favorable SMT mode and compilation options, as described in “Evaluation of performance on a favorable number of cores” on page 307. After that, use that SMT mode and vary the number of cores to get the scalability characteristics. Put the results of the measurements in the **Performance** row of Table A-6 (the table is similar to Table A-4 on page 307). Compute values for the **Speed-up** row based on the performance results. The number of columns in your version of the table depends on the maximum number of cores you obtained when you probed the scalability. For more information, see “Probing the scalability” on page 306).

Table A-6 A skeleton of a scalability evaluation results table

Number of cores	1	2	3	…	10	12	14	16	18	20
Number of sockets	1	1	1	…	1	2	2	2	2	2
Number of cores per socket	1	2	3	…	10	6	7	8	9	10
Performance										
Speed-up	—									

Note: When we probed the scalability, we had been running the cores in ST mode. In the scalability evaluation step, we ran the cores in an SMT mode that we determined in the performance evaluation step. The total number of threads in each run of the scalability evaluation is equal to the number of used cores multiplied by the number of threads per core in a chosen SMT mode.

Conclusions

As a result of the benchmark, you will have the following information for each application:

- The level of the application scalability in ST mode
- A favorable SMT mode and compiler options that delivered the highest performance
- Scalability characteristics in the favorable SMT mode

Summary

To summarize, use this step-by-step instruction that can facilitate your benchmarking.

1. Define the purpose of the benchmarking (see “Defining the purpose of performance benchmarking” on page 302) and choose which of the following options you will use for performance tuning:
   - Choice between different compilers
   - Choice of compiler optimization options
   - Choice of an SMT mode
   - Choice of the number of computing cores
   - Choice of runtime system parameters and environment variables
   - Choice of operating system parameters
2. Create the benchmarking plan (see “Plan for benchmarking” on page 303).
3. Define the performance metric (see “Defining the performance metric and constraints” on page 303.)
4. Define the success criteria (see “Defining the success criteria” on page 304).
5. Verify that the application works correctly (see “Correctness and determinacy” on page 304).
6. Probe the scalability (see “Probing the scalability” on page 306) to determine the limits of the application scalability.
   a. Complete the questionnaire on a scalability model and scalability criteria (see page 306).
   b. Complete a results table (see Table A-4 on page 307).
7. Obtain favorable SMT mode and compilation options (see “Evaluation of performance on a favorable number of cores” on page 307) by completing Table A-5 on page 308.
8. Evaluate the scalability (see “Evaluation of scalability” on page 308) by completing Table A-6 on page 308.

Sample code for the construction of thread affinity strings

Example A-22 provides the source code `t_map.c` for the program `t_map`. You can employ this small utility to construct a text string that describes the mapping of OpenMP threads of an application to logical processors. Text strings of this kind are intended to be assigned to OpenMP thread affinity environment variables.

Example A-22 Source code t_map.c (in C programming language) for the program t_map

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

#define MAX_TPC 8 // 8 is for the POWER8 processor (max SMT mode is SMT8)
```

```
#define MAX_CPS 10 // 8 is for a 16-core system, 10 is for a 20-core system
#define MAX_SPS 2
#define MAX_THR (MAX_TPC * MAX_CPS * MAX_SPS)

void Print_Map(int sps, int cps, int tpc, int base) {
  const int maps[MAX_TPC][MAX_TPC] = {
    { 0                      },
    { 0,          4          },
    { 0,    2,    4          },
    { 0,    2,    4,    6    },
    { 0, 1, 2,    4,    6    },
    { 0, 1, 2,    4, 5, 6    },
    { 0, 1, 2, 3, 4, 5, 6    },
    { 0, 1, 2, 3, 4, 5, 6, 7 }
  };

  const int sep = ',';

  int thread, core, socket;

  int tot = sps * cps * tpc;
  int cur = 0;

  for (socket = 0; socket < sps; ++socket) {
    for (core = 0; core < cps; ++core) {
      for (thread = 0; thread < tpc; ++thread) {
        int shift = socket * MAX_CPS * MAX_TPC +
                                core * MAX_TPC;
        shift += base;
        ++cur;
        int c = (cur != tot) ? sep : '\n';
        printf("%d%c", shift + maps[tpc-1][thread], c);
      }
    }
  }
}

void Print_Usage(char **argv) {
    fprintf(stderr, "Usage: %s "
        "threads_per_core=[1-%d] "
        "cores_per_socket=[1-%d] "
        "sockets_per_system=[1-%d] "
        "base_thread=[0-%d]\n",
        argv[0], MAX_TPC, MAX_CPS, MAX_SPS, MAX_THR-1);
}

int main(int argc, char **argv) {
  const int num_args = 4;

  if (argc != num_args+1) {
    fprintf(stderr, "Invalid number of arguments (%d). Expecting %d "
        "arguments.\n", argc-1, num_args);
    Print_Usage(argv);
    exit(EXIT_FAILURE);
  }
```

```
  int tpc = atoi(argv[1]);
  int cps = atoi(argv[2]);
  int sps = atoi(argv[3]);
  int base = atoi(argv[4]);

  if (tpc < 1 || tpc > MAX_TPC ||
      cps < 1 || cps > MAX_CPS ||
      sps < 1 || sps > MAX_SPS) {
    fprintf(stderr, "Invalid value(s) specified in the command line\n");
    Print_Usage(argv);
    exit(EXIT_FAILURE);
  }

  int tot = sps * cps * tpc;

  if (base < 0 || base+tot-1 >= MAX_THR) {
    fprintf(stderr, "Invalid value specified for the base thread (%d). "
        "Expected [0, %d]\n", base, MAX_THR-tot);
    Print_Usage(argv);
    exit(EXIT_FAILURE);
  }

  Print_Map(sps, cps, tpc, base);

  return EXIT_SUCCESS;
}
```

Note: Example A-22 lists a revised version of the code that originally appeared in Appendix D of *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263. The code was adjusted to better fit the architecture of the IBM Power System S822LC servers.

To use the tool, you need to compile the code first with the C compiler of your choice. For example, with **gcc** compiler you need to execute the following command:

```
$ gcc -o t_map t_map.c
```

If you run the tool without any arguments, you get a brief hint on the usage:

```
$ ./t_map
Invalid number of arguments (0). Expecting 4 arguments.
Usage: ./t_map threads_per_core=[1-8] cores_per_socket=[1-10]
sockets_per_system=[1-2] base_thread=[0-159]
```

The utility needs you to specify the amount and location of resources you want to employ:

- Number of threads per core
- Number of cores per socket
- Number of sockets
- The initial logical processor number (counting from zero)

Example A-23 shows how to generate a thread mapping string for an OpenMP application that usse the following resources of a 20-core IBM Power Systems S822LC server:

- Twenty OpenMP threads in total
- Two threads on each core
- Ten cores on each socket
- Only the second socket

Example A-23 A sample command for the generation of a thread mapping string

```
$ ./t_map 2 10 1 80
80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
```

The runtime system of an OpenMP application obtains the thread mapping string from an environment variable. You need to use different OpenMP thread affinity environment variables depending on the compiler you use for your OpenMP application. Table A-7 provides a reference on OpenMP thread affinity environment variables.

Table A-7 OpenMP thread affinity environment variables

Compiler family	Some compilers from the compiler family	OpenMP thread affinity environment variable
GNU Compiler Collection (GCC)	**gcc** **g++** **gfortran**	**GOMP_CPU_AFFINITY**
IBM XL compilers	**xlc_r** **xlc++_r** **xlf2008_r**	**XLSMPOPTS**, suboption **procs**

The information in Table A-7 also applies to the OpenMP applications built with the derivatives of GCC and IBM XL compilers (for example, MPI wrappers).

Example A-24 shows how to assign values to OpenMP thread environment variables to implement the scenario in Example A-23.

Example A-24 Assigning values to the OpenMP thread affinity environment variables

```
$ export XLSMPOPTS=procs="`t_map 2 10 1 80`"
$ echo $XLSMPOPTS
procs=80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
$ export GOMP_CPU_AFFINITY="`t_map 2 10 1 80`"
$ echo $GOMP_CPU_AFFINITY
80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
```

Note: Example A-24 implies that the `t_map` program is in your **PATH**. You need to specify the full path to the tool if the operating system does not find it in your **PATH**.

Example A-24 shows the value of the environment variables with the **echo** command just to demonstrate the result of the assignment. This command does not affect the affinity.

ESSL performance results

The ESSL library includes the implementation of the famous DGEMM routine, which is used in a large spectrum of libraries, benchmarks, and other ESSL routines. Therefore, its performance is significant.

DGEMM implements the following formula:

*C = alpha*A*B + beta*C*

Here, *alpha* and *beta* are real scalar values, *A*, *B*, and *C* are matrixes of conforming shape.

Example A-25 contains sample Fortran program with multiple calls of DGEMM for different sizes of input matrixes.

Example A-25 ESSL Fortran example source code dgemm_sample.f

```
program dgemm_sample
implicit none

real*8 diff
integer n, m, k
integer maxn
integer step
integer i
real*8,allocatable :: a(:,:), b(:,:), c(:,:)
real*8,allocatable :: at(:,:), bt(:,:), ct(:,:)
real*8 rmin
real*8 seed1, seed2, seed3
real*8 dtime, mflop
real*8 flop
integer tdummy, tnull, tstart, tend, trate, tmax

maxn = 20000
step = 1000

seed1 = 5.0d0
seed2 = 7.0d0
seed3 = 9.0d0
rmin = -0.5d0

call system_clock(tdummy,trate,tmax)
call system_clock(tstart,trate,tmax)
call system_clock(tend,trate,tmax)
tnull = tend - tstart

allocate( at(maxn, maxn) )
allocate( bt(maxn, maxn) )
allocate( ct(maxn, maxn) )
allocate( a(maxn, maxn) )
allocate( b(maxn, maxn) )
allocate( c(maxn, maxn) )

call durand(seed1, maxn*maxn, at)
call dscal(maxn*maxn, 1.0d0, at, 1)
call daxpy(maxn*maxn, 1.0d0, rmin, 0, at, 1)
```

```
      call durand(seed2, maxn*maxn, bt)
      call dscal(maxn*maxn, 1.0d0, bt, 1)
      call daxpy(maxn*maxn, 1.0d0, rmin, 0, bt, 1)

      call durand(seed3, maxn*maxn, ct)
      call dscal(maxn*maxn, 1.0d0, ct, 1)
      call daxpy(maxn*maxn, 1.0d0, rmin, 0, ct, 1)

      do i = 1, maxn/step
        n = i*step
        m = n
        k = n

        flop = dfloat(n)*dfloat(m)*(2.0d0*(dfloat(k)-1.0d0))

        call dcopy(n*k, at, 1, a, 1)
        call dcopy(k*m, bt, 1, b, 1)
        call dcopy(n*m, ct, 1, c, 1)

        call system_clock(tstart,trate,tmax)
        call dgemm('N','N',m,n,k,1.0d0,a,n,b,k,1.0d0,c,n);
        call system_clock(tend,trate,tmax)

        dtime = dfloat(tend-tstart)/dfloat(trate)
        mflop = flop/dtime/1000000.0d0

        write(*,1000) n, dtime, mflop

      enddo

 1000 format(I6,1X,F10.4,1X,F14.2)

      end program dgemm_sample
```

Commands from Example A-26 compile and execute this program using different types of ESSL library (serial, SMP, and SMP CUDA). For SMP runs, it uses 20 SMP threads with each thread bound to a different POWER8 physical core.

Example A-26 Compilation and execution of dgemm_sample.f

```
echo "Serial run"
xlf_r -O3 -qnosave dgemm_sample.f -lessl -o dgemm_fserial
./dgemm_fserial

echo "SMP run"
export
XLSMPOPTS=parthds=20:spins=0:yields=0:PROCS=0,8,16,24,32,40,48,56,64,72,80,88,96,1
04,112,120,128,136,144,152
xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmp -o dgemm_fsmp
./dgemm_fsmp

echo "SMP CUDA run with 4 GPUs hybrid mode"
xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmpcuda -lcublas -lcudart
-L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -o dgemm_fcuda
./dgemm_fcuda
```

```
echo "SMP CUDA run with 4 GPUs non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1,2
./dgemm_fcuda

echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=1,2
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (1st) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (1st) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (4th) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=3
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (4th) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
```

Figure A-28 shows the dependence of performance in MFlops to size of matrixes for different calls from Example A-25 on page 313.

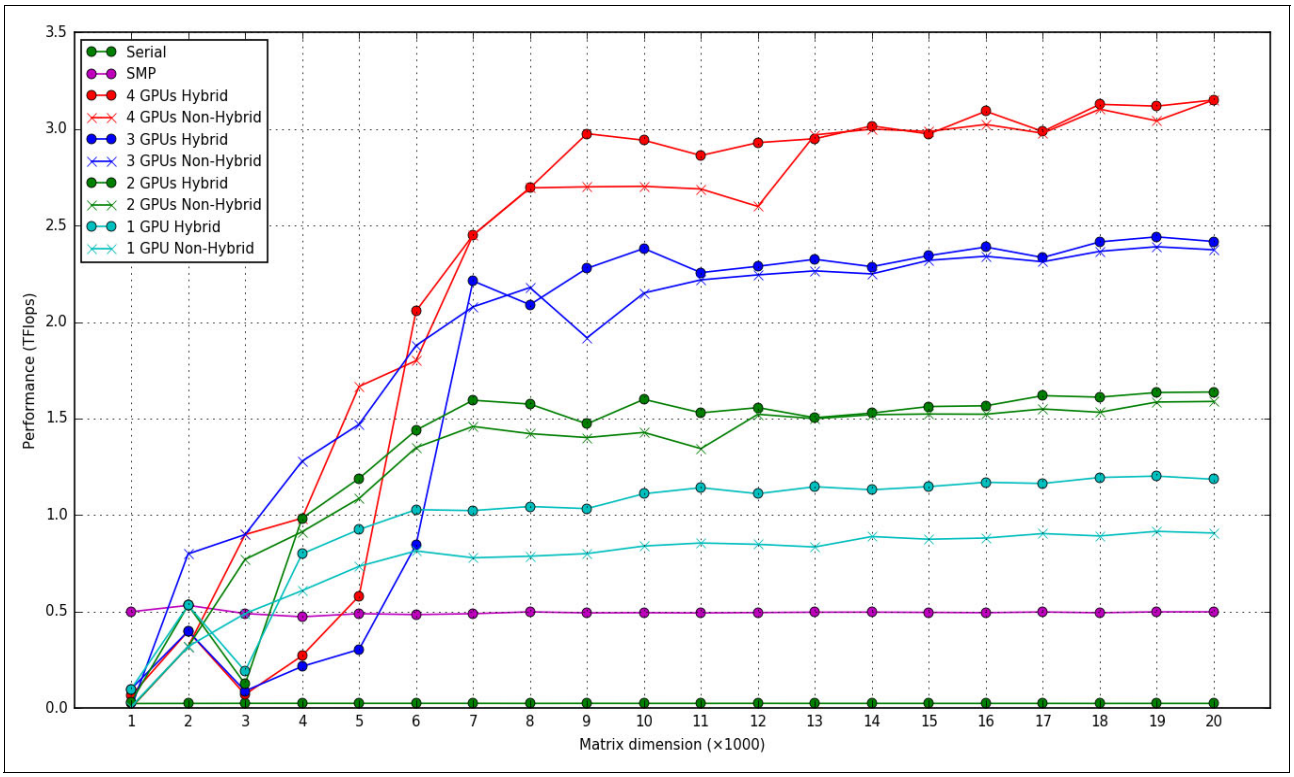

Figure A-28 DGEMM performance results for different type of ESSL library

The chart shows that the GPU gives advantage in performance starting from the 3000 - 5000 problem size. A size less than that is not enough to run the computation in the NVIDIA (using GPU) card, and it is better to let the computation run in the CPU using the ESSL SMP library.

Another good conclusion from these performance results is to use hybrid calls of the ESSL SMP CUDA library for large problem sizes, especially for runs with one GPU where improvement of performance is about 20%. You can look closer at performance of one GPU case in Figure A-29, and additionally can compare runs with different GPUs in the system.

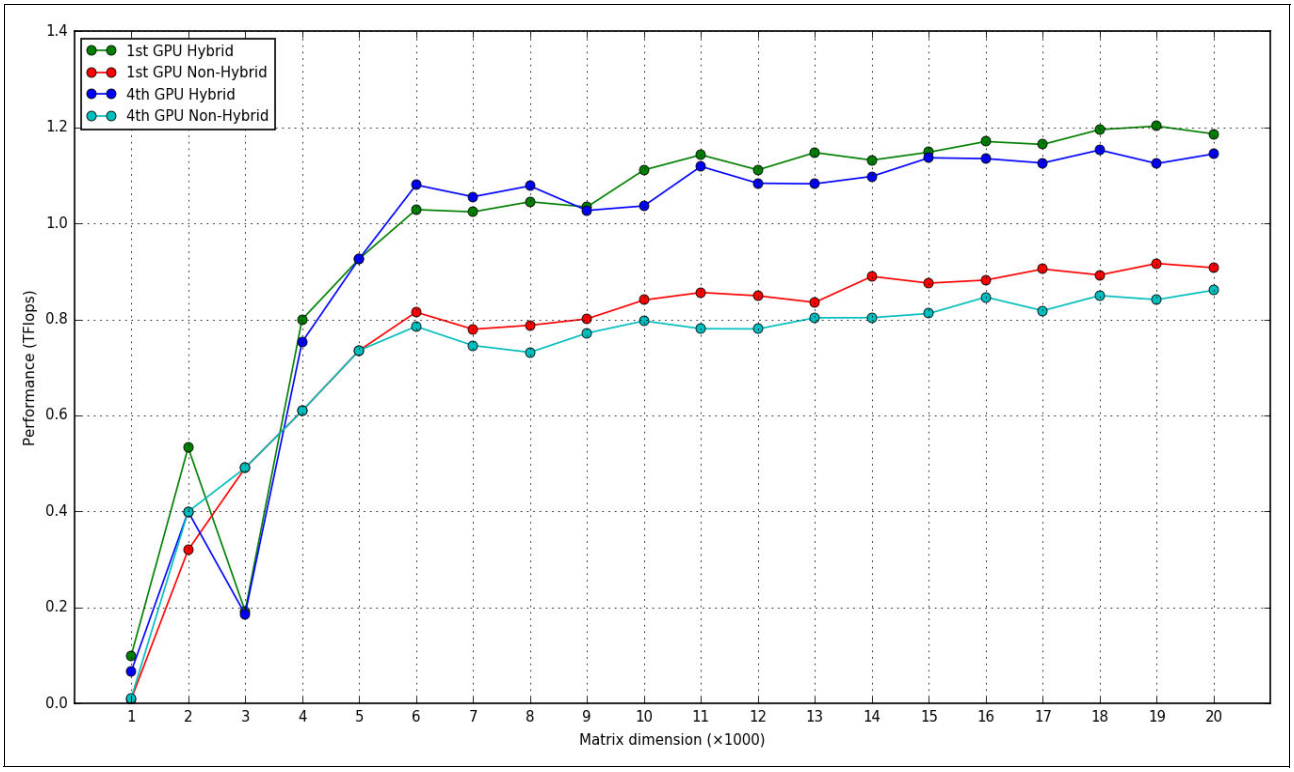

Figure A-29 ESSL SMP CUDA runs with one GPU

The first GPU has slightly better results than the fourth GPU. It is possible because these GPUs connected to different NUMA nodes and connection delays can occur. Run your program on different GPUs to find the environment with the best performance results.

Figure A-30 shows the performance chart for different combinations of two GPUs calls.

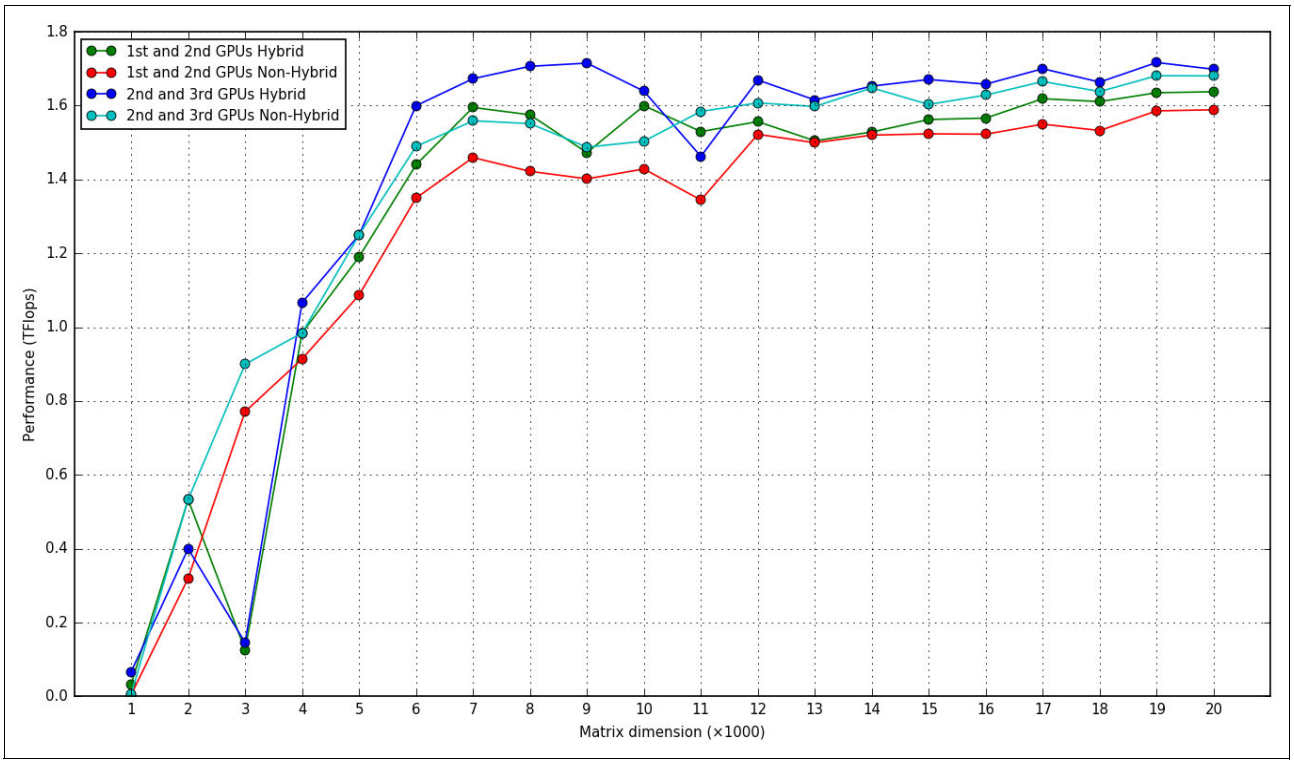

Figure A-30 ESSL SMP CUDA runs with two GPUs

Note: Performance results can have drops due other jobs are running in the system at the same time.

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

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *IBM Power Systems S822LC Technical Overview and Introduction*, REDP-5283
- *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263
- *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*, SG24-8171

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- OpenPOWER Foundation

  http://openpowerfoundation.org
- IBM Knowledge Center Power Systems POWER8 8335-GTA (Power System S822LC)

  http://www.ibm.com/support/knowledgecenter/HW4M4/p8hdx/8335_gta_landing.htm
- xCAT (Extreme Cloud/Cluster Administration Toolkit)

  http://xcat.org
- IBM Platform Cluster Manager

  http://www.ibm.com/support/knowledgecenter/SSDV85/product_welcome_pcm.html
- IBM Platform HPC

  http://www.ibm.com/support/knowledgecenter/SSENRW/product_welcome_hpc.html
- IBM Spectrum LSF

  http://www.ibm.com/systems/spectrum-computing/products/lsf/index.html
- System Planning Tool

  http://www.ibm.com/systems/support/tools/systemplanningtool/
- IBM System Storage

  http://www.ibm.com/systems/storage/disk

- NVIDIA GPU Tesla

  http://www.nvidia.com/object/tesla-servers.html
- IBM ESSL

  http://www.ibm.com/support/knowledgecenter/#!/SSFHY8/essl_welcome.html
- IBM Parallel ESSL

  http://www.ibm.com/support/knowledgecenter/#!/SSNR5K/pessl_welcome.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

Implementing an IBM High-Performance Computing Solution on

SG24-8280-00

ISBN 0738441872

IBM

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8280-00

ISBN 0738441872

Printed in U.S.A.