Redbooks
ibm.com/redbooks

Implementing an IBM High-Performance Computing Solution on IBM POWER8

Dino Quintero
Wei Li
Wainer dos Santos Moschetta
Mauricio Faria de Oliveira
Alexander Pozdneev

Cloud

Power Systems

IBM

Redbooks

International Technical Support Organization

Implementing an IBM High-Performance Computing Solution on IBM POWER8

September 2015

SG24-8263-00

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

First Edition (September 2015)

This edition applies to IBM Spectrum Scale 4.1.0-6, Ubuntu Server 14.04.1 LTS, xCAT 2.9, Mellanox OFED 2.3-2.0.0, XLC 13.1.1-0, XL Fortran 15.1.1-0, IBM Advance Toolchain 8.0 (provides GCC-4.9.2), Parallel Environment Runtime 2.1.0.0, Parallel Environment Developer Edition 2.1.0.0, ESSL 5.3.1-0, PESSL 5.1.0-0, IBM JDK 7.1-2.0, Platform LSF 9.1.3, and NVIDIA CUDA Toolkit 5.5-54 (on Ubuntu 14.10).

Contents

Notices

Trademarks

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication documents and addresses topics to provide step-by-step programming concepts to tune the applications to use IBM POWER8® hardware architecture with the technical computing software stack. This publication explores, tests, and documents how to implement an IBM high-performance computing (HPC) solution on POWER8 by using IBM technical innovations to help solve challenging scientific, technical, and business problems.

This book demonstrates and documents that the combination of IBM HPC hardware and software solutions delivers significant value to technical computing clients in need of cost-effective, highly scalable, and robust solutions.

This book targets technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) who are responsible for delivering cost-effective HPC solutions that help uncover insights among clients' data so that they can act to 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 Senior Certified IT Specialist with the ITSO in Poughkeepsie, NY. 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.

Wei Li is a Staff Software Engineer in Blue Gene and a member of the Fortran compiler test team with the IBM Canada Software Laboratory, Toronto. He joined IBM in 2008 and has nine years of experience in the IT industry. His areas of expertise include IBM AIX®, Linux, Blue Gene, IBM System p® hardware, and HPC software solutions. He holds an IBM Certification in pSeries AIX 6 System Support. He holds a Bachelor of Engineering and a Master of Engineering degree in Information and Communication Engineering from the University of Science and Technology of China.

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 PowerLinux™ project. He has six years of experience with designing and implementing software development tools for Linux on IBM platforms. Wainer holds a Bachelor degree in Computer Science from the University of São Paulo. He co-authored IBM Redbooks publications, *IBM Parallel Environment (PE) Developer Edition*, SG24-8075, and *Performance Optimization and Tuning Techniques for IBM Power Systems processors, including IBM POWER8,* SG24-8171, has published articles and videos for the IBM developerWorks® website, and contributes to the IBM PowerLinux technical community blog.

Mauricio Faria de Oliveira is a Staff Software Engineer in the IBM Linux Technology Center, Brazil. His areas of expertise include Linux performance and Debian/Ubuntu distributions on IBM Power Systems™. He worked with official benchmark publications and the early development of Debian on little-endian 64-bit IBM PowerPC®. Mauricio holds a Master of Computer Science and Technology degree and a Bachelor of Engineering degree in Computer Engineering from Federal University of Itajubá.

Alexander Pozdneev is a Technical Consultant in STG Lab Services at IBM East Europe/Asia, Moscow, Russia. He has 11 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 Buslovich
International Technical Support Organization, Poughkeepsie Center

Ruzhu Chen, Gerald Gantz, Holly Gross, Brian Hart, Xinghong He, John Lemek, Bill LePera, Joan McComb, Mark Perez, David Shapiro, Tzy-hwa K Tzeng, Duane Witherspoon
IBM US

Guang Cheng Li, Xiao Lu Qi, Si Si Shen, Yu Song, Er Tao Zhao
IBM China

Laura Flores Garnier
IBM Costa Rica

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

This publication describes the implementation of an IBM high-performance computing (HPC) solution on IBM Power scale-out/Linux servers with the IBM POWER8 processor. This implementation uses the IBM cluster software stack and InfiniBand interconnect.

This publication covers topics, such as environment setup, administration, and monitoring (for system administrators), and application development, execution, and tuning (for application developers and users).

This chapter provides an overview of the IBM HPC solution. Chapters 2, 3, and 4 describe the HPC environment. Chapters 5, 6, and 7 provide insight and tools for applications. Chapter 8 covers the Graphics Processing Unit (GPU) functionality. The appendixes cover problem determination, references for operation and migration, and application performance.

The following sections are presented in this chapter:

- Why implement an IBM HPC solution on POWER8
- IBM Power System S824L
- High-performance Interconnect
- Storage
- Software stack

1.1 Why implement an IBM HPC solution on IBM POWER8

In early 2014, the release of the IBM POWER8 processor represented to technical computing a step further toward the Exascale era due to the introduction of a series of innovations in business and technology. High-performance systems with incredible performance and efficiency were launched at affordable acquisition and maintenance prices.

With POWER8 systems, such as the new IBM Power System S824L, the total cost of ownership (TCO) is lowered drastically because the server delivers more performance per core (processor) with notable energy efficiency. Fewer cores and machines are required, which ultimately reduces the licensing costs for your HPC cluster.

No industry-leading high-performance system is achieved without technological innovations. Therefore, the IBM POWER8 processor was conceived with new and unique features for technical computing to offer the following advantages:

- Help deliver the highest performance per core
- Achieve energy efficiency with in-chip technologies
- Deliver scalability
- Provide reliability

The IBM POWER® processor family is now sustained by open innovations that were created through a partnership with technology company members of the recently announced OpenPower™ Foundation. The foundation is a joint effort to build an open and innovative ecosystem of software and hardware around the POWER processors. Learn more about OpenPower at the following website:

http://openpowerfoundation.org

An example of OpenPower collaboration is the new IBM Power System S824L that emerged from a partnership among NVidia, Mellanox, and IBM to build a scale-out system. This system features high-speed and low-latency node interconnection with Mellanox InfiniBand technologies. It combines POWER8 with NVidia GPUs to perform highly optimized parallel tasks that are focused on delivering low latency and throughput.

A deeper technical description of the IBM POWER8 system is in 1.2.1, “The IBM POWER8 processor chip” on page 4.

Another advantage of implementing HPC in POWER8 is the rich ecosystem of software that is built around it and ready to take advantage of its strength. The software stack was implemented to satisfy the needs of technical computing users, administrators, and developers. The software stack consists of the following components:

- Cluster management software
- Parallel runtime environment
- Parallel file system
- Libraries
- Compilers
- Parallel development tools

Also, the new support for the Ubuntu server 14.04 LTS Linux distribution that runs in little-endian mode promotes systems migration from Intel to Power Systems.

1.2 IBM Power System S824L

In October 2014, IBM announced a new range of systems that were targeted at handling massive amounts of computational data[1].

IBM Power System S824L[2] (Figure 1-1) is the first IBM Power Systems server that features NVIDIA GPU. This offering delivers a new class of technology that maximizes performance and efficiency for scientific, engineering, Java, big data analytics, and other technical computing workloads. Designed to empower the ecosystem of open source development, these new servers support the Ubuntu server Linux distribution that runs in little-endian and non-virtualized mode.

Figure 1-1 Front view of the IBM Power System S824L server

The IBM POWER8 processor was designed for data-intensive technical computing workloads, big data, and analytics applications. All aspects of the IBM POWER8 processor design are optimized to deal with today's exploding data sizes. The POWER8 cache hierarchy was architected to provide data bandwidth that is suitable for running large data sets and to accommodate their large footprints.

The POWER8 processor comes in two versions. One version is targeted toward large symmetric multiprocessing (SMP) systems at the enterprise level. The other version is specifically designed for scale-out servers that are building blocks for current and future IBM HPC systems. The two latter chips are installed in pairs in a dual-chip module (DCM). DCM connects its chips with SMP links and plugs into a socket in the planar of the system. Functionally, DCM works as a single processor, and the operating system considers DCM as a Non-Uniform Memory Access (NUMA) node. The IBM POWER8 scale-out servers contain one or two sockets that are populated with DCMs.

The IBM Power System S824L server (model 8247-42L) is a two-socket system. The server is available in two-processor configurations:

- Two 10-core 3.42 GHz POWER8 DCMs (20 cores per system)
- Two 12-core 3.02 GHz POWER8 DCMs (24 cores per system)

[1] See *IBM Provides Clients Superior Alternative to x86-Based Commodity Servers*, which is at this website:
http://www.ibm.com/press/us/en/pressrelease/45006.wss

[2] The model name follows a simple convention. The first letter ("S") stands for "scale-out". The first digit indicates a POWER8 processor. The second digit is for the number of sockets. The third digit reflects the server height in standard rack units. The trailing "L" letter indicates "scale-out/Linux". IBM Power scale-out/Linux servers are designed specifically to run the Linux operating system. If you are interested in running the IBM AIX or IBM i operating system on POWER scale-out servers, you need to consider server models that do not have the letter "L" suffix in their names (S814, S822, and S824 at the time that this publication was written).

The IBM Power System S824L server has 16 slots for dynamic device reconfiguration 3 (DDR3) error-correction code (ECC) memory[3] (eight slots for each DCM) and supports memory modules of 16 GB, 32 GB, and 64 GB. The maximum configurable system memory is 1 TB. The form factor for the memory modules is Custom dual inline memory module (DIMM) (CDIMM).

Note: The introduction of new technology in memory modules can increase the maximum system memory that is configurable for a system. However, that maximum is subject to support, compatibility, and other considerations.

The minimum system configuration includes one NVIDIA Tesla K40 GPU (more details in 8.1.1, “NVIDIA Tesla K40 GPU” on page 192). The maximum system configuration is two GPUs.

The S824L planar features 11 PCIe Gen3 slots:

- Four ×16 slots (one or two are used for NVIDIA GPU cards)
- Seven ×8 slots (one is used for the integrated LAN adapter)

The server has a rack-mounted form factor and takes 4U (four units) of rack space as depicted in Figure 1-1 on page 3. The lower-left front part of the server features 12 small form factor bays for internal storage that is based on hard disk drives (HDDs) or solid-state drives (SSDs), which are not supported on all system models/configurations. Two USB 3.0 ports and a DVD bay are visible on the lower-right front of the server.

For more detailed information about the server options, search the IBM Offering Information website for the IBM Power System S824L at the following website:

http://www.ibm.com/common/ssi/index.wss

The official technical documentation for the IBM Power System S824L server is at the following website:

http://www.ibm.com/support/knowledgecenter/8247-42L/p8hdx/8247_42l_landing.htm

To acquire a better technical understanding of the IBM Power System S824L server and its reliability, availability, and serviceability features, see *IBM Power System S824L Technical Overview and Introduction*, REDP-5139.

1.2.1 The IBM POWER8 processor chip

The POWER8 processor is fabricated with IBM 22 nanometer (22 nm) silicon-on-insulator (SOI) technology that uses copper interconnects and 15 layers of metal. The scale-out version of the POWER8 chip is shown in Figure 1-2 on page 5. It contains six cores[4] and is 362 square millimeters (mm) in size. Each core has 512 KB of static RAM (SRAM) second-level cache (L2), and 8 MB of embedded DRAM (eDRAM) third-level cache (L3). The L3 cache is shared among all cores of the chip. The chip also includes a memory controller with four memory interfaces, PCIe Gen3 controllers, SMP interconnect, hardware accelerators, and coherent accelerator processor interface (CAPI). The interconnection system links all components within the chip.

[3] Error-correcting code (ECC) memory is a type of random access memory that can detect and correct spontaneous runtime failures of memory cells or memory access logic.

[4] The number of cores that is activated depends on a server offering.

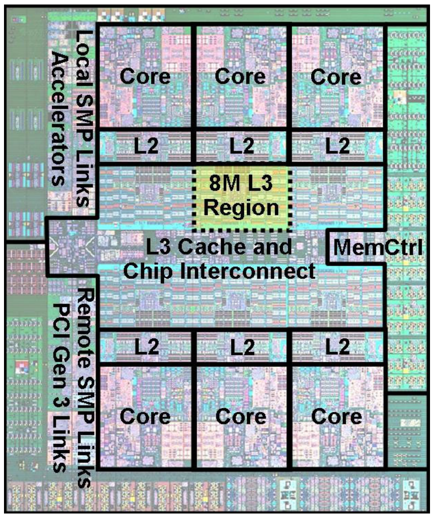


Figure 1-2 The POWER8 scale-out processor chip

The SMP fabric of the IBM POWER8 scale-out chip has two types of links:

- An "X" bus connects a pair of chips to form a DCM.
- An "A" bus provides intersocket communications.

Two POWER8 scale-out chips are installed in pairs in a DCM that plugs into a socket in a system planar of a computing server. In a maximum configuration, 12 POWER8 cores of a DCM share 96 MB of L3 cache and two memory controllers address 512 GB of system memory through eight memory interfaces.

POWER8 core

The POWER8 is a 64-bit processor that is based on the IBM Performance Optimization With Enhanced RISC[5] (Power) Instruction Set Architecture (ISA) version 2.07. The POWER8 computing core is a superscalar out-of-order eight-way simultaneously multithreaded (SMT)[6] microprocessor. POWER8 is the first processor that is based on the IBM POWER Architecture that supports little-endian and big-endian byte ordering in virtualized and non-virtualized (or bare-metal) mode[7].

The primary components of a POWER8 core are shown in Figure 1-3 on page 6:

- Instruction fetch unit (IFU)
- Instruction sequencing unit (ISU)
- Load/store unit (LSU)
- Fixed point unit (FXU)
- Vector and scalar unit (VSU)
- Decimal floating point unit (DFU)

[5] RISC is an acronym for "reduced instruction set computer". The first prototype computer to use RISC architecture was designed by IBM researcher John Cocke and his team in the late 1970s. For the historical perspective, see http://www.ibm.com/ibm/history/ibm100/us/en/icons/risc.

[6] Simultaneous multithreading capabilities of the POWER8 processor core allow eight independent software threads to efficiently share the core resources. This mode is known as SMT8. In a single-threaded mode, almost all of the resources of the highly parallel POWER8 core are used by the single thread.

[7] The byte ordering (big-endian or little-endian) for a storage access is specified by the operating system.

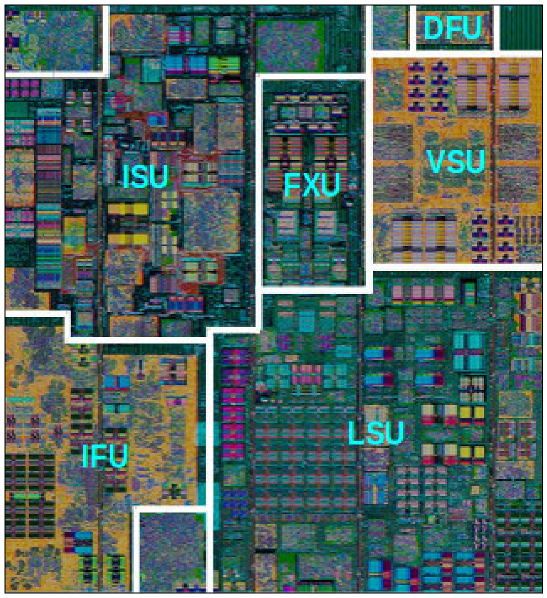


Figure 1-3 The IBM POWER8 processor core

On each cycle, the IFU of a POWER8 core selects a thread and fetches eight quadword-aligned[8] instructions from the 32 KB eight-way first-level instruction cache into a thread instruction buffer. POWER8 instructions are always 4 bytes long and word-aligned. As a result, an eight-instruction block is 32 bytes long.

In SMT modes, the POWER8 thread priority logic selects two threads for each cycle for instruction group formation. Groups are independently formed by reading a maximum of three nonbranches and one branch from the instruction buffer of each thread, for a total of eight instructions. After group formation, the simple instructions are immediately decoded, and the complex instructions are routed to special microcode hardware that breaks them into a series of simple internal operations.

The POWER8 core dispatches instructions on a group basis as soon as all resources are available for the instructions in a group. In SMT modes, two groups are dispatched independently. The ISU dispatches eight instructions per cycle into three separate issue queues:

- A unified queue
- A branch issue queue
- A condition register issue queue

The ISU is also responsible for register renaming and instruction completion.

The POWER8 core saves dispatched instructions in the issued queues and then issues them to the execution units. Instructions can be issued in order or out of order from all of these queues. An instruction in the issue queue is selected for issuing when all source operands for that instruction are available. The issue queues together can issue a total of ten instructions for each cycle in the following manner:

- One branch instruction to the branch execution unit
- One condition register logical instruction to the condition register execution unit
- Two fixed-point instructions to the FXU

[8] Halfword, word, doubleword, and quadword are 2-, 4-, 8-, and 16-byte memory entries. One, two, three, and four least significant bits of a halfword-, word-, doubleword-, and quadword-aligned memory address are zeros.

- Two load/store and two load instructions to the LSU
- Two instructions to the VSU

In addition to load/store operations, the LSU can accept and execute simple fixed-point instructions that come from each of its four execution pipelines.

VSU includes the following subunits:

- Two single precision vector floating point units (FPUs)
- Two double precision vector FPUs
- Two scalar FPUs
- Two fixed-point vector units
- One decimal floating point unit (DFU)
- One cryptographic operations unit

VSU vector subunits operate on 128-bit registers. To manipulate vectors, VSU implements vector multimedia extension (VMX) and vector scalar extension (VSX) instructions[9]. The POWER8 core is able to issue up to two fused multiply-add instructions on vectors in each cycle. Therefore, the POWER8 core is able to deliver 16 single precision or eight double precision floating point operations per cycle during peak workloads.

PCI Express

The IBM POWER8 server system components connect through the Peripheral Component Interconnect Express Gen3 (PCI Express Gen3 or PCIe Gen3) bus.

In server systems, certain PCIe devices connect directly to the PCIe Gen3 buses on the processors, and other devices connect to these buses through PCIe Gen3 switches. PCIe slots are used to plug in devices, such as InfiniBand cards, 10 GbE network interface cards, Fibre Channel (FC) adapters, serial-attached SCSI (SAS) cards for internal disks, and external SAS ports. Certain PCIe slots are Coherent Accelerator Processor Interface (CAPI)-enabled to connect custom acceleration engines (for example, field-programmable gate arrays (FPGAs)) to the coherent fabric of the POWER8 chip.

The IBM POWER8 scale-out chip has 24 PCIe Gen3 full duplex lanes that provide 7.877 Gbps of effective bandwidth in each direction. The total effective theoretical bandwidth for a DCM is calculated in the following manner:

```
2 chips × 24 lanes × 7.877 Gbit/s × 2 directions = 94.524 Gbyte/s
```

Note: The bandwidth that is listed here can appear slightly differently from other materials. In our case, we consider the bandwidth overhead that originated from the PCIe Gen3 encoding scheme. We also avoid rounding where possible.

Energy awareness

The energy that is required to power and cool servers contributes significantly to the overall operational efficiency of the computing infrastructure. To address this challenge, IBM developed the EnergyScale™ technology for IBM Power Systems servers. This technology helps to control the power consumption and performance of POWER8 servers. For more details, see the *IBM EnergyScale for POWER8 Processor-Based Systems* paper at the following website:

http://public.dhe.ibm.com/common/ssi/ecm/po/en/pow03125usen/POW03125USEN.PDF

[9] The IBM VMX and VSX instructions implement and extend the AltiVec specifications that were introduced to the Power ISA at its 2.03 revision.

The IBM POWER8 chip has an embedded IBM PowerPC 405 processor with 512 KB of SRAM. This processor, which is known as an on-chip controller (OCC), runs a real-time control firmware. The purpose of the OCC is to respond timely to workload variations. The OCC adjusts the per-core frequency and voltage based on activity, thermal, voltage, and current sensors. The OCC real-time operating system was released as open source software.

See the open-power/OCC at the following website:

`http://github.com/open-power/occ`

On-chip accelerators

The IBM POWER8 processor has the on-chip accelerators that provide the following functions:

- On-chip encryption
- On-chip compression
- On-chip random number generation

The Linux kernel uses on-chip accelerators through specialized drivers[10].

The on-chip cryptography accelerator, which is also known as the *cryptographic module*, provides high-speed encryption capabilities. The operating system can use this facility to offload certain latency-tolerant operations, for example, file system encryption and securing Internet Protocol communications.

The on-chip compression engine, which is sometimes referred to as the *memory compression module*, was designed as a facility that helps the operating system to compress the least frequently used virtual memory pages.

The on-chip random number generator (RNG) provides the operating system with the source of hardware-based random numbers. This generator was architected to be cryptographically stronger than software-based pseudo-random number generators. In certain instances, a performance advantage exists.

In addition to on-chip accelerators, each core has built-in functions that facilitate the implementation of cryptographic algorithms. An application developer can use these capabilities by employing compiler intrinsics in their source code or by writing in assembly language.

Coherent accelerator processor interface

Many computing workloads can benefit from running on specialized hardware, such as field-programmable gate arrays (FPGAs) or GPUs, rather than on a general-purpose processor. Traditionally, external accelerators communicate with the processor through the I/O bus, which follows the model of I/O device operation. To simplify the interaction between the processor and accelerators, IBM introduced the CAPI with the POWER8 processor. CAPI attaches devices to the SMP fabric of the POWER8 chip. As a result, CAPI enables off-chip accelerators to access the main system memory and participate in the system memory coherence protocol as a peer of other caches in the system. CAPI allows a specialized accelerator to be seen as merely an additional processor in the system.

The conforming accelerator is to be plugged into a standard PCIe Gen3 slot that is marked as CAPI-enabled. The CAPI coherency protocol is tunneled over a standard PCIe Gen3 bus.

[10] Available only in selected offerings.

1.2.2 Memory subsystem

The IBM POWER8 processor was designed with a strong memory subsystem to meet the bandwidth, latency, and capacity demands of data and compute-intensive applications.

Each core of the POWER8 processor has first level (L1) and second level (L2) private caches:

- 32 KB of L1 instruction cache
- 64 KB of L1 data cache
- 512 KB of L2 unified cache

Cache memory of a third level (L3) is shared among all cores of the POWER8 chip. A core has faster access to its local 8 MB L3 cache region.

The scale-out version of the POWER8 processor chip has one memory controller with four memory interfaces. Memory slots are populated with custom DIMM (CDIMM) memory modules. *CDIMM* is a memory card that houses a set of industry standard dynamic random access memory (DRAM) memory chips and a memory buffer chip. The memory controller of a processor accesses system memory through that external memory buffer chip. The memory buffer chip isolates the processor memory controller from the interaction with the actual DRAM memory chips. When this publication was written, the POWER8 server offerings included DDR3 memory options. However, the described technology allows easy transition to DDR4 memory later. Figure 1-4 illustrates the POWER8 memory organization.

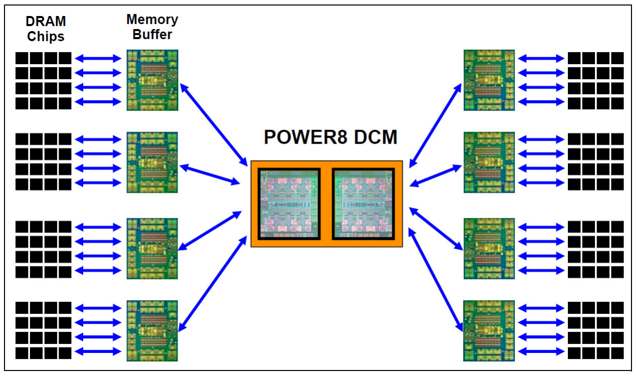


Figure 1-4 POWER8 memory organization

The memory buffer chip that is shown in Figure 1-5 on page 10 consists of the following components:

- POWER8 processor link
- Memory cache
- Memory scheduler
- Memory manager
- DDR3 interfaces

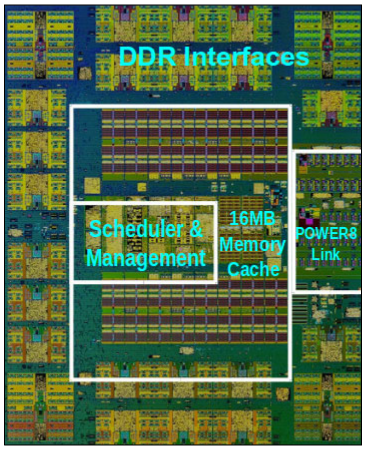


Figure 1-5 Memory buffer chip

The 16 MB memory buffer chip cache constitutes the fourth level (L4) of cache hierarchy. The memory scheduler supports prefetch and write optimization. The memory manager is responsible for reliability, availability, and serviceability (RAS) decisions[11] and energy management.

Hardware transactional memory

Transactional memory (TM) is a computer science concept that was created to facilitate the design of multithreaded algorithms. The TM makes a series of loads and stores appear as a single atomic operation or *transaction* that either succeeds or fails. The TM is primarily intended to be used in situations where software engineers typically employ locks to ensure the atomicity of a set of operations.

Algorithms exist in which the probability of a memory conflict between application threads is negligible. Nevertheless, even in such cases, software engineers need to use locks to ensure that the algorithm is always correct. The use of locks in frequently executed pieces of code can dramatically affect performance.

One algorithmic approach to avoid acquiring locks in this situation is a lock elision technique. A thread needs to mark the beginning and end of a transaction on shared variables with the TM directives or instructions that are provided by a programming language or system software. In an optimistic execution scenario, no conflicts exist, and transactions always succeed. In an expected highly improbable conflict situation, the TM reports that a transaction failed. In this case, an algorithm developer can either try a transaction again with a TM, or fall back to a lock-based if-then-else branch in the code.

The TM was first implemented as a pure software-based mechanism. Hardware support for the TM became a commercial reality only recently. To the best of our knowledge, the first commercially available product with the hardware transactional memory (HTM) capabilities was the IBM Blue Gene/Q® (BG/Q) supercomputer in 2011. The BG/Q compute chip was based on PowerPC A2 processor cores and implemented transactional execution primarily in the L2 cache, which served as the point of coherence.

In 2012, IBM started to ship IBM zEnterprise® EC12 (zEC12) IBM System z® mainframe solutions that targeted business audiences. With the zEC12 microprocessor, IBM offered multiprocessor transactional support.

[11] The RAS decisions deal with reliability, availability, and serviceability features and functions.

When this publication was written, the latest IBM product with support for HTM was the POWER8 processor. The IBM POWER architecture support for HTM includes instructions and registers that control the transactional state of a thread. The Power ISA version 2.07 specification also describes the behavior of transactions regarding the execution and memory models of the processor. It also includes more complex behaviors, such as suspend mode, which allows non-transactional operations inside a transaction.

1.3 High-performance Interconnect

The IBM Power System S824L includes one or two high bandwidth and low latency Mellanox Connect-IB Dual-Port FDR InfiniBand Host Channel Adapters (HCAs). These HCAs connect in the bus through PCIe 3.0 x16, which is able to deliver 56 Gbps of data at each FDR port. Among many features, the following features can be highlighted:

- Delivers more than 100 Gb per second overall throughput
- InfiniBand Architecture specification version 1.2.1 compliant
- Implements Virtual Protocol Interconnect (VPI)
- Higher message rate, which supports more than 130 M messages per second
- Takes over transport operations to offload the CPU
- Supports noncontinuous memory transfers
- Supported by IBM Parallel Edition (PE)

For more information about the InfiniBand card, see *Connect-IB Single and Dual QSFP+ Port PCI Express Gen3 x16 Adapter Card User Manual* at the following website:

http://www.mellanox.com/page/products_dyn?product_family=142&mtag=connect_ib

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

1.4 Storage

Today's never-ending data growth is challenging traditional storage and data management solutions. These outdated systems are expensive to administer and scale, in addition to limiting data access, performance, and reliability that today's data-intensive computing environments require, particularly when data is accessed on a global scale. Application performance is affected by data access bottlenecks that delay schedules and waste expensive resources.

IBM Spectrum™ Scale is a proven, scalable, high-performance data and file management solution (formerly IBM GPFS™) that is being used extensively across multiple industries worldwide. Spectrum Scale provides simplified data management and integrated information lifecycle tools that can manage petabytes of data and billions of files to help control the growing cost of managing ever growing amounts of data. Spectrum Scale version 4.1 software-defined storage for cloud, big data, and analytics introduces enhanced security, flash-accelerated performance, and improved usability for the world's most demanding data-intensive enterprises.

Spectrum Scale features enhanced security, increased performance, and improved usability. Spectrum Scale provides world-class storage management that is delivered in software and used on any hardware, with extreme scalability flash-accelerated performance, and automatic policy-based storage tiering from flash through disk to tape.

Spectrum Scale can help reduce storage costs by 90% and simultaneously improve security and management efficiency in cloud, big data, and analytics environments.

1.5 Software stack

The HPC solution on POWER8 that is described throughout this publication contains the software components that are shown in Table 1-1.

Table 1-1 HPC solution on POWER8 software stack

Category	Software component	Landing page
Application development		
Compiler	IBM Xpertise Library (XL) C/C++	http://www.ibm.com/support/knowledgecenter/SSXVZZ/welcome
	IBM XL Fortran	http://www.ibm.com/support/knowledgecenter/SSAT4T/welcome
	GNU Compiler Collection (GCC)	http://gcc.gnu.org
	NVIDIA NVCC	http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc
Integrated development environment (IDE)	IBM Parallel Environment Developer Edition (PE DE)	http://www.ibm.com/support/knowledgecenter/SSFK5S/pedev_welcome.html
	NVIDIA Nsight Eclipse Edition	http://developer.nvidia.com/nsight-eclipse-edition
Performance analysis and tuning tools	IBM HPC Toolkit (part of IBM PE DE)	http://www.ibm.com/support/knowledgecenter/SSFK5S/pedev21/com.ibm.cluster.pedev.v2r1.pedev100.doc/bl7ug_about.htm
	Oprofile	http://oprofile.sourceforge.net
	Perf	https://perf.wiki.kernel.org
	gprof	http://www.gnu.org/software/binutils
	NVIDIA nvprof	http://docs.nvidia.com/cuda/profiler-users-guide/#nvprof-overview

Category	Software component	Landing page
Debugger	PTP Parallel Debugger (part of IBM PE DE)	http://help.eclipse.org/luna/topic/org.eclipse.ptp.doc.user/html/06parDebugging.html?cp=42_12
	PDB (part of IBM PE)	http://www.gnu.org/software/gdb
	CUDA GDB	http://docs.nvidia.com/cuda/cuda-gdb
Engineering and Scientific Math library	IBM Engineering and Scientific Subroutine Library (ESSL)	http://www.ibm.com/support/knowledgecenter/SSFHY8/essl_welcome.html
	IBM Parallel ESSL (PESSL)	http://www.ibm.com/support/knowledgecenter/SSNR5K_4.2.0/pessl.v4r2_welcome.html
	IBM Mathematical Acceleration Subsystem (MASS)	http://www.ibm.com/software/products/en/mass
NVIDIA CUDA	CUDA Toolkit for POWER8	https://developer.nvidia.com/cuda-downloads-power8
GPU Accelerated libraries	CUDA math API, CUBLAS, CUFFT, CURAND, CUSPARSE, NPP, and Thrust (all part of CUDA Toolkit)	https://developer.nvidia.com/gpu-accelerated-libraries
Java	IBM SDK, Java Technology Edition	http://www.ibm.com/support/knowledgecenter/SSYKE2
Base run time		
Firmware	OpenPower Abstraction Layer (OPAL)	https://github.com/open-power
Operating system	Ubuntu Server (ppc64el)	http://cdimage.ubuntu.com/releases/
InfiniBand Host Channel Adapter (HCA)	Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux	http://www.mellanox.com/page/products_dyn?product_family=142&mtag=connect_ib
Parallel Runtime	IBM Parallel Environment (PE) Runtime Edition (PE RTE)	http://www.ibm.com/support/knowledgecenter/SSFK3V/pe_welcome.html
Parallel File System	IBM Spectrum Scale™, based on IBM General Parallel File System (GPFS)	http://www.ibm.com/support/knowledgecenter/SSFKCN/gpfs_welcome.html
Systems deployment and management		
Firmware management	Intelligent Platform Management Interface (IPMI) Tool	http://sourceforge.net/projects/ipmitool
Deployment and management	Extreme Cluster/Cloud Administration Toolkit (xCAT)	http://xcat.sourceforge.net

Category	Software component	Landing page
GPU	NVIDIA System Management Interface (SMI)	`http://developer.nvidia.com/nvidia-system-management-interface`
Resource management and scheduling		
Workload and resource management	IBM Platform Load Sharing Facility (LSF®)	`http://www.ibm.com/systems/platformcomputing/products/lsf/index.html`

2

Planning for your high-performance computing environment

When you plan for a high-performance computing (HPC) solution, you need to think about what your hardware and software environment will be and the strategy for using this environment. You also need to ensure that all of the necessary system requirements are met.

This chapter provides an HPC architecture reference with the suggested hardware and software stacks to implement a high-performance solution that uses IBM POWER8 scale-out Linux servers.

The following topics are described in this chapter:

- Reference architecture
- Power, packaging, and cooling

2.1 Reference architecture

This section considers two main scenarios for user interaction with the HPC system and outlines the hardware and software pieces that are needed to achieve the goal.

The simple case to consider is a single-user environment with and without support for distributed jobs. This case implies that it is the users' responsibility to ensure that code development and post-processing activities do not affect the running jobs.

The other scenario implies that the HPC system has a set of computing resources that are dedicated to computing tasks only. Therefore, users do not access nodes directly, so their activity cannot affect the running jobs. In this scenario, users access a front-end system and use it for any pre-running and post-processing activities, such as file editing, compilation, or compressing.

The users also use the front-end system to submit the job to the computing resources. The computing resources are typically configured so that users have exclusive access to the requested resources. The submitted jobs are executed in a batch mode. The job scheduler manages tasks from multiple users and sends the jobs for execution in adherence to scheduling policies.

Important: Consider the use of a parallel file system, a high-performance network, job scheduling facilities, and system management facilities.

The HPC system needs to contain the following key component subsystems:

- Front-end node: Provides user access and the ability to compile the source code and submit tasks
- Service node: Runs job scheduler server and system monitoring tools
- Compute node: Uses dedicated resources to run compute-intensive jobs without external intervention
- Storage system: Provides data persistence

The HPC system contains the following high-level facilities:

User interface	For application developers and application users, this component serves as a gateway to the system. For users, this facility is available to edit source files and compile applications, launch computing jobs, perform post-processing of computed data, and conduct other interacting activities.
System administration	This component provides the system administrator with the tools and services for system provisioning, deployment, management, monitoring, and report generating.
Job scheduling	The job scheduler is responsible for managing the jobs queue, scheduling policies, task planning and execution, and advanced reservations for computing resources, application program license management, interaction with parallel runtime and operating environment, and computing quota governing. Basic schedulers work on a first-in first-out principle. More sophisticated schedulers use complicated scheduling algorithms that include many parameters and typically try to maximize the system usage level.

Parallel environment	A parallel runtime and operating environment distributes user application tasks among the computing nodes.
Computing	This component provides the actual computing resources.
Storage	This component manages many storage systems to best fit user application needs. It can provide local storage, local cache, parallel storage, and elastic storage.

These components are united by the following communication environment components:

- Computing network.
- Data network.
- Management network.
- Internally, storage can contain a Fibre Channel (FC) network.

Multiple options exist to map these components into the actual hardware.

2.1.1 Hardware

The computing nodes in the HPC solution that are described in this book consist of IBM Power System S824L that implement POWER8 processor technology. POWER8 processor technology is introduced in 1.3, “High-performance Interconnect” on page 11. For a more comprehensive description of this system, see *IBM Power System S824L Technical Overview and Introduction*, REDP-5139.

2.1.2 Firmware

The OpenPower Abstraction Layer (OPAL) firmware is available on IBM Power Systems scale-out Linux servers with POWER8 processors[1]. This feature supports running Linux in non-virtualized (or bare-metal) mode[2] and virtualized mode (guest or virtual machine) with kernel-based virtual machine (KVM) acceleration. OPAL firmware provides an interface to the underlying hardware.

One of the most noticeable changes that was introduced with OPAL is out-of-band management through the Intelligent Platform Management Interface (IPMI) rather than the Hardware Management Console (HMC). In fact, OPAL cannot be enabled with a HMC connection in place. The usage of IPMI is favorable to environments with many systems (clusters, for example) due to its applicability in automated, network-based system management.

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

https://github.com/open-power

2.1.3 Software

The major software building blocks of the HPC solution on POWER8 that are presented in this book are introduced in the next sections.

[1] Also available on certain IBM POWER7® and IBM POWER7+™ systems under an agreement (not generally available).

[2] The non-virtualized/bare-metal terms are not precisely correct because the kernel performs calls to the OPAL firmware. However, their usage is popular because Linux on IBM Power Systems traditionally ran in IBM PowerVM®.

Deployment and management

Extreme Cluster/Cloud Administration Toolkit (xCAT) performs the major role of the overall deployment and management of the software stack of the HPC cluster. Among many services, the toolkit controls the node discovery process and management, operating system provision, and software stack installation and configuration.

xCAT is an open source software. For more information, see the project's website:

http://xcat.sourceforge.net

IBM Parallel Environment

IBM Parallel Environment (PE) is a development and execution environment for parallel applications. IBM PE offers optimized routines and a runtime environment for more efficient computing resource usage and integrated application lifecycle management.

IBM PE consists of the following offerings:

- IBM Parallel Environment (PE) Runtime Edition (IBM PE Runtime Edition) helps you develop, debug, and run parallel applications.
- IBM Parallel Environment Developer Edition (PE DE) offers an Eclipse-based application development environment for technical computing and HPC systems. The Developer Edition also includes integrated scalable performance profiling tools.

This section covers only IBM PE Runtime Edition. The IBM Parallel Environment Developer Edition is described in 5.3, "IBM Parallel Environment Developer Edition" on page 155.

The IBM PE Runtime Edition product is a set of software components that are intended for development, debugging, and running parallel programs on an HPC cluster:

- Message passing and collective communication API subroutine libraries
- IBM OpenSHMEM Runtime Library
- Parallel operating environment (POE)
- Debugger for parallel programs, which is called PDB
- Scalable Communication Infrastructure (SCI)
- Parallel Active Messaging Interface (PAMI)
- Sample programs
- Documentation

Note: As of version 2.1, IBM PE Runtime Edition runs on IBM Power Systems servers with IBM POWER8 technology in little-endian (LE) mode running the Ubuntu Server 14.04.01 for IBM Power Systems only.

The PE Runtime Edition provides all resources to develop and execute C, C++, and Fortran applications. You use either the Single Program Multiple Data (SPMD) or Multiple Program Multiple Data (MPMD) parallel programming models with the Message Passing Interface (MPI) and Parallel Active Messaging Interface (PAMI) APIs. The following considerations for MPI implementation within IBM PE Runtime Edition:

- Implements MPI version 3.0 on the MPICH[3] specification
- Uses the PAMI protocol as a common transport layer
- Supports 64-bit applications only
- The I/O component of MPI (or MPI-IO) uses the implementation of MPICH 3.1.2 from the Argonne National Laboratory's ROMIO[4]. It is also optimized for production-level use with the IBM Spectrum Scale.

[3] The MPICH standard MPI specification website address is http://www.mpich.org.
[4] The ROMIO project website address is http://press3.mcs.anl.gov/romio.

The high-speed and low overhead message passing communication is achieved by the support to InfiniBand (IB) host channel adapters (HCAs) on specific hardware, and an implementation of a mechanism where messages bypass the Linux kernel stack, therefore providing user-space direct access to the IB HCA interface. However, if all of the tasks of a parallel job are on a single system, they can be run by using shared memory for performance improvement, so no communication is through the IBM HCA in this context.

The OpenSHMEM[5] API for the Global Shared memory programming model is also supported through the IBM OpenSHMEM Runtime Library, but only for C applications.

To use the provided APIs, it is required that you have a working compiler. IBM PE Runtime Edition for Linux on Power supports parallel development by using the IBM Xpertise Library (XL) family and GNU Compile Collection (GCC) compilers.

The entry point for submitting and managing parallel jobs within PE Runtime Edition is the Parallel Operating Environment (POE). It provides the **poe** command-line tool that is used to start, stop, and cancel parallel jobs. With a rich set of environment variables, it is possible to finely control the parallel application's run time.

A debugger for parallel programs (PDB) accompanies the IBM PE Runtime Edition. PDB is described in 7.4, "Debugging tools" on page 188.

The general description of the IBM Parallel Environment software offering is available at the following website:

http://www.ibm.com/systems/power/software/parallel/

The detailed documentation about the installation and usage of the IBM Parallel Environment Runtime Edition is available at the IBM Knowledge Center at this website:

http://www.ibm.com/support/knowledgecenter/SSFK3V/pe_welcome.html

Application compilers

The IBM XL compiler family and GNU Compiler Collection (GCC) are choice compilers to work with the parallel compiler scripts that are provided within IBM Parallel Environment (PE) to build HPC applications that target the IBM POWER8 scale-out Linux compute nodes. They support POWER8 processor and are fully capable of deliver highly optimized code by using each of its features. For information about using the compilers for application development, see 5.1, "Compilers" on page 144.

The IBM XL C/C++ for Linux on POWER8 software offerings are available at following website:

http://www.ibm.com/software/products/en/xlcpp-linux

The IBM XL Fortran for Linux on POWER8 software offerings are available at following website:

http://www.ibm.com/software/products/en/xlfortran-linux

Performance libraries

HPC libraries are an essential building block of any technical computing application today. Among the many available options that support Linux on POWER8, we recommend that you use the IBM Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL (PESSL).

[5] The OpenSHMEM project website address is http://www.openshmem.org.

The IBM ESSL is a collection of high-performance mathematical subroutines for C, C++, and Fortran that provide a wide range of functions for any common scientific and engineering applications. Parallel ESSL is the scalable counterpart for use with the Message Passing Interface (MPI) application on HPC clusters of IBM Power scale-out/Linux servers.

The IBM ESSL and PESSL software offerings are available on the following website:

http://www.ibm.com/systems/power/software/essl/

Operating system and device drivers

The IBM Power System scale-out Linux compute nodes are installed with Ubuntu Linux Server 14.04.1 LTS with support for little-endian 64-bit PowerPC and running in nonvirtualized mode. For more information about Ubuntu for IBM POWER8, see the following website:

http://www.ubuntu.com/download/server/power8

No device drivers other than the device drivers in the Ubuntu distribution are required for the installation and use of the IBM Power System S824L compute node's devices, such as the network cards and FC adapters. The only exceptions are the device drivers for the InfiniBand HCAs that are obtained from Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux (MLNX_OFED). For more information, see 1.3, "High-performance Interconnect" on page 11.

Parallel file system

Typically, HPC systems use a commonly accessed, shared, or parallel file system. Several advantages exist for using a parallel file system. Often, the parallel file system operates over high-speed and low latency networks, such as InfiniBand Quad Data Rate (QDR) interconnect networks. (See 1.3, "High-performance Interconnect" on page 11 and 2.1.4, "Network" on page 21.)

IBM Spectrum Scale is a distributed, high-performance, massively scalable enterprise file system solution that addresses the most challenging demands in high-performance computing. Therefore, it is adopted in the solution that is presented in this book as the technology for the parallel file system.

Spectrum Scale provides online storage management, scalable access, and integrated information lifecycle management tools to manage petabytes of data and billions of files. Spectrum Scale delivers concurrent and fast access to a single file system or a set of file systems from multiple nodes. Virtualizing the file storage space and allowing multiple systems and applications to share common pools of storage provide you the flexibility to transparently administer the infrastructure without disrupting applications. With Spectrum Scale, administrators can add more storage capacity or change the file system attributes without affecting the application while a file system is mounted, simplifying administration for any type of workload, even in large environments.

The IBM Spectrum Scale offering is available in the following website:

http://www.ibm.com/software/products/en/software

The parallel file system role is also intrinsic with system storage design, which is described in 2.1.5, "Storage" on page 21.

Resource management and scheduling

IBM Platform Load Sharing Facility (LSF) performs the resources management, workload management, and scheduling of parallel jobs in the compute nodes. IBM Platform LSF integrates well with PE Runtime Edition, which provides interfaces with POE, to submit and manage parallel jobs.

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

http://www.ibm.com/systems/platformcomputing/products/lsf/

2.1.4 Network

The network design of any supercomputer is a major concern due to its possible high impact on the overall system performance. Two main aspects must be correctly addressed:

- Cluster topology
- Interconnections

When the HPC cluster scales from a few compute nodes to hundreds (or thousands) of compute nodes, the physical organization of systems becomes critical to the overall communication of parallel tasks inside the cluster system. Several common network topologies are available to technical computing, for example:

- Fat tree (full bandwidth and blocking)
- Island
- Torus (3D and 4D)
- Dragonfly

The fat tree topology is one of most popular topologies for HPC cluster systems. It is a layered multi-root tree hierarchy of network switches. Think of the compute nodes as the “leaves”.

The choice of network interconnect devices is also critical to the supercomputing design. The goal is the highest bandwidth and lowest latency links. Common options are listed:

- Ethernet (1 Gigabit (1 GbE) and 10 Gigabit (10 GbE))
- InfiniBand (QDR and Fourteen Data Rate (FDR))

2.1.5 Storage

In a typical HPC solution, certain types of data need to be shared and accessed simultaneously across multiple nodes with different formats, types, and sizes. Usually, the HPC solution needs to scale its storage systems to be able to store the exponential amount of growing data. The ability to keep the data stored, secured, shared, and accessible can be challenging and critical.

Two options are available for storage solutions in the HPC area: a Network File System (NFS) solution or a parallel file system. For years, NFS was the dominant protocol for sharing data between compute nodes. NFS was no-charge and relatively straightforward to configure and set up. NFS consists of a single NFS server, or two NFS servers in a redundant configuration, which are connected to storage arrays, exporting data to compute nodes as NFS clients. In an NFS solution, many compute nodes access data on a single server with metadata and data that are stored in a central location.

However, an NFS cannot handle situations when throughput requirements exceed 1 GBps and storage capacity needs to scale beyond a couple of 100 terabytes in a single name space. Traditional NFS protocol today does not allow for scalability outside a single server for a specific file system, and it does not separate file metadata from data, both of which limit throughput and scalability.

File system performance is often a major component of overall HPC system performance. File system performance depends on the nature of the applications that generate the load. The I/O can become the bottleneck for those applications that perform many data reads and writes in a large cluster environment with hundreds of nodes where many jobs compete for limited I/O bandwidth.

Due to the limitations on NFS, more HPC systems use parallel file systems for HPC storage needs. Parallel file systems separate data from metadata and support multiple storage servers that work together in parallel to create a single name space to increase storage throughput and capacity. Compute nodes, by accessing a parallel file system architecture, can read and write data from multiple storage servers and devices in parallel, greatly improving performance over a traditional NFS solution. Separating the metadata function from the data path to dedicated servers and using faster spindles to match the metadata I/O workload increase file system performance and scalability.

IBM Spectrum Scale is a high-performance and proven product that is used to store the data for thousands of mission-critical commercial installations worldwide. IBM launched the IBM Elastic Storage™ Server, which is an integrated software-defined storage appliance that combines the IBM POWER8 server with IBM Storage and Spectrum Scale. By pairing IBM Flash System storage systems with Spectrum Scale software, organizations can derive rapid value from their ever-growing volumes of data. These two complementary solutions deliver extreme performance for the most compute-intensive applications. They help to remove both storage bottlenecks and data access bottlenecks.

2.2 Power, packaging, and cooling

This section is a brief overview of the IBM Power System S824L power, packaging, and cooling features. For more information, see the following IBM Redpaper™ publication: *IBM Power System S824L Technical Overview and Introduction*, REDP-5139.

The IBM Power System S824L chassis dimension is 17.5 inches x 6.9 inches x 29.7 inches (width x height x depth). It is available in a rack-mounted form factor and takes 4U of rack space. Three rack options (all built in a 19-inch EIA 310D standard) are available: IBM 7014 model T00 (Feature Code (FC) 0551), IBM 7014 model T42 (FC0553), and IBM Slim Rack model 7965-94Y. The rack physical specifications are shown in Table 2-1.

Table 2-1 Physical specifications of racks 7014-T00 (FC0551), 7014-T42 (FC0553), and Slim Rack

Rack model	EIA units	Height	Width	Depth	Weight (empty)
7014-T00 (FC0551)	36U	71.0-inch height with standard AC power 75.8-inch height with -48 volt DC power	25.4-inch width with side panels 24.5-inch width without side panels	43.3-inch depth with front and rear doors (FC6097 or FC6068) 45.2-inch depth with front and rear doors (FC6088) 41.0-inch depth with rear door only	244 kg (535 lb)

Rack model	EIA units	Height	Width	Depth	Weight (empty)
7014-T42 (FC0553)	42U	79.3-inch height with standard AC power	25.4-inch width with side panels 24.5-inch width without side panels	43.3-inch depth with front and rear doors (FC6083 or FC6069) 45.2-inch depth with front and rear doors (FC6089) 41.0-inch depth with rear door only	261 kg (575 lb)
Slim Rack 7965-94Y	42U	79.3-inch height with standard AC power	23.6-inch width with side panels	43.3-inch depth with front and rear doors	187 kg (412 lb)

Four hot-plug, redundant 1400 watt 200 - 240 V AC power supplies (FCEL1B) are supported on the IBM Power System S824L server. Its energy management is an IBM EnergyScale technology feature that provides functions to help the user understand and dynamically optimize processor performance versus processor energy consumption and system workload to control the IBM Power Systems power and cooling usage.

Table 2-2 shows the power requirements of the IBM 7014 model T00 (FC0551), IBM 7014 model T42 (FC0553), and IBM Slim Rack model 7965-94Y racks.

Table 2-2 Power requirements of 7014-T00 (FC0551), 7014-T42 (FC0553), and Slim Rack racks

Rack model	Operating voltage	Power distribution unit (PDU)
7014-T00 (FC0551)	200 - 240 V AC 50/60 Hz Optional DC system available on model T00 only: -48 V DC	Up to 4 PDUs. Provides up to 4.8 kilovolt-ampere (kVA) of power source loading per PDU. Twelve AC power outlets on each PDU.
7014-T42 (FC0553)	200 - 240 V AC 50/60 Hz	Up to 4 PDUs. Up to 4.8 kVA of power source loading per PDU. Twelve AC power outlets on each PDU.
Slim Rack 7965-94Y	200 - 240 V AC 50/60 Hz	Up to 8 PDUs. Up to 4.8 kVA of power source loading per PDU.

3

Software deployment and configuration

This chapter describes how to install and configure Extreme Cluster and Cloud Administration Toolkit (xCAT) and the software stack. This chapter includes software and firmware updates, and a few operating system configuration options.

The following sections provide an overview of the environment, and the tools that are used in this chapter:

- Hardware, firmware, and the software stack
- OPAL firmware and the ASM interface
- Intelligent Platform Management Interface (IPMI)

The next sections describe the concepts and setup of the xCAT cluster, and the installation of the software stack:

- xCAT overview
- xCAT management node
- xCAT node discovery
- xCAT compute nodes: Operating system
- xCAT compute nodes: Software stack

The last section explains updates to the software stack and tuning the system firmware:

- Software and firmware updates
- System tuning

3.1 Hardware, firmware, and the software stack

The following environment is used and described throughout this chapter:

- IBM Power System S824L (or other IBM Power scale-out/Linux server) with the following adapters:
  - InfiniBand adapter: Mellanox Connect-IB
  - Fibre Channel (FC) adapter: Emulex LPe12002 host bus adapter (HBA)
- IBM Open Power™ Abstraction Layer (OPAL) firmware (with Intelligent Platform Management Interface (IPMI) functionality)
- Ubuntu Server 14.04.1 Long Term Support (LTS) in non-virtualized mode, and the following adapter software/drivers:
  - Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux (MLNX_OFED)
  - Fibre Channel adapter's device driver that is shipped with the Linux distribution
- Extreme Cluster/Cloud Administration Toolkit (xCAT)
- Xpertise Library (XL) C/C++ for Linux (IBM Compilers), including the following libraries:
  - Mathematical Acceleration Subsystem (MASS)
  - Basic Linear Algebra Subprograms (BLAS)
- IBM SDK, Java Technology Edition (IBM Java)
- Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL (PESSL)
- Parallel Environment Runtime, including the Message Passing Interface (MPI) library
- Platform Load Sharing Facility (LSF)
- Spectrum Scale (formerly GPFS)

Note: The supported operating system version for certain system models and software products is *Ubuntu Server 14.04.2 LTS*, which was not generally available at the time that this publication was written. Similar cases can happen regarding the supported versions of certain software products.

The differences to the instructions that are provided in this book are minimal and do not affect the applicability of this document (for example, version number differences in the commands).

For installation instructions and navigation steps for the IBM Power System S824L, see the IBM Knowledge Center at the following address:

http://www.ibm.com/support/knowledgecenter

Select **Power Systems** → **POWER8** → **8247-42L (IBM Power System S824L)** → **Installing and configuring the system**.

For instructions about an InfiniBand Switch and Mellanox Unified Fabrics Manager (UFM), see the product documentation and support at the following website:

http://www.mellanox.com/page/products_dyn?product_family=142&mtag=connect_ib

To run a Linux distribution in a non-virtualized mode, the firmware type must be set to OPAL. The OPAL firmware allows certain management, monitoring, and configuration operations to be performed on the system's Flexible Service Processor (FSP) through IPMI. The IPMI functionality is used in several operations that are performed by xCAT.

3.2 OPAL firmware and the ASM interface

This section describes the steps to set the firmware type to OPAL (if it is set to PowerVM, which is available on certain systems), and to reset the FSP (suggested in the next sections). To perform both operations, you first need to power off the system.

To perform the steps that are described in this section, access the system's Advanced System Management interface (ASMI) as an administrator:

1. Point your web browser to the system's Flexible Service Processor (FSP) address:

   `https://`*`fsp-address`*

2. Log in as the administrator (`User ID: admin`).

3.2.1 Power off

We suggest that you power off from the operating system if an operating system is installed, if possible. On non-virtualized (or bare-metal) mode, powering off the OS powers off the system. Otherwise, proceed in the ASMI:

1. Expand **Power/Restart Control** and click **Immediate Power Off**.
2. Click **Continue**.
3. Click **Power On/Off System** and verify that the `Current system power state` is `Off`. (You need to repeat this step until the system is eventually powered off because as the page does not refresh automatically.)

3.2.2 Set firmware type

To change the firmware type to OPAL, the system must be powered off and cannot be managed by a Hardware Management Console (HMC). To change the firmware, proceed in the ASMI:

1. Power off the system. (See 3.2.1, "Power off" on page 27.)
2. Remove the HMC connections:
   a. Expand **System Configuration** and click **Hardware Management Consoles**.
   b. Select any HMC connections.
   c. Click **Disconnect**.
   d. Click **Reset to non-HMC system**.
3. Change the firmware type to OPAL:
   a. Expand **System Configuration** and click **Firmware Configuration**.
   b. From the Firmware Type list, select **OPAL**.
   c. Click **Continue**.
   d. Expand **Power/Restart Control** and click **Power On/Off System**.
   e. From the Server firmware start policy list, select **Running (Auto-Start Always)**.
   f. Click **Save settings**.

3.2.3 Reset the FSP

To reset the FSP, the system must be powered off. Proceed in the ASMI:

1. Power off the system. (See 3.2.1, “Power off” on page 27.)
2. Reset the FSP:
   a. Expand **System Service Aids** and click **Reset Service Processor**.
   b. Click **Continue**.

3.3 Intelligent Platform Management Interface (IPMI)

The following steps consist of setting an IPMI password for the system and instructions to manage the system by using IPMI.

Access the system’s Advanced System Management Interface (ASMI) as the administrator:

1. Point your web browser to the system’s FSP address:

   `https://`*`fsp-address`*

2. Log in as the administrator (`User ID: admin`).

Set the IPMI password

Use the following steps to set the IPMI password:

1. Expand **Login Profile** and click **Change Password**.
2. From the User ID to change list, select **IPMI**.
3. In the Current password for user ID admin field, enter the administrator password.
4. In the New password for user and New password again fields, enter the IPMI password.
5. Click **Continue**.

Manage the system by using IPMI

The IPMItool is a command-line interface program to manage IPMI devices that use the network. For example, the interface can be used to power on and off a system, and access its console[1].

You can install the IPMItool from the Linux distribution packages in your workstation or another server (preferably on the same network as the installed server). For example, in Ubuntu, use this command:

```
$ sudo apt-get install ipmitool
```

Or, you can build it from the source, which is available at the following site:

http://sourceforge.net/projects/ipmitool

Use the IPMItool version 1.8.14 or later, which includes fixes and improvements for IBM Power Systems. You can check the version with the command:

```
$ ipmitool -V
```

[1] Also known as Serial over LAN (SOL) session

The following list contains common **ipmitool** commands:

- Power on:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password power on
  ```

- Power off:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password power off
  ```

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

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password power cycle
  ```

- Power status:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password power status
  ```

- Open console session:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password sol activate
  ```

- Close console session:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password sol deactivate
  ```

  You can also close the current console session with the following keystrokes:

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

    `Enter, ~ (tilde`[2]`), . (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 FSP:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password bmc reset cold
  ```

- Values of sensors:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password sdr list full
  ```

- Specific sensor value:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password sdr entity XY[.Z]
  ```
  [3]

- Display FSP Ethernet Port 1 (HMC1; X=1) or 2 (HMC2; X=2) network configuration:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password lan print X
  ```

- Set FSP Ethernet Port 1 (HMC1; X=1) or 2 (HMC2; X=2) for Dynamic Host Configuration Protocol (DHCP) IP address:

  ```
  $ ipmitool -I lanplus -H fsp-address -P ipmi-password lan set X ipsrc dhcp
  ```

[2] On keyboards with dead keys (certain non-English languages), the tilde mark requires two keystrokes: tilde and space.

[3] Sensor entity: *XY* (entity ID) or *XY.Z* (entity ID and instance ID); listed with the **sdr elist full** IPMI command. (See the fourth field of its output.)

- Set FSP Ethernet Port 1 (HMC1; X=1) or 2 (HMC2; X=2) for Static IP address:

```
$ ipmitool -I lanplus -H fsp-address -P ipmi-password lan set X ipsrc static
$ ipmitool -I lanplus -H fsp-address -P ipmi-password lan set X ipaddr a.b.c.d
$ ipmitool -I lanplus -H fsp-address -P ipmi-password lan set X netmask e.f.g.h
$ ipmitool -I lanplus -H fsp-address -P ipmi-password lan set X defgw i.j.k.l
```

Note: The IPMItool is not available on certain platforms. For an alternative IPMI utility, which is not covered in this publication, see the IPMI Management Utilities project at the following website:

`http://sourceforge.net/projects/ipmiutil`

3.4 xCAT overview

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

3.4.1 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 or compute). The type of network depends on its traffic and interconnected nodes (for example, operating system-level management and provisioning, FSP-level/hardware management, application intercommunication, and external or 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, execution, and monitoring.
- Compute nodes: Perform operations that are specified by the management node, for example, operating system provisioning, executing commands, and starting applications.
- Other applications (for example, job schedulers) can control the execution of jobs and applications in compute nodes after the correct operating system provisioning and configuration steps. The compute nodes are referred to simply as *nodes*.
- Optional: Service nodes[4]: Perform operations that are delegated by the management node on groups of nodes, acting as intermediary management nodes on large clusters. An xCAT cluster with service nodes is known as a *hierarchical cluster*.

  Note: The availability of a service node for a particular operating system or version is subject to support by the xCAT release.

[4] Not to be confused with service nodes in the Reference Architecture's context.

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 operating system), for example, node discovery, operating system provisioning, and management
  - Interconnects the management node, service nodes (if any), and compute nodes (in-band network interface controller (NIC))
  - Possibly segregated into separate virtual LANs (VLANs) for each service node[5], if applicable
  - 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 system's FSP/baseboard management controller (BMC[6])), for example, power control, console sessions, and platform-specific functions
  - Interconnects the management node, service nodes (if any), and compute nodes (out-of-band NIC)
  - Possibly combined with the management network (same physical network).
  - Usually an Ethernet network; not necessarily 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
- Optional: 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 accessing the management node, and services that are provided by compute nodes (if any)
  - Interconnects the site gateway, management node, and compute nodes
  - Possibly provides the cluster with access to the Internet
  - Possibly combined with the management node (same physical network)
  - Usually an Ethernet network

For more information about the xCAT architecture, see the xCAT project wiki page, xCAT Overview, Architecture, and Planning, at the following website:

http://sourceforge.net/p/xcat/wiki/XCAT_Overview,_Architecture,_and_Planning

3.4.2 xCAT database: Tables and objects

The xCAT database contains all information that relates to the cluster, either that is defined by an administrator or automatically obtained from the cluster, for example, nodes, networks, services, and settings.

[5] For example, to limit broadcast domains for groups of compute nodes.

[6] In this context, FSP and BMC are interchangeable terms. Several xCAT tools denote the FSP as PBMC, which might stand for Power BMC.

All data is stored in tables, and all data can be manipulated 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:

- Tables:
  - View: **`tabdump`**
  - Change: **`tabedit`** or **`chtab`**
- Objects:
  - View: **`lsdef`**
  - Create: **`mkdef`**
  - Change: **`chdef`**
  - Remove: **`rmdef`** (option: `-t` *`type`*)

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 corresponding compute node.

The xCAT database is stored in a SQLite database by default, but other database management systems can be used on larger clusters.

For more information, see the xCAT database manual page:

```
$ man 5 xcatdb
```

3.4.3 xCAT node booting

The xCAT management node can control the boot method of the nodes at the bootloader (*petitboot*) level. This control requires the correct autoboot configuration in the bootloader, and active configuration of DHCP and Trivial File Transfer Protocol (TFTP) servers with general and per-node settings.

You can 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).

The nodes' bootloader must be configured to automatically boot from the network interface on the xCAT management network (to obtain network and boot configuration through DHCP/TFTP from the xCAT management node), and boot from disk if that fails.

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 contains an operating system installation that is already provisioned.

3.4.4 xCAT node discovery

The xCAT node discovery (or node discovery process) consists of booting a new or undiscovered node with a special-purpose boot image, which is called *genesis*. Genesis informs the xCAT management node about the node and allows the xCAT management node to configure and control the node. This process is described in the following sequence:

1. The new or undiscovered node requests a network address and boot configuration.
2. The xCAT management node does not recognize the node. The xCAT management node 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' Media Access Control (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, 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.

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.

3.4.5 xCAT FSP discovery

The xCAT FSP discovery automatically occurs after node discovery. It is described in the following sequence:

1. The xCAT management node attempts to match the node object definition to a temporary FSP object definition for new or undiscovered FSPs.
2. If a match is identified, the xCAT management node configures the FSP according to the FSP attributes in the node object definition, for example, network configuration.

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

The FSP becomes a discovered FSP, and the corresponding node supports out-of-band operations, for example, power control and monitoring, from the xCAT management node.

3.4.6 xCAT operating system installation types: Disks and state

The xCAT management node can support different methods for provisioning an operating system (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 state 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 either through the network file system (NFS) or initial RAM disk. Changes are written to memory and can be stored (persistent) or discarded (not persistent).

For more information about operating system provisioning and state persistency, see the xCAT online documentation at the following website:

http://sourceforge.net/p/xcat/wiki/XCAT_Linux_Statelite

Note: The availability of a particular installation type for an operating system or version is subject to support by the xCAT release.

3.4.7 xCAT network adapters: Primary and secondary or additional

In xCAT terminology, a network adapter or interface[7] in a node can be either *primary* or *secondary* (also known as *additional*). Only one primary network adapter exists, and zero or more secondary or additional network adapters.

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 or secondary 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 online documentation at the following website:

http://sourceforge.net/p/xcat/wiki/Configuring_Secondary_Adapters

3.4.8 xCAT Software Kits

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

[7] The term *xCAT Network Interface* is more precise and unambiguous in that a single network *adapter* with multiple ports provides multiple network *interfaces* (that is, one interface per port) to the operating system. However, the term is not widely adopted in the documentation.

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 that are provided in the product's distribution packages (incomplete or partial kits are 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 following wiki pages:

- `http://sourceforge.net/p/xcat/wiki/Using_Software_Kits_in_OS_Images`
- `http://sourceforge.net/p/xcat/wiki/IBM_HPC_Software_Kits`

3.4.9 xCAT version

This section describes xCAT version 2.9, which includes the following functionalities and restrictions:

- Ubuntu Server 14.04 and 14.04.1 support on PowerPC64LE (little-endian 64-bit PowerPC):
  - Node types: management node and compute node (no service node/hierarchy support)
  - Provisioning types: Diskful/stateful and diskless/stateless (no diskless/statelite support)
  - Support for software kits
  - Support for non-virtualized (or bare-metal) mode
- POWER8 systems with OPAL firmware:
  - Hardware discovery for FSPs
  - Hardware management with IPMI

For more information about the xCAT 2.9 release, see the following website:

`http://sourceforge.net/p/xcat/wiki/XCAT_2.9_Release_Notes`

Update: After this publication was written, xCAT 2.9.1 was announced. It includes support for Ubuntu Server 14.04.2 on PowerPC64LE (little endian) and provides fixes for several of the issues that are described in this chapter.

For more information about xCAT 2.9.1, see the following website:

`http://sourceforge.net/p/xcat/wiki/XCAT_2.9.1_Release_Notes`

3.4.10 xCAT cluster scenario: Networks, IP addresses, and hostnames

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 (Ethernet): 10.*1*.0.0/16
- Service network (Ethernet): 10.*2*.0.0/16
- Application network (InfiniBand): 10.*3*.0.0/16

- Application network (InfiniBand): 10.*4*.0.0/16
- Site network (Ethernet): 10.*5*.0.0/16

The management and service networks can be combined in a single network; they are separated in this chapter for illustration.

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 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`

3.5 xCAT management node

This section describes the steps to install and configure the xCAT management node. It uses Ubuntu 14.04.1 LTS in non-virtualized mode on an IBM Power System S824L.

3.5.1 Obtain super-user privileges

Perform all operations with super-user (`root`) privileges (prompt `#`), unless specified otherwise (Example 3-1).

Example 3-1 xCAT: Obtaining super-user (root) privileges

```
$ whoami
<username>
$ sudo -i
[sudo] password for <username>:
# whoami
root
```

3.5.2 Install xCAT packages

Install the xCAT GNU Privacy Guard (GPG) key for *Advanced Package Tool* (APT) to verify the authenticity of xCAT packages (Example 3-2).

Example 3-2 xCAT: installing the xCAT GPG public key

```
# wget -O - http://sourceforge.net/projects/xcat/files/ubuntu/apt.key/download |
apt-key add -
<...> - written to stdout [<...>]

OK
```

You can verify the xCAT GPG key installation as shown in Example 3-3 on page 37.

Example 3-3 xCAT: Verifying the xCAT GPG key

```
# apt-key list
/etc/apt/trusted.gpg
--------------------
<...>
pub   1024D/DA736C68 2008-06-27
uid                  Jarrod Johnson <jbjohnso@us.ibm.com>
sub   2048g/FC0752D5 2008-06-27
```

Configure the xCAT Ubuntu package repositories[8] as shown in Example 3-4.

Example 3-4 xCAT: Configuring xCAT Ubuntu package repositories from Source Forge

```
# cat > /etc/apt/sources.list.d/xcat.list <<EOF
deb http://sourceforge.net/projects/xcat/files/ubuntu/2.9/xcat-core trusty main
deb http://sourceforge.net/projects/xcat/files/ubuntu/xcat-dep trusty main
EOF
```

Verify that the Ubuntu package repositories are enabled to satisfy the dependencies of the xCAT packages. The required components are `main`, `restricted`, and `universe`. These components are available from the pockets `release` (default), `updates`, `security`, and `backports`.

The mentioned repositories are enabled, by default, in `/etc/apt/sources.list`. If the repositories are not enabled in `/etc/apt/sources.list` in your system, proceed as shown in Example 3-5, which describes how to manually add them and verify the setting in the configuration file.

Example 3-5 xCAT: Configuring Ubuntu package repositories

```
# for pocket in "" -updates -security -backports
do
   echo "deb http://ports.ubuntu.com/ubuntu-ports trusty$pocket main restricted
universe"
done >> /etc/apt/sources.list
# tail -n4 /etc/apt/sources.list
deb http://ports.ubuntu.com/ubuntu-ports trusty main restricted universe
deb http://ports.ubuntu.com/ubuntu-ports trusty-updates main restricted universe
deb http://ports.ubuntu.com/ubuntu-ports trusty-security main restricted universe
deb http://ports.ubuntu.com/ubuntu-ports trusty-backports main restricted universe
```

Now, update the package manager's definitions (Example 3-6).

Example 3-6 xCAT: Updating APT package repository definitions

```
# apt-get update
<...>
Hit http://ports.ubuntu.com trusty/main ppc64el Packages
Hit http://ports.ubuntu.com trusty/restricted ppc64el Packages
Hit http://ports.ubuntu.com trusty/universe ppc64el Packages
<...>
```

[8] To use Source Forge mirrors (for speed or other considerations), the URLs differ slightly:
`http://<mirror>.dl.sourceforge.net/project/xcat/ubuntu/2.9/xcat-core`
`http://<mirror>.dl.sourceforge.net/project/xcat/ubuntu/xcat-dep`
You can check the list of Source Forge mirrors at http://sourceforge.net/p/forge/documentation/Mirrors or during the process of manually downloading a file from a project in Source Forge (for example, 'heanet').

```
Hit http://ports.ubuntu.com trusty-updates/main ppc64el Packages
Hit http://ports.ubuntu.com trusty-updates/restricted ppc64el Packages
Hit http://ports.ubuntu.com trusty-updates/universe ppc64el Packages
<...>
Hit http://ports.ubuntu.com trusty-security/main ppc64el Packages
Hit http://ports.ubuntu.com trusty-security/restricted ppc64el Packages
Hit http://ports.ubuntu.com trusty-security/universe ppc64el Packages
<...>
Hit http://ports.ubuntu.com trusty-backports/main ppc64el Packages
Hit http://ports.ubuntu.com trusty-backports/restricted ppc64el Packages
Hit http://ports.ubuntu.com trusty-backports/universe ppc64el Packages
<...>
Hit http://heanet.dl.sourceforge.net trusty/main ppc64el Packages
<...>
Hit http://heanet.dl.sourceforge.net trusty/main ppc64el Packages
<...>
```

The `xcat` meta-package must be available for installation (Example 3-7).

Example 3-7 xCAT: Verifying the xcat meta-package

```
# apt-cache show xcat
Package: xcat
Version: 2.9-snap20141208
Architecture: ppc64el
<...>
```

Install the `xcat` meta-package, which includes many packages as dependencies, such as HTTP, domain name service (DNS), TFTP, and NFS servers. It performs several configuration procedures during the process and starts the xCAT server (Example 3-8).

Example 3-8 xCAT: Installing the xcat meta-package

```
# apt-get --yes install xcat
<...>
11 upgraded, 218 newly installed, 0 to remove and 72 not upgraded.
<...>
 * Starting web server apache2
<...>
 * Starting domain name service... bind9
<...>
isc-dhcp-server start/running, process <PID>
<...>
 Adding system startup for /etc/init.d/conserver ...
<...>
tftpd-hpa start/running, process <PID>
<...>
xinetd start/running, process <PID>
<...>
 * Not starting NFS kernel daemon: no exports.
<...>
 Adding system startup for /etc/init.d/xcatd ...
<...>
```

```
Generating RSA private key, 2048 bit long modulus
<...>
xCAT is now running, <...>
```

The xCAT packages are now installed, and the xCAT server is running. The xCAT commands are automatically available on new login shells. To use them in the current shell, run the following command:

```
# source /etc/profile.d/xcat.sh
```

You can check the local xCAT server by checking its version and basic configuration with the **lsxcatd** command (Example 3-9).

Example 3-9 xCAT: Checking version and basic configuration of the xCAT server

```
# lsxcatd -a
Version 2.9 (git commit 32345bc57c5b88b25448b7e89b31bd6d4e58b3ac, built Mon Dec  8
02:22:58 EST 2014)
This is a management node
dbengine=SQLite
```

3.5.3 Configure the xCAT networks

The xCAT networks configuration is stored in the `networks` table. It can be automatically populated with the **makenetworks** command, and manually modified later. The **makenetworks** command collects the information for the `networks` table from the active network configuration. Add the network interface configuration for the management and service network interfaces to `/etc/network/interfaces` (Example 3-10).

Example 3-10 xCAT: Adding network interface configuration to /etc/network/interfaces

```
# cat >> /etc/network/interfaces <<EOF

# xCAT Management Network interface
auto eth5
iface eth5 inet static
    address 10.1.0.1
    netmask 255.255.0.0

# xCAT Service Network interface
auto eth4
iface eth4 inet static
    address 10.2.0.1
    netmask 255.255.0.0

EOF
```

Activate the configuration by restarting the network interfaces with the **ifdown** and **ifup** commands (Example 3-11).

Example 3-11 xCAT: Restarting the network interfaces

```
# ifdown eth5; ifup eth5
# ifdown eth4; ifup eth4
```

Tip: The **ifdown** command is necessary only if the network interface was active or up previously. The command is included here for a typical example.

Note: Restarting the network with the **service network restart** command is not supported in the Ubuntu Server 14.04 LTS series.

(Verify in `/var/log/upstart/networking.log`).

The network interfaces can now be assigned the defined IP addresses and network masks (Example 3-12).

Example 3-12 xCAT: Verifying the IP address and network masks of the network interfaces

```
# ifconfig eth5 | grep 'inet addr'
     inet addr:10.1.0.1  Bcast:10.1.255.255  Mask:255.255.0.0

# ifconfig eth4 | grep 'inet addr'
     inet addr:10.2.0.1  Bcast:10.2.255.255  Mask:255.255.0.0
```

Now, populate the `networks` table with the **makenetworks** command (no output):

```
# makenetworks
```

You can list the collected information in the `networks` table with the **tabdump** or **lsdef** command (Example 3-13).

Example 3-13 xCAT: Listing the networks table with the tabdump command

```
# tabdump networks
#netname,net,mask,mgtifname,gateway,dhcpserver,tftpserver,nameservers,<...>
"fd55:faaf:e1ab:3ce::/64","fd55:faaf:e1ab:3ce::/64","/64","eth0",<...>
"9_114_39_0-255_255_255_0","9.114.39.0","255.255.255.0","eth0",<...>
"192_168_122_0-255_255_255_0","192.168.122.0","255.255.255.0","virbr0",<...>
"10_1_0_0-255_255_0_0","10.1.0.0","255.255.0.0","eth5",<...>
"10_2_0_0-255_255_0_0","10.2.0.0","255.255.0.0","eth4",<...>

Or:

# lsdef -t network
10_1_0_0-255_255_0_0         (network)
10_2_0_0-255_255_0_0         (network)
fd55:faaf:e1ab:3ce::/64      (network)
9_114_39_0-255_255_255_0     (network)
192_168_122_0-255_255_255_0  (network)
```

Usually, the non-xCAT networks need to be removed from that file. To remove them, use the **tabedit** command (text editor) or the **rmdef** command with a comma-separated list of the networks' `netname` attributes (Example 3-14) or one command per network.

Example 3-14 xCAT: Removing networks from the networks table

```
# tabedit networks
<use the text editor to remove the non-xCAT networks lines; save; quit>

Or:
# rmdef -t network
"fd55:faaf:e1ab:3ce::/64","9_114_39_0-255_255_255_0","192_168_122_0-255_255_255_0"
3 object definitions have been removed.
```

Verify that only xCAT networks are now listed in the `networks` table with the **tabdump** or **lsdef** command (Example 3-15).

Example 3-15 xCAT: Verify the networks listing

```
# tabdump networks
#netname,net,mask,mgtifname,gateway,dhcpserver,tftpserver,nameservers,<...>
"10_1_0_0-255_255_0_0","10.1.0.0","255.255.0.0","eth5",<...>
"10_2_0_0-255_255_0_0","10.2.0.0","255.255.0.0","eth4",<...>

Or:
# lsdef -t network
10_1_0_0-255_255_0_0  (network)
10_2_0_0-255_255_0_0  (network)
```

For more information, see the manual page of the `networks` table:

```
# man 5 networks
```

For more information, see the manual page of the **makenetworks** command:

```
# man 8 makenetworks
```

3.5.4 Configure the xCAT management node host name

Ensure that the management node host name is in the `/etc/hosts` file (Example 3-16).

Example 3-16 xCAT: Checking that the management node host name is in the /etc/hosts

```
# hostname xcat-mn.cluster-xcat
# echo 'xcat-mn.cluster-xcat' > /etc/hostname
# echo '10.1.0.1 xcat-mn.cluster-xcat xcat-mn' >> /etc/hosts

# hostname
xcat-mn.cluster-xcat

# hostname -f
xcat-mn.cluster-xcat


# hostname -s
xcat-mn
```

```
# hostname -d
cluster-xcat

# ping -c1 xcat-mn
PING xcat-mn.cluster-xcat (10.1.0.1) 56(84) bytes of data.
64 bytes from xcat-mn.cluster-xcat (10.1.0.1): icmp_seq=1 ttl=64 time=0.022 ms
<...>
```

3.5.5 Configure the xCAT DNS server

The xCAT management node provides DNS services to the cluster so that the nodes can resolve other nodes' names with a single, centralized configuration. Requests that cannot be resolved by the management node (that is, the resolution of non-cluster names, which are names outside of the cluster or unrelated to the cluster) are forwarded to existing DNS servers, for example, the site's DNS servers or external DNS servers.

The xCAT management node uses the following attributes of the `site` table to configure the DNS server and DNS server information in the compute nodes (informed through DHCP):

- `dnsinterfaces`: 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*>, which indicates 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:

```
# man 5 site
```

You can set and verify the attributes that are mentioned in the `site` table with the **chdef** and **lsdef** commands (Example 3-17).

Example 3-17 xCAT: Using the chdef and lsdef commands with the site table

```
< shell line breaks added for clarity >

Set:
# chdef -t site \
   dnsinterfaces=eth5,eth4 \
   domain=cluster-xcat \
   forwarders=9.12.16.2 \
   master=10.1.0.1 \
   nameservers='<xcatmaster>'

Verify:
# lsdef -t site -i dnsinterfaces,domain,forwarders,master,nameservers
Object name: clustersite
    dnsinterfaces=eth5,eth4
    domain=cluster-xcat
    forwarders=9.12.16.2
    master=10.1.0.1
    nameservers=<xcatmaster>
```

Typically, the DNS records must be updated whenever the following changes occur:

- Changes to DNS attributes, such as the previous DNS attributes
- Changes to nodes' network addresses, such as changes in IP addresses and hostnames, and the addition or removal of nodes. (In this case, **`makedns`** can run after **`makehosts`**, which is described later in this section.)

The DNS records can be updated with the **`makedns`** command (Example 3-18). It uses information from the `site`, `networks`, and `hosts` tables, and the `/etc/hosts` file. For more information, see the manual pages of the `networks` and `hosts` tables, and the **`makedns`** and **`makehosts`** commands:

```
# man 5 networks
# man 5 hosts
# man 8 makedns
# man 8 makehosts
```

Example 3-18 xCAT: Updating DNS records with the makedns command

```
# makedns -n
<...>
Updating DNS records, this may take several minutes for a large cluster.
Completed updating DNS records.
```

Important: Ensure the line "`Completed updating DNS records`" is shown at the end of the output.

If the line "`Completed updating DNS records`" is not shown at the end of the output, the DNS records are incomplete. Therefore, the DNS services will not work correctly. To prevent this issue, watch for and address any `Error` lines that are reported by the **`makedns`** command, as shown in Example 3-19. Any lines that are similar to `Ignoring <host>` for non-cluster hosts are not problems.

Example 3-19 xCAT: Checking for and addressing Error lines from the makedns command

```
# makedns -n | grep -i error
Error: Make sure xcat-mn.cluster-xcat exist either in /etc/hosts or DNS.

# echo '10.1.0.1 xcat-mn.cluster-xcat xcat-mn' >> /etc/hosts

# makedns -n | grep -i error
#

# makedns -n
<...>
   Completed updating DNS records.
```

3.5.6 Configure the xCAT DHCP server

The xCAT management node provides DHCP services to the cluster. These services extend beyond providing network configuration information to the nodes. The advanced usage of DHCP services in the xCAT cluster enables several functions in the cluster administration process to happen in a flexible manner, for example:

- Node detection (listening to IP address requests).
- Node identification (matching the requester's MAC address).
- Specification of boot images and arguments by using DHCP attributes, according to the status of a node:
  - Node discovery for new nodes
  - Operating system provisioning for specific nodes
  - Starting the existing operating system for already provisioned nodes

The xCAT management node uses the following attributes from the `site` and `networks` tables to configure the DHCP server:

- `site` table `dhcpinterfaces`

  Network interfaces for the DHCP server to listen on.

- `networks` table `dynamicrange`

  Range of IP addresses that are *temporarily* assigned during the node discovery process, which are required in the xCAT management and service networks. When a non-discovered node requests an IP address, it is assigned an IP address from this range until its discovery process finishes, when it is assigned its reserved IP address. This range cannot overlap with the range of IP addresses that are reserved for discovered nodes.

For more information, see the manual pages of the `site` and `networks` tables with the following commands:

```
# man 5 site
# man 5 networks
```

You can set and verify the `dhcpinterfaces` and `dynamicrange` attributes with the **`chdef`** and **`lsdef`** commands (Example 3-20).

Example 3-20 xCAT: Setting and verifying the dhcpinterfaces and dynamicrange attributes

```
Set the attributes:

# chdef -t site dhcpinterfaces=eth5,eth4
1 object definitions have been created or modified.

# chdef -t network 10_1_0_0-255_255_0_0 dynamicrange="10.1.254.1-10.1.254.254"
1 object definitions have been created or modified.

# chdef -t network 10_2_0_0-255_255_0_0 dynamicrange="10.2.254.1-10.2.254.254"
1 object definitions have been created or modified.

Verify the attributes:

# lsdef -t site -i dhcpinterfaces
Object name: clustersite
    dhcpinterfaces=eth5,eth4
```

```
# lsdef -t network -i dynamicrange
Object name: 10_1_0_0-255_255_0_0
    dynamicrange=10.1.254.1-10.1.254.254
Object name: 10_2_0_0-255_255_0_0
    dynamicrange=10.2.254.1-10.2.254.254
```

You can update the DHCP server configuration with the **makedhcp** command. It uses information from the `site` and `networks` tables, and node address/*hostname* information from the `/etc/hosts` file and DNS service. Restart the DHCP server to ensure that the changes are activated.

If changes occur to the nodes' network addresses, such as the IP address/*hostname* or node addition or removal, which were described earlier, the **makedhcp** command must be run after the **makehosts** and **makedns** commands. For more information, see the manual pages of the **makedhcp, makedns**, and **makehosts** commands:

```
# man 8 makedhcp
# man 8 makedns
# man 8 makehosts
```

Example 3-21 describes the general process to update the DHCP server configuration.

Example 3-21 xCAT: Updating the DHCP server configuration with the makedhcp command

```
Required with changes to nodes' names and addresses:
# makehosts
# makedns -n
<...>
   Completed updating DNS records.

Then:
# makedhcp -n
Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak

# makedhcp -a

# service isc-dhcp-server restart
isc-dhcp-server stop/waiting
isc-dhcp-server start/running, process <PID>
```

You can verify the DHCP server configuration in the `/etc/dhcp/dhcpd.conf` file. This file contains definitions for the xCAT management and service networks, including general network boot configuration and the `dynamicrange` attribute (Example 3-22).

Example 3-22 Verifying the DHCP server configuration in the /etc/dhcp/dhcpd.conf file

```
# cat /etc/dhcp/dhcpd.conf
#xCAT generated dhcp configuration
<...>
shared-network eth5 {
  subnet 10.1.0.0 netmask 255.255.0.0 {
<...>
    option domain-name "cluster-xcat";
    option domain-name-servers  10.1.0.1;
    option domain-search  "cluster-xcat";
<...>
zone 1.10.IN-ADDR.ARPA. {
```

```
<...>
    } 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";
<...>
    range dynamic-bootp 10.1.254.1 10.1.254.254;
  } # 10.1.0.0/255.255.0.0 subnet_end
} # eth5 nic_end
<...>
<and similarly for eth4>
```

3.5.7 Configure the xCAT IPMI password

The xCAT management node uses the IPMI protocol to perform hardware management functions on the nodes through FSP, such as power control, console sessions, and FSP network configuration.

The IPMI password for the nodes can be stored in the `passwd` table and in the `bmcpassword` attribute of a node. If the `bmcpassword` attribute of a node is set, it is used as the password; otherwise, the password in the `passwd` table is used. With this design, you can use a common IPMI password for many nodes, if convenient, and individual IPMI passwords, if and when required.

You can set the IPMI password in the `passwd` table with the **`chtab`** or **`tabedit`** commands, and verify it with the **`tabdump`** command, or you can set it for individual nodes with the **`chdef`** command, and verify it with the **`lsdef`** command (Example 3-23).

Example 3-23 xCAT: Setting and verifying IPMI passwords with chtab, tabdump, chdef, and lsdef

```
Using the passwd table:
# chtab key=ipmi passwd.username='' passwd.password='<ipmi-password>'

That is:
- key: ipmi
- username: null
- password: <ipmi-password>
- remaining fields: null

# tabdump passwd
#key,username,password,cryptmethod,authdomain,comments,disable
<...>
"ipmi",,"<ipmi-password>",,,,

For individual nodes (that are not created yet):
# chdef <node> bmcpassword='<ipmi-password>'
1 object definitions have been created or modified.

# lsdef <node> -i bmcpassword
Object name: <node>
    bmcpassword=<ipmi-password>
```

For more information, see the manual page of the `passwd` table:

```
# man 5 passwd
```

Note: Although an IPMI `username` field exists in the `passwd` table (and `bmcusername` node attribute), it is not used for IPMI management on IBM Power scale-out/Linux servers.

3.5.8 Configure the xCAT console server

The xCAT management node uses a console server to access and share console sessions to the nodes. The console server automatically opens, maintains, and provides access to console sessions. You can access and share console sessions with the **rcons** command.

The console server configuration must be updated and its service must be restarted (for the changes to take effect) whenever nodes are added or removed, or the nodes' `conserver` attribute is changed. You can update the console server configuration with the **makeconservercf** command as shown in Example 3-24.

Example 3-24 xCAT: Updating the configuration of the console server and restarting it

```
# makeconservercf
# service conserver restart
Restarting console server daemon
```

For more information, see the manual pages of the **makeconservercf**, **conserver**, and **rcons** commands:

```
# man 8 makeconservercf
# man 8 conserver
# man 1 rcons
```

3.6 xCAT node discovery

This section describes the required steps for the node discovery process.

3.6.1 Define the xCAT nodes

The xCAT management node requires the configuration of the nodes to be defined in the xCAT database. This configuration is required to perform the node discovery process and to correctly configure the nodes afterward.

Initially, we create the node object definitions for 30 nodes (five racks and six systems per rack) by using the `node range` feature in xCAT.

For more information, see the `noderange` manual page:

```
# man 3 noderange
```

Also, you can add the nodes as members of a group, that is, a group of nodes or *node group,* so that you can refer to the members of that group by simply using the group name. We use the `all` group. The attributes are already assigned for the default node object definition (Example 3-25).

Example 3-25 xCAT: Defining nodes using a noderange and assigning group membership

```
# mkdef p8r[1-5]n[1-6] groups=all
30 object definitions have been created or modified.
```

```
# lsdef
p8r1n1  (node)
p8r1n2  (node)
p8r1n3  (node)
p8r1n4  (node)
p8r1n5  (node)
p8r1n6  (node)
p8r2n1  (node)
p8r2n2  (node)
<...>
p8r5n5  (node)
p8r5n6  (node)

#lsdef --short all
< same output >

# lsdef p8r4n2
Object name: p8r4n2
    groups=all
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
```

Because the basic configuration attributes are common and uniform across the nodes, you can conveniently assign them to a `group` object definition, and add the nodes as members of that group. The nodes are automatically assigned the group's attributes, and you are not required to repetitively define the same set of attributes across the nodes.

For the scenario that is used in this book, we create the `power8_nodes` group. We define the basic configuration attributes that relate to the IP network address, out-of-band management (including the console server), network boot mechanism, and operating system provisioning. You can create and manage a group with the object definition commands (Example 3-26).

Note: Throughout this chapter, the xCAT commands refer to the *p8r4n2* node instead of the *power8_nodes* node group, which is necessary for the clarity and brevity of the output to avoid mixing concurrent and repetitive output from different nodes.

To apply any xCAT command to all nodes, specify *power8_nodes* instead of *p8r4n2*.

Example 3-26 xCAT: Defining a group with attributes and assigning nodes to the group

```
< shell linebreaks added for clarity >

# mkdef -t group -o power8_nodes \
   mgt=ipmi \
   cons=ipmi \
   conserver=10.1.0.1 \
   netboot=petitboot \
   installnic=mac \
   primarynic=mac \
   ip='|p8r(\d+)n(\d+)|10.1.($1+0).($2+0)|' \
   bmc='|p8r(\d+)n(\d+)|10.2.($1+0).($2+0)|'

Warning: Cannot determine a member list for group 'power8_nodes'.
1 object definitions have been created or modified.
```

```
# chdef p8r[1-5]n[1-6] --plus groups=power8_nodes
30 object definitions have been created or modified.

# lsdef p8r4n2
Object name: p8r4n2
    bmc=10.2.4.2
    cons=ipmi
    conserver=10.1.0.1
    groups=power8_nodes
    installnic=mac
    ip=10.1.4.2
    mgt=ipmi
    netboot=petitboot
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    primarynic=mac
```

The IP addresses of a node (`ip` attribute) and its FSP (`bmc` attribute) refer to the permanent IP addresses that the node will be assigned by the node discovery process. The permanent IP addresses replace the temporary IP addresses that were acquired previously.

Note: The IP address attributes use regular expressions so that you can set the value of attributes according to each node's name, therefore simplifying the node definition task. For more information about xCAT regular expressions, check the `xcatdb` manual page:

```
# man 5 xcatdb
```

You must propagate the IP address changes to the `/etc/hosts` file with the **`makehosts`** command. The `/etc/hosts` file uses information from the node object definitions (node name as hostname, and `ip` attribute as IP address) and the `site` or `networks` tables (`domain` attribute as DNS domain name), which is depicted in Example 3-27.

Example 3-27 xCAT: Propagating node and IP address changes to /etc/hosts file with makehosts

```
# makehosts

# cat /etc/hosts
127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

10.1.0.1 xcat-mn.cluster-xcat xcat-mn
10.1.1.1 p8r1n1 p8r1n1.cluster-xcat
10.1.1.2 p8r1n2 p8r1n2.cluster-xcat
10.1.1.3 p8r1n3 p8r1n3.cluster-xcat
10.1.1.4 p8r1n4 p8r1n4.cluster-xcat
10.1.1.5 p8r1n5 p8r1n5.cluster-xcat
10.1.1.6 p8r1n6 p8r1n6.cluster-xcat
10.1.2.1 p8r2n1 p8r2n1.cluster-xcat
10.1.2.2 p8r2n2 p8r2n2.cluster-xcat
<...>
```

```
10.1.5.5 p8r5n5 p8r5n5.cluster-xcat
10.1.5.6 p8r5n6 p8r5n6.cluster-xcat
```

You must also propagate the changes to the DNS server and records with the **makedns** command. You can verify that the DNS name resolution works correctly with the **nslookup** command (Example 3-19 on page 43).

Example 3-28 xCAT: Propagating changes and verifying the DNS server with makedns and nslookup

```
# makedns -n
<...>
   Completed updating DNS records.

Working correctly (name is resolved):
# nslookup p8r4n4.cluster-xcat 10.1.0.1
Server:         10.1.0.1
Address:        10.1.0.1#53

Name:   p8r4n4.cluster-xcat
Address: 10.1.4.4

Not working correctly (name is not resolved):
# nslookup p8r4n4.cluster-xcat 10.1.0.1
Server:         10.1.0.1
Address:        10.1.0.1#53

** server can't find p8r4n4.cluster-xcat: NXDOMAIN
```

Note: If DNS name resolution does not work correctly, the node discovery process on the nodes can fail, looping indefinitely with the following error message:

"`Unrecognized directive`"

Propagate the changes to the DHCP server with the following commands:

```
# makedhcp -n
# makedhcp -a
# service isc-dhcp-server restart
```

Finally, propagate the nodes' configuration to the console server with the **makeconservercf** command and restart its service:

```
# makeconservercf
# service conserver restart
```

3.6.2 Generate the boot image for node discovery

The client-side part (the part that runs on the nodes) of the node discovery process is performed by a small, special-purpose Linux operating system image that is known as the *genesis image*. The genesis image consists of a kernel and an initial RAM disk (initrd) that contains the scripts and programs to communicate with the xCAT server during the node discovery process.

You can generate the genesis image with the **mknb** command. Verify that the `xcat-genesis-scripts` and `xcat-genesis-base-ppc64` packages are installed. (They are automatically installed with the `xcat` meta-package). See Example 3-29 on page 51.

Example 3-29 xCAT: Generating the genesis image with the mknb command

```
# dpkg -l | grep xcat-genesis
ii  xcat-genesis-base-ppc64          2.9-snap201410130243         all
xCAT Genesis netboot image
ii  xcat-genesis-scripts             2.9-snap20141208             ppc64el
xCAT genesis

# mknb ppc64
Creating genesis.fs.ppc64.lzma in /tftpboot/xcat
```

The xCAT installation automatically generates the genesis image (kernel and initrd). However, changes to the xCAT networks configuration, such as adding or removing networks, can require the image to be generated again. See Example 3-30.

Example 3-30 xCAT: Verifying the need to generate the genesis image

```
Checking the default network boot image in the DHCP server configuration:
# 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";

Checking whether it is present (it is not, in this case):
# ls -1 /tftpboot/pxelinux.cfg/p/
127.0.0.0_8
192.168.122.0_24
9.114.39.0_24

Generate the genesis image:
# mknb ppc64
Creating genesis.fs.ppc64.lzma in /tftpboot/xcat

Rechecking whether it is present (it is, now):
# ls -1 /tftpboot/pxelinux.cfg/p/
10.1.0.0_16
10.2.0.0_16
127.0.0.0_8
192.168.122.0_24
9.114.39.0_24
```

This following list depicts the behavior of petitboot if it cannot retrieve either the configuration file, kernel, or initrd, when it scans the network interfaces or boots an entry:

- Configuration file not found

  Petitboot does not display the `xCAT` entry.

- Kernel not found

  Petitboot displays an error message in the last line (Example 3-31).

  Example 3-31 Petitboot error message for kernel not found

```
Petitboot (dev.20141013)                                   8247-42L 211E4BA
 ????????????????????????????????????????????????????????????????????????????????
  [Network: eth0 / 40:f2:e9:d3:89:40]
 *  xCAT

<...>

  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
 Error: Couldn't load kernel image
```

- Initial RAM disk (initrd) not found

  Petitboot displays an error message in the last line (Example 3-32).

Example 3-32 Petitboot error message for initrd not found

```
Petitboot (dev.20141013)                                      8247-42L 211E4BA
??????????????????????????????????????????????????????????????????????????????
  [Network: eth0 / 40:f2:e9:d3:89:40]
 *  xCAT

<...>

  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
 Error: Couldn't load initrd
```

Tip: You can modify the genesis image, for example, to perform debugging and other customizations for the node discovery process. Modify the files that are used in the genesis image, and regenerate it. The files are placed in the following location:

`/opt/xcat/share/xcat/netboot/genesis/ppc64/`

3.6.3 Configure bootloader for node discovery and booting

The xCAT management node coordinates the boot method of the nodes by using the network boot (netboot) configuration (or its absence) that is informed by the DHCP protocol. Configuring the bootloader to automatically boot (autoboot) from network interfaces by using DHCP is required.

This strategy offers more flexibility than directly controlling the bootloader. For example, it enables new or undiscovered nodes (that is, nodes with unrecognized MAC addresses) to boot the node discovery image. It enables known or discovered nodes (that is, with recognized MAC addresses) to boot an arbitrary boot image. The boot image can be either the node discovery again, a shell for firmware upgrades, an operating system installation (provisioning), or that operating system when already installed on disk (by not informing a network boot configuration, so the bootloader falls back to disks).

To configure the bootloader (petitboot) to automatically boot (autoboot) from network interfaces by using DHCP, perform the following steps:

1. Power on (or power cycle, if already on) the system:

   ```
   $ ipmitool -I lanplus -H fsp-address -P ipmi-password power on # or cycle
   ```

2. Open a console session:

   ```
   $ ipmitool -I lanplus -H fsp-address -P ipmi-password sol activate
   ```

3. Wait a few minutes for the Petitboot window.
4. In the Petitboot window, select **System configuration**, which takes you to the Petitboot System Configuration window.
5. From the Autoboot list, select **Autoboot from any disk/network device**, which instructs the bootloader to boot from any network device if it can find a DHCP configuration for network boot. Otherwise, it instructs the bootloader to boot from any disk with an installed operating system (despite the option that shows “disk” before “network device”).

   Note: It is not possible to select **Only Autoboot from a specific disk/network device** with xCAT diskful or stateful operating system provisioning (described in this book). This option restricts the bootloader from falling back to disk if no network boot configuration is found, which is the method that is used by xCAT to boot a diskful or stateful operating system installation.

   This option is not a problem for xCAT diskless, stateless, NFS-root, or statelite operating system provisioning (not covered in this book), although it can require reconfiguring the specific network device in the case of changes to the network interface cards and cables.

6. From the Network list, select **DHCP on all active interfaces**. This option instructs the bootloader to request DHCP configuration on all active network interfaces, that is, interfaces with the state showing `link: up`.

   Note: It is possible to select **DHCP on a specific interface**, although it can require reconfiguring the specific network interface in the case of changes to the network interface cards and cables.

   This option can be useful in environments with more than one DHCP server that is accessible by the nodes. We recommend that you avoid this scenario to prevent unintended issues with booting the nodes as specified by the xCAT management node.

7. Select **OK**, which returns you to the Petitboot window.
8. In a few seconds, the bootloader appends a new entry with the name `xCAT` to the boot options list, which indicates that the bootloader can reach the xCAT DHCP server and that it retrieved a network boot configuration.

The bootloader does not autoboot this time due to the presence of manual intervention. At this stage, you can leave the system waiting at the bootloader prompt (do not boot the `xCAT` entry).

3.6.4 Configure the FSP network for node discovery

The xCAT management node can also manage FSPs to provide functions, such as power control and console sessions for nodes. For this capability, it is necessary for the FSPs to be accessible by the xCAT management node through the network (conceptually, on the xCAT service network).

Important: You are required to use *FSP Ethernet Port 1,* which is also known as *HMC1*, for the xCAT service network. *Do not use Port 2 (HMC2).* A limitation in xCAT 2.9 assigns the FSP its permanent IP address on Port 1/HMC1 regardless of the port/interface that is on the xCAT service network.

The FSP Ethernet Port 2 (HMC2) can be used for other purposes, for example, maintenance and service. Using HMC2 offers another IP address, which is known and independent of xCAT, to access the FSP for debugging problems.

Initially, the xCAT DHCP server provides network configuration for the FSPs, assigning temporary IP addresses from the reserved address range (the `dynamicrange` attribute in the service network's definition). Later, during the node discovery process, the FSPs are assigned permanent IP addresses (the `bmc` attribute in the nodes' definition). Differently from nodes, which are assigned permanent IP addresses with dynamic IP configuration (DHCP) based on MAC-address matching, FSPs are assigned permanent IP addresses with static IP configuration.

You can use one of two methods to set the FSP Ethernet Port number 1 (HMC1) to acquire network configuration with DHCP (if it is not already set): either through IPMI or in the ASM interface:

- IPMI

  You can use the **`ipmitool`** command to configure the "LAN channels". Each FSP Ethernet Port is a LAN channel (Port 1: LAN channel 1 and Port 2: LAN channel 2). Example 3-33 describes how to configure the FSP Etherport Port 1 for DHCP network configuration and the FSP Ethernet Port 2 for static network configuration.

  Example 3-33 xCAT: Configuring the FSP Ethernet Port with the ipmitool command

```
Set FSP Ethernet Port 1 (HMC1) to DHCP IP address:
# ipmitool -I lanplus -H <fsp-address> -P <password> lan set 1 ipsrc dhcp

Set FSP Ethernet Port 2 (HMC2) to Static IP address:
# ipmitool -I lanplus -H <fsp-address> -P <password> lan set 2 ipsrc static
# ipmitool -I lanplus -H <fsp-address> -P <password> lan set 2 ipaddr 10.0.0.42
# ipmitool -I lanplus -H <fsp-address> -P <password> lan set 2 netmask
255.0.0.0
# ipmitool -I lanplus -H <fsp-address> -P <password> lan set 2 defgw 10.0.0.1

Display FSP Ethernet Port 2 (HMC2) network configuration:
# ipmitool -I lanplus -H <fsp-address> -P <password> lan print 2
Set in Progress         : Set Complete
IP Address Source       : Static Address
IP Address              : 10.0.0.42
```

```
Subnet Mask             : 255.0.0.0
MAC Address             : 98:be:94:4b:82:8c
Default Gateway IP      : 10.10.0.1
Cipher Suite Priv Max   : Not Available


Description of several ipmitool options for a LAN channel:
# ipmitool -I lanplus -H <fsp-address> -P <password> lan set
<...>
   ipaddr <x.x.x.x>              Set channel IP address
   netmask <x.x.x.x>             Set channel IP netmask
   macaddr <x:x:x:x:x:x>         Set channel MAC address
   defgw ipaddr <x.x.x.x>        Set default gateway IP address

   ipsrc <source>                Set IP Address source
      none   = unspecified source
      static = address manually configured to be static
      dhcp   = address obtained by BMC running DHCP
      bios   = address loaded by BIOS or system software
<...>
```

- ASMI:
  - Expand **Network Services** and click **Network Configuration**.
  - In the Service Processor: Primary section (which was the only section at the time that this publication was written, but in a future system with two FSP processors, depending on whether your system has redundant/two FSPs, you can select **Secondary**), select **IPv4**.
  - Click **Continue**.
  - In the Network interface eth0 section, perform these steps:
    - Select **Configure this interface**.
    - For the field or label IPv4, select **Enabled**.
    - For the field or label Type of IP address, select **Dynamic**.
  - On the bottom of the page, click **Continue**.
  - On the next page, verify that the network configuration in the field/frame eth0 is correct.
  - Click **Save Settings**.

In a few seconds[9], the xCAT DHCP server receives the resulting DHCP request and services it by offering an IP address from the temporary IP address range. Then, the DHCP server receives periodic DHCP requests from the FSP to renew the address lease of that IP address (Example 3-34).

Example 3-34 xCAT: Example of DHCP request handling on the xCAT management node system log

```
# tail /var/log/syslog
<...>
<...> dhcpd: DHCPDISCOVER from 98:be:94:4b:82:8c via eth4
<...> dhcpd: DHCPOFFER on 10.2.254.1 to 98:be:94:4b:82:8c via eth4
<...> dhcpd: DHCPREQUEST for 10.2.254.1 (10.2.0.1) from 98:be:94:4b:82:8c via eth4
<...> dhcpd: DHCPACK on 10.2.254.1 to 98:be:94:4b:82:8c via eth4
<...>
```

[9] Approximately 5 seconds for IPMI, and 15 seconds for ASMI, as of this writing.

```
<...> dhcpd: DHCPREQUEST for 10.2.254.1 from 98:be:94:4b:82:8c via eth4
<...> dhcpd: DHCPACK on 10.2.254.1 to 98:be:94:4b:82:8c via eth4
<...>
<...> dhcpd: DHCPREQUEST for 10.2.254.1 from 98:be:94:4b:82:8c via eth4
<...> dhcpd: DHCPACK on 10.2.254.1 to 98:be:94:4b:82:8c via eth4
<...>
```

You can verify the IP configuration that was obtained by the FSP with the **`ipmitool`** command (Example 3-35). You can also check the progress of the DHCP configuration. When it is complete, the `Set in Progress` field status changes from `Set In Progress` to `Set Complete`.

Example 3-35 xCAT: Verifying the FSP Ethernet Port 1 (HMC1) network configuration with ipmitool

```
# ipmitool -I lanplus -H <fsp-address> -P <password> lan print 1
Set in Progress         : Set Complete
IP Address Source       : DHCP Address
IP Address              : 10.2.254.1
Subnet Mask             : 255.255.0.0
MAC Address             : 98:be:94:4b:82:8c
Default Gateway IP      : 10.2.0.1
Cipher Suite Priv Max   : Not Available
```

Important: If the IP address is not acquired during the FSP boot process (that is, either when the physical system is powered on or when the FSP is reset), the FSP will not respond to the Service Location Protocol (SLP) requests (from the **`lsslp`** command that is described in the next sections) with an IPv4 address, but with an IPv6 address. This problem happens, for example, if the FSP network configuration changes from a static IP address to a dynamic IP address (at the time that this publication was written):

```
# lsslp -i 10.2.0.1 -s PBMC
Sending SLP request on interfaces: 10.2.0.1 ...
Received 4 responses.
Sending SLP request on interfaces: 10.2.0.1 ...
Received 0 responses.
<... 2 repetitions of the “Sending/Received” block ...>
4 requests with 4 responses.  Now processing responses.  <...>
device  type-model  serial-number  side  ip-addresses  hostname
pbmc    8247-42L    211E49A             fe80::9abe:94ff:fe4b:828cServer-<...>
```

This situation causes communication problems between xCAT and the FSP, therefore affecting the node discovery (nodes and the related FSPs are not matched).

To resolve this issue, reset the FSP by either the ASMI (see 3.2.3, “Reset the FSP” on page 28) or the **`ipmitool`** command (3.3, “Intelligent Platform Management Interface (IPMI)” on page 28). Alternatively, you can power cycle the physical system through power cables.

3.6.5 Configure the xCAT FSP nodes and compute nodes for node discovery

The xCAT management node associates FSPs with their related (compute) nodes so that functions, such as power control and console sessions of a node, which are actually performed on its FSP, can be requested by specifying the node instead of its FSP (more intuitively).

To associate the FSP and node, xCAT matches the *Machine-Type (and) Model* (MTM) and *Serial Number* (SN) information of a system between a node object definition and a (temporary) FSP object definition (specifically, the `mtm` and `serial` attributes).

This list provides an overview of that process:

1. During the node discovery process, the genesis image collects system information, including the MTM and SN, and transfers it to the xCAT server.
2. The xCAT server attempts to match the MTM and SN to a node object definition to identify the node. This action is required to assign the system information that was collected to the correct node and to obtain configuration information to configure the node, for example, the permanent IP address and next steps.
3. The xCAT server attempts to match the MTM and SN to an FSP object definition to identify the temporary IP address of that FSP. This action is required to configure the FSP with the configuration information that was obtained from the node object definition, for example, permanent FSP IP address and IPMI password.
4. Finally, the xCAT server configures the FSP, for example, the network configuration. The FSP is now accessible through node object definition, and the FSP object definition is removed (which is why it is temporary).

To create the temporary FSP object definitions, use the **`lsslp`** command, which uses the SLP to discover FSPs on a network and obtain information, such as the MTM and SN.

The **`lsslp`** command accepts the following options:

- `-i`: IP address of the network interface in the network that you want (default: All interfaces)
- `-s`: Service type to discover (for example, PBMC for FSPs that are running the OPAL firmware)
- `-w`: Option to write discovery results to object definitions (It is useful to verify results first, then write.)

Example 3-36 describes the creation of one FSP object definition. (Only one system was powered on for clarity.) The **`lsslp`** command options specify the discovery of the FSPs (**`-s PBMC`**) on the xCAT service network (`-i 10.2.0.1`) and the storage of results as object definitions (**`-w`**) after they are verified. The object definitions define the `name`, `mtm`, and `serial` attributes according to the **`lsslp`** command output. The attributes are available for xCAT commands similar to other object definitions (for example, the **`lsdef`** and **`rpower`** commands).

Example 3-36 xCAT: Discovering FSPs with the lsslp command

```
# lsslp -s PBMC -i 10.2.0.1
Sending SLP request on interfaces: 10.2.0.1 ...
Received 4 responses.
Sending SLP request on interfaces: 10.2.0.1 ...
Received 0 responses.
Sending SLP request on interfaces: 10.2.0.1 ...
Received 0 responses.
Sending SLP request on interfaces: 10.2.0.1 ...
Received 0 responses.
4 requests with 4 responses.  Now processing responses.  This will take 0-1
minutes...
Begin to write into Database, this may change node name
device  type-model  serial-number  side  ip-addresses  hostname
pbmc    8247-42L    211E49A              10.2.254.1    Server-8247-42L-SN211E49A

# lsslp -s PBMC -i 10.2.0.1 -w
```

```
<... identical output ...>

# lsdef Server-8247-42L-SN211E49A
Object name: Server-8247-42L-SN211E49A
    bmc=10.2.254.1
    groups=pbmc,all
    hidden=0
    mgt=ipmi
    mtm=8247-42L
    nodetype=ppc
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=211E49A

# lsdef Server-8247-42L-SN211E49A -i mtm,serial
Object name: Server-8247-42L-SN211E49A
    mtm=8247-42L
    serial=211E49A

# rpower Server-8247-42L-SN211E49A stat
Server-8247-42L-SN211E49A: on
```

To configure the MTM and SN on node object definitions, use the **`chdef`** command to set the `mtm` and `serial` attributes for each node (Example 3-37).

Example 3-37 xCAT: Defining the machine-type model and serial number attributes for a node

```
# chdef p8r4n2 mtm=8247-42L serial=211E49A
1 object definitions have been created or modified.
```

Note: Obtain the Machine-Type and Model and Serial Number information from the system chassis or by using the **`lsslp`** command (powering on one system/FSP at a time).

With the configured node and FSP object definitions, the system is ready for the node discovery process.

3.6.6 Perform the node discovery process

Now that the xCAT object definitions for the nodes and FSPs are configured, you are required to power on or power cycle the nodes. The node discovery starts automatically. Petitboot requests a network and boot configuration, which is provided by the xCAT DHCP server, then downloads the boot image from the xCAT TFTP server, and boots it.

The operation to power on a node is addressed to its FSP. You can use either a known or static IP address, for example, the address that is assigned to the FSP’s Ethernet Port 2, or the temporary FSP object definition, which is listed in the output of the **`lsslp`** command. With an IP address, you can use either the **`ipmitool`** command (see 3.3, “Intelligent Platform Management Interface (IPMI)” on page 28) or the ASMI (see 3.2, “OPAL firmware and the ASM interface” on page 27). With a temporary FSP object definition, you can use the **`rpower`** command in the xCAT management node. The commands are shown:

- Use the **`ipmitool`** command:

  ```
  # ipmitool -I lanplus -H <fsp-address> -P <ipmi-password> power on # or cycle
  ```

- Use the **rpower** command with the temporary FSP object definition:

```
# rpower Server-8247-42L-SN211E49A on
```

You can watch the node discovery process on a specific node in several ways. On the xCAT management node, the xCAT daemon prints log messages, which are available in the `/var/log/syslog` file, about the process to the system log, and you can access the node's console with the **rcons** or **ipmitool** command. On a node that is not the xCAT management node, you can access the node's console directly with the **ipmitool** command by using one of the FSP's IP addresses (either FSP Ethernet Port 1 or 2):

- Use the **ipmitool** command:

```
# ipmitool -I lanplus -H <fsp-address> -P <ipmi-password> sol activate
```

- Use the **rcons** command with the temporary FSP object definition:

```
# rcons Server-8247-42L-SN211E49A
```

Important: The FSP discovery, which was performed automatically after node discovery, changes the FSP's IP address that was assigned to FSP Ethernet Port 1 from a temporary IP address to its permanent IP address. This change causes the current console session to stop working. You must reopen the console session on the newly assigned FSP's IP address or switch from the temporary FSP object definition to the node object definition with the **rcons** command. Another option is to continue to use the FSP's IP address (which did not change) that was assigned to FSP Ethernet Port 2.

The node discovery process as shown in the system log of the xCAT management node is depicted in Example 3-38 on page 60, which registers the following sequence of log messages (despite the successful process, several error messages and duplicated messages exist):

1. Two DHCP IP address requests from the node are shown, from the same MAC address:
   - The first DHCP IP address is from the petitboot bootloader (temporary IP address 10.1.254.1), which did not release the provided IP address.
   - The second DHCP IP address is from the genesis image (temporary IP address 10.1.254.2), which gets a different IP address because the previously provided IP address was not released.
2. The node asks the xCAT management node for its credentials, which are required to establish communication with the xCAT daemon.
3. The node sends a discovery request with its collected hardware information, which is matched, identified, and processed by the xCAT management node.
4. The xCAT management node also matches the FSP temporary object definition, then configures the FSP network and removes the temporary object definition.
5. The node performs a DHCP IP address renewal to be provided its permanent IP address. During the IP address renewal process, the old IP address is not acknowledged (that is, refused) by the server, the node then asks for a new IP address, and the DHCP server provides the node with its permanent address.
6. A failure exists in notifying the node (from the FSP worker instance) that it is the node with the specified hostname. However, that log message likely occurs because the notification occurs on all IP addresses that are known for that node, including the just released temporary address, which causes the obvious failure.
7. The node receives the *hostname* notification correctly on its permanent IP address and requests the credentials again with the permanent IP address, which the xCAT management node correctly resolved to its *hostname*, `p8r4n2`.

Example 3-38 xCAT management node log during the node discovery process of a node

```
# tail -f /var/log/syslog
<...>
<...> dhcpd: DHCPDISCOVER from 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPOFFER on 10.1.254.1 to 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPREQUEST for 10.1.254.1 (10.1.0.1) from 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPACK on 10.1.254.1 to 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPDISCOVER from 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPOFFER on 10.1.254.2 to 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPREQUEST for 10.1.254.2 (10.1.0.1) from 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPACK on 10.1.254.2 to 40:f2:e9:d3:85:90 via eth5
<...> xCAT[27538]: xCAT: Allowing getcredentials x509cert
<...> xCAT[20858]: xcatd: Processing discovery request from 10.1.254.2
<...> xCAT[20858]: Discovery Error: Could not find any node.
<...> xCAT[20858]: Discovery Error: Could not find any node.
<...> xCAT[20858]: xcatd: Processing discovery request from 10.1.254.2
<...> xCAT[20858]: Discovery info: configure password for
pbmc_node:Server-8247-42L-SN211E49A.
<...> xCAT[27575]: xCAT: Allowing rspconfig to Server-8247-42L-SN211E49A password=
for root from localhost
<...> xCAT[20858]: Discover info: configure ip:10.2.4.2 for
pbmc_node:Server-8247-42L-SN211E49A.
<...> xCAT[27586]: xCAT: Allowing rspconfig to Server-8247-42L-SN211E49A
ip=10.2.4.2 for root from localhost
<...> xCAT[20858]: Discovery info: remove pbmc_node:Server-8247-42L-SN211E49A.
<...> xCAT[27597]: xCAT: Allowing rmdef Server-8247-42L-SN211E49A for root from
localhost
<...> xcatd: Discovery worker: fsp instance: nodediscover instance: p8r4n2 has
been discovered
<...> dhcpd: DHCPRELEASE of 10.1.254.2 from 40:f2:e9:d3:85:90 via eth5 (found)
<...> dhcpd: DHCPDISCOVER from 40:f2:e9:d3:85:90 via eth5
<...> xCAT[20858]: Discovery info: configure password for
pbmc_node:Server-8247-42L-SN211E49A.
<...> xCAT[20858]: Discover info: configure ip:10.2.4.2 for
pbmc_node:Server-8247-42L-SN211E49A.
<...> xCAT[20858]: Discovery info: remove pbmc_node:Server-8247-42L-SN211E49A.
<...> xCAT[27633]: xCAT: Allowing rmdef Server-8247-42L-SN211E49A for root from
localhost
<...> dhcpd: DHCPOFFER on 10.1.254.2 to 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPREQUEST for 10.1.254.2 (10.1.0.1) from 40:f2:e9:d3:85:90 via
eth5: lease 10.1.254.2 unavailable.
<...> dhcpd: DHCPNAK on 10.1.254.2 to 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPDISCOVER from 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPOFFER on 10.1.4.2 to 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPREQUEST for 10.1.4.2 (10.1.0.1) from 40:f2:e9:d3:85:90 via eth5
<...> dhcpd: DHCPACK on 10.1.4.2 to 40:f2:e9:d3:85:90 via eth5
<...> xcatd: Discovery worker: fsp instance: nodediscover instance: Failed to
notify 10.1.254.2 that it's actually p8r4n2.
<...> xCAT[27637]: xCAT: Allowing getcredentials x509cert from p8r4n2
<...> xCAT: credentials: sending x509cert
```

Note: Log messages that are similar to the following log message indicate connectivity problems between the node and the xCAT management node (`xcat-mn` in this example):

```
<...> xCAT[27589]: Error dispatching request to xcat-mn:3001: Connection
failure: IO::Socket::INET6: getaddrinfo: Name or service not known at
/opt/xcat/lib/perl/xCAT/Client.pm line 236.
```

The node discovery process as shown in the node's console is depicted in Example 3-39. At the end of the process, the node is instructed to periodically check the xCAT management node for instructions.

Example 3-39 xCAT: Node's console during the node discovery process

```
<...>
 Petitboot (dev.20141013)                                      8247-42L 211E49A
 ??????????????????????????????????????????????????????????????????????????????
[Network: eth0 / 40:f2:e9:d3:85:90]
*   xCAT

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 10 sec: xCAT
<...>
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
 -> smp_release_cpus()
spinning_secondaries = 159
 <- smp_release_cpus()
 <- setup_system()
<...>

Done
ERROR: BOOTIF missing, can't detect boot nic
Could not load host key: /etc/ssh/ssh_host_ecdsa_key
Generating private key...Done
Setting IP via DHCP...
[<...>] bnx2x: [bnx2x_dcbnl_set_dcbx:2353(enP3p9s0f0)]Requested DCBX mode 5 is
beyond advertised capabilities
[<...>] bnx2x: [bnx2x_dcbnl_set_dcbx:2353(enP3p9s0f1)]Requested DCBX mode 5 is
beyond advertised capabilities
```

```
[<...>] bnx2x: [bnx2x_dcbnl_set_dcbx:2353(enP3p10s0f2)]Requested DCBX mode 5 is
beyond advertised capabilities
Acquiring network addresses..Acquired IPv4 address on enP3p9s0f0: 10.1.254.2/16
Could not open device at /dev/ipmi0 or /dev/ipmi/0 or /dev/ipmidev/0: No such file
or directory
Could not open device <...repetition...>
Could not set IPMB address: Bad file descriptor
Could not open device <...repetition...>
Get Device ID command failed

Beginning node discovery process
Waiting for nics to get addresses
enP3p9s0f0|10.1.254.2
rsyslogd: error: option -c is no longer supported - ignored
Network configuration complete, commencing transmit of discovery packets
Done waiting
/usr/bin/doxcat: line 314:  1215 Terminated              dhclient -6 -pf
/var/run/dhclient6.$bootnic.pid $bootnic -lf /var/lib/dhclient/dhclient6.leases

Received request to retry in a bit, will call xCAT back in 166 seconds
```

After the node discovery and FSP discovery finish, you can check the FSP for its permanent IP address with the **ipmitool** command. You can check the out-of-band connectivity with the **rpower** command. And, you can verify the node for in-band connectivity with the **nodestat** command (Example 3-40).

Example 3-40 xCAT: Checking a node's FSP IP address, and in-band and out-of-band connectivity

```
# ipmitool -I lanplus -H <fsp-address> -P <ipmi-password> lan print 1
Set in Progress         : Set Complete
IP Address Source       : Static Address
IP Address              : 10.2.4.2
Subnet Mask             : 255.255.0.0
MAC Address             : 98:be:94:4b:82:8c
Default Gateway IP      : 10.2.0.1
Cipher Suite Priv Max   : Not Available

# rpower p8r4n4 status
p8r4n2: on

# nodestat p8r4n2
p8r4n2: sshd
```

You can check the node's object definition to verify that the hardware attributes were collected during the node discovery process (Example 3-41).

Example 3-41 xCAT: Listing a node's object definition with its hardware attributes

```
# lsdef p8r4n2
Object name: p8r4n2
    arch=ppc64
    bmc=10.2.4.2
    cons=ipmi
    conserver=10.1.0.1
    cpucount=160
    cputype=POWER8E (raw), altivec supported
    groups=all,power8_nodes
    installnic=mac
    ip=10.1.4.2
    mac=40:f2:e9:d3:85:90
    memory=261379MB
    mgt=ipmi
    mtm=8247-42L
    netboot=petitboot
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    primarynic=mac
    serial=211E49A
    status=standingby
    statustime=<...>
    supportedarchs=ppc64
```

3.7 xCAT compute nodes: Operating system

This section describes the steps to provision (install) the operating system (OS) to the xCAT compute nodes, including the required configuration in the xCAT management node.

The operating system provisioning is an operating system network installation (sometimes referred to as a *netboot* installation) that is modified to suit the xCAT cluster. The xCAT management node combines the contents of the operating system installation media with the operating system network installation mechanism, and performs configuration changes, then makes it available to the nodes through DHCP network and boot configuration. Then, you can boot a node into an automatic and unattended operating system network installation, which comes configured correctly for cooperating with the xCAT management node.

Note: At the time that this publication was written, a few challenges existed with the netboot files of Ubuntu Server 14.04.1 LTS. These challenges require additional steps in the configuration of the xCAT management node with xCAT 2.9. The following list summarizes the challenges, and the sections that describe and address them:

- The installation media does not contain the netboot files, which are required for the operating system provisioning. (This challenge is addressed in 3.7.4, “Download the netboot files” on page 65.)
- The online netboot files contain a kernel and modules that are incompatible with other modules that are in the installation media. (This challenge is addressed in 3.7.5, “Create a package repository for installer modules that are compatible with the netboot files” on page 66.)

3.7.1 Set node attributes for operating system network installation

To perform the operating system provisioning, several commands and scripts in the xCAT management node, for example, **nodeset** and **debian.pm**, require attributes to be defined in the xCAT Node object definitions. These attributes are used, for example, in the kernel command line to boot the operating system network installation, in pointers to operating system package repositories (in the xCAT management node), and in console configuration settings.

You can use the **chdef** command to set the `nfsserver`, `tftpserver`, `serialport`, and `serialspeed` attributes in the node group. You can use the **lsdef** command to verify the setting (Example 3-42).

Note: Several attributes that are also required for operating system provisioning were set in 3.6.1, “Define the xCAT nodes” on page 47.

Example 3-42 xCAT: Setting the nfsserver, tftpserver, serialport, and serialspeed attributes

```
# chdef -t group power8_nodes \
    nfsserver=10.1.0.1 \
    tftpserver=10.1.0.1
1 object definitions have been created or modified.

# lsdef p8r4n2 -i nfsserver,tftpserver
Object name: p8r4n2
    nfsserver=10.1.0.1
    tftpserver=10.1.0.1

# chdef -t group -o power8_nodes \
    serialport=0 \
    serialspeed=115200
1 object definitions have been created or modified.

# lsdef p8r4n2 -i serialport,serialspeed
Object name: power8_nodes
    serialport=0
    serialspeed=115200
```

3.7.2 Set the root password for nodes

xCAT uses the password information in the `passwd` table to configure the `root` password in the nodes.

You can set the `root` password in the `passwd` table with the **chtab** command, and you can verify the setting with the **tabdump** command (Example 3-43).

Example 3-43 xCAT: Setting the root password for the nodes and verifying the setting

```
# chtab key=system passwd.username=root passwd.password=<root-password>

That is,
- key: system
- username: root
- password: <root-password>
- remaining fields: null

Verify it:

# tabdump passwd
#key,username,password,cryptmethod,authdomain,comments,disable
"omapi","xcat_key","QXRHUlJocW1TSzdYNEdOODBtQUcwT3YzYXcydOx3eDU=",,,,
"ipmi",,"<ipmi-password>",,,,
"system","root","<root-password>",,,,
```

3.7.3 Create the operating system image object definitions

xCAT stores the information that relates to the operating systems that are available for provisioning (that is, the installation profiles) in object definitions of the `osimage` type, which are also referred to as *OS images*.

You can list the available operating system images with the **lsdef** command:

```
# lsdef -t osimage
```

You can use the **copycds** command with an operating system installation media to automatically define the operating system images, to configure the xCAT management node to store its contents in the `/install` directory, and to make it available for operating system network installations (Example 3-44).

Example 3-44 xCAT: Creating operating system images from the OS installation media

```
# lsdef -t osimage
Could not find any object definitions to display.

# copycds /path/to/ubuntu-14.04.1-server-ppc64el.iso
Copying media to /install/ubuntu14.04.1/ppc64el
Media copy operation successful

# lsdef -t osimage
ubuntu14.04.1-ppc64el-install-compute  (osimage)
ubuntu14.04.1-ppc64el-install-kvm  (osimage)
ubuntu14.04.1-ppc64el-netboot-compute  (osimage)
ubuntu14.04.1-ppc64el-statelite-compute  (osimage)
```

The **copycds** command creates several installation profiles from an installation media, according to the possible installation types (as described in 3.4.6, “xCAT operating system installation types: Disks and state” on page 34) and other variables, for example, virtualization methods.

For more information about the **copycds** command, see the manual page:

```
# man copycds
```

3.7.4 Download the netboot files

At the time that this publication was written, the installation media of Ubuntu Server 14.04.1 LTS for IBM POWER8 did not contain the netboot files, which are required for operating system provisioning. It is necessary to manually download the netboot files to the xCAT management node of the xCAT 2.9 release.

You can download the netboot files (`vmlinux` and `initrd.gz`) for Ubuntu releases at the following address:

http://cdimage.ubuntu.com/netboot

Example 3-45 on page 66 describes the steps to download the netboot files for the Ubuntu Server 14.04 LTS series (which is also known as *Trusty Tahr*) to the appropriate location that is defined by the **copycds** command (`/install/ubuntu14.04.1/ppc64el`). The example uses variables to store the common URL directory and the destination directory.

Note: The downloaded `vmlinux` file overwrites the file that is distributed in the installation media, and only the downloaded `initrd.gz` (not `vmlinux`) file is placed in the `netboot` directory.

Example 3-45 xCAT: Downloading the netboot files for Ubuntu 14.04 LTS

```
# dir='/install/ubuntu14.04.1/ppc64el'
#
url='http://ports.ubuntu.com/ubuntu-ports/dists/trusty-updates/main/installer-ppc6
4el/current/images/netboot/ubuntu-installer/ppc64el'

# mkdir -p $dir/install/netboot
# wget $url/vmlinux -O $dir/install/vmlinux
# wget $url/initrd.gz -O $dir/install/netboot/initrd.gz
```

3.7.5 Create a package repository for installer modules that are compatible with the netboot files

The netboot files (a kernel and an initial RAM disk) are based on the kernel packages and are therefore equally subject to updates. Eventually, an update introduces incompatibility with previous kernel versions, which is reflected as a version change in the kernel, and kernel-related packages (for example, install-time packages with kernel modules).

The installer verifies the running kernel version, and attempts to obtain and load install-time packages with kernel modules that are a compatible version. If the installer cannot find any, the installation fails.

This process causes a problem for operating system provisioning with xCAT because the installer can only obtain the contents and packages that are served by the xCAT management node, which are based on the installation media. The relevant packages in the installation media are eventually made incompatible with the netboot files that are available online (currently required, as described in 3.7.4, “Download the netboot files” on page 65) by kernel package updates, therefore causing the operating system provisioning to fail.

Currently, to address this problem, a manual process is used to create the package repository for updates, which is expected by the installer, in the xCAT management node, and to provide the install-time packages with kernel modules that are up-to-date with the netboot kernel. This process is demonstrated in Example 3-46 and includes the following steps:

1. Identify the kernel version in the netboot files.
2. Copy the indexes of the package repository with updates, filtering the data that is relevant to the kernel-related install-time packages in the identified version.
3. Download the required install-time packages.

Example 3-46 xCAT: Create a package repository for current install-time packages with kernel modules

```
# dir='/install/ubuntu14.04.1/ppc64el/'

# mirror='http://ports.ubuntu.com/ubuntu-ports'
# dist='trusty-updates'
# component='main'
# arch='ppc64el'
# path="$component/debian-installer/binary-$arch"
# packages_file='Packages.gz'
# packages="$dir/dists/$dist/$path/$packages_file"
# release_file='Release'
# release="$dir/dists/$dist/$release_file"

# mkdir -p $dir/dists/$dist/$path

# vmlinux="$dir/install/vmlinux"
# version="$(grep 'Linux version' --text $vmlinux | cut -d' ' -f3)"

# wget -O- "$mirror/dists/$dist/$path/$packages_file" \
  | gunzip --stdout \
  | sed -n -e "/^Package:/,/^\$/ H;" -e "/^\$/ { x; /Kernel-Version: $version/ 
s/\n//p }" \
  | gzip \
  > $packages

# wget -O- $mirror/dists/$dist/$release_file \
  | sed -e '/:$/,$ d' -e "/^Components:/ s/: .*/: $component/"  \
  > $release

# size="$(wc --bytes < $packages | tr -d ' ')"
# (
    cd $dir/dists/$dist
    file=$path/$packages_file
    echo 'MD5Sum:'
    md5sum $file
    echo 'SHA1:'
    sha1sum $file
    echo 'SHA256:'
    sha256sum $file
  ) \
  | sed "s/  / $size /" \
  >> $release

# gunzip --stdout $packages \
  | awk '/^Filename:/ { print $2 }' \
  | while read pool_file
    do
      wget -nv $mirror/$pool_file -O $dir/$pool_file
    done
```

You can verify the resulting package repository and downloaded files as described in Example 3-47.

Example 3-47 xCAT: Verifying the created package repository with up-to-date packages

```
# find $dir/dists/$dist/
/install/ubuntu14.04.1/ppc64el//dists/trusty-updates/
/install/ubuntu14.04.1/ppc64el//dists/trusty-updates/main
/install/ubuntu14.04.1/ppc64el//dists/trusty-updates/main/debian-installer
/install/ubuntu14.04.1/ppc64el//dists/trusty-updates/main/debian-installer/binary-
ppc64el
/install/ubuntu14.04.1/ppc64el//dists/trusty-updates/main/debian-installer/binary-
ppc64el/Packages.gz
/install/ubuntu14.04.1/ppc64el//dists/trusty-updates/Release
```

```
# find $dir/pool/ -name "*$version*"
/install/ubuntu14.04.1/ppc64el//pool/main/l/linux/scsi-modules-3.13.0-44-generic-d
i_3.13.0-44.73_ppc64el.udeb
<...>
/install/ubuntu14.04.1/ppc64el//pool/main/l/linux/block-modules-3.13.0-44-generic-
di_3.13.0-44.73_ppc64el.udeb
```

3.7.6 Set the operating system image of the nodes

The operating system image (or *installation profile*) that was used in this book is the diskful/stateful installation for compute nodes. This installation profile is defined in the `ubuntu14.04.1-ppc64el-install-compute` osimage object definition.

You can set the operating system image to be provisioned to a node with the **`nodeset`** command (Example 3-48).

Example 3-48 xCAT: Setting the operating system image for a compute node

```
# nodeset p8r4n2 osimage=ubuntu14.04.1-ppc64el-install-compute
p8r4n2: install ubuntu14.04.1-ppc64el-compute
```

The **`nodeset`** command performs automatic configuration of several components for the node, for example, attributes in a node object definition, DHCP server, and the boot configuration (Example 3-49).

Example 3-49 xCAT: Automatic configuration that is performed by the nodeset command

```
# lsdef p8r4n2
Object name: p8r4n2
    arch=ppc64el
    bmc=10.2.4.2
    cons=ipmi
    cpucount=160
    cputype=POWER8E (raw), altivec supported
    currchain=boot
    currstate=install ubuntu14.04.1-ppc64el-compute
    groups=all,power8_nodes
    initrd=xcat/osimage/ubuntu14.04.1-ppc64el-install-compute/initrd.img
    installnic=mac
    ip=10.1.4.2
    kcmdline=nofb utf8 auto url=http://10.1.0.1/install/autoinst/p8r4n2
xcatd=10.1.0.1 mirror/http/hostname=10.1.0.1
netcfg/choose_interface=40:f2:e9:d3:85:90 console=tty0 console=hvc0,115200
locale=en_US priority=critical hostname=p8r4n2
live-installer/net-image=http://10.1.0.1/install/ubuntu14.04.1/ppc64el/install/fil
esystem.squashfs
    kernel=xcat/osimage/ubuntu14.04.1-ppc64el-install-compute/vmlinuz
    mac=40:f2:e9:d3:85:90
    memory=261379MB
    mgt=ipmi
    mtm=8247-42L
    netboot=petitboot
    nfsserver=10.1.0.1
    os=ubuntu14.04.1
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    primarynic=mac
    profile=compute
    provmethod=ubuntu14.04.1-ppc64el-install-compute
    serial=211E49A
    serialport=0
    serialspeed=115200
    status=standingby
    statustime=01-19-2015 21:34:47
    supportedarchs=ppc64
    tftpserver=10.1.0.1

# tail /var/lib/dhcp/dhcpd.leases
<...>
host p8r4n2 {
  dynamic;
  hardware ethernet 40:f2:e9:d3:85:90;
  fixed-address 10.1.4.2;
        supersede server.ddns-hostname = "p8r4n2";
        supersede host-name = "p8r4n2";
        supersede server.next-server = 0a:01:00:01;
        supersede conf-file = "http://10.1.0.1/tftpboot/petitboot/p8r4n2";
}

# cat /tftpboot/petitboot/p8r4n2
#install ubuntu14.04.1-ppc64el-compute
default xCAT
label xCAT
        kernel
http://10.1.0.1:80/tftpboot/xcat/osimage/ubuntu14.04.1-ppc64el-install-compute/vml
inuz
        initrd
http://10.1.0.1:80/tftpboot/xcat/osimage/ubuntu14.04.1-ppc64el-install-compute/ini
trd.img
        append "nofb utf8 auto url=http://10.1.0.1/install/autoinst/p8r4n2
xcatd=10.1.0.1 mirror/http/hostname=10.1.0.1
netcfg/choose_interface=40:f2:e9:d3:85:90 console=tty0 console=hvc0,115200
locale=en_US priority=critical hostname=p8r4n2
live-installer/net-image=http://10.1.0.1/install/ubuntu14.04.1/ppc64el/install/fil
esystem.squashfs"
```

3.7.7 Start the operating system provisioning

After the operating system image is already set, the operating system provisioning starts automatically when a node boots, as specified in the DHCP network/boot configuration of that node.

You can power on or power cycle (reset) a node with the **rpower** command:

```
# rpower p8r4n2 on
# rpower p8r4n2 reset
```

You can watch a node's operating system provisioning on its console by using the **rcons** command (Example 3-50 on page 70).

Example 3-50 xCAT: Watching the OS provisioning of a node with the rcons command

```
# rcons p8r4n2

<...>
Petitboot (dev.20141013)                                     8247-42L 211E49A
 ????????????????????????????????????????????????????????????????????????????????
[Network: eth0 / 40:f2:e9:d3:85:90]
* xCAT

  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 10 sec: xCAT
<...>

  ??????????????????????? Finishing the installation ????????????????????????
  ?                                                                          ?
  ?                                  95%                                     ?
  ?                                                                          ?
The system is going down NOW!                                                ?
Sent SIGTERM to all processes                                                ?
Sent SIGKILL to all processes????????????????????????????????????????????????
Requesting system reboot
[ 2563.357250] reboot: Restarting system
<...>
```

After the operating system provisioning finishes, the system automatically reboots. On the node's console, the boot configuration that is provided by the xCAT management node is now different (Example 3-51) as a result of a diskful/stateful operating system provisioning. No `xCAT` entry exists to boot from network (therefore allowing the bootloader to boot from disk). Instead, entries exist for the Ubuntu operating system that is installed on a disk. These entries are obtained from the bootloader (`grub2`) configuration by *petitboot*. The default entry is automatically booted, which leads to the node's login prompt.

Example 3-51 xCAT: Shows the petitboot entries and boot for diskful/stateful OS provisioning

```
<...>
Petitboot (dev.20141013)                                     8247-42L 211E49A
 ????????????????????????????????????????????????????????????????????????????????
  [Disk: sdd2 / 0ec3b12c-2b85-4418-8c29-2671fbda657a]
    Ubuntu, with Linux 3.13.0-32-generic (recovery mode)
    Ubuntu, with Linux 3.13.0-32-generic
    Ubuntu







  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!
Info: Booting in 10 sec: Ubuntu
<...>
Sent SIGTERM to all processes
Sent SIGKILL to all processes
[    0.000000] OPAL V3 detected !
[    0.000000] Using PowerNV machine description
<...>

Ubuntu 14.04.1 LTS p8r4n2 hvc0

p8r4n2 login:
```

You can check connectivity from the xCAT management node to the provisioned node, which includes network connectivity and SSH functionality, by running a program with the **xdsh** command (Example 3-52).

Example 3-52 xCAT: Checking connectivity to the provisioned node with the xdsh command

```
# xdsh p8r4n2 uptime
p8r4n2: Warning: Permanently added 'p8r4n2,10.1.4.2' (ECDSA) to the list of known 
hosts.
p8r4n2:  19:27:02 up 2 min,  0 users,  load average: 0.15, 0.13, 0.05
```

You can verify the changes to the node object definition and boot configuration (Example 3-53).

Example 3-53 xCAT: Verifying changes to the node object definition and boot configuration

```
# lsdef p8r4n2
Object name: p8r4n2
    <...>
    currstate=boot
    <...>

# cat /tftpboot/petitboot/p8r4n2
#boot
```

3.8 xCAT compute nodes: Software stack

This section describes how to install the software stack (other than the operating system) in the xCAT compute nodes from the xCAT management node.

3.8.1 Configure nodes for access to the Internet

The installation process of several components of the software stack requires access to the Internet to install packages, for example, the Mellanox OFED for Linux, from the Ubuntu package repositories that are not available in the installation media. Access to the Internet can also be used to install package updates to the Ubuntu operating system.

You can configure the xCAT cluster to access the Internet in either one of the following ways:

- Through the management node: The Internet connection of the management node is shared by the compute nodes through network address translation (NAT). All of the Internet-related network traffic goes through the management node.
- Through the secondary network interface: Each compute node has an additional (also known as *secondary*) network interface, which can access the Internet directly, for example, on a different/site network.

The main difference between the two methods is the configuration of the network *gateway*: Either the management node (for NAT) or the site gateway (for Secondary Network Interface).

The following sections provide the instructions for each method:

- Internet access through the management node
- "Internet access through the secondary network interface" on page 74

Internet access through the management node

First, ensure that the management network has the `gateway` attribute set to the management node (the default setting is the `<xcatmaster>` alias to the management node). If any change is required, it is necessary to update the DHCP server configuration (reflected in the `routers` option). This operation is demonstrated in Example 3-54.

Example 3-54 xCAT: Configuring and verifying the gateway attribute of the management network

```
# chdef -t network 10_1_0_0-255_255_0_0 gateway='<xcatmaster>'
1 object definitions have been created or modified.

# lsdef -t network 10_1_0_0-255_255_0_0 -i gateway
Object name: 10_1_0_0-255_255_0_0
    gateway=<xcatmaster>

# makedhcp -n
# makedhcp -a
# service isc-dhcp-server restart

# 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
```

On the compute nodes, the gateway setting is reflected in the default network route. You can verify the default network route with the **ip** command in the compute nodes. The DHCP network configuration must be refreshed in the compute nodes that acquired the previous configuration with a reboot or with the **ifdown** and **ifup** commands (Example 3-55).

Example 3-55 xCAT: Refreshing the DHCP network configuration on a compute node

```
The default network route still is not set (no output):

# xdsh p8r4n2 'ip route list | grep default'
#

Restart all network interfaces with the ifdown/ifup commands:

# xdsh p8r4n2 -s 'bash -c "ifdown -a; ifup -a"'
<...>
p8r4n2: DHCPRELEASE on eth0 to 10.1.0.1 port 67 (xid=0x5bcdb97e)
<...>
p8r4n2: DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3
(xid=0x5fcece70)
p8r4n2: DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3
(xid=0x5fcece70)
p8r4n2: DHCPREQUEST of 10.1.4.2 on eth0 to 255.255.255.255 port 67
(xid=0x5fcece70)
p8r4n2: DHCPOFFER of 10.1.4.2 from 10.1.0.1
p8r4n2: DHCPACK of 10.1.4.2 from 10.1.0.1
<...>
p8r4n2: bound to 10.1.4.2 -- renewal in 17414 seconds.

The default network route is now set (IP address of the management node):
# xdsh p8r4n2 'ip route list | grep default'
p8r4n2: default via 10.1.0.1 dev eth0
```

Now, the compute nodes' DHCP network configuration, which points to the management node as the network gateway, is up-to-date.

To configure NAT on the management node, first enable IPv4 Forwarding in the `/etc/sysctl.conf` file (Example 3-56).

Example 3-56 xCAT: Verifying and enabling IPv4 Forwarding in the sysctl.conf file

```
Verify if the net.ipv4.ip_forward option is enabled (1) and uncommented (no
heading # in the line):
# grep net.ipv4.ip_forward /etc/sysctl.conf
net.ipv4.ip_forward = 1

In this case, it is enabled and uncommented.
If it's not, you can add it with:
# echo 'net.ipv4.ip_forward=1' >> /etc/sysctl.conf

Activate the new setting:
# sysctl -p
```

Then, add NAT rules for the iptables firewall in `/etc/rc.local` (Example 3-57) for translating network addresses between the xCAT management node, which is defined in the `mgmt_net` variable, and the Internet network interface (gateway on the management node). The `rc.local` script is run automatically during the boot.

Example 3-57 xCAT: Adding iptables firewall rules in rc.local

```
Temporarily remove the last line (“exit 0”), which exits the script:
# sed '/exit 0/ d' -i /etc/rc.local

Insert the iptables rules for NAT in /etc/rc.local
Note: EOF is quoted.
# cat >> /etc/rc.local <<'EOF'
# xCAT: NAT for management network/CIDR-netmask & internet network interface.
mgmt_net='10.1.0.0/16'
internet_if="$(ip route list | awk '/^default/ { print $5 }')"

if [ -n "$mgmt_net" ] && [ -n "$internet_if" ]
then
   iptables -t nat -A POSTROUTING -s $mgmt_net -o $internet_if -j MASQUERADE
   iptables -A FORWARD -s $mgmt_net -o $internet_if -j ACCEPT
   iptables -A FORWARD -d $mgmt_net -i $internet_if -m state --state
ESTABLISHED,RELATED -j ACCEPT
fi
EOF

Put the last line (“exit 0”) back:
# echo 'exit 0' >> /etc/rc.local

Run the rc.local script manually, this time (it runs automatically during boot):
# service rc.local start
```

You can test Internet connectivity on the nodes with the **`ping`** command (Example 3-58).

Example 3-58 xCAT: Testing Internet connectivity on the nodes with the ping command

```
# xdsh p8r4n2 ping -c1 www.ibm.com
p8r4n2: PING <...> (23.206.157.66) 56(84) bytes of data.
p8r4n2: 64 bytes from <...> (23.206.157.66): icmp_seq=1 ttl=49 time=7.01 ms
p8r4n2:
p8r4n2: --- <...> ping statistics ---
p8r4n2: 1 packets transmitted, 1 received, 0% packet loss, time 0ms
p8r4n2: rtt min/avg/max/mdev = 7.016/7.016/7.016/0.000 ms
```

For more information about NAT configuration on Ubuntu Server 14.04 LTS, see the corresponding online documentation:

https://help.ubuntu.com/lts/serverguide/firewall.html#ip-masquerading

Internet access through the secondary network interface

A secondary or additional network interface consists of a network interface (on the compute nodes) that is configured for an xCAT network other than the management or service networks.

First, ensure that the management network does *not* have the `gateway` attribute set (the default setting is the `<xcatmaster>` alias to the management node). If any change is required, it is necessary to update the DHCP server configuration (reflected as the no *routers* option), then propagate the DHCP network configuration changes to the nodes that acquired the previous network configuration. This operation is similar to the operation that is described in "Internet access through the management node" on page 72 and is demonstrated in Example 3-59.

Example 3-59 xCAT: Disabling and verifying the gateway attribute of the management network

```
# chdef -t network 10_1_0_0-255_255_0_0 gateway=''
1 object definitions have been created or modified.

# lsdef -t network 10_1_0_0-255_255_0_0 -i gateway
Object name: 10_1_0_0-255_255_0_0
    gateway=

# makedhcp -n
# makedhcp -a
# service isc-dhcp-server restart

# grep '\(subnet\|routers\)' /etc/dhcp/dhcpd.conf
  subnet 10.1.0.0 netmask 255.255.0.0 {
  } # 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

Restart all network interfaces on the nodes:
# xdsh p8r4n2 -s 'bash -c "ifdown -a; ifup -a" '
<...>

Verify there is no default network route (no output) on the nodes:
# xdsh p8r4n2 'ip route list | grep default'
#
```

It is necessary to create another xCAT network (an object definition of type *network*) with the network configuration of the site or public network; it must have the `gateway` attribute defined. Example 3-60 describes the steps to create an Ethernet network with an arbitrary name, `eth1net` (an allusion to the `eth1` interface, but not a reference to it), address 9.114.39.0/24, and gateway address 9.114.39.254.

Example 3-60 xCAT: Creating and assigning an Ethernet xCAT network

```
# mkdef -t network -o eth1net net=9.114.39.0 mask=255.255.255.0
gateway=9.114.39.254
1 object definitions have been created or modified.
```

It is also necessary to assign the xCAT network to the compute nodes by defining several network-related attributes in the `nics` table or in the node object definitions.

Those network-related attributes are originally stored in the `nics` table in a per-row way. Each row contains the attributes of either a node or node group, and each row contains all of the xCAT networks and network interfaces that are associated with that node or node group. In several cases, a single attribute contains information from all network interfaces of the node or node group.

Alternatively, the attributes can be manipulated directly in the node object definitions, and they are automatically converted between the node object definition and the `nics` table. In this manner, you can manipulate the per network-interface attributes by using a *network-interface subattribute* (that is "`.<network-interface>`").

Note: To use regular expressions with the attributes from the `nics` table that contain information from multiple network interfaces (usually the attributes that can be manipulated in a per network-interface way, for example, the IP address), you are required to manually set the attribute by considering the multiple network interfaces (that is, not in an individual network-interface manner).

Consider the existing values of the network interfaces, which can use regular expressions or not, and add or modify only the appropriate values, for example, regular-expression prefix, field separators, variable values, and suffix, leaving any static values in the correct places in the regular-expression fields.

For more details and examples, see 3.8.3, "Mellanox InfiniBand" on page 79.

The `nics` table cannot have more than one row that matches any node (for example, a row for a particular node, and another row for its node group). When a match for a node is attempted, only one row is considered (the first match); any other row is not considered, therefore, its value is not used (as of xCAT version 2.9).

Note: The remainder of this section creates a row for an individual node (the example node, `p8r4n2`) that is part of a node group (`power8_nodes`) for a simple demonstration. However, consider the use of regular expressions in the node group, as described in 3.8.3, "Mellanox InfiniBand" on page 79.

The network is then assigned to the node group in the `eth1` network interface with the **`chdef`** command, and a static IP address is assigned to the example node `p8r4n2` (see the previous note). The **`makehosts`** and **`makedns`** commands propagate the configuration changes to the `/etc/hosts` file and the DNS server (Example 3-61). The following network-related attributes are used:

- `nictypes.<interface>`: The type of the xCAT network, for example, Ethernet or InfiniBand.
- `nicnetworks.<interface>`: The name of the xCAT network (as in the `nics` table).
- `nicips.<interface>`: The IP address in that xCAT network.
- Optional: `nichostnamesuffixes.<interface>`: Create aliases to the node's IP address in the xCAT network. These aliases are based on the node's *hostname* plus the specified suffixes (added to `/etc/hosts` and the DNS server).

For more information about the `nics` table and network-related attributes, see the manual page:

```
# man nics
```

Example 3-61 xCAT: Defining and checking network-related attributes

```
# chdef -t group power8_nodes \
   nictypes.eth1=Ethernet \
   nicnetworks.eth1=eth1net \
   nichostnamesuffixes.eth1="-eth1"

# chdef p8r4n2 nicips.eth1=9.114.39.106
# lsdef p8r4n2 -i
nictypes.eth1,nicnetworks.eth1,nichostnamesuffixes.eth1,nicips.eth1
Object name: p8r4n2
    nichostnamesuffixes.eth1=-eth1
    nicips.eth1=9.114.39.106
    nicnetworks.eth1=eth1net
    nictypes.eth1=Ethernet

# makehosts

# grep p8r4n2 /etc/hosts
10.1.4.2 p8r4n2 p8r4n2.cluster-xcat
9.114.39.106 p8r4n2-eth1 p8r4n2-eth1.cluster-xcat

# makedns -n
<...>
Handling p8r4n2-eth1 in /etc/hosts.
<...>
Completed updating DNS records.

# No DHCP changes implied. No makedhcp required.
```

To apply the network configuration changes to the nodes, you can use the **confignics** script.

Note: The **confignics** script needs to be modified to correctly apply the `gateway` attribute (as of xCAT version 2.9); this one-time procedure is described in Example 3-62.

Example 3-62 xCAT: Applying a fix to the confignics script for the gateway attribute

```
# sed \
    -e '/num_v4num=$5/ { h; s/num_v4num/str_v4gateway/p; x; s/5/6/ }' \
    -e '/network ${str_v4net}/ { p; s/net\(work\)\?/gateway/g }' \
    -e '/^[ \t]\+configipv4/ s/$num_ipv4_index/$str_gateway &/' \
    -i /install/postscripts/configeth
```

The **confignics** script must be added to the list of scripts to run automatically on boot (*postscripts*) and to optionally run manually for immediate effect (Example 3-63).

Example 3-63 xCAT: Using the confignics script to apply the networked configuration

```
Run the confignics script automatically during boot (the postscripts list):
# chdef p8r4n2 --plus postscripts=confignics

Run the confignics script for immediate effect:
# updatenode p8r4n2 --scripts confignics
p8r4n2: Sun Feb  8 20:04:08 EST 2015 Running postscript: confignics
p8r4n2: confignics on p8r4n2: config install nic:0, remove: 0, iba ports:
p8r4n2: eth1!9.114.39.106
p8r4n2: confignics on p8r4n2: call 'configeth eth1 9.114.39.106 eth1net|'
p8r4n2: configeth on p8r4n2: os type: debian
p8r4n2: configeth on p8r4n2: old configuration: 3: eth1: <BROADCAST,MULTICAST> mtu
1500 qdisc noop state DOWN group default qlen 1000
p8r4n2:     link/ether 40:f2:e9:d3:85:92 brd ff:ff:ff:ff:ff:ff
p8r4n2: configeth on p8r4n2: new configuration
p8r4n2:        9.114.39.106, 9.114.39.0, 255.255.255.0, 9.114.39.254
```

```
p8r4n2: configeth on p8r4n2: eth1 changed, modify the configuration files
p8r4n2: Postscript: confignics exited with code 0
p8r4n2: Running of postscripts has completed.
```

You can verify the network configuration on the nodes with the **ifconfig** command, and you can verify any aliases to the nodes with the **ping** command (Example 3-64).

Example 3-64 xCAT: Verifying the network configuration of the confignics script

```
# xdsh p8r4n2 "ifconfig eth1 | grep 'inet addr' "
p8r4n2:           inet addr:9.114.39.106  Bcast:9.114.39.255  Mask:255.255.255.0

# ping -c1 p8r4n2-eth1
PING p8r4n2-eth1 (9.114.39.106) 56(84) bytes of data.
64 bytes from p8r4n2-eth1 (9.114.39.106): icmp_seq=1 ttl=64 time=0.090 ms

--- p8r4n2-eth1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.090/0.090/0.090/0.000 ms
```

You can also verify the default network route (for Internet access) with the **ip** command; connectivity to the Internet can be verified with the **ping** command (Example 3-65).

Example 3-65 xCAT: Verifying access to the Internet with the ip and ping commands

```
# xdsh p8r4n2 'ip route list | grep default'
p8r4n2: default via 9.114.39.254 dev eth1

# xdsh p8r4n2 ping -c1 www.ibm.com
p8r4n2: PING <...> (23.206.157.66) 56(84) bytes of data.
p8r4n2: 64 bytes from <...> (23.206.157.66): icmp_seq=1 ttl=49 time=6.48 ms
p8r4n2:
p8r4n2: --- <...> ping statistics ---
p8r4n2: 1 packets transmitted, 1 received, 0% packet loss, time 0ms
p8r4n2: rtt min/avg/max/mdev = 6.482/6.482/6.482/0.000 ms
```

For more information about the configuration of secondary or additional network interfaces, see the xCAT online documentation at the following website:

http://sourceforge.net/p/xcat/wiki/Configuring_Secondary_Adapters

3.8.2 Check for errors in package updates

If the nodes have access to package updates (for example, through to the Internet, or a local package repository mirror), a particular package update can fail. This failure can cause problems to other package installations and updates, therefore preventing several software administration operations to complete. Therefore, we suggest that you check for errors in package updates before you proceed with any other package installation or update step, or other steps that can trigger it.

Errors in package updates are usually listed in the output of the **updatenode** command (described in more detail later in this chapter), which is extensively used to install and update software and to run xCAT scripts in the nodes.

You can check for errors in package updates with the **updatenode** command, during the execution of the **ospkgs** postscript (specifically, the **apt-get** command), as illustrated in Example 3-66.

Note: We suggest that you first perform this operation on a single node (not the node group) to identify any errors in package updates, and create a solution to the problem, if required. Only then proceed to more nodes, possibly repeating the commands to perform the updates and solve the problems.

For more information and resolution instructions for errors in package updates, see 3.9.4, “Software updates: Problems in package updates” on page 118.

Example 3-66 xCAT: Checking for errors in package updates with the updatenode command

```
# updatenode p8r4n2
<...>
p8r4n2: <...> Running postscript: ospkgs
p8r4n2: === apt-get -y upgrade
<...>
p8r4n2: *** <...> dpkg: error processing package <package> (--configure):
<...>
p8r4n2: Errors were encountered while processing:
p8r4n2: <package>
p8r4n2: E: Sub-process /usr/bin/dpkg returned an error code (1)
<...>
```

3.8.3 Mellanox InfiniBand

To configure Mellanox InfiniBand on the nodes, you must first install the Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux, which is also known as *MLNX_OFED* or *MOFED,* in the nodes. Then, provide the InfiniBand network configuration in the management node and compute nodes.

You can install the MOFED in the nodes by performing the following steps:

1. Download the MOFED:
   a. Go to the Mellanox website:

      http://mellanox.com

   b. Navigate to **Support/Education** → **InfiniBand/VPI Drivers** → **Mellanox OpenFabrics Enterprise Distribution for Linux (MLNX_OFED)** → **Download** → **version**[10] **2.3-2.0.0** → **Ubuntu** → **Ubuntu 14.04** → **ppc64le** → **tgz**.

   c. Download the `MLNX_OFED_LINUX-2.3-2.0.0-ubuntu14.04-ppc64le.tgz` file.

   d. Transfer it to the management node.

2. Configure the management node to provide the MOFED to the nodes:
   a. Extract and provide the MOFED in the `ofed` directory in the operating system image’s postinstallation `otherpkgs` directory:

```
# dir=/install/post/otherpkgs/ubuntu14.04.1/ppc64el
# mkdir -p $dir
# cd $dir
```

[10] The text mentions the version that was available as of this writing. Adopt more recent versions as available.

```
# tar xf /path/to/MLNX_OFED_LINUX-2.3-2.0.0-ubuntu14.04-ppc64le.tgz
# mv MLNX_OFED_LINUX-2.3-2.0.0-ubuntu14.04-ppc64le ofed
```

b. Copy the **mlnxofed_ib_install** script to the postinstallation scripts directory:

```
# cp /opt/xcat/share/xcat/ib/scripts/Mellanox/mlnxofed_ib_install
/install/postscripts/
```

3. Install the MOFED in the nodes:

a. Run the **mlnxofed_ib_install** script in the nodes with the **updatenode** command (Example 3-67).

Example 3-67 xCAT: Installing the MOFED in the nodes with the mlnxofed_ib_install script

```
# updatenode p8r4n2 --scripts mlnxofed_ib_install
p8r4n2: <...> Running postscript: mlnxofed_ib_install
p8r4n2: p8r4n2 's operating system is Ubuntu.
p8r4n2: If you want to install Mellanox_OFED in p8r4n2, p8r4n2 must have
ability to access public network.
p8r4n2: checking p8r4n2 's ability to access public network...........[OK]
<...>
p8r4n2: ARCH=powerpc perl -x mlnxofedinstall --without-fw-update
--without-32bit --force
p8r4n2: Log: /tmp/ofed.build.log
p8r4n2: Logs dir: /tmp/OFED.<number>.logs
<...>
p8r4n2: This program will install the MLNX_OFED_LINUX package on your
machine.
p8r4n2: Note that all other Mellanox, OEM, OFED, or Distribution IB packages
will be removed.
<...>
p8r4n2: Device (04:01:0):
p8r4n2:         0004:01:00.0 Network controller [0207]: Mellanox
Technologies MT27600 [Connect-IB]
<...>
p8r4n2:
p8r4n2: Installation passed successfully
p8r4n2: stop: Unknown instance:
p8r4n2: openibd start/running
p8r4n2: Postscript: mlnxofed_ib_install exited with code 0
p8r4n2: Running of postscripts has completed.
```

The InfiniBand network adapter is an additional or secondary network adapter. (See 3.4.7, "xCAT network adapters: Primary and secondary or additional" on page 34.)

The network configuration is stored in the `nics` table and applied by the **confignics** script. The `nics` table contains attributes that are applicable to the management node and compute nodes.

Note: The InfiniBand network adapter that was used contains two ports, therefore providing two network interfaces to the operating system, which are assigned to different xCAT networks. (See 3.4.10, "xCAT cluster scenario: Networks, IP addresses, and hostnames" on page 35.)

The instructions that are provided in this section contain steps for both the 1-port (one network interface) and 2-port (two network interfaces) cases.

You can provide the InifiniBand network configuration by following these steps:

1. Configure the xCAT networks for InfiniBand on the management node:

   a. Create the xCAT networks with the **mkdef** command, which is reflected in the `nics` table:

      - For one port:

        ```
        # mkdef -t network -o ib0net net=10.3.0.0 mask=255.255.0.0
        ```

      - For two ports (additionally):

        ```
        # mkdef -t network -o ib1net net=10.4.0.0 mask=255.255.0.0
        ```

   b. Define the xCAT networks attributes (except the IP address) for the nodes. You can define the per network-interface attributes by using a network-interface subattribute (that is, “.<*network-interface*>”).

      - For one port:

        ```
        # chdef -t group power8_nodes \
           nictypes.ib0=Infiniband \
           nicnetworks.ib0=ib0net \
           nichostnamesuffixes.ib0=-ib0
        ```

      - For two ports (additionally):

        ```
        # chdef -t group power8_nodes \
           nictypes.ib1=Infiniband \
           nicnetworks.ib1=ib1net \
           nichostnamesuffixes.ib1=-ib1
        ```

   c. Define the IP address in the xCAT networks for the nodes:

      Note: It is not possible to define the IP address attribute with regular expressions by using network-interface subattributes. You are required to define IP addresses at the attribute level, considering the values for all network interfaces of the node group, due to an incompatibility between the attribute format xCAT adopts for regular expressions and the IP addresses of network interfaces.

      - For one port:

        ```
        # chdef -t group power8_nodes \
        nicips='|p8r(\d+)n(\d+)|ib0!10.3.($1+0).($2+0)|'
        ```

      - For two ports (alternatively):

        ```
        # chdef -t group power8_nodes \
        nicips='|p8r(\d+)n(\d+)|ib0!10.3.($1+0).($2+0),ib1!10.4.($1+0).($2+0)|'
        ```

      For an additional or secondary network interface, for example, a site or public network, see “Internet access through the secondary network interface” on page 74:

      ```
      # chdef -t group power8_nodes \
      nicips='|p8r(\d+)n(\d+)|ib0!10.3.($1+0).($2+0),ib1!10.4.($1+0).($2+0),eth1!1
      0.5.($1+0).($2+0)|'
      ```

      You can also manipulate the `nics` table directly, instead of the object definitions:

      ```
      # chtab node=power8_nodes \
      nics.nicips='|p8r(\d+)n(\d+)|ib0!10.3.($1+0).($2+0),ib1!10.4.($1+0).($2+0),e
      th1!10.5.($1+0).($2+0)|'
      ```

      You can verify that the IP address attribute is correct for each network interface by checking a particular node (Example 3-68 on page 82).

Example 3-68 xCAT: Checking several network attributes of a node with the lsdef command

```
# lsdef p8r4n2 -i nictypes,nicnetworks,nichostnamesuffixes,nicips
Object name: p8r4n2
    nichostnamesuffixes.ib0=-ib0
    nichostnamesuffixes.eth1=-eth1
    nichostnamesuffixes.ib1=-ib1
    nicips.ib0=10.3.4.2
    nicips.ib1=10.4.4.2
    nicips.eth1=10.5.4.2
    nicnetworks.ib1=ib1net
    nicnetworks.eth1=eth1net
    nicnetworks.ib0=ib0net
    nictypes.ib0=Infiniband
    nictypes.eth1=Ethernet
    nictypes.ib1=Infiniband
```

d. Update the nodes' *hostname* and address information for name resolution with the **makehosts** and **makedns** commands (Example 3-69).

Example 3-69 xCAT: Updating the nodes' hostname and address

```
# makehosts

# grep p8r4n2 /etc/hosts
10.1.4.2 p8r4n2 p8r4n2.cluster-xcat
10.5.4.2 p8r4n2-eth1 p8r4n2-eth1.cluster-xcat p8r5n6-eth1
10.4.4.2 p8r4n2-ib1 p8r4n2-ib1.cluster-xcat
10.3.4.2 p8r4n2-ib0 p8r4n2-ib0.cluster-xcat

# makedns -n
<...>
Completed updating DNS records.
```

2. Configure the InfiniBand network interfaces on the nodes:

a. Run the **confignics** script in the nodes, passing the number of ports in the InfiniBand adapter in the **ibaports** parameter (Example 3-70).

Example 3-70 xCAT: Configuring the InfiniBand network interfaces in the nodes with confignics

```
# updatenode p8r4n2 --scripts 'confignics --ibaports=2'
p8r4n2: Sun Feb  8 20:50:27 EST 2015 Running postscript: confignics
--ibaports=2
p8r4n2: confignics on p8r4n2: config install nic:0, remove: 0, iba ports: 2
p8r4n2: ib0!10.3.4.2
p8r4n2: ib1!10.4.4.2
<...>
p8r4n2: confignics on p8r4n2: executed script: configib for nics: ib0,ib1,
ports: 2
p8r4n2: openibd start/running
p8r4n2: openibd start/running
p8r4n2: openibd start/running
p8r4n2: Postscript: confignics --ibaports=2 exited with code 0
p8r4n2: Running of postscripts has completed.
```

b. You can verify the IP addresses of the network interfaces in the nodes with the **ifconfig** command (Example 3-71 on page 83).

Example 3-71 xCAT: Verifying the IP addresses in the nodes with the ifconfig command

```
# xdsh p8r4n2 "ifconfig ib0 | grep 'inet addr' "
p8r4n2:          inet addr:10.3.4.2  Bcast:10.3.255.255  Mask:255.255.0.0

# xdsh p8r4n2 "ifconfig ib1 | grep 'inet addr' "
p8r4n2:          inet addr:10.4.4.2  Bcast:10.4.255.255  Mask:255.255.0.0
```

c. Add the **`configncs`** to the list of scripts to run automatically on boot (postscripts):

```
# chdef p8r4n2 --plus postscripts=configncs
```

For more information about the Mellanox InfiniBand network with xCAT, and secondary or additional network adapters/interfaces, see the following xCAT wiki pages:

- http://sourceforge.net/p/xcat/wiki/Managing_the_Mellanox_Infiniband_Network1
- http://sourceforge.net/p/xcat/wiki/Configuring_Secondary_Adapters

3.8.4 XL C/C++ compiler

The xCAT project provides a partial kit for the IBM XL C/C++ compiler. (See 3.4.8, “xCAT Software Kits” on page 34.) The partial kit is available in the following wiki page:

http://sourceforge.net/p/xcat/wiki/IBM_HPC_Software_Kits

First, add the kit to the management node, then install it in the nodes.

To add the kit to the management node, perform the following steps:

1. Download the partial kit:

```
# wget http://sourceforge.net/projects/xcat/files/kits/hpckits/2.9/
Ubuntu/ppc64_Little_Endian/xlc-13.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2
```

2. Convert the partial kit to a complete kit by providing the product’s distribution packages with the **`buildkit addpkgs`** command, which creates another tarball for the complete kit:

```
# ls -1 /path/to/xl-c-c++-media/*.deb
libxlc_13.1.1.0-141105_ppc64el.deb
libxlc-devel.13.1.1_13.1.1.0-141105_ppc64el.deb
libxlmass-devel.8.1.0_8.1.0.0-141027_ppc64el.deb
libxlsmp_4.1.0.0-141010_ppc64el.deb
libxlsmp-devel.4.1.0_4.1.0.0-141010_ppc64el.deb
xlc.13.1.1_13.1.1.0-141105_ppc64el.deb
xlc-license.13.1.1_13.1.1.0-141105_ppc64el.deb

# ls -1
xlc-13.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2

# buildkit addpkgs xlc-13.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir
/path/to/xl-c-c++-media/
Extracting tar file
/root/xcat-kits/xlc/xlc-13.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2. Please
wait.
<...>
Creating tar file /root/xcat-kits/xlc/xlc-13.1.1-0-ppc64el.tar.bz2.
Kit tar file /root/xcat-kits/xlc/xlc-13.1.1-0-ppc64el.tar.bz2 successfully
built.
```

```
# ls -1
xlc-13.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2
xlc-13.1.1-0-ppc64el.tar.bz2
```

3. Add the kit to the management node with the **`addkit`** command:

```
# addkit xlc-13.1.1-0-ppc64el.tar.bz2
Adding Kit xlc-13.1.1-0-ppc64el
Kit xlc-13.1.1-0-ppc64el was successfully added.

# lsdef -t kit
xlc-13.1.1-0-ppc64el  (kit)
```

You can examine the kit and kit components with the **`lsdef`**, **`lskit`**, and **`lskitcomp`** commands. The kit component of the IBM XL C/C++ compiler for compute nodes is `xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el` (Example 3-72).

Example 3-72 xCAT: Examining a software kit with the lsdef, lskit, and lskitcomp commands

```
# lsdef -t kit -l xlc-13.1.1-0-ppc64el
Object name: xlc-13.1.1-0-ppc64el
    basename=xlc
    description=XLC for Ubuntu
    kitdeployparams=xlc-13.1.1-0-ppc64el_xlc.env
    kitdir=/install/kits/xlc-13.1.1-0-ppc64el
    ostype=Linux
    release=0
    version=13.1.1

# lsdef -t kitcomponent
xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el  (kitcomponent)
xlc.license-compute-13.1.1-0-ubuntu-14.04-ppc64el  (kitcomponent)
xlc.rte-compute-13.1.1-0-ubuntu-14.04-ppc64el  (kitcomponent)

# lskit xlc-13.1.1-0-ppc64el

----------------------------------------------------
kit:
    basename=xlc
    description=XLC for Ubuntu
    kitdeployparams=xlc-13.1.1-0-ppc64el_xlc.env
    kitdir=/install/kits/xlc-13.1.1-0-ppc64el
    kitname=xlc-13.1.1-0-ppc64el
    ostype=Linux
    release=0
    version=13.1.1

kitrepo:
    kitname=xlc-13.1.1-0-ppc64el

kitrepodir=/install/kits/xlc-13.1.1-0-ppc64el/repos/xlc-13.1.1-0-ubuntu-14.04-ppc6
4el
    kitreponame=xlc-13.1.1-0-ubuntu-14.04-ppc64el
    osarch=ppc64el
    osbasename=ubuntu
    osmajorversion=14
    osminorversion=04
```

```
kitcomponent:
    basename=xlc.rte-compute
    description=XLC13 for runtime kitcomponent
    kitcompdeps=xlc.license-compute
    kitcompname=xlc.rte-compute-13.1.1-0-ubuntu-14.04-ppc64el
    kitname=xlc-13.1.1-0-ppc64el
    kitpkgdeps=libxlc,libxlsmp
    kitreponame=xlc-13.1.1-0-ubuntu-14.04-ppc64el
    release=0
    serverroles=compute
    version=13.1.1

kitcomponent:
    basename=xlc.compiler-compute
    description=XLC13 for compiler kitcomponent
    kitcompdeps=xlc.rte-compute
    kitcompname=xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el
    kitname=xlc-13.1.1-0-ppc64el

kitpkgdeps=xlc.13.1.1,libxlc-devel.13.1.1,libxlmass-devel.8.1.0,libxlsmp-devel.4.1
.0
    kitreponame=xlc-13.1.1-0-ubuntu-14.04-ppc64el
    release=0
    serverroles=compute
    version=13.1.1

kitcomponent:
    basename=xlc.license-compute
    description=XLC13 license kitcomponent
    kitcompname=xlc.license-compute-13.1.1-0-ubuntu-14.04-ppc64el
    kitname=xlc-13.1.1-0-ppc64el
    kitpkgdeps=xlc-license.13.1.1
    kitreponame=xlc-13.1.1-0-ubuntu-14.04-ppc64el
    release=0
    serverroles=compute
    version=13.1.1

# lskit xlc-13.1.1-0-ppc64el | grep kitcomp
kitcomponent:
    description=XLC13 for runtime kitcomponent
    kitcompdeps=xlc.license-compute
    kitcompname=xlc.rte-compute-13.1.1-0-ubuntu-14.04-ppc64el
kitcomponent:
    description=XLC13 for compiler kitcomponent
    kitcompdeps=xlc.rte-compute
    kitcompname=xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el
kitcomponent:
    description=XLC13 license kitcomponent
    kitcompname=xlc.license-compute-13.1.1-0-ubuntu-14.04-ppc64el

# lskitcomp xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el

----------------------------------------------------
kit:
    kitname=xlc-13.1.1-0-ppc64el

kitcomponent:
    basename=xlc.compiler-compute
    description=XLC13 for compiler kitcomponent
    kitcompdeps=xlc.rte-compute
    kitcompname=xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el
    kitname=xlc-13.1.1-0-ppc64el

kitpkgdeps=xlc.13.1.1,libxlc-devel.13.1.1,libxlmass-devel.8.1.0,libxlsmp-devel.4.1
.0
    kitreponame=xlc-13.1.1-0-ubuntu-14.04-ppc64el
    release=0
    serverroles=compute
    version=13.1.1
```

To install the kit to the nodes, perform the following steps:

1. Add the kit component and its dependencies to the operating system image that is provisioned to the nodes with the **addkitcomp** command:

```
# lsdef p8r4n2 -i provmethod
Object name: p8r4n2
    provmethod=ubuntu14.04.1-ppc64el-install-compute

# addkitcomp --adddeps -i ubuntu14.04.1-ppc64el-install-compute \
xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el
Assigning kit component xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el to
osimage ubuntu14.04.1-ppc64el-install-compute
Kit components xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el were added to
osimage ubuntu14.04.1-ppc64el-install-compute successfully
```

Optional tip: You can verify the compatibility of a kit component (and its other kit component dependencies, which can be listed by the **lskit** command) with an operating system image by using the **chkkitcomp** command.

Note: No spaces between the commas and the kit components in the command.

```
# chkkitcomp \
    -i ubuntu14.04.1-ppc64el-install-compute \
xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el,\
xlc.rte-compute-13.1.1-0-ubuntu-14.04-ppc64el,\
xlc.license-compute-13.1.1-0-ubuntu-14.04-ppc64el
Kit components
xlc.rte-compute-<...>,xlc.compiler-compute-<...>,xlc.license-compute-<...>
are compatible with osimage ubuntu14.04.1-ppc64el-install-compute
```

2. For new operating system provisions, the kit installation occurs automatically during provisioning. If the node is already provisioned, you can install the kit with the **updatenode** command by using the **otherpkgs** script (Example 3-73).

Example 3-73 xCAT: Installing an xCAT kit to a node with the updatenode command

```
# xdsh p8r4n2 xlc -qversion
p8r4n2: bash: xlc: command not found

# xdsh p8r4n2 xlC -qversion
```

```
p8r4n2: bash: xlC: command not found

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 xlc -qversion
p8r4n2: IBM XL C/C++ for Linux, V13.1.1 (5725-C73, 5765-J08)
p8r4n2: Version: 13.01.0001.0000

# xdsh p8r4n2 xlC -qversion
p8r4n2: IBM XL C/C++ for Linux, V13.1.1 (5725-C73, 5765-J08)
p8r4n2: Version: 13.01.0001.0000
```

You can observe the changes that relate to the kit component in the operating system image definition and xCAT files as demonstrated in Example 3-74.

Example 3-74 xCAT: Observing changes that relate to the kit component in the OS image definition

```
# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute
Object name: ubuntu14.04.1-ppc64el-install-compute
    imagetype=linux

kitcomponents=xlc.license-compute-13.1.1-0-ubuntu-14.04-ppc64el,xlc.rte-compute-13
.1.1-0-ubuntu-14.04-ppc64el,xlc.compiler-compute-13.1.1-0-ubuntu-14.04-ppc64el
    osarch=ppc64el
    osname=Linux
    osvers=ubuntu14.04.1
    otherpkgdir=/install/post/otherpkgs/ubuntu14.04.1/ppc64el

otherpkglist=/install/osimages/ubuntu14.04.1-ppc64el-install-compute/kits/KIT_DEPL
OY_PARAMS.otherpkgs.pkglist,/install/osimages/ubuntu14.04.1-ppc64el-install-comput
e/kits/KIT_COMPONENTS.otherpkgs.pkglist
    pkgdir=/install/ubuntu14.04.1/ppc64el
    pkglist=/opt/xcat/share/xcat/install/ubuntu/compute.pkglist
    profile=compute
    provmethod=install
    template=/opt/xcat/share/xcat/install/ubuntu/compute.tmpl


# ls -1 /install/post/otherpkgs/ubuntu14.04.1/ppc64el/
ofed
xlc-13.1.1-0-ubuntu-14.04-ppc64el

# cat
/install/osimages/ubuntu14.04.1-ppc64el-install-compute/kits/KIT_COMPONENTS.otherp
kgs.pkglist
xlc-13.1.1-0-ubuntu-14.04-ppc64el/xlc.compiler-compute
xlc-13.1.1-0-ubuntu-14.04-ppc64el/xlc.rte-compute
xlc-13.1.1-0-ubuntu-14.04-ppc64el/xlc.license-compute
```

3.8.5 XL Fortran compiler

The xCAT project provides a partial kit for the IBM XL Fortran compiler. (See 3.4.8, “xCAT Software Kits” on page 34.) The partial kit is available in the following wiki page:

http://sourceforge.net/p/xcat/wiki/IBM_HPC_Software_Kits

First, add the kit to the management node, then install it in the nodes, as described in the following steps:

1. Download the partial kit:

   ```
   # wget http://sourceforge.net/projects/xcat/files/kits/hpckits/2.9/
   Ubuntu/ppc64_Little_Endian/xlf-15.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2
   ```

2. Convert the partial kit to a complete kit:

   ```
   # ls -1 /path/to/xl-f-media/*.deb
   libxlf_15.1.1.0-141105_ppc64el.deb
   libxlf-devel.15.1.1_15.1.1.0-141105_ppc64el.deb
   libxlmass-devel.8.1.0_8.1.0.0-141027_ppc64el.deb
   libxlsmp_4.1.0.0-141010_ppc64el.deb
   libxlsmp-devel.4.1.0_4.1.0.0-141010_ppc64el.deb
   xlf.15.1.1_15.1.1.0-141105_ppc64el.deb
   xlf-license.15.1.1_15.1.1.0-141105_ppc64el.deb

   # ls -1
   xlf-15.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2

   # buildkit addpkgs xlf-15.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir
   /path/to/xl-f-media/
   Extracting tar file
   /root/xcat-kits/xlf/xlf-15.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2. Please
   wait.
   <...>
   Creating tar file /root/xcat-kits/xlf/xlf-15.1.1-0-ppc64el.tar.bz2.
   Kit tar file /root/xcat-kits/xlf/xlf-15.1.1-0-ppc64el.tar.bz2 successfully
   built.

   # ls -1
   xlf-15.1.1-0-ppc64el.NEED_PRODUCT_PKGS.tar.bz2
   xlf-15.1.1-0-ppc64el.tar.bz2
   ```

3. Add the kit to the management node:

   ```
   # addkit xlf-15.1.1-0-ppc64el.tar.bz2
   Adding Kit xlf-15.1.1-0-ppc64el
   Kit xlf-15.1.1-0-ppc64el was successfully added.
   ```

4. Add the kit component (`xlf.compiler-compute-15.1.1-0-ubuntu-14.04-ppc64el`) to the operating system image (`ubuntu14.04.1-ppc64el-install-compute`):

   ```
   # lsdef -t kitcomponent
   <...>
   xlf.compiler-compute-15.1.1-0-ubuntu-14.04-ppc64el  (kitcomponent)
   xlf.license-compute-15.1.1-0-ubuntu-14.04-ppc64el  (kitcomponent)
   xlf.rte-compute-15.1.1-0-ubuntu-14.04-ppc64el  (kitcomponent)
   ```

```
# addkitcomp --adddeps -i ubuntu14.04.1-ppc64el-install-compute
xlf.compiler-compute-15.1.1-0-ubuntu-14.04-ppc64el
Assigning kit component xlf.compiler-compute-15.1.1-0-ubuntu-14.04-ppc64el to
osimage ubuntu14.04.1-ppc64el-install-compute
Kit components xlf.compiler-compute-15.1.1-0-ubuntu-14.04-ppc64el were added to
osimage ubuntu14.04.1-ppc64el-install-compute successfully
```

5. For any already provisioned nodes, you can install the kit with the **updatenode** command by using the **otherpkgs** script:

```
# xdsh p8r4n2 xlf -qversion
p8r4n2: bash: xlf: command not found

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 xlf -qversion
p8r4n2: IBM XL Fortran for Linux, V15.1.1 (5725-C75, 5765-J10)
p8r4n2: Version: 15.01.0001.0000
```

3.8.6 Parallel Environment Runtime Edition (PE RTE)

The IBM Parallel Environment Runtime Edition (PE RTE) distribution media provides a complete kit. Any updates, which are also known as *program temporary fixes* (PTFs), are available in IBM Fix Central, and include a complete kit also (without the program license).

Note: At the time that this publication was written, a mandatory PTF, which is version 2.1.0.1, was available for PE RTE version 2.1.0.0.

The kit component with the program license is only available in the kit from the distribution media; the kits from PTFs depend on that kit component only. Therefore, you are required to add both kits to the management node but to install only the kit from the PTF.

You can download the PE RTE PTF from IBM Fix Central with the following steps:

1. Go to IBM Fix Central at the following address:

   http://www.ibm.com/support/fixcentral

2. Select the following options:

   a. Product Group: **Cluster software**

   b. Cluster software: **Parallel Environment Runtime Edition**

   c. Installed Version: **2.1.0**

   d. Platform: **Linux 64-bit, pSeries**

3. Click **Continue**.
4. Click **Browse for fixes**.
5. Click **Continue**.
6. Select **fix pack: PERTE-2.1.0.1-power-Linux** (or any later fix pack).
7. Click **Continue**.
8. Proceed with authentication.

9. Download the `PERTE-2.1.0.1-power-Linux.tar.gz` file and transfer it to the management node. Or, you can copy the download link to download it in the management node.
   The file contains the PE RTE packages and the complete xCAT kit (Example 3-75).

Example 3-75 xCAT: Downloading and extracting the PE RTE PTF

```
# mkdir -p /path/to/pe-rte-ptf/
# cd /path/to/pe-rte-ptf/
# wget https://<...>.ibm.com/<...>/PERTE-2.1.0.1-power-Linux.tar.gz
# tar xf PERTE-2.1.0.1-power-Linux.tar.gz

# ls -1
PERTE-2.1.0.1-power-Linux.tar.gz
poe.linux.README
ppe-rte-2101-2.1.0.1-s001a.ppc64el.deb
pperte-2.1.0.1-s001a.ppc64el.deb
pperte-2.1.0.1-s001a-ppc64el.tar.bz2
ppe-rte-man-2101-2.1.0.1-s001a.ppc64el.deb
pperteman-2.1.0.1-s001a.ppc64el.deb
ppe-rte-samples-2101-2.1.0.1-s001a.ppc64el.deb
ppertesamples-2.1.0.1-s001a.ppc64el.deb
```

To install the PE RTE license and PTF to the compute nodes, perform the following steps:

1. Add the kit, which includes the kit component with the program license, from the distribution media:

```
# addkit /path/to/pe-rte-media/pperte-2.1.0.0-1445a-ppc64el.tar.bz2
Adding Kit pperte-2.1.0.0-1445a-ppc64el
Kit pperte-2.1.0.0-1445a-ppc64el was successfully added.

# lsdef -t kit
pperte-2.1.0.0-1445a-ppc64el  (kit)
<...>

# lsdef -t kitcomponent
min-pperte-compute-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-compute-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-license-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-login-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
<...>
```

2. Add the kit from the PTF, providing identical kit components for both PE RTE kits (two different versions) except for the program license:

```
# addkit pperte-2.1.0.1-s001a-ppc64el.tar.bz2
Adding Kit pperte-2.1.0.1-s001a-ppc64el
Kit pperte-2.1.0.1-s001a-ppc64el was successfully added.

# lsdef -t kit
pperte-2.1.0.0-1445a-ppc64el  (kit)
pperte-2.1.0.1-s001a-ppc64el  (kit)
<...>

# lsdef -t kitcomponent
min-pperte-compute-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
min-pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el  (kitcomponent)
```

```
pperte-compute-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-license-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-login-2.1.0.0-1445a-ubuntu-14.04-ppc64el  (kitcomponent)
pperte-login-2.1.0.1-s001a-ubuntu-14.04-ppc64el  (kitcomponent)
<...>
```

3. The kit component in the PTF kit has a dependency on a license kit component, which is satisfied by the kit from the distribution media:

```
# lsdef -t kitcomponent pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el -i
kitcompdeps
Object name: pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el
    kitcompdeps=pperte-license
```

4. Add the kit component from the PTF in the operating system image:

```
# addkitcomp --adddeps -i ubuntu14.04.1-ppc64el-install-compute
pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el
Assigning kit component pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el to
osimage ubuntu14.04.1-ppc64el-install-compute
Kit components pperte-compute-2.1.0.1-s001a-ubuntu-14.04-ppc64el were added to
osimage ubuntu14.04.1-ppc64el-install-compute successfully
```

5. For any already provisioned nodes, you can install the kit with the **updatenode** command by using the **otherpkgs** script:

```
# xdsh p8r4n2 ls -ld '/opt/ibmhpc/pe*'
p8r4n2: ls: cannot access /opt/ibmhpc/pe*: No such file or directory

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 ls -ld '/opt/ibmhpc/pe*'
p8r4n2: drwxr-xr-x <...> /opt/ibmhpc/pe2100
p8r4n2: drwxr-xr-x <...> /opt/ibmhpc/pe2101
p8r4n2: lrwxrwxrwx <...> /opt/ibmhpc/pecurrent -> /opt/ibmhpc/pe2101/
p8r4n2: lrwxrwxrwx <...> /opt/ibmhpc/pelatest -> /opt/ibmhpc/pe2101/
```

6. Verify that the PE RTE configuration also points to the PTF version in the `ppe.cfg` file:

```
# xdsh p8r4n2 cat /etc/ppe.cfg
p8r4n2: PE_LATEST_LEVEL: /opt/ibmhpc/pe2101
p8r4n2: DEFAULT_PE_BASE: /opt/ibmhpc/pe2101
p8r4n2: PE_RELEASE_2101: /opt/ibmhpc/pe2101
p8r4n2: PE_SECURITY_METHOD: SSH poesec_ossh
/opt/ibmhpc/pecurrent/base/gnu/lib64/poesec_ossh.so
m[t=-1,v=1.0.0,l=/lib/powerpc64le-linux-gnu/libcrypto.so.1.0.0]
```

If you want to change PE RTE versions (for example, for further updates, debugging, or testing), we suggest that you use the **pelinks** command, as described in PE RTE documentation, for example, with a hypothetical version 2.1.0.2:

```
# xdsh p8r4n2 pelinks 2102
```

However, the **pelinks** command requires the Korn Shell. It is provided by the *ksh* package, which is available in the Ubuntu Server installation media (therefore through the xCAT management node), although it is not installed by default.

To install the ksh package, you can perform either one of the following two procedures:

- Add the ksh package to the default package list, and run the **updatenode** command with the **ospkgs** script (Example 3-76).
- Add the ksh package to another package list, add it to the operating system image object definition, and run the **updatenode** command with the **otherpkgs** script (Example 3-77).

Example 3-76 xCAT: Installing the ksh package with the ospkgs script

```
# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i pkglist
Object name: ubuntu14.04.1-ppc64el-install-compute
    pkglist=/opt/xcat/share/xcat/install/ubuntu/compute.pkglist

# cat /opt/xcat/share/xcat/install/ubuntu/compute.pkglist
openssh-server
ntp
gawk
nfs-common
snmpd

# echo ksh >> /opt/xcat/share/xcat/install/ubuntu/compute.pkglist

# updatenode p8r4n2 --scripts ospkgs
p8r4n2: <...> Running postscript: ospkgs
p8r4n2: === apt-get -y upgrade
<...>
p8r4n2: ===  apt-get -q -y --force-yes install  openssh-server
p8r4n2: ntp
p8r4n2: gawk
p8r4n2: nfs-common
p8r4n2: snmpd
p8r4n2: ksh
<...>
p8r4n2: The following NEW packages will be installed:
p8r4n2:   ksh
<...>
p8r4n2: update-alternatives: using /bin/ksh93 to provide /bin/ksh (ksh) in auto
mode
p8r4n2: Postscript: ospkgs exited with code 0
```

Example 3-77 xCAT: Installing the ksh package with the otherpkgs script

```
# list='/install/custom/ubuntu14.04.1-ppc64el-install-compute/otherpkglist/
pe_rte.pkglist'
# mkdir -p $(dirname $list)
# echo ksh > $list
# chdef -t osimage ubuntu14.04.1-ppc64el-install-compute --plus
otherpkglist=$list
1 object definitions have been created or modified.

# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i otherpkglist
Object name: ubuntu14.04.1-ppc64el-install-compute

otherpkglist=/install/osimages/ubuntu14.04.1-ppc64el-install-compute/kits/KIT_D
EPLOY_PARAMS.otherpkgs.pkglist,/install/osimages/ubuntu14.04.1-ppc64el-install-
```

```
compute/kits/KIT_COMPONENTS.otherpkgs.pkglist,/install/custom/ubuntu14.04.1-ppc
64el-install-compute/otherpkglist/pe_rte.pkglist

# xdsh p8r4n2 ksh --version
p8r4n2: bash: ksh: command not found

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 ksh --version
p8r4n2:   version         sh (AT&T Research) 93u+ 2012-08-01
```

3.8.7 Parallel Environment Developer Edition (PE DE)

The IBM Parallel Environment Developer Edition (PE DE) distribution media provides a complete kit. You can install PE DE in the nodes with the following steps:

1. Add the kit to the management node:

```
# addkit /path/to/pe-de-media/ppedev-2.1.0-0.tar.bz2
Adding Kit ppedev-2.1.0-0
Kit ppedev-2.1.0-0 was successfully added.

# lsdef -t kit
ppedev-2.1.0-0  (kit)
<...>
```

2. Add the kit component to the operating system image:

```
# lskit ppedev-2.1.0-0 | grep kitcompname
    kitcompname=ppedev-compute-2.1.0-0-ubuntu-14.04-ppc64el
    kitcompname=ppedev.license-all-2.1.0-0-ubuntu-14.04-ppc64el
    kitcompname=ppedev-login-2.1.0-0-ubuntu-14.04-ppc64el

# addkitcomp --adddeps -i ubuntu14.04.1-ppc64el-install-compute
ppedev-compute-2.1.0-0-ubuntu-14.04-ppc64el
Assigning kit component ppedev-compute-2.1.0-0-ubuntu-14.04-ppc64el to osimage
ubuntu14.04.1-ppc64el-install-compute
Kit components ppedev-compute-2.1.0-0-ubuntu-14.04-ppc64el were added to
osimage ubuntu14.04.1-ppc64el-install-compute successfully
```

3. For any already provisioned nodes, you can install the kit with the **updatenode** command by using the **otherpkgs** script:

```
# xdsh p8r4n2 'ls -1d /opt/ibmhpc/ppedev*'
p8r4n2: ls: cannot access /opt/ibmhpc/ppedev*: No such file or directory

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 'ls -1d /opt/ibmhpc/ppedev*'
p8r4n2: /opt/ibmhpc/ppedev.hpct
p8r4n2: /opt/ibmhpc/ppedev.sci
```

3.8.8 Engineering and Scientific Subroutine Library (ESSL)

The IBM Engineering and Scientific Subroutine Library (ESSL) distribution media provides a complete kit.

You can install ESSL in the nodes with the following steps:

1. Add the kit to the management node:

```
# addkit /path/to/essl-media/essl-5.3.1-0-ppc64el.tar.bz2
Adding Kit essl-5.3.1-0-ppc64el
Kit essl-5.3.1-0-ppc64el was successfully added.
```

2. Add the kit component to the operating system image:

```
# lskit essl-5.3.1-0-ppc64el | grep kitcompname
    kitcompname=essl-loginnode-5.3.1-0-ubuntu-14.04-ppc64el
    kitcompname=essl-license-5.3.1-0-ubuntu-14.04-ppc64el
    kitcompname=min-essl-compute-5.3.1-0-ubuntu-14.04-ppc64el
    kitcompname=essl-compute-5.3.1-0-ubuntu-14.04-ppc64el

# addkitcomp --adddeps -i ubuntu14.04.1-ppc64el-install-compute
essl-compute-5.3.1-0-ubuntu-14.04-ppc64el
Assigning kit component essl-compute-5.3.1-0-ubuntu-14.04-ppc64el to osimage
ubuntu14.04.1-ppc64el-install-compute
Kit components essl-compute-5.3.1-0-ubuntu-14.04-ppc64el were added to osimage
ubuntu14.04.1-ppc64el-install-compute successfully
```

3. For any already provisioned nodes, you can install the kit with the **updatenode** command by using the **otherpkgs** script:

```
# xdsh p8r4n2 ls -1 /opt/ibmmath/essl
p8r4n2: ls: cannot access /opt/ibmmath/essl: No such file or directory

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 ls -1 /opt/ibmmath/essl
p8r4n2: 5.3
```

3.8.9 Parallel Engineering and Scientific Subroutine Library (PESSL)

The IBM Parallel Engineering and Scientific Subroutine Library (PESSL) distribution media provides a complete kit. Its kit components depend on the kit components from Parallel Environment Runtime Edition (PE RTE), which was described in 3.8.6, "Parallel Environment Runtime Edition (PE RTE)" on page 89.

You can install PESSL in the nodes with the following steps:

1. Add the kit to the management node:

```
# addkit /path/to/pessl-media/pessl-5.1.0-0-ppc64el.tar.bz2
Adding Kit pessl-5.1.0-0-ppc64el
Kit pessl-5.1.0-0-ppc64el was successfully added.
```

2. Add the kit component to the operating system image:

```
# lskit pessl-5.1.0-0-ppc64el | grep kitcompname
    kitcompname=pessl-compute-5.1.0-0-ubuntu-14.04-ppc64el
    kitcompname=min-pessl-compute-5.1.0-0-ubuntu-14.04-ppc64el
    kitcompname=pessl-loginnode-5.1.0-0-ubuntu-14.04-ppc64el
    kitcompname=pessl-license-5.1.0-0-ubuntu-14.04-ppc64el

# addkitcomp --adddeps -i ubuntu14.04.1-ppc64el-install-compute
pessl-compute-5.1.0-0-ubuntu-14.04-ppc64el
Assigning kit component pessl-compute-5.1.0-0-ubuntu-14.04-ppc64el to osimage
ubuntu14.04.1-ppc64el-install-compute
Kit components pessl-compute-5.1.0-0-ubuntu-14.04-ppc64el were added to osimage
ubuntu14.04.1-ppc64el-install-compute successfully
```

3. For any already provisioned nodes, you can install the kit with the **updatenode** command by using the **otherpkgs** script:

```
# xdsh p8r4n2 ls -1 /opt/ibmmath/pessl
p8r4n2: ls: cannot access /opt/ibmmath/pessl: No such file or directory

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: Sun Jan 25 06:44:55 EST 2015 Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 ls -1 /opt/ibmmath/pessl
p8r4n2: 5.1
```

3.8.10 SDK, Java Technology Edition (IBM Java)

The Java Standard Edition (SE) platform is a dependency for the installation of the Load Sharing Facility (LSF). The IBM SDK, Java Technology Edition (which is commonly referred to as IBM Java) for Linux (or IBM Developer Kit for Linux, Java Edition) is available at the following page:

http://www.ibm.com/developerworks/java/jdk/linux

No xCAT kit for IBM Java was available at the time that this publication was written. The following steps consist of copying the `ibm-java-ppc64le-sdk-7.1-2.0.bin` install file of IBM Java to a particular location in the management node and creating a shell script to download, install, and uninstall it in the nodes (Example 3-78):

1. Create the following directory in the management node, and copy the install file into it:

```
# dir=/install/custom/ubuntu14.04.1-ppc64el-install-compute/download
# mkdir -p $dir
# cp /path/to/ibm-java-ppc64le-sdk-7.1-2.0.bin $dir
```

2. Create the **ibm-java** xCAT script in the management node, according to the IBM Java install file and installation path. The installation path can be trivially derived from the name of the install file or obtained by running the installation until the installation directory prompt, as described in Example 3-78.

Example 3-78 xCAT: Creating the ibm-java xCAT script for installing or uninstalling IBM Java

```
Notice: EOF is quoted.

# cat > /install/postscripts/ibm-java <<"EOF"
#!/bin/bash

set -e
```

```
# Steps:
# 0) Check/Remove (option: -r) existing IBM Java
# 1) Download IBM Java install file
# 2) Run it (non-interactive: -i silent)
# 3) Configure environment variables (PATH,JAVA_HOME)

# IBM Java install file name
file='ibm-java-ppc64le-sdk-7.1-2.0.bin'

# IBM Java install path
path='/opt/ibm/java-ppc64le-71'

# IBM Java environment variables file
profile='/etc/profile.d/ibm-java.sh'

# Path in xCAT server (PROVMETHOD: osimage name; defined by xCAT)
server_path="/install/custom/$PROVMETHOD/download"

# IP of xCAT server (XCATSERVER: IP:PORT format; defined by xCAT)
server_ip="$(echo $XCATSERVER | cut -d: -f1)"

# Check existing installation
if [ -f "$path/jre/bin/java" ]
then

    if [ "$1" != '-r' ]
    then
        echo "ERROR. Existing installation found. Remove with $(basename $0)
-r, and re-run."
        exit 1
    fi

    echo "Uninstalling ($path)"

    # Attept proper uninstall, or fallback.
    uninstaller="$path/_uninstall/uninstall"
    if [ ! -x "$uninstaller" ] || ! $uninstaller
    then
        echo '.. removing files'
        rm -rf $path
        rm -f /var/.com.zerog.registry.xml
    fi

    # Remove profile and its source line
    rm -f $profile
    sed "/^source.*$(basename $profile)/d" -i ~/.bashrc

    echo 'Uninstall finished.'
    exit 0
fi

# Temp file
temp="/tmp/$file"

# Download
```

```
wget -nv $server_ip/$server_path/$file -O $temp

# Run (non-interactive)
chmod +x $temp
$temp -i silent

# Clean
rm -f $temp

# Configure env vars in profile
echo "export JAVA_HOME=$path" > $profile
echo 'export PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin' >> $profile

# Configure env vars also in non-interactive/login shells.
# (Ubuntu: insert source line before the exit command)
sed "/# If not running interactively/i source $profile" -i ~/.bashrc

# Verify current jre/jdk in non-interactive shell
source $profile
for command in java javac
do
    echo -n "$command: "
    if ! bash -c "which $command | grep '^$path'"
    then
        echo "ERROR. Not found."
        exit 1
    fi
done

EOF
```

3. Install IBM Java in the nodes with the **updatenode** command by using the **ibm-java** script that you created:

```
# xdsh p8r4n2 'java -fullversion; javac -fullversion'
p8r4n2: bash: java: command not found
p8r4n2: bash: javac: command not found

# updatenode p8r4n2 --scripts ibm-java
p8r4n2: <...> Running postscript: ibm-java
p8r4n2: <...>
URL:http://10.1.0.1//install/custom/ubuntu14.04.1-ppc64el-install-compute/downl
oad/ibm-java-ppc64le-sdk-7.1-2.0.bin [129032176/129032176] ->
"/tmp/ibm-java-ppc64le-sdk-7.1-2.0.bin" [1]
p8r4n2: strings: '/lib/libc.so.6': No such file
p8r4n2: java: /opt/ibm/java-ppc64le-71/bin/java
p8r4n2: javac: /opt/ibm/java-ppc64le-71/bin/javac
p8r4n2: Postscript: ibm-java exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 'java -fullversion; javac -fullversion'
p8r4n2: java full version JRE 1.7.0 IBM Linux build
pxl6470_27sr2-20141101_01(SR2)
p8r4n2: javac full version "1.7.0-foreman_2014_10_04_12_30-b00"
```

3.8.11 IBM Platform Load Sharing Facility (LSF)

No xCAT kit was available in the IBM Platform Load Sharing Facility (LSF) at the time that this publication was written. Two xCAT scripts exist for handling the LSF installation and configuration in the nodes, if you are provided the installation tarballs from the LSF distribution media.

For more information about the Platform LSF and its installation, and configuration, see the following IBM Knowledge Center page and go to **Products: Platform Computing** → **Platform LSF** → **Platform LSF 9.1.3** (the version that was used during the development of this publication):

http://www.ibm.com/support/knowledgecenter/

Considerations, requirements, and implementation

The following considerations and requirements apply to the installation and configuration of LSF:

- The LSF destination path, which is also known as `TOP` dir, is shared by all compute nodes. Therefore, it must be available at the same mount point for all compute nodes.
- The LSF installation is performed by only one compute node, which installs LSF in the destination path that is shared by all compute nodes. Therefore, the installation files must be available to that compute node.

To address these points, we used the following implementation:

- The management node exports the files through the Network File System (NFS) to the nodes. (Other network file systems can be used instead, but for simplicity and demonstration, we used the NFS.)
  - The LSF top directory is `/usr/share/lsf/top`.
  - The LSF distribution directory for the installation files is `/usr/share/lsf/distrib`.
  - The NFS export directory is `/usr/share/lsf`.
- All nodes mount the NFS exports from the management node at the same mount point.

  The directory path is the same across the management node (with local contents) and compute nodes (with file system mount points over NFS).

The following xCAT scripts are used in the xCAT-specific procedure:

- The **`install_lsf`** installation script can run in one node.
- The **`lsf_startup`** configuration script can run in all nodes, including the one node that is used for the installation.

Installation instructions

First, use the following steps to configure the NFS exports in the management node:

1. Create the LSF `top` and distribution (`distrib`) directories:

```
# mkdir -p /usr/share/lsf/top /usr/share/lsf/distrib
```

2. Export the LSF parent directory through NFS in the xCAT management network:

```
# cat <<EOF >>/etc/exports
/usr/share/lsf 10.1.0.0/255.255.0.0(rw,sync,no_root_squash,no_subtree_check)
EOF
# exportfs -ra
```

3. Configure all nodes to mount the LSF parent directory through NFS from the management node. Verify whether its mount point lists the `top` and `distrib` directories. (The **`xdsh`** command's `--stream` option is used for watching the remote command output. The shell's `-x` option is set for watching a more detailed script execution.)

   ```
   # xdsh p8r4n2 --stream 'set -x; dir=/usr/share/lsf; mkdir -p $dir && echo
   "10.1.0.1:$dir $dir nfs auto 0 0" >> /etc/fstab && mount $dir; ls -1 $dir'
   p8r4n2: + dir=/usr/share/lsf
   p8r4n2: + mkdir -p /usr/share/lsf
   p8r4n2: + echo '10.1.0.1:/usr/share/lsf /usr/share/lsf nfs auto 0 0'
   p8r4n2: + mount /usr/share/lsf
   p8r4n2: + ls -1d /usr/share/lsf
   p8r4n2: distrib
   p8r4n2: top
   <... xCAT variables setting>
   ```

To perform the LSF installation in one node, follow these steps:

1. Copy the LSF install files from tarballs (`lsf9.1.3_no_jre_lsfinstall.tar.Z` and `lsf9.1.3_lnx313-lib219-ppc64le.tar.Z`) and copy the LSF entitlement file (`platform_lsf_<edition>_entitlement.dat`) to the LSF distribution directory:

   ```
   # cd /path/to/lsf-media/
   # cp lsf9.1.3_no_jre_lsfinstall.tar.Z \
        lsf9.1.3_lnx313-lib219-ppc64le.tar.Z \
        platform_lsf_adv_entitlement.dat \
        /usr/share/lsf/distrib/
   ```

2. Create the LSF installation and configuration file (`install.config`) at the xCAT postscripts directory (`/install/postscripts`).

   For more information, see the following IBM Knowledge Center page and go to **Products: Platform Computing** → **Platform LSF** → **Platform LSF 9.1.3** (the version that was used for writing this book) → **Installing, Upgrading, and Migrating** → **install.config**:

   http://www.ibm.com/support/knowledgecenter/

   Example 3-79 demonstrates the commands to create the `install.config` file according to the following scenario:

   – LSF top directory: `/usr/share/lsf/top`
   – LSF distribution directory: `/usr/share/lsf/distrib`
   – LSF entitlement file: `platform_lsf_adv_entitlement.dat` (in the LSF distribution directory)
   – LSF administrator username: `lsfadmin`
   – LSF cluster name: `cluster-lsf`
   – LSF master/server nodes: `p8r1n1` and `p8r4n2`
   – LSF non-master/server nodes: All other nodes in the `power8_nodes` node group

   Example 3-79 xCAT: Creating the LSF install.config file with two master nodes and other nodes

   ```
   # master_nodes="p8r1n1 p8r4n2"

   # all_nodes="$(nodels power8_nodes)"
   # non_master_nodes="$(for node in $master_nodes $all_nodes; do echo $node; done
   | sort | uniq -u | tr '\n' ' ')"

   # cat > /install/postscripts/install.config <<EOF
   ```

```
LSF_TOP="/usr/share/lsf/top"
LSF_TARDIR="/usr/share/lsf/distrib"
LSF_ENTITLEMENT_FILE="/usr/share/lsf/distrib/platform_lsf_adv_entitlement.dat"
LSF_ADMINS="lsfadmin"
LSF_CLUSTER_NAME="cluster-lsf"
LSF_MASTER_LIST="$master_nodes"
LSF_ADD_SERVERS="$non_master_nodes"
EOF
```

3. Create the LSF administrator user (`lsfadmin`) on the nodes:

   ```
   # xdsh p8r4n2 'user="lsfadmin"; password="<lsfadmin-password>"; useradd
   --create-home --shell /bin/bash $user && echo "$user:$password" | chpasswd'
   ```

4. Run the xCAT script for the LSF installation in one node, for example, `p8r4n2`. All nodes in the `LSF_MASTER_LIST` and `LSF_ADD_SERVERS` must be accessible by that node.

   ```
   # updatenode p8r4n2 --scripts install_lsf
   p8r4n2: <...> Running postscript: install_lsf
   p8r4n2: LSF_TOP="/usr/share/lsf/top"
   p8r4n2: LSF_ADMINS="lsfadmin"
   p8r4n2: LSF_CLUSTER_NAME="cluster-lsf"
   p8r4n2: LSF_MASTER_LIST="<...>"
   p8r4n2:
   LSF_ENTITLEMENT_FILE="/usr/share/lsf/distrib/platform_lsf_adv_entitlement.dat"
   p8r4n2: LSF_TARDIR="/usr/share/lsf/distrib"
   p8r4n2: LSF_ADD_SERVERS="<...>"
   p8r4n2: INFO: We will untar the lsfinstall TAR file
   /usr/share/lsf/distrib/lsf9.1.3_no_jre_lsfinstall.tar.Z.
   <...>
   p8r4n2: INFO: Installation script DONE.
   p8r4n2: INFO: Updating LSF Cluster Configuration Files lsf.conf and lsb.hosts
   p8r4n2: Postscript: install_lsf exited with code 0
   p8r4n2: Running of postscripts has completed.
   ```

To perform the LSF configuration on all nodes, follow these steps:

1. The `lsf_startup` script requires a fix for an error in identifying the LSF directory, as of the writing of this book (xCAT 2.9).

   a. The error causes the following message:

      ```
      # updatenode p8r4n2 --scripts lsf_startup
      p8r4n2: <...< Running postscript: lsf_startup
      p8r4n2: INFO: Run hostsetup on each node.
      p8r4n2: ./lsf_startup: line 48: /usr/share/lsf/top//install/hostsetup: No
      such file or directory
      <...>
      ```

   b. This error can be resolved by changing the `/install/postscripts/lsf_startup` file:

      - Change the file from this line:

        ```
        LSF_VERSION=`find /$LSF_TOP -name hostsetup|head -1|awk '{print $5}'`
        ```

      - To this line:

        ```
        LSF_VERSION=`find /$LSF_TOP -name hostsetup|head -1|sed
        "s,.*/\([0-9.]\+\)/.*,\1,"`
        ```

- Or, you can perform the change with the following command:

  ```
  # sed '/^LSF_VERSION=/ s:awk.*:sed "s,.*/\\([0-9.]\\+\\)/.*,\\1,"\`: ' \
  -i /install/postscripts/lsf_startup
  ```

2. Run the **lsf_startup** script on all nodes (Example 3-80).

Example 3-80 xCAT: Running the lsf_startup script with the updatenode command

```
# updatenode p8r4n2 --scripts lsf_startup
p8r4n2: <...> Running postscript: lsf_startup
p8r4n2: INFO: Run hostsetup on each node.
p8r4n2: Logging installation sequence in /usr/share/lsf/top/log/Install.log
p8r4n2: /usr/share/lsf/top/9.1/install/instlib/lsflib.sh: line 2667:
/lib64/libc.so.6: No such file or directory
p8r4n2:
p8r4n2: ------------------------------------------------------------
p8r4n2:     L S F    H O S T S E T U P    U T I L I T Y
p8r4n2: ------------------------------------------------------------
p8r4n2: This script sets up local host (LSF server, client or slave)
environment.
p8r4n2: Setting up LSF server host "p8r4n2" ...
p8r4n2: /usr/share/lsf/top/9.1/install/instlib/lsflib.sh: line 2667:
/lib64/libc.so.6: No such file or directory
p8r4n2: Checking LSF installation for host "p8r4n4" ... Done
p8r4n2: /usr/share/lsf/top/9.1/install/instlib/lsflib.sh: line 2667:
/lib64/libc.so.6: No such file or directory
p8r4n2: Copying /etc/init.d/lsf, /etc/rc2.d/S95lsf and /etc/rc1.d/K05lsf
p8r4n2: Installing LSF RC scripts on host "p8r4n2" ... Done
p8r4n2: LSF service ports are defined in /usr/share/lsf/top/conf/lsf.conf.
p8r4n2: Checking LSF service ports definition on host "p8r4n2" ... Done
p8r4n2: You are installing IBM Platform LSF -  Advanced Edition.
p8r4n2: /usr/share/lsf/top/9.1/install/instlib/lsflib.sh: line 2667:
/lib64/libc.so.6: No such file or directory
p8r4n2:
p8r4n2: ... Setting up LSF server host "p8r4n2" is done
p8r4n2: ln: failed to create symbolic link '/usr/lib64/libpermapi.so': No such
file or directory
p8r4n2: ... LSF host setup is done.
p8r4n2: INFO: Set LSF environment for root and LSF_ADMINS
p8r4n2: INFO: Start LSF Cluster.
p8r4n2: Starting up LIM on <p8r4n2.cluster-xcat> ...... done
p8r4n2: Starting up RES on <p8r4n2.cluster-xcat> ...... done
p8r4n2: Starting up slave batch daemon on <p8r4n2.cluster-xcat> ...... done
p8r4n2: Postscript: lsf_startup exited with code 0
p8r4n2: Running of postscripts has completed.
```

3. You can verify the LSF cluster setup on a node by running the **lsclusters** command as the LSF administrator user:

   ```
   # xdsh p8r4n2 'su lsfadmin -l -c lsclusters'
   p8r4n2: CLUSTER_NAME   STATUS   MASTER_HOST              ADMIN    HOSTS
   SERVERS
   p8r4n2: cluster-lsf    ok       p8r4n2.cluster-xca     lsfadmin    <...> <...>
   <...>
   ```

3.8.12 IBM Spectrum Scale (formerly GPFS)

No xCAT kit was available for IBM Spectrum Scale at the time that this publication was written. To install Spectrum Scale in the xCAT compute nodes, this section uses the xCAT **`otherpkgs`** script (for packages that are not distributed with the Linux distribution).

Note: Throughout this section, Spectrum Scale is referred to as GPFS where it still applied at the time that this publication was written, for example, in the navigation steps in the IBM Knowledge Center and distribution package file names.

Considerations and scenario

You can configure a Spectrum Scale cluster in many ways based on the number of nodes, their roles, the considerations and requirements for data replication, throughput and latency, the number and types of licenses, and so on. Many commands are available for the installation and configuration procedures. For the related documentation, see the Spectrum Scale documentation in the IBM Knowledge Center:

- For more information about Spectrum Scale concepts and installation, see the following IBM Knowledge Center pages and go to **Products: Cluster software** → **General Parallel File System** → **General Parallel File System 4.1.0.4** (as of this writing) → **GPFS V4.1.0.4** → **GPFS Concepts, Planning, and Installation Guide**:

  `http://www.ibm.com/support/knowledgecenter/`

  See the following specific pages:

  – *Steps to establishing and starting your GPFS cluster*
  – *Installing GPFS on Linux nodes*

- For more information about Spectrum Scale commands, see the following IBM Knowledge Center page and go to **Products: Cluster software** → **General Parallel File System** → **General Parallel File System 4.1.0.4** (as of this writing) → **GPFS V4.1.0.4** → **GPFS Administration and Programming Reference** → **GPFS Commands**:

  `http://www.ibm.com/support/knowledgecenter/`

We used the following Spectrum Scale cluster scenario in this section:

- xCAT management node:
  – Node designation: Manager, quorum
  – Cluster configuration server: Primary
  – License: Server
  – Network Shared Disk (NSD): Provides one NSD to the cluster
  – Package installation steps: Manual for reference and demonstration purposes (for an automatic mode, use an xCAT compute node)
- xCAT compute nodes:
  – Node designation: Non-manager, non-quorum
  – Cluster configuration server: Non-primary
  – License: Client
  – Network Shared Disk (NSD): Mount the NSD that is provided by the management node
  – Package installation steps: Automatic (through the xCAT management node)

Instructions for the management node

Perform the tasks that relate to the role of the xCAT management node in the GPFS cluster. Install and update GPFS, build the GPFS Portability Layer (GPL), which is also known as the *port layer*, create the GPFS cluster, and export a disk as a Network Shared Disk (NSD).

Packages from the distribution media

To install the packages from the distribution media, perform the following steps:

1. Install the package dependencies for the GPFS base package and to build the GPFS portability layer:

   ```
   # apt-get install ksh libaio1 m4
   # apt-get install build-essential linux-headers-$(uname -r)
   ```

2. Configure the environment variables for running GPFS commands, including in non-interactive or non-login shells:

   ```
   # profile='/etc/profile.d/gpfs.sh'
   # echo 'export PATH=$PATH:/usr/lpp/mmfs/bin' > $profile
   # source $profile

   # sed "/# If not running interactively/i source $profile" -i ~/.bashrc
   ```

3. Install the packages from the distribution media with the **dpkg -i** command:

   ```
   # ls -1 /path/to/gpfs-media/*.deb
   gpfs.base_4.1.0-5_ppc64el.deb
   gpfs.crypto_4.1.0-5_ppc64el.deb
   gpfs.docs_4.1.0-5_all.deb
   gpfs.ext_4.1.0-5_ppc64el.deb
   gpfs.gpl_4.1.0-5_all.deb
   gpfs.gskit_8.0.50-32_ppc64el.deb
   gpfs.msg.en-us_4.1.0-5_all.deb

   # dpkg -i /path/to/gpfs-media/*.deb
   <...>
   ```

4. Verify that the packages are installed with the **dpkg -l** command (note version 4.1.0-5):

   ```
   # dpkg -l | grep gpfs
   ii  gpfs.base                      4.1.0-5                    ppc64el
   GPFS File Manager
   ii  gpfs.crypto                    4.1.0-5                    ppc64el
   GPFS Cryptographic Subsystem
   ii  gpfs.docs                      4.1.0-5                    all
   GPFS Server Manpages and Documentation
   ii  gpfs.ext                       4.1.0-5                    ppc64el
   GPFS Extended Features
   ii  gpfs.gpl                       4.1.0-5                    all
   GPFS Open Source Modules
   ii  gpfs.gskit                     8.0.50-32                  ppc64el
   GPFS GSKit Cryptography Runtime
   ii  gpfs.msg.en-us                 4.1.0-5                    all
   GPFS Server Messages - U.S. English
   ```

Packages from the product updates

To install the packages from product updates, perform the following steps:

1. Download the updates from IBM Fix Central:

   a. Go to IBM Fix Central:

      http://www.ibm.com/support/fixcentral/

   b. Navigate through the following sequence:

      i. Click the **Select product** tab.

      ii. From the Product Group list, select **Cluster software**.

      iii. From the Select from Cluster software list, select **General Parallel File System <*edition*> Edition**.

      iv. From the Installed Version list, select **4.1.0** (or the appropriate version).

      v. From the Platform list, select **Linux Power PC 64 Little Endian**.

      vi. Click **Continue**.

      vii. Select **Browse for fixes**.

      viii.Click **Continue**.

      ix. Select **fix pack: GPFS-<*edition/version*>-powerLE-Linux**. (Use the current version, which was `GPFS-4.1.0.6-powerLE-Linux.advanced.tar.gz`, as of the writing of this book.)

      x. Click **Continue**.

      xi. Proceed with authentication.

   c. Download the `GPFS-4.1.0.6-powerLE-Linux.advanced.tar.gz` file and transfer it to the management node. Or, you can copy the download link to download it in the management node and then extract it:

```
# mkdir -p /path/to/gpfs-update/
# cd /path/to/gpfs-update/
# wget
https://<...>.ibm.com/<...>/GPFS-4.1.0.6-powerLE-Linux.advanced.tar.gz
# tar xf GPFS-4.1.0.6-powerLE-Linux.advanced.tar.gz
```

   d. Check the list of versions on which the update can be applied in the `readme` file:

```
# cat README
<...>
Update to Version:

      4.1.0-6

Update from Version:

      4.1.0-0
      4.1.0-1
      4.1.0-2
      4.1.0-3
      4.1.0-4
      4.1.0-5
<...>
```

2. Install the packages from IBM Fix Central with the **dpkg -i** command:

```
# cd /path/to/gpfs-update/

# ls -1 *.deb
gpfs.base_4.1.0-6_ppc64el_update.deb
gpfs.docs_4.1.0-6_all.deb
gpfs.ext_4.1.0-6_ppc64el_update.deb
gpfs.gpl_4.1.0-6_all.deb
gpfs.gskit_8.0.50-32_ppc64el.deb
gpfs.msg.en-us_4.1.0-6_all.deb

# dpkg -i *.deb
<...>
```

3. Verify that several packages are updated with the **dpkg -l** command (note version 4.1.0-6):

```
# dpkg -l | grep gpfs
ii  gpfs.base                    4.1.0-6                      ppc64el
GPFS File Manager
ii  gpfs.crypto                  4.1.0-5                      ppc64el
GPFS Cryptographic Subsystem
ii  gpfs.docs                    4.1.0-6                      all
GPFS Server Manpages and Documentation
ii  gpfs.ext                     4.1.0-6                      ppc64el
GPFS Extended Features
ii  gpfs.gpl                     4.1.0-6                      all
GPFS Open Source Modules
ii  gpfs.gskit                   8.0.50-32                    ppc64el
GPFS GSKit Cryptography Runtime
ii  gpfs.msg.en-us               4.1.0-6                      all
GPFS Server Messages - U.S. English
```

GPFS portability layer

To build the GPFS portability layer, perform the following steps:

1. Build and install the GPFS portability layer (GPL):

   Five files are relevant: Three kernel modules and two utility executables.

```
# cd /usr/lpp/mmfs/src && make Autoconfig && make World && make InstallImages
<...>
        @/bin/mkdir -p $(INSTALL_MOD_PATH)/lib/modules/`cat $(MODPATH)`/extra
        @$(CP) mmfs26.ko $(INSTALL_MOD_PATH)/lib/modules/`cat $(MODPATH)`/extra
        @$(CP) mmfslinux.ko $(INSTALL_MOD_PATH)/lib/modules/`cat
$(MODPATH)`/extra
        @$(CP) tracedev.ko $(INSTALL_MOD_PATH)/lib/modules/`cat
$(MODPATH)`/extra
        @$(INSTALL) -c -m 0500 lxtrace $(LPPBIN)/lxtrace-`cat $(MODPATH)`
        @$(INSTALL) -c -m 0500 kdump $(LPPBIN)/kdump-`cat $(MODPATH)`
```

2. Ensure that the SSH key-based authentication is enabled from the management node to itself:

```
# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

GPFS cluster

To create the GPFS cluster, perform the following steps:

1. Create the GPFS cluster initially with *one* node, the xCAT management node, and the name `cluster-gpfs` with the **`mmcrcluster`** command:

   ```
   # mmcrcluster -N xcat-mn.cluster-xcat:manager-quorum -p xcat-mn.cluster-xcat -C
   cluster-gpfs
   mmcrcluster: Performing preliminary node verification ...
   mmcrcluster: Processing quorum and other critical nodes ...
   mmcrcluster: Finalizing the cluster data structures ...
   mmcrcluster: Command successfully completed
   mmcrcluster: Warning: Not all nodes have proper GPFS license designations.
       Use the mmchlicense command to designate licenses as needed.
   ```

2. Change the GPFS license of the xCAT management node to `server` to export an NSD with the **`mmchlicense`** command:

   ```
   # mmchlicense server --accept -N xcat-mn.cluster-xcat
   The following nodes will be designated as possessing GPFS server licenses:
           xcat-mn.cluster-xcat
   mmchlicense: Command successfully completed
   ```

3. You can list the GPFS cluster information with the **`mmlscluster`** command:

   ```
   # mmlscluster

   GPFS cluster information
   ========================
     GPFS cluster name:         cluster-gpfs.cluster-xcat
     GPFS cluster id:           13844065848665742146
     GPFS UID domain:           cluster-gpfs.cluster-xcat
     Remote shell command:      /usr/bin/rsh
     Remote file copy command:  /usr/bin/rcp
     Repository type:           CCR

    Node  Daemon node name      IP address  Admin node name       Designation
   ------------------------------------------------------------------------
      1   xcat-mn.cluster-xcat  10.1.0.1    xcat-mn.cluster-xcat  quorum-manager
   ```

4. Start the GPFS cluster with the **`mmstartup`** command:

   ```
   # mmstartup -a
   Sat Feb  7 18:12:57 EST 2015: mmstartup: Starting GPFS ...
   ```

GPFS Network Shared Disk (NSD)

To create the GPFS Network Shared Disk (NSD), perform the following steps:

1. Create a stanza file for the `/dev/sdc` disk with the NSD name `nsd_sdc`:

```
# cat > nsd.sdc.stanzafile <<EOF
%nsd: device=/dev/sdc
  nsd=nsd_sdc
  servers=xcat-mn.cluster-xcat
EOF
```

2. Create the NSD from the stanza file with the **mmcrnsd** command:

```
# mmcrnsd -F nsd.sdc.stanzafile
mmcrnsd: Processing disk sdc
```

3. You can list the NSDs with the **mmlsnsd** command:

Note: The NSD is listed as a free disk; that is, it is not used by a GPFS file system.

```
# mmlsnsd

 File system   Disk name    NSD servers
---------------------------------------------------------------------------
 (free disk)   nsd_sdc      xcat-mn.cluster-xcat
```

GPFS file system

To create a GPFS file system on the NSD, perform the following steps:

1. Create a GPFS file system (for example, `/dev/gpfs_fs0`) on the NSD (`nsd_sdc`) by using the **mmcrfs** command with the stanza file:

```
# ls /dev/gpfs_fs0
ls: cannot access /dev/gpfs_fs0: No such file or directory

# mmcrfs /dev/gpfs_fs0 -F nsd.sdc.stanzafile

The following disks of gpfs_fs0 will be formatted on node xcat-mn.cluster-xcat:
    nsd_sdc: size 270648 MB
Formatting file system ...
<...>
Completed creation of file system /dev/gpfs_fs0.

# ls -lh /dev/gpfs_fs0
brw-r--r-- 1 root root 239, 150 <...> /dev/gpfs_fs0
```

2. You can verify that the NSD is now used by the `gpfs_fs0` file system with the **mmlsnsd** command:

```
# mmlsnsd

 File system   Disk name    NSD servers
---------------------------------------------------------------------------
 gpfs_fs0      nsd_sdc      xcat-mn.cluster-xcat
```

3. You can list the GPFS file system details (for example, its default mount point) with the **mmlsfs** command:

```
# mmlsfs /dev/gpfs_fs0
flag                value                    description
------------------- ------------------------ -----------------------------------
 -f                 8192                     Minimum fragment size in bytes
 -i                 4096                     Inode size in bytes
 -I                 16384                    Indirect block size in bytes
 <...>
 -T                 /gpfs/gpfs_fs0           Default mount point
 --mount-priority   0                        Mount priority
```

4. You can mount the file system with the **mmmount** command:

```
# mmmount /dev/gpfs_fs0
<...>: mmmount: Mounting file systems ...
```

5. The file system is now available in its mount point (usually in the `/gpfs` directory).
   You can perform normal file system operations on the file system, for example, to list and create files:

```
# ls -l /gpfs/gpfs_fs0/
total 0

# touch /gpfs/gpfs_fs0/hello-world

# ls -l /gpfs/gpfs_fs0/
total 0
-rw-r--r-- 1 root root 0 <...> hello-world
```

Instructions for the compute nodes

Now, perform the tasks that relate to the role of the xCAT compute nodes in the Spectrum Scale cluster. Install and update GPFS, build and distribute the GPFS portability layer, add the nodes to the GPFS cluster, and mount the NSD that is exported by the management node.

The method that we used to install Spectrum Scale in the xCAT compute nodes consists of providing the GPFS distribution packages in a package repository in the management node and then installing GPFS distribution packages in the nodes with the xCAT *other packages* (**otherpkgs**) script. This script is used for installing packages that are not distributed with the Linux distribution. This method is applied for the packages from the distribution media and for packages from the product updates that are available in IBM Fix Central.

Information: xCAT provides two methods for installing Linux packages in the nodes, depending on whether the packages are part of the Linux distribution installation media (therefore in the management node) or its online package repository. Each method uses a different directory for storing the packages and a different list of packages to install:

- Operating system packages:
  - Packages are part in the Linux distribution? *Yes*.
  - Script for installation: **ospkgs**
  - Package directory attribute: `pkgdir`
  - Package lists attribute: `pkglist`
- Other packages:
  - Packages are part in the Linux distribution? *No*.
  - Script for installation: **otherpkgs**
  - Package directory attribute: `otherpkgdir`
  - Package lists attribute: `otherpkglist`

You can verify the "other packages" attributes for an operating system image with the following command:

```
# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i
otherpkgdir,otherpkglist
```

For more information about installing additional packages in xCAT, see the following xCAT wiki pages:

- http://sourceforge.net/p/xcat/wiki/Ubuntu_Quick_Start/#customizing-additional-packages
- http://sourceforge.net/p/xcat/wiki/Using_Updatenode/#install-additional-non-os-rpms

Packages from the distribution media

To install the packages from the distribution media, perform the following steps:

1. Install the `dpkg-dev` package to create simple package repositories:

   ```
   # apt-get install dpkg-dev
   ```

2. Create the package repository with packages from the distribution media in the `otherpkgs` directory for the operating system image (in the subdirectory `gpfs-install`). You can use the **dpkg-scanpackages** command:

   ```
   # cd /install/post/otherpkgs/ubuntu14.04.1/ppc64el/

   # dir=gpfs-install
   # mkdir $dir
   # cd $dir
   # cp /path/to/gpfs-media/*.deb .

   # ls -1 *.deb
   gpfs.base_4.1.0-5_ppc64el.deb
   gpfs.crypto_4.1.0-5_ppc64el.deb
   gpfs.docs_4.1.0-5_all.deb
   gpfs.ext_4.1.0-5_ppc64el.deb
   gpfs.gpl_4.1.0-5_all.deb
   ```

```
gpfs.gskit_8.0.50-32_ppc64el.deb
gpfs.msg.en-us_4.1.0-5_all.deb

# dpkg-scanpackages . > Packages
dpkg-scanpackages: info: Wrote 7 entries to output Packages file.

# grep '^Package:' Packages
Package: gpfs.base
Package: gpfs.crypto
Package: gpfs.docs
Package: gpfs.ext
Package: gpfs.gpl
Package: gpfs.gskit
Package: gpfs.msg.en-us
```

3. Create the package list (in the “*subdirectory/package-name*” format) of the repository:

```
# install_list='/install/custom/ubuntu14.04.1-ppc64el-install-compute/otherpkg
list/gpfs-install.pkglist'
# mkdir -p $(dirname $install_list)
# sed -n "s,^Package: ,$dir/,p" Packages > $install_list

# cat $install_list
gpfs-install/gpfs.base
gpfs-install/gpfs.crypto
gpfs-install/gpfs.docs
gpfs-install/gpfs.ext
gpfs-install/gpfs.gpl
gpfs-install/gpfs.gskit
gpfs-install/gpfs.msg.en-us
```

4. Add the package list to the operating system image with the **chdef** command and verify it with the **lsdef** command:

```
# chdef -t osimage ubuntu14.04.1-ppc64el-install-compute --plus
otherpkglist="$install_list"
1 object definitions have been created or modified.

# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i otherpkglist
Object name: ubuntu14.04.1-ppc64el-install-compute
    otherpkglist=<...>,/install/custom/ubuntu14.04.1-ppc64el-install-compute/
otherpkglist/gpfs-install.pkglist
```

5. Install the packages in the nodes with the **updatenode** command and the **otherpkgs** script. (You can check the `gpfs` packages and versions with the **dpkg -l** command.)

```
# xdsh p8r4n2 'dpkg -l | grep gpfs'
#

# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 0
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 'dpkg -l | grep gpfs'
p8r4n2: ii  gpfs.base                        4.1.0-5
ppc64el      GPFS File Manager
p8r4n2: ii  gpfs.crypto                      4.1.0-5
ppc64el      GPFS Cryptographic Subsystem
```

```
p8r4n2: ii  gpfs.docs                          4.1.0-5
all          GPFS Server Manpages and Documentation
p8r4n2: ii  gpfs.ext                           4.1.0-5
ppc64el      GPFS Extended Features
p8r4n2: ii  gpfs.gpl                           4.1.0-5
all          GPFS Open Source Modules
p8r4n2: ii  gpfs.gskit                         8.0.50-32
ppc64el      GPFS GSKit Cryptography Runtime
p8r4n2: ii  gpfs.msg.en-us                     4.1.0-5
all          GPFS Server Messages - U.S. English
```

Packages from the product updates

To install the packages from product updates, perform the following steps:

1. To install the updates, start by removing the package list from the distribution media from the operating system image with the **chdef** command and verify it with the **lsdef** command. (The `install_list` variable is defined in the previous steps.)

```
# chdef -t osimage ubuntu14.04.1-ppc64el-install-compute --minus
otherpkglist="$install_list"
1 object definitions have been created or modified.

# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i otherpkglist
Object name: ubuntu14.04.1-ppc64el-install-compute
    otherpkglist=<... no list item with name gpfs-install ... >
```

2. Create the package repository with packages from the product updates (notice the `update` suffix in the file names) in the `otherpkgs` directory for the operating system image (in the subdirectory `gpfs-update`). You can use the **dpkg-scanpackages** command:

```
# cd /install/post/otherpkgs/ubuntu14.04.1/ppc64el/

# dir=gpfs-update
# mkdir $dir
# cd $dir

# cp /path/to/gpfs-update/*.deb .
# ls -1 *.deb
gpfs.base_4.1.0-6_ppc64el_update.deb
gpfs.crypto_4.1.0-6_ppc64el_update.deb
gpfs.docs_4.1.0-6_all.deb
gpfs.ext_4.1.0-6_ppc64el_update.deb
gpfs.gpl_4.1.0-6_all.deb
gpfs.gskit_8.0.50-32_ppc64el.deb
gpfs.msg.en-us_4.1.0-6_all.deb

# dpkg-scanpackages . > Packages
dpkg-scanpackages: info: Wrote 7 entries to output Packages file.
```

3. Create the package list (in the "`subdirectory/package-name`" format) of the repository:

```
# update_list='/install/custom/ubuntu14.04.1-ppc64el-install-compute/otherpkg
list/gpfs-update.pkglist'
# mkdir -p $(dirname $update_list)
# sed -n "s,^Package: ,$dir/,p" Packages > $update_list

# cat $update_list
gpfs-update/gpfs.base
gpfs-update/gpfs.crypto
```

```
gpfs-update/gpfs.docs
gpfs-update/gpfs.ext
gpfs-update/gpfs.gpl
gpfs-update/gpfs.gskit
gpfs-update/gpfs.msg.en-us
```

4. Add the package list to the operating system image with the **chdef** command and verify it with the **lsdef** command:

```
# chdef -t osimage ubuntu14.04.1-ppc64el-install-compute --plus
otherpkglist="$update_list"
1 object definitions have been created or modified.

# lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i otherpkglist
Object name: ubuntu14.04.1-ppc64el-install-compute
    otherpkglist=<...>,
                 /install/custom/ubuntu14.04.1-ppc64el-install-compute/
otherpkglist/gpfs-update.pkglist
```

5. Install the packages in the nodes with the **updatenode** command and the **otherpkgs** script. (You can check the GPFS packages and versions with the **dpkg -l** command.)

```
# updatenode p8r4n2 --scripts otherpkgs
p8r4n2: Sun Feb  8 07:30:36 EST 2015 Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 100
p8r4n2: Running of postscripts has completed.

# xdsh p8r4n2 'dpkg -l | grep gpfs'
p8r4n2: ii  gpfs.base                         4.1.0-6
ppc64el      GPFS File Manager
p8r4n2: ii  gpfs.crypto                       4.1.0-6
ppc64el      GPFS Cryptographic Subsystem
p8r4n2: ii  gpfs.docs                         4.1.0-6
all          GPFS Server Manpages and Documentation
p8r4n2: ii  gpfs.ext                          4.1.0-6
ppc64el      GPFS Extended Features
p8r4n2: ii  gpfs.gpl                          4.1.0-6
all          GPFS Open Source Modules
p8r4n2: ii  gpfs.gskit                        8.0.50-32
ppc64el      GPFS GSKit Cryptography Runtime
p8r4n2: ii  gpfs.msg.en-us                    4.1.0-6
all          GPFS Server Messages - U.S. English
```

GPFS portability layer

Note: The GPFS portability layer must be built initially and for every kernel upgrade that includes an Application Binary Interface (ABI) change.

On Ubuntu, the kernel ABI version is listed in the kernel-related packages name and version, as the first number after the dash, for example:

Kernel package name `linux-image-3.13.0-`**`32`**`-generic` at version `3.13.0-`**`32`**`.33`

To build the GPFS portability layer, perform the following steps:

1. Build the GPL in one node with the following command. (Check for the "`exit code 0`" message.)

   ```
   # xdsh p8r4n2 --stream 'apt-get install build-essential linux-headers-$(uname
   -r) && cd /usr/lpp/mmfs/src && make Autoconfig && make World && make
   InstallImages; echo exit code $?'
   <...>
   p8r4n2: Verifying that tools to build the portability layer exist....
   <...>
   p8r4n2:   CC      /usr/lpp/mmfs/src/gpl-linux/mmfs26.mod.o
   p8r4n2:   LD [M]  /usr/lpp/mmfs/src/gpl-linux/mmfs26.ko
   p8r4n2:   CC      /usr/lpp/mmfs/src/gpl-linux/mmfslinux.mod.o
   p8r4n2:   LD [M]  /usr/lpp/mmfs/src/gpl-linux/mmfslinux.ko
   p8r4n2:   CC      /usr/lpp/mmfs/src/gpl-linux/tracedev.mod.o
   p8r4n2:   LD [M]  /usr/lpp/mmfs/src/gpl-linux/tracedev.ko
   <...>
   p8r4n2: /usr/bin/install -c -m 0500 lxtrace /usr/lpp/mmfs/src/bin/lxtrace-`cat
   //usr/lpp/mmfs/src/gpl-linux/gpl_kernel.tmp.ver`
   p8r4n2: /usr/bin/install -c -m 0500 kdump   /usr/lpp/mmfs/src/bin/kdump-`cat
   //usr/lpp/mmfs/src/gpl-linux/gpl_kernel.tmp.ver`
   <...>
   p8r4n2: exit code 0
   ```

2. You can verify that the five GPL-required files (three kernel modules and two utility executables) are correctly built and installed with the following command:

   ```
   # xdsh p8r4n2 'ls -1 /lib/modules/$(uname -r)/extra/{mmfs26.ko,mmfslinux.ko,
   tracedev.ko} /usr/lpp/mmfs/bin/{kdump,lxtrace}'
   p8r4n2: /lib/modules/3.13.0-32-generic/extra/mmfs26.ko
   p8r4n2: /lib/modules/3.13.0-32-generic/extra/mmfslinux.ko
   p8r4n2: /lib/modules/3.13.0-32-generic/extra/tracedev.ko
   p8r4n2: /usr/lpp/mmfs/bin/kdump
   p8r4n2: /usr/lpp/mmfs/bin/lxtrace
   ```

3. Create a tarball with the GPL-required files and copy it back to the management node:

   ```
   # xdsh p8r4n2 --stream 'tar cvjf /tmp/gpfs-gpl_$(uname -r).tar.bz2
   /lib/modules/$(uname -r)/extra/{mmfs26.ko,mmfslinux.ko,tracedev.ko}
   /usr/lpp/mmfs/bin/{kdump,lxtrace}'
   p8r4n2: tar: Removing leading `/' from member names
   p8r4n2: /lib/modules/3.13.0-32-generic/extra/mmfs26.ko
   p8r4n2: /lib/modules/3.13.0-32-generic/extra/mmfslinux.ko
   p8r4n2: /lib/modules/3.13.0-32-generic/extra/tracedev.ko
   p8r4n2: /usr/lpp/mmfs/bin/kdump
   p8r4n2: /usr/lpp/mmfs/bin/lxtrace

   # dir='/install/custom/ubuntu14.04.1-ppc64el-install-compute/download/gpfs-gpl'
   # mkdir -p $dir

   # scp p8r4n2:/tmp/gpfs-gpl_3.13.0-32-generic.tar.bz2 $dir
   gpfs-gpl_3.13.0-32-generic.tar.bz2
   100%  <...>
   ```

4. Download and extract that tarball on all nodes:

```
# xdsh p8r4n2 --stream
'url="http://10.1.0.1/install/custom/ubuntu14.04.1-ppc64el-install-compute/down
load/gpfs-gpl/gpfs-gpl_3.13.0-32-generic.tar.bz2" && tarball="/tmp/$(basename
$url)" && wget -nv $url -O $tarball && tar xvf $tarball --directory=/ && rm -f
$tarball'
p8r4n2: 2015-02-08 22:10:08
URL:http://10.1.0.1/install/custom/ubuntu14.04.1-ppc64el-install-compute/downlo
ad/gpfs-gpl/gpfs-gpl_3.13.0-32-generic.tar.bz2 [973352/973352] ->
"/tmp/gpfs-gpl_3.13.0-32-generic.tar.bz2" [1]
p8r4n2: lib/modules/3.13.0-32-generic/extra/mmfs26.ko
p8r4n2: lib/modules/3.13.0-32-generic/extra/mmfslinux.ko
p8r4n2: lib/modules/3.13.0-32-generic/extra/tracedev.ko
p8r4n2: usr/lpp/mmfs/bin/kdump
p8r4n2: usr/lpp/mmfs/bin/lxtrace
```

5. You can verify that the GPL-required files are in a node with the following command:

```
# xdsh p8r4n2 'ls -1 /lib/modules/$(uname -r)/extra/{mmfs26.ko,mmfslinux.ko,
tracedev.ko} /usr/lpp/mmfs/bin/{kdump,lxtrace}'
p8r4n2: /lib/modules/3.13.0-32-generic/extra/mmfs26.ko
p8r4n2: /lib/modules/3.13.0-32-generic/extra/mmfslinux.ko
p8r4n2: /lib/modules/3.13.0-32-generic/extra/tracedev.ko
p8r4n2: /usr/lpp/mmfs/bin/kdump
p8r4n2: /usr/lpp/mmfs/bin/lxtrace
```

GPFS cluster

To add the nodes to the Spectrum Scale cluster with the `client` license, perform the following steps (for a particular node or all nodes):

1. Add the node to the GPFS cluster with the **mmaddnode** command:

   – For a particular node:

```
# mmaddnode -N p8r4n2
<...>: mmaddnode: Processing node p8r4n2
mmaddnode: Command successfully completed
mmaddnode: Warning: Not all nodes have proper GPFS license designations.
    Use the mmchlicense command to designate licenses as needed.
mmaddnode: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
```

   – For all nodes (the `power8_nodes` node group):

```
# mmaddnode -N $(nodels power8_nodes | tr '\n' ',')
```

2. Set the license for the node with the **mmchlicense** command:

   – For a particular node:

```
# mmchlicense client -N p8r4n2 --accept

The following nodes will be designated as possessing GPFS client licenses:
        p8r4n2
mmchlicense: Command successfully completed
mmchlicense: Propagating the cluster configuration data to all
  affected nodes.  This is an asynchronous process.
```

   – For all nodes (the `power8_nodes` node group):

```
# mmchlicense client -N $(nodels power8_nodes | tr '\n' ',') --accept
```

3. List the GPFS cluster information and details with the **lscluster** command:

```
# mmlscluster

GPFS cluster information
========================
  GPFS cluster name:         cluster-gpfs.cluster-xcat
  GPFS cluster id:           13844065848665742146
  GPFS UID domain:           cluster-gpfs.cluster-xcat
  Remote shell command:      /usr/bin/rsh
  Remote file copy command:  /usr/bin/rcp
  Repository type:           CCR

 Node  Daemon node name      IP address  Admin node name       Designation
----------------------------------------------------------------------------
   1   xcat-mn.cluster-xcat  10.1.0.1    xcat-mn.cluster-xcat  quorum-manager
   4   p8r4n2                10.1.4.2    p8r4n2
```

4. Start the GPFS cluster in the nodes with the **mmstartup** command:

```
# mmstartup -a
Sun Feb  8 22:13:14 EST 2015: mmstartup: Starting GPFS ...
xcat-mn.cluster-xcat:  The GPFS subsystem is already active.
```

5. Configure the nodes to automatically load and start GPFS on boot (rather than using the **mmstartup** command):

   a. List the current configuration with the **mmlsconfig** command:

   ```
   # mmlsconfig autoload
   autoload no
   ```

   b. Change the current configuration with the **mmchconfig** command:

   - For a particular node:

     ```
     # mmchconfig autoload=no -N p8r4n2
     mmchconfig: Command successfully completed
     mmchconfig: Propagating the cluster configuration data to all
       affected nodes.  This is an asynchronous process.

     # mmlsconfig autoload
     autoload yes
     autoload no [p8r4n2]
     ```

   - For all nodes:

     ```
     # mmchconfig autoload=yes
     <...>

     # mmlsconfig autoload
     autoload yes
     ```

Tip: To restart the GPFS cluster on all nodes, you can use the **mmshutdown** and **mmstartup** commands:

```
# mmshutdown -a
<...> mmshutdown: Starting force unmount of GPFS file systems
<...> mmshutdown: Shutting down GPFS daemons
p8r4n2:  Shutting down!
xcat-mn.cluster-xcat:  Shutting down!
p8r4n2:  Unloading modules from /lib/modules/3.13.0-32-generic/extra
xcat-mn.cluster-xcat:  'shutdown' command about to kill process 61400
xcat-mn.cluster-xcat:  Unloading modules from
/lib/modules/3.13.0-32-generic/extra
xcat-mn.cluster-xcat:  Unloading module mmfs26
xcat-mn.cluster-xcat:  Unloading module mmfslinux
<...> mmshutdown: Finished

# mmstartup -a
<...> mmstartup: Starting GPFS ...
```

GPFS file system

The GPFS file system must be already available while the GPFS cluster daemon is running. To access the previously created GPFS file system, access its mount point in `/gpfs`:

```
# xdsh p8r4n2 'ls -l /dev/gpfs_fs0; ls -lR /gpfs/gpfs_fs0'
p8r4n2: /dev/gpfs_fs0
p8r4n2: /gpfs/gpfs_fs0:
p8r4n2: total 0
p8r4n2: -rw-r--r-- 1 root root 0 Feb  7 18:25 hello-world
```

3.9 Software and firmware updates

This section covers updates to the software stack and the system firmware. It also describes how to handle problems during updates with the package manager. It explains the definitions and particularities of package updates in the Ubuntu server distribution.

3.9.1 Software updates: Operating system packages

The xCAT *operating system packages* script (**ospgks**) is available with the **updatenode** command. The **ospgks** script runs the correct package manager operations in the nodes to check for and install any updates to the installed packages that are available in the updates channel of the Linux distribution (usually a package repository).

Note: This process requires that the nodes can access the updates channel of the Linux distribution. Access can be through the Internet. (See 3.8.1, "Configure nodes for access to the Internet" on page 72). Or, access can be a local mirror of the package repository, which is not covered in this book.

The **updatenode** command runs the **ospkgs** script automatically if you do not specify the scripts to run. To run only the **ospkgs** script, use the following command:

```
# updatenode <node or node-group> --scripts ospkgs
<node>: <...> Running postscript: ospkgs
<...>
<node>: Postscript: ospkgs exited with code <exit code>
```

Note: We suggest that you first perform this operation on a single node (not in a node group) to verify whether any errors exist in the package updates or whether additional processes (for example, problem resolution or configuration steps) exist. Then, only *after* you complete the process on a single node, proceed to more nodes or the node group.

3.9.2 Software updates: Other packages

The xCAT *other packages* script (**otherpkgs**) is similar to the **ospkgs** script, but **otherpkgs** applies to the packages that are not part of the Linux distribution, for example, packages that are distributed with the xCAT Software Kits. These package repositories and lists are defined by the xCAT commands for installing kits, as described in 3.8, “xCAT compute nodes: Software stack” on page 72.

Note: The procedures to update xCAT Software Kits can differ between products and possibly require special instructions for a particular product. Several products provide kits with a completely functional set of packages, while other products provide only a differential set of packages between an original version and the update version.

We suggest that you see the documentation of a particular product and kit for the steps to install and update its packages in the xCAT cluster, if the product is not described in or it differs from 3.8, “xCAT compute nodes: Software stack” on page 72.

The **updatenode** command runs the **otherpkgs** script automatically if you do not specify the specific scripts to run. To run only the **otherpkgs** script, use the following command:

```
# updatenode <node or node-group> --scripts otherpkgs
<node>: <...> Running postscript: otherpkgs
<...>
<node>: Postscript: otherpkgs exited with code <exit code>
```

Note: We suggest that you first perform this operation on a single node (not in a node group) to verify whether any errors exist in the package updates or whether additional processes (for example, problem resolution or configuration steps) exist. Then, only *after* you complete the process on a single node, proceed to more nodes or the node group.

3.9.3 Software updates: Package updates in Ubuntu server

The Ubuntu server Linux distribution uses the *Advanced Package Tool* (APT), which is mostly known for the **apt-get** command, as the default high-level interface for the `dpkg` package manager.

The **apt-get** command provides three commands that relate to package updates, which despite the similar names, perform distinct functions:

- **apt-get update**: This command updates the package index files only (that is, the lists of available packages, versions, and related metadata, which are also known as *package definitions*) from the package repositories. *This command does not install, update, or remove any packages.*
- **apt-get upgrade**: This command updates the installed packages to the up-to-date available version. *This command updates the installed packages; however, it does not install any new packages or remove any packages.*
- **apt-get dist-upgrade**: This command updates the installed packages to the up-to-date available version. It can modify the list of installed packages. *This command updates the installed packages and, depending on the updates, installs new packages and removes unnecessary packages.*

Note: The difference between **apt-get upgrade** and **apt-get dist-upgrade** allows the option to perform updates to the fundamental packages or to determine whether any changes pose a risk to the system. This feature relies on coordination by the package maintainers, ensuring that new versions of packages with such changes actually become packages with different names (usually by including a version number in the package name), which are therefore interpreted as new packages.

For example, this mechanism is used by the kernel and compiler toolchain packages (for example: `linux-image-<`*`version`*`>-<`*`ABI version`*`>-generic`, `gcc-<`*`version`*`>`), which can introduce changes to binary compatibility and generation, or changes to default options, possibly rendering the system unable to boot or load out-of-tree kernel modules.

The following commands are often used to update the package definitions and, if that finishes without errors, then perform package updates:

```
# apt-get update && apt-get upgrade
# apt-get update && apt-get dist-upgrade
```

For the xCAT nodes, you can use the **xdsh** command (quote the command to prevent interpretation by the local shell):

```
# xdsh p8r4n2 --stream 'apt-get update && apt-get upgrade'
# xdsh p8r4n2 --stream 'apt-get update && apt-get dist-upgrade'
```

For more information about the **apt-get** command, see its manual page with either the following command or the weblink:

```
# man apt-get
```

http://manpages.ubuntu.com/manpages/trusty/man8/apt-get.8.html

3.9.4 Software updates: Problems in package updates

Package updates can encounter errors for several reasons, including the following examples:

- Presenting an interactive prompt. (The xCAT scripts are not interactive.)
- Providing a new default configuration file when the installed configuration file contains modifications (and therefore differ from the old default), which triggers an interactive prompt
- Failing due to not found or incorrectly specified dependency packages.

Depending on the error, the package update can cause problems to other package management operations, for example, other updates, installation, removal, or dependency resolution, therefore possibly rendering the system in a problematic state.

The errors that are encountered by the package manager when you install and update packages are listed in the output of the **updatenode** command for the **ospkgs** and **otherpkgs** scripts, which run the **apt-get** command. Typically, the errors can be identified by a *nonzero return/exit code* in the script/command chain. See the output excerpts in Example 3-81.

Example 3-81 xCAT: Errors in package updates during the updatenode command

```
Excerpt 1:
p8r4n2: <...> Running postscript: ospkgs
p8r4n2: === apt-get -y upgrade
<...>
p8r4n2: *** <...> dpkg: error processing package <package> (--configure):
<...>
p8r4n2: Errors were encountered while processing:
p8r4n2: <package>
p8r4n2: E: Sub-process /usr/bin/dpkg returned an error code (1)

Excerpt 2:
p8r4n2: <...> Running postscript: ospkgs
<...>
p8r4n2: Postscript: ospkgs exited with code 100

Excerpt 3:
p8r4n2: <...> Running postscript: otherpkgs
p8r4n2: Postscript: otherpkgs exited with code 100
```

The required steps to resolve the problem vary with the package and error, but they usually involve reproducing the error manually in one of the problematic nodes to obtain evidence for searching in documentation and debug information.

You can obtain the failing command from the output of the **updatenode** command, for example, `apt-get -y upgrade`, as listed in Example 3-81. You can reproduce and run the failing command in the nodes with the **xdsh** command (use the **--stream** argument for improved readability):

```
# xdsh p8r4n2 --stream 'apt-get -y upgrade'
```

Example: Yes/No confirmation prompts

In several cases, a simple `yes/no` confirmation prompt is presented by the **apt-get** command before it can continue.

Usually, *yes* must be answered. Sometimes, the **--yes** (or **-y**) argument is not enough (for security reasons) and **--force-yes** must be used:

```
# xdsh p8r4n2 --stream 'apt-get --yes --force-yes upgrade'
```

If *no* is the intended answer, the **--assume-no** argument can be used, instead:

```
# xdsh p8r4n2 --stream 'apt-get --assume-no upgrade'
```

Example: New version of default configuration files

If a package update provides a new version of a default configuration file, for example, several of the files in `/etc`, the package update can only update the currently installed configuration file if the configuration file is exactly the same as the old version of the default configuration file. That is, the configuration file was not modified at all since the installation. This rule ensures that any user-specified configuration is not lost during a package update.

When the described conflict happens in a package update, an interactive option prompt for conflict resolution is presented by the **`apt-get`** command. Because the xCAT scripts are not interactive, the **`apt-get`** command fails due to the lack of input.

One frequent example is the `rsyslog` package, which has a configuration file that is modified by xCAT (to perform remote logging from the compute nodes to the management node), and a package update is available (with Ubuntu Server 14.04.1 LTS). The error is depicted in Example 3-82.

Example 3-82 xCAT: Error during the update of the rsyslog package

```
# updatenode p8r4n2
<...>
p8r4n2: <...> Running postscript: ospkgs
p8r4n2: === apt-get -y upgrade
<...>
p8r4n2: Setting up rsyslog (7.4.4-1ubuntu2.3) ...
p8r4n2:
p8r4n2: Configuration file '/etc/rsyslog.conf'
p8r4n2:  ==> Modified (by you or by a script) since installation.
p8r4n2:  ==> Package distributor has shipped an updated version.
p8r4n2:    What would you like to do about it ?  Your options are:
p8r4n2:     Y or I  : install the package maintainer's version
p8r4n2:     N or O  : keep your currently-installed version
p8r4n2:       D     : show the differences between the versions
p8r4n2:       Z     : start a shell to examine the situation
p8r4n2:  The default action is to keep your current version.
p8r4n2: *** rsyslog.conf (Y/I/N/O/D/Z) [default=N] ? dpkg: error processing
package rsyslog (--configure):
p8r4n2:  EOF on stdin at conffile prompt
p8r4n2: Errors were encountered while processing:
p8r4n2:  rsyslog
p8r4n2: E: Sub-process /usr/bin/dpkg returned an error code (1)
<...>
```

To resolve this sort of conflict with configuration files, you can specify the preferred option in the **`apt-get`** command, with option arguments for the **`dpkg`** package manager. The following conflict resolution options for **`dpkg`** are available:

- `--force-conf`**`def`**: Choose the *default* action, if available (preferred over other options).
- `--force-conf`**`old`**: Choose to keep the *old* version (if `--force-confdef` is not specified).
- `--force-conf`**`new`**: Choose to install the *new* version (if `--force-confdef` is not specified).

You can specify **`dpkg`** options in the **`apt-get`** command with multiple arguments `--option` (or `-o`) **`Dpkg::Options::="`*`<dpkg-option>`*`"`**. For example, to run the **`apt-get`** upgrade, preferably use the default action if available, and the old configuration file otherwise. (Line breaks were added for clarity.)

```
# xdsh p8r4n2 'apt-get
```

```
-o Dpkg::Options::="--force-confdef"
-o Dpkg::Options::="--force-confold"
upgrade'
```

Considering the example with the `rsyslog` package, you can resolve it by performing a similar command to choose the default action (Example 3-83).

Example 3-83 xCAT: Resolving the conflict with the configuration file in the rsyslog package update

```
# xdsh p8r4n2 'apt-get -o Dpkg::Options::="--force-confdef" --yes install rsyslog'
<...>
p8r4n2: Setting up rsyslog (7.4.4-1ubuntu2.5) ...
<...>
p8r4n2: Configuration file '/etc/rsyslog.conf'
p8r4n2:  ==> Modified (by you or by a script) since installation.
p8r4n2:  ==> Package distributor has shipped an updated version.
p8r4n2:  ==> Keeping old config file as default.
<...>
```

For more information about **dpkg** options, see the manual page (section `OPTIONS`, subsection `--force-things`) by using the following command or weblink:

```
# man dpkg
```

http://manpages.ubuntu.com/manpages/trusty/man1/dpkg.1.html

Example: Missing package dependencies

Sometimes, a kit or package does not completely or correctly specify the other packages that it depends on.

For an example and solution steps, see the **pelinks**/**ksh** case that is described in 3.8.6, "Parallel Environment Runtime Edition (PE RTE)" on page 89, which describes how to add a package to the list of packages that are automatically installed in an operating system image.

Alternatively, install the packages in order in a node (for example, for test/verification purposes) without modifying the list of packages that are automatically installed, you can use the following commands:

- Packages that are available in a package repository: **apt-get install**

  ```
  # xdsh p8r4n2 'apt-get install <package1> [package2] [...]'
  ```

- Packages that are available as *deb* package files: **dpkg -i**

  ```
  # xdsh p8r4n2 'dpkg -i </path/to/package1.deb> [/path/to/package2.deb]'
  ```

For more information about the **apt-get** command, see the manual page by using either the following command or weblink:

```
# man apt-get
```

http://manpages.ubuntu.com/manpages/trusty/man8/apt-get.8.html

For more information about the **dpkg** command, see the manual page by using either the following command or weblink:

```
# man dpkg
```

http://manpages.ubuntu.com/manpages/trusty/man1/dpkg.1.html

3.9.5 Software updates: Kernel package updates

The updates to the kernel packages are not installed by default because the **ospkgs** script runs **apt-get upgrade**, not **dist-upgrade**. See 3.9.3, “Software updates: Package updates in Ubuntu server” on page 117.

Updates to the kernel package are available in the output of the **ospkgs** script. They are run by default by the **updatenode** command. You will see messages of packages with the name `linux-`*`something`* that are *kept back*, that is, not updated (Example 3-84).

Example 3-84 xCAT: Checking for kernel packages that are not updated

```
# updatenode p8r4n2
<...>
p8r4n2: <...> Running postscript: ospkgs
p8r4n2: === apt-get -y upgrade
<...>
p8r4n2: The following packages have been kept back:
p8r4n2:   linux-generic linux-headers-generic linux-image-generic
p8r4n2: 0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
<...>
p8r4n2: Postscript: ospkgs exited with code 0
<...>
p8r4n2: Running of postscripts has completed.
```

You can update the kernel and other packages with the **apt-get dist-upgrade** command. (See 3.9.3, “Software updates: Package updates in Ubuntu server” on page 117.)

You can update the kernel packages and no other packages by requesting an update of the *linux-generic* meta-package. This meta-package depends on the `linux-image-generic` and `linux-headers-generic` meta-packages, which in turn depend on the latest version of the concrete kernel image and kernel headers packages.

```
# xdsh p8r4n2 'apt-get install --yes --force-yes linux-generic'
```

The new version is installed as the default version to boot the system. The old versions remain installed, and they can be used to boot the system if the new version has issues.

You can reboot the system through the operating system by using the following command:

```
# xdsh p8r4n2 reboot
```

3.9.6 Software updates: Early package updates

Although early package updates are not popular in cluster environments, several updates can provide the hardware enablement, performance improvements, and critical security that you want or bug fixes. These updates might be available earlier for testing and verification, for example.

Two updates channels (package repositories) are on the Ubuntu Linux distribution:

- Regular updates channel: the `<release>-updates` (for example, `trusty-updates`) repository, which is enabled by default. It contains the package updates that are tested and verified, for example, by bug reporters, Ubuntu teams, and other users.
- Early updates channel: the `<release>-proposed` (for example, `trusty-proposed`) repository, which is *not* enabled by default. It contains the package updates that are *not* yet tested and verified, which are made available for this testing and verification usually for bug reporters.

To enable the `proposed` repository, which is also known as "the proposed *pocket*," and obtain its package definitions, run the following command:

- General command:

```
# echo "deb http://ports.ubuntu.com/ubuntu-ports trusty-proposed main
restricted universe" >> /etc/apt/sources.list && apt-get update
```

- In the nodes:

```
# xdsh p8r4n2 'echo "deb http://ports.ubuntu.com/ubuntu-ports trusty-proposed
main restricted universe" >> /etc/apt/sources.list && apt-get update'
```

You can verify the package versions that are available from each repository or "pocket" with the **`apt-cache madison`** command:

```
# xdsh p8r4n2 'apt-cache madison <package>'
```

3.9.7 Firmware updates

This section describes the process to perform a firmware update (or upgrade) on the compute nodes by providing the up-to-date firmware image in the management node. Follow these steps:

1. Download the up-to-date firmware image package from IBM Fix Central:
   a. Go to the IBM Fix Central:

      `http://www.ibm.com/support/fixcentral/`

   b. Click **Select product**.

   c. From the Product Group list, select **Power**.

   d. From the Product list, select the *machine-type/model* (MTM) code, for example, `8247-42L`, for the IBM Power System S824L system.

   e. From the Select from <*MTM*> list, select the firmware major version that you want, for example, `SV810` or `SV830`.

   f. Click **Continue**.

   g. Select **POWER8 System Firmware <*major_minor version*> (FW<*version*>)**, for example, the most up-to-date version or another version that you want.

   h. Click **Continue**.

   i. Proceed with authentication.

   j. On the Download options page, select **Download using bulk FTP**.

   k. Click **Continue**.

   l. If required, provide the system's machine-type and serial number, and click **Continue**.

m. In the “Download files using bulk FTP” window, verify the terms and conditions, and read the required FTP properties and hints in the Fix package location section, for example:

```
FTP server: delivery04-mul.dhe.ibm.com
User ID: anonymous
Password: Send any password
Directory: /ecc/hsb/H74564367

FTP hints
<...>
ftp> binary
ftp> prompt
ftp> mget *
```

n. Download the file by using the **ftp** command and the specified properties:

```
# ftp delivery04-mul.dhe.ibm.com
<...>
Name (delivery04-mul.dhe.ibm.com:ubuntu): anonymous
331 Guest login ok, send any password.
Password:
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /ecc/hsb/H74564367
250 CWD command successful.
ftp> binary
200 Type set to I.
ftp> prompt
Interactive mode off.
ftp> mget *
<...>
150 Opening BINARY mode data connection for 01SV810_108_081.rpm (... bytes).
226 Transfer complete.
ftp> quit
<...>
221 Goodbye.
```

o. The firmware image package download is available in the current directory:

```
# ls -1
01SV810_108_081.rpm
```

2. Provide the firmware image in the management node for the compute nodes:

   a. Convert the RPM package to a DEB package with the **alien** command:

```
# apt-get install alien
<...>

# alien --to-deb 01SV810_108_081.rpm
<...>
01sv810-108-081_1.1-2_all.deb generated

# ls -1
01sv810-108-081_1.1-2_all.deb
01SV810_108_081.rpm
```

b. Extract the package contents with the **dpkg-deb** command. It is *not* necessary to install the package.

```
# dpkg-deb -x 01sv810-108-081_1.1-2_all.deb .
```

c. The firmware image is available in the `tmp/fwupdate` directory:

```
# ls -1 tmp/fwupdate/
01SV810_108_081.img
```

d. Copy the firmware image to a directory in the management node that is accessible by the compute nodes:

```
# dir='/install/custom/ubuntu14.04.1-ppc64el-install-compute/download/
firmware'
# mkdir -p $dir
# cp tmp/fwupdate/01SV810_108_081.img $dir
```

3. Download and flash (that is, install) the firmware image in the compute nodes:

a. You can use the **update_flash_nv** command to flash the firmware image in a non-virtualized environment. It is available in the Ubuntu server, automatically installed by xCAT, and in the petitboot bootloader shell. The following command describes the command options that are used:

```
# update_flash_nv -h
USAGE: update_flash {-h | -s | -r | -c | -d | [-v|-n] -f <image filename>}

   -h              Print this message.
   -s              Determine whether partition has access to
                   perform flash image management.
   -r              Reject temporary image.
   -c              Commit temporary image.
   -d              Display current firmware version.
   -v              Validate the given image file.
   -n              Do not overwrite Permanent side image automatically.
   -f <filename>   Update with given image file. If possible, the image is
                   automatically validated before the update.
```

b. Verify that the nodes can flash the firmware image (that is, perform firmware/flash image management) with the following command:

```
# xdsh p8r4n2 ‘update_flash_nv -s’
p8r4n2: update_flash: Firmware image management is supported.
```

c. If supported, proceed to download the firmware image in the nodes from the management node and flash the firmware image with the following command:

```
# xdsh p8r4n2 --stream 'url="http://10.1.0.1/install/custom/ubuntu14.04.1-
ppc64el-install-compute/download/firmware/01SV810_108_081.img"; wget $url
-nv -O /tmp/firmware.img && update_flash_nv -f /tmp/firmware.img'
p8r4n2: <...> URL:<...>/01SV810_108_081.img <...> -> "/tmp/firmware.img" [1]
p8r4n2: info: Temporary side will be updated with a newer or identical image
p8r4n2: Projected Flash Update Results:
p8r4n2: Current T Image: SV810_101
p8r4n2: Current P Image: SV810_101
p8r4n2: New T Image:     SV810_108
p8r4n2: New P Image:     SV810_101
p8r4n2:
p8r4n2: FLASH: Image ready...rebooting the system...
p8r4n2: FLASH: This will take several minutes.
p8r4n2: FLASH: Do not power off!
```

The following output can be observed in the node console:

```
Broadcast message from root@p8r4n2
<...>
The system is going down for reboot NOW!
<...>
[<...>] reboot: Restarting system
[<...>] FLASH: Flashing new firmware
[<...>] FLASH: Image is <...> bytes
[<...>] FLASH: Performing flash and reboot/shutdown
[<...>] FLASH: This will take several minutes. Do not power off!
```

d. After the node boots successfully, verify the version of the firmware image in the boot side or temporary side:

```
# xdsh p8r4n2 ‘update_flash_nv -d’
p8r4n2: Current firwmare version :
p8r4n2:   P side    : FW810.20 (SV810_101)
p8r4n2:   T side    : FW810.21 (SV810_108)
p8r4n2:   Boot side : FW810.21 (SV810_108)
```

e. You can commit the firmware image to the permanent side with this command:

```
# xdsh p8r4n2 ‘update_flash_nv -c’
p8r4n2: Success: Committed temporary firmware image.
```

3.10 System tuning

This section outlines operating system tuning options.

3.10.1 CPU frequency scaling

Linux provides CPU frequency scaling with the `cpufreq` mechanism. It can adjust the processors’ clock frequency over time, according to certain policies, which are called *governors*.

The default scaling governor in the Ubuntu server is `ondemand`, which dynamically adjusts CPU frequency according to system load, on a processor-core level. This governor enables power savings without compromising performance. A core’s frequency is maintained low when idle, and increased under load.

On highly used HPC systems, the preferable scaling governor is `performance`, which statically sets CPU frequency to the highest available frequency. This governor ensures that a core is clocked at maximal frequency at all times and eliminates any frequency-scaling overhead and jitter.

The cpufreq settings are per-CPU (hardware thread) files that are in this directory:

`/sys/devices/system/cpu/cpu<number>/cpufreq/`

Although the `cpufreq` settings are exposed *per-CPU* (or *per-hardware thread)*, the `cpufreq` settings are actually *per-core* on POWER8. That is, they are equal among all hardware threads in a core. This approach is reflected in the `related_cpus` file, as shown in Example 3-85 on page 127, which shows that CPU (or hardware thread) 0 shares settings with other CPUs (hardware threads) of its core (eight hardware threads in SMT8 mode).

Example 3-85 The related_cpus file

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/related_cpus
0 1 2 3 4 5 6 7
```

For example, the current scaling governor can be displayed or changed by using the `scaling_governor` file and the available scaling governors are displayed in the `scaling_available_governors` file (Example 3-86).

Example 3-86 Accessing the scaling_governor and scaling_available_governors files

```
# xdsh p8r4n2 cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
p8r4n2: ondemand

# xdsh p8r4n2 cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
p8r4n2: conservative userspace powersave ondemand performance

# xdsh p8r4n2 ‘echo performance > /sys/devices/system/cpu/cpu0/cpufreq/
scaling_governor’

# xdsh p8r4n2 cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
p8r4n2: performance
```

Set the scaling governor to the performance governor on all CPUs

The `cpufrequtils` package provides tools for managing `cpufreq` settings on all CPUs/hardware-threads (system-wide) and making changes persistent across reboots. Follow these steps:

1. Install the `cpufrequtils` package:
   a. Add the package to the `ospkgs` package list in the operating system image:
   ```
   # lsdef -t osimage ubuntu14.04.1-ppc64el-install-compute -i pkglist
   Object name: ubuntu14.04.1-ppc64el-install-compute
       pkglist=/opt/xcat/share/xcat/install/ubuntu/compute.pkglist

   # echo cpufrequtils >> /opt/xcat/share/xcat/install/ubuntu/compute.pkglist
   ```
   b. Install the package by running the **`ospkgs`** script with the **`updatenode`** command:
   ```
   # updatenode p8r4n2 --scripts ospkgs
   ```
2. Add the line `GOVERNOR=performance` to the file `/etc/default/cpufrequtils` to make the setting persistent across reboots:
   ```
   # xdsh p8r4n2 'echo GOVERNOR=performance >> /etc/default/cpufrequtils'
   ```
3. Disable the system’s default `ondemand` governor setting:
   ```
   # xdsh p8r4n2 'update-rc.d ondemand disable'
   ```
4. Finally, activate the `performance` governor without rebooting the system:
   ```
   # xdsh p8r4n2 'service cpufrequtils start'
   ```

You can use the **cpufreq-info** command to obtain the status of the cpufreq settings. See Example 3-87 to verify the number of CPUs/hardware-threads that are running with a certain governor.

Example 3-87 Number of CPUs with the specified scaling governor

```
# xdsh p8r4n2 “cpufreq-info | grep ‘The governor’ | uniq -c”
p8r4n2: 160      The governor “performance” <...>
```

4

Cluster monitoring

This chapter describes the available monitoring tools for the IBM POWER8 System S824L and the Linux high-performance computing (HPC) solution, and the key features of the new monitoring software. IBM Platform Load Sharing Facility (LSF) administrators and xCAT administrators can use these commands to monitor the cluster.

This chapter describes the monitoring tools for the following components:

- IBM Platform LSF administration
- xCAT cluster monitoring

4.1 IBM Platform LSF administration

The IBM Platform LSF product family is a workload management platform for demanding, distributed, and mission-critical high-performance computing (HPC) environments. It provides a set of intelligent, policy-driven scheduling features so that users can fully take advantage of all of compute infrastructure resources and ensure optimal application performance. For more information about Platform LSF, see the following website:

http://www.ibm.com/systems/platformcomputing/products/sf/index.html

4.1.1 Platform LSF definitions

To demonstrate how to administer Platform LSF, this chapter provides an example to simulate the user scenario, including Platform LSF job queue definitions and user group definitions. This chapter provides a set of Platform LSF commands to manage cluster resources and scheduling policies for job scheduling and dispatch.

Typically, Platform LSF cluster administrators define a number of job queues to distinguish different types of workloads and resource requirements. Platform LSF administrators also define a number of user groups to give different roles and privileges to each user group.

Job queues

A defined queue includes the information about which users can access and manage this queue, hosts and resources of this queue, job types and maximum execution time, and restrictions, such as exclusive execution. Platform LSF has default queues that are defined in the `lsb.params` file. By default, the queue is the normal queue and the interactive queue. When a user submits a job, Platform LSF chooses the best queue to send the job to. If the user does not specify a queue, the best queue is chosen based on the input parameters of the job.

The Platform LSF administrator can also define new queues. All definitions for queues are stored in the `lsb.queues` file. In the scenario for this book, eight job queues are defined for different workloads as shown in Table 4-1.

Table 4-1 Job queue definitions

Queue name	Description
Priority	Jobs that are submitted for this queue are scheduled as urgent jobs. Jobs in this queue can preempt jobs in lower-priority queues.
Night	For large, heavy-duty jobs that run during off hours and weekends. Scheduled with higher priority.
Small	Small job, 4-hour maximum wall-clock time (wall time), 2 GB maximum.
Medium	Medium job, 8-hour maximum wall time, 4 GB maximum, run job with restricted user and host.
Large	Large job, 10-day maximum wall time, 8 GB maximum, run job with restricted user and host.
Normal	For normal low-priority jobs that run only if hosts are lightly loaded.
Idle	Running only if the machine is idle and lightly loaded.
Interactive	For interactive jobs only, 48-hour maximum wall time.

To define these job queues in Platform LSF, you need to modify the `lsb.queue` file as shown in Example 4-1. This file is installed by default in the location:

`<LSB_CONFDIR>/<cluster_name>/configdir`

Example 4-1 lsb.queue file

```
Begin Queue
QUEUE_NAME   = interactive
PRIORITY     = 30
INTERACTIVE  = ONLY
NEW_JOB_SCHED_DELAY = 0
RUNLIMIT     = 48:00
DESCRIPTION  = For interactive jobs only, 48-hours max wall-time
End Queue

Begin Queue
QUEUE_NAME   = normal
PRIORITY     = 30
INTERACTIVE  = NO
TASKLIMIT    = 5             # job task limit
RUNLIMIT     = 8:00          # 8 wall-clock hours
USERS        = all           # users who can submit jobs to this queue
HOSTS        = all           # hosts on which jobs in this queue can run
DESCRIPTION  = For normal low priority jobs, running only if hosts are lightly
loaded
End Queue

Begin Queue
QUEUE_NAME    = priority
PRIORITY      = 43
QJOB_LIMIT    = 15           # 10 max concurrent jobs
PREEMPTION    = PREEMPTIVE
DESCRIPTION   = Jobs submitted for this queue are scheduled as urgent\
jobs. Jobs in this queue can preempt jobs in lower priority queues.
End Queue

Begin Queue
QUEUE_NAME   = idle
PRIORITY     = 20
r15s         = 0.3/1.5
r1m          = 0.3/1.5
pg           = 4.0/15
it           = 10/1
DESCRIPTION  = Running only if the machine is idle and lightly loaded.
End Queue

Begin Queue
QUEUE_NAME   = night
PRIORITY     = 40
RUN_WINDOW   = 5:19:00-1:8:30 20:00-8:30
r1m          = 0.8/2.5
DESCRIPTION  = For large heavy duty jobs, running during off hours and \
weekends. Scheduled with higher priority.
End Queue

Begin Queue
```

```
QUEUE_NAME   = small
PRIORITY     = 38
QJOB_LIMIT   = 80             # job limit of the queue
UJOB_LIMIT   = 40             # job limit per user
r1m          = 0.7/2.0        # loadSched/loadStop
RUNLIMIT     = 4:00           # 4 wall-clock hours
MEMLIMIT     = 2000000        # jobs bigger than 2GB will be niced
USERS        = all            # users who can submit jobs to this queue
HOSTS        = all
DESCRIPTION  = small job, 4-hours max wall-time, 2GB max
End Queue

Begin Queue
QUEUE_NAME   = medium
PRIORITY     = 35
QJOB_LIMIT   = 40              # job limit of the queue
UJOB_LIMIT   = 30              # job limit per user
r1m          = 0.7/2.0         # loadSched/loadStop
ut           = 0.5
RUNLIMIT     = 8:00            # 8 wall-clock hours
MEMLIMIT     = 4000000         # jobs bigger than 4GB will be niced
USERS        = weili mauricfo   # users who can submit jobs to this queue
HOSTS        = c656f6n05       # hosts on which jobs in this queue can run
DESCRIPTION  = medium job, 8-hours max wall-time, 4GB max, run job with restrict
user and host
End Queue

Begin Queue
QUEUE_NAME   = large
PRIORITY     = 30
QJOB_LIMIT   = 25              # job limit of the queue
UJOB_LIMIT   = 25              # job limit per user
r1m          = 0.7/2.0         # loadSched/loadStop
ut           = 0.7
RUNLIMIT     = 240:00          # 10 days wall-clock hours
MEMLIMIT     = 8000000         # jobs bigger than 8GB will be niced
USERS        = dino            # users who can submit jobs to this queue
HOSTS        = c656f6n06       # hosts on which jobs in this queue can run
DESCRIPTION  = large job, 10-days max wall-time, 8GB max, run job with restrict
user and host
End Queue
```

User groups

LSF administrators can define roles and privileges for user groups. All definitions for users are stored in the `lsb.users` file. Two user groups, `student` and `prof`, are in this scenario. The group `student` includes three users, and the group `prof` has one defined user. The user `dino` has a maximum of 800 pending jobs. The group `student` has a total threshold of 1,000 pending jobs. By default, any user and group has a threshold of 100 jobs.

To define user groups in Platform LSF, you need to modify the `lsb.users` file as shown in Example 4-2 on page 133. This file is installed by default in the following location:

`<`*`LSB_CONFDIR`*`>/<`*`cluster_name`*`>/configdir`

Example 4-2 lsb.users file

```
Begin UserGroup
GROUP_NAME       GROUP_MEMBER              USER_SHARES
student          (weili mauricfo wainersm)  ()
prof             (dino)                     ()
End UserGroup

Begin User
USER_NAME        MAX_PEND_JOBS
dino             800            # dino has pend job threshold of 800
student          1000           # collectively this group has threshold of 1000
default          100            # default, any user/group has threshold of 100
End User
```

By running the **`badmin reconfig`** command after you update the definitions, the updated queues and user groups are enabled in the Platform LSF cluster. Platform LSF administrators and users can use Platform LSF commands to monitor the Platform LSF cluster.

4.1.2 Platform LSF administration commands

The common LSF administration commands are shown:

- `lsclusters`
- `lsid`
- `lshosts`
- `lsload`
- `bparams`
- `badmin`
- `bhosts`
- `bjobs`
- `bqueues`

lsclusters

Use the **`lsclusters`** command to identify the cluster administrator and to see a summary of the current cluster as shown in Example 4-3.

Example 4-3 lsclusters command output

```
$ lsclusters
CLUSTER_NAME   STATUS   MASTER_HOST            ADMIN    HOSTS  SERVERS
redbook-cluste ok       c656f6n05.pok.stgl     lsfadmin      2        2
```

lsid

Use the **`lsid`** command to show the cluster name, master host, and Platform LSF version in the current cluster as shown in Example 4-4 on page 134.

Example 4-4 lsid command output

```
$ lsid
IBM Platform LSF Advanced 9.1.3.0, Oct 30 2014
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 redbook-cluster
My master name is c656f6n05.pok.stglabs.ibm.com
```

lshosts

Use the **lshosts** command to display a list of the resources that are defined on a specific host as shown in Example 4-5.

Example 4-5 lshosts command output

```
$ lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
c656f6n05.p LINUXPP   POWER8 250.0    20 255.2G  10.7G    Yes (mg)
c656f6n06.p LINUXPP   POWER8 250.0    20 255.2G  10.7G    Yes (mg)
$ lshosts -l
HOST_NAME:  c656f6n05.pok.stglabs.ibm.com
type            model  cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server
nprocs ncores nthreads
LINUXPPC64      POWER8 250.0    20      1 255.2G  10.7G 255432M      0    Yes
1     20        8

RESOURCES: (mg)
RUN_WINDOWS:  (always open)

LOAD_THRESHOLDS:
  r15s   r1m  r15m   ut    pg    io   ls   it   tmp   swp   mem
     -   3.5     -    -     -     -    -    -     -     -     -

HOST_NAME:  c656f6n06.pok.stglabs.ibm.com
type            model  cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server
nprocs ncores nthreads
LINUXPPC64      POWER8 250.0    20      1 255.2G  10.7G 255432M      0    Yes
1     20        8

RESOURCES: (mg)
RUN_WINDOWS:  (always open)

LOAD_THRESHOLDS:
  r15s   r1m  r15m   ut    pg    io   ls   it   tmp   swp   mem
     -   3.5     -    -     -     -    -    -     -     -     -
```

lsload

Use the **lsload** command to display the current state of the hosts as shown in Example 4-6.

Example 4-6 lsload command output

```
$ lsload
HOST_NAME       status  r15s   r1m  r15m   ut    pg  ls    it   tmp   swp   mem
c656f6n06.pok.s     ok   0.0   0.0   0.0   1%   0.0   1     0  225G 10.7G  252G
c656f6n05.pok.s     ok  32.0  31.3  19.4  20%   0.0   1     0  203G 10.7G  252G
```

bparams

Use the **bparams** command to display the generic configuration parameters of Platform LSF. These parameters include the default queues, job dispatch interval, job checking interval, and job accepting interval as shown in Example 4-7.

Example 4-7 bparams command output

```
$ bparams
Default Queues:  normal interactive
MBD_SLEEP_TIME used for calculations:  10 seconds
Job Checking Interval:  7 seconds
Job Accepting Interval:  0 seconds
```

badmin

Use the **badmin showstatus** command to display a summary of the current Platform LSF runtime information about the entire cluster, including information about hosts, jobs, users, user groups, and mbatchd startup and reconfiguration, as shown in Example 4-8.

Example 4-8 badmin showstatus command output

```
$ badmin showstatus
LSF runtime mbatchd information
    Available local hosts (current/peak):
        Clients:               0/0
        Servers:               2/2
           CPUs:               2/2
          Cores:               40/40
          Slots:               30/30

    Number of servers:         2
        Ok:                    2
        Closed:                0
        Unreachable:           0
        Unavailable:           0

    Number of jobs:            0
        Running:               0
        Suspended:             0
        Pending:               0
        Finished:              0

    Number of users:           6
    Number of user groups:     2
    Number of active users:    0
    Latest mbatchd start:       Wed Dec 10 08:48:23 2014
    Active mbatchd PID:         21773

    Latest mbatchd reconfig:    Wed Dec 10 11:19:40 2014
```

bhosts

Use the **`bhosts`** command to display hosts and their static and dynamic resources. By default, this command returns the following information about all hosts: host name, host status, job state statistics, and job slot limits, as shown in Example 4-9.

Example 4-9 bhosts command output

```
$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
c656f6n05.pok.stgl ok              -     10      0      0      0      0      0
c656f6n06.pok.stgl ok              -     20      0      0      0      0      0
```

Typically, a user can also use the **`bhosts -l`** command to check whether the hosts configured enough slots for the jobs as shown in Example 4-10.

Example 4-10 bhosts -l command output

```
$ bhosts -l
HOST  c656f6n05.pok.stglabs.ibm.com
STATUS           CPUF  JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
DISPATCH_WINDOW
ok             250.00     -     10      0      0      0      0      0      -

 CURRENT LOAD USED FOR SCHEDULING:
              r15s   r1m  r15m    ut    pg    io   ls    it   tmp   swp   mem
slots
 Total         0.0   0.0   0.0    0%   0.0     1    3     5  202G 10.7G 251.9G
10
 Reserved      0.0   0.0   0.0    0%   0.0     0    0     0    0M    0M    0M
-


 LOAD THRESHOLD USED FOR SCHEDULING:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -      -      -      -
 loadStop    -     -     -     -       -     -    -     -      -      -      -


HOST  c656f6n06.pok.stglabs.ibm.com
STATUS           CPUF  JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
DISPATCH_WINDOW
ok             250.00     -     20      0      0      0      0      0      -

 CURRENT LOAD USED FOR SCHEDULING:
              r15s   r1m  r15m    ut    pg    io   ls    it   tmp   swp   mem
slots
 Total         0.0   0.0   0.0    0%   0.0     0    1    85  225G 10.7G 251.7G
20
 Reserved      0.0   0.0   0.0    0%   0.0     0    0     0    0M    0M    0M
-
```

```
LOAD THRESHOLD USED FOR SCHEDULING:
          r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
loadSched   -     -     -     -       -     -    -     -     -      -      -
loadStop    -     -     -     -       -     -    -     -     -      -      -
```

bjobs

Use the **bjobs** command to display job information. By default, the **bjobs** command displays information for the user who invoked the command.

You can run the **bjobs -u all** command to display all jobs for all users as shown in Example 4-11.

Example 4-11 bjobs -u all command output

```
$ bjobs -u all
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
396     weili   RUN   priority   c656f6n05.p c656f6n06.p prio1      Dec 10 12:06
392     weili   RUN   small      c656f6n05.p c656f6n06.p demo1      Dec 10 12:04
395     weili   RUN   normal     c656f6n05.p c656f6n06.p para1      Dec 10 12:05
                                             c656f6n06.pok.stglabs.ibm.com
                                             c656f6n06.pok.stglabs.ibm.com
                                             c656f6n06.pok.stglabs.ibm.com
```

You can run **bjobs -sum** to display summary information about unfinished jobs as shown in Example 4-12.

Example 4-12 bjobs -sum command output

```
$ bjobs -sum
RUN        SSUSP      USUSP      UNKNOWN    PEND       FWD_PEND
6          0          0          0          0          0
```

bqueues

The **bqueues** command displays the status of a particular queue or all queues. The **bqueues** command also displays the available queues in the cluster as shown in Example 4-13.

Example 4-13 bqueues command output

```
$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
priority         43  Open:Active      15    -    -    -     1     0     1     0
night            40  Open:Inact        -    -    -    -     0     0     0     0
small            38  Open:Active      80   40    -    -     1     0     1     0
medium           35  Open:Active      40   30    -    -     0     0     0     0
normal           30  Open:Active       -    -    -    -     4     0     4     0
large            30  Open:Active      25   25    -    -     0     0     0     0
interactive      30  Open:Active       -    -    -    -     0     0     0     0
idle             20  Open:Active       -    -    -    -     0     0     0     0
```

For more information about Platform LSF commands, see the following website:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_kc_cmd_ref.dita

4.2 xCAT cluster monitoring

For the xCAT component and its subcategories, this section presents examples of monitoring commands that are commonly used to obtain the status of services, nodes, and hardware availability. We also present the hardware discovery commands that are important when you perform problem determination tasks.

The commands that are shown in this section are organized into the following distinct areas:

- General view
- Hardware discovery
- Remote console

4.2.1 General view

The following commands are used for general viewing:

- `rpower`
- `nodestat`

rpower

For the **`rpower`** command, the help description is shown in Example 4-14.

Example 4-14 rpower command flags

```
# rpower -h
Usage: rpower <noderange> [--nodeps]
[on|onstandby|off|suspend|reset|stat|state|boot] [-V|--verbose] [-m
table.colum==expectedstatus][-m table.colum==expectedstatus...] [-r <retrycount>]
[-t <timeout>]
       rpower [-h|--help|-v|--version]
     KVM Virtualization specific:
       rpower <noderange> [boot] [ -c <path to iso> ]
     PPC (with IVM or HMC) specific:
       rpower <noderange> [--nodeps] [of] [-V|--verbose]
     CEC (with HMC) specific:
       rpower <noderange> [on|off|reset|boot|onstandby]
     LPAR(with HMC) specific:
       rpower <noderange> [on|off|reset|stat|state|boot|of|sms|softoff]
     CEC(using Direct FSP Management) specific:
       rpower <noderange> [on|onstandby|off|stat|state|resetsp]
     Frame(using Direct FSP Management) specific:
       rpower <noderange> [stat|state|rackstandby|exit_rackstandby|resetsp]
     LPAR(using Direct FSP Management) specific:
       rpower <noderange> [on|off|reset|stat|state|boot|of|sms]
     Blade(using Direct FSP Management) specific:
       rpower <noderange> [on|onstandby|off|cycle|state|sms]
     Blade(using AMM) specific:
       rpower <noderange> [cycle|softoff] [-V|--verbose]
     zVM specific:
       rpower noderange [on|off|reset|stat|softoff]
     MIC specific:
       rpower noderange [stat|state|on|off|reset|boot]
```

The **rpower** command remotely controls nodes and retrieves their status. The format to run the command is `rpower <noderange> <action>`.

Example 4-15 is the typical output of the **rpower** command. The output of the command displays the status of all of the machines in xCAT group `f6`.

Example 4-15 rpower command output example for all machines in the xCAT f6 group

```
# rpower f6 stat
c656f6n05: on
c656f6n06: on
```

nodestat

For the **nodestat** command, the help description is shown in Example 4-16.

Example 4-16 nodestat command flags

```
# nodestat -h
Usage:
  nodestat [noderange] [-m|--usemon] [-p|powerstat] [-u|--updatedb]
  nodestat [-h|--help|-v|--version]
```

The **nodestat** command returns information about the status of the node's operating system and power state. The **nodestat** command updates the xCAT database with the information. This command also supports custom application status through an xCAT table and flag. The following input and output values are used:

- Command:

  `nodestat <noderange>`

- Flags:

  -p: Returns information about the power state

  -m: Uses the xCAT monsetting table to monitor more services with customized outputs

  -u: Updates the xCAT database with returned values

- Outputs:
  - `[no-flag]`
  - `<noderange>: <status>`
  - `[-p]`
  - `<noderange>: <status>(<power_state>)`

- Attributes:

  – ***<noderange>***: Nodes that are listed in the xCAT database that belong to a hardware type.

  – ***<status>***: Information about the level of detail of the checked item.

  – ***<power_state>***: When the state is not on or running, the command reports the output of the **rpower *<noderange>* stat** command.

Example 4-17 shows typical examples of using the **nodestat** command to query nodegroup `f6` and a certain node with the *hostname* and option `-p`.

Example 4-17 nodestat command output

```
# nodestat f6
c656f6n05: sshd
c656f6n06: sshd
# nodestat c656f6n06 -p
c656f6n06: sshd
```

4.2.2 Hardware discovery

This section describes the hardware discovery command on the IBM POWER8 System S824L. The help description of the **lsslp** command is shown in Example 4-18.

Example 4-18 lsslp command flags

```
# lsslp -h
Usage: lsslp [-h|--help|-v|--version]
       lsslp [<noderange>][-V|--verbose][-i ip[,ip..]][-w][-r|-x|-z][-n][-I][-s
FRAME|CEC|MM|IVM|RSA|HMC|CMM|IMM2|FSP]
             [-u] [--range IPranges][-t tries][--vpdtable][-C counts][-T timeout]
```

The Flexible Service Processor (FSP)/baseboard management controllers (BMCs) are automatically powered on after the physical machine is powered on. Currently, the **lsslp** command can use Service Location Protocol (SLP) to identify all of the FSPs for the POWER8 LE (little-endian) host as shown in Example 4-19.

Example 4-19 lsslp command output

```
# lsslp -s FSP
device  type-model  serial-number  side  ip-addresses               hostname
fsp     8247-42L    211E4EA        A-0   10.128.128.64              10.128.128.64
fsp     8247-42L    211E4AA        A-0   10.128.128.66              10.128.128.66
```

4.2.3 Remote console

The **rcons** command is used to open a remote console to a node to monitor the installation progress or log in remotely. If the **rcons** command fails to open the remote console, the conserver might not be configured.

To monitor the network installation of a node, or to open a remote console of the node, first configure the conserver by using the **makeconservercf** command. Then, open a remote console to the node by using the following command:

```
rcons <nodename>
```

The output is shown in Example 4-20.

Example 4-20 rcons command output

```
# rcons c656f6n06
[Enter `^Ec?' for help]
Info: SOL payload already de-activated
[SOL Session operational.  Use ~? for help]

Ubuntu 14.10 c656f6n06 hvc0

c656f6n06 login:
```

In the remote console, enter `` `^Ec?' `` for help. To terminate the remote console, use `^Ec.'` to quit the session.

5

Application development

This section provides details about application development on an IBM POWER8.

The following sections are described in this chapter:

- Compilers
- Performance libraries
- IBM Parallel Environment Developer Edition
- IBM POWER8 features
- Migration considerations

5.1 Compilers

Several compilers, including IBM Xpertise Library (XL) compilers and the GNU Compiler Collection (GCC), support the latest IBM POWER8 processor features.

5.1.1 IBM XL compilers

IBM XL compilers are updated periodically to improve application performance and add processor-specific tuning and capabilities. IBM XL C/C++ v13.1.1 and XL Fortran v15.1.1 compilers are the first versions to support IBM POWER8 with little-endian Linux distributions.

XL C/C++/Fortran

XL C/C++ for Linux, v13.1.1 and XL Fortran for Linux, v15.1.1 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/SSXVZZ_13.1.1/com.ibm.compilers.linux.doc/welcome.html?lang=en

For more information about XL Fortran, see the following website:

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.1/com.ibm.compilers.linux.doc/welcome.html?lang=en

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++ v13.1.1, the available compiler options to maximize application development performance are described in Table 5-1.

Table 5-1 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 by calling the equivalent vector functions in the Mathematical Acceleration Subsystem libraries (MASS), 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 on Table 5-2, 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 5-2 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 (HOT) 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`, 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/SSXVZZ_13.1.1/com.ibm.xlcpp1311.lelinux.doc/compiler_ref/rucopt.html?lang=en

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.1/com.ibm.xlf1511.lelinux.doc/proguide/optimization.html?lang=en

OpenMP 4.0 support

The IBM XL C/C++ for Linux v13.1.1 and the IBM XL Fortran for Linux v15.1.1 fully support the OpenMP Application Program Interface (API) version 3.1 specification and partially support the OpenMP API version 4.0 specification. The XL C/C++/Fortran implementation is based on the IBM interpretation of the OpenMP API version 4.0.

This version of XL C/C++ supports the following OpenMP 4.0 features:

- The `update` clause and `capture` clause enhancements

  The `update` and `capture` clauses of the atomic construct are extended to support more expression forms.
- **`OMP_DISPLAY_ENV`** environment variable

  You can use the **`OMP_DISPLAY_ENV`** environment variable to display the values of the internal control variables (ICVs) that are associated with the environment variables and the build-specific information about the runtime library.

This version of XL Fortran supports the following OpenMP 4.0 features:

- The `capture` clause enhancements

  The `capture` clause of the atomic construct is extended to support more syntax forms.
- **`OMP_DISPLAY_ENV`** environment variable

  You can use the **`OMP_DISPLAY_ENV`** environment variable to display the values of the ICVs that are associated with the environment variables and the build-specific information about the runtime library.

5.1.2 GCC and the IBM Advance Toolchain

For GCC compilers on Linux, the IBM Advance Toolchain 8.0 (GCC-4.9.2) enables POWER8 with support for both big endian (ppc64) and little endian (ppc64le).

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-4.9 through the `-mcpu=power8` and `-mtune=power8` options.

Specifying the `-mveclibabi=mass` option and linking to the Mathematical Acceleration Subsystem (MASS) libraries enables more loops for `-ftree-vectorize`. The MASS libraries support only static archives for linking. So, 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-4.9/

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

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/IBM%20Advance%20Toolchain%20for%20PowerLinux%20Documentation.co/1CsNsDs

Optimization parameters

The following commonly used optimization options are shown in Table 5-3.

Table 5-3 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.
-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.

5.2 Performance libraries

Many numerical libraries for high-performance mathematical computing are provided by IBM. These advanced software components can provide significant performance improvements for certain workloads on POWER processor-based systems.

5.2.1 ESSL and Parallel ESSL

Engineering and Scientific Subroutine Library (ESSL) is a collection of high performance mathematical subroutines that provide a wide range of functions for any common scientific and engineering applications. Parallel ESSL is a scalable mathematical subroutine library for stand-alone clusters or clusters of servers that connect through a switch.

All of the libraries are designed to provide high levels of performance for numerically intensive computing jobs and to provide mathematically equivalent results. The ESSL subroutines are called from application programs, which are written in Fortran, C, and C++, that run on AIX and Linux operating systems.

Example 5-1 is Fortran example source code about Gaussian elimination without pivoting. This code uses three methods to implement the algorithm and compare the execution times:

- Straightforward formulas
- Fortran 90/95 features
- Basic Linear Algebra Subprograms (BLAS) routines

Example 5-1 ESSL example source code essl_demo.f90

```
! Gaussian elimination without pivoting:
! straightforward formulas, Fortran 90/95 features, BLAS routines
!   ELIMINATION
!   for k = 1: n-1
!       for i = k+1: n
!           a(i, k) /= a(k, k)
!       for i = k+1: n
!           for j = k+1: n
!               a(i, j) -= a(i, k) * a(k, j)
!           end
!       end
!   end
!   BACKSUBSTITUTION  -  L U y = f  =>  L x = f  =>  x = L \ f  =>  y = U \ x
!   for i = 1: n
!       x(i) = f(i)
!       for j = 1: i-1
!           x(i) -= a(i, j) * x(j)
!       end
!   end
!   for i = n: 1
!       y(i) = x(i)
!       for j = n: i+1
!           y(i) -= a(i, j) * y(j)
!       end
!       y(i) /= a(i, i)
!   end
!!-----------------------------------------------------------------------
program ge
!!-----------------------------------------------------------------------
    use omp_lib, only: omp_get_wtime
    implicit none
    ! Matrix of coefficients; the one is filled by random_number()
    real, dimension(:, :), allocatable :: A
    ! "Analytical" solution; the one is filled by random_number()
    real, dimension(:), allocatable :: u
    ! Right-hand side (RHS); the one is calculated as f = A * u
    ! Numerical solution (NS) of the equation A y = f
    ! RHS is overwritten by NS
    real, dimension(:), allocatable :: y

    ! Size of matrix
    integer, parameter :: n = 500

    ! Time marks
    real(kind(0.d0)) :: elimination_start, elimination_finish
    real(kind(0.d0)) :: backsubstition_start, backsubstition_finish

    ! Allocate memory
    allocate(A(1: n, 1: n))
    allocate(u(1: n))
    allocate(y(1: n))

    ! Algorithm uses straightforward formulas
    call Generate_Data()
    call Straightforward_Elimination()
    call Straightforward_Backsubstition()
    call Print_Norms()
    call Print_Times()

    ! Algorithm uses Fortran 90/95 features
    call Generate_Data()
    call Fortran9x_Elimination()
    call Fortran9x_Backsubstition()
    call Print_Norms()
    call Print_Times()

    ! Algorithm uses BLAS
    call Generate_Data()
    call BLAS_Elimination()
    call BLAS_Backsubstition()
    call Print_Norms()
    call Print_Times()

    ! Free memory
    deallocate(A)
    deallocate(u)
    deallocate(y)
!!-----------------------------------------------------------------------------
contains
!!-----------------------------------------------------------------------------
subroutine Print_Norms()
    write (*, *) "Norms:", maxval(abs(u)), maxval(abs(y - u))
end subroutine Print_Norms
!!-----------------------------------------------------------------------------
subroutine Print_Times()
    write (*, *) "Times:", &
            elimination_finish - elimination_start, &
            backsubstition_finish - backsubstition_start
end subroutine Print_Times
!!-----------------------------------------------------------------------------
! This version is a simplified modification of
! http://gcc.gnu.org/onlinedocs/gfortran/RANDOM_005fSEED.html
subroutine Init_Random_Seed()
    integer :: i, n
    integer, dimension(:), allocatable :: seed

    call random_seed(size = n)
    allocate(seed(n))

    seed = 37 * (/ (i - 1, i = 1, n) /)
    call random_seed(put = seed)

    deallocate(seed)
end subroutine Init_Random_Seed
!!---------------------------------------------------------------------------
subroutine Generate_Data()
    call Init_Random_Seed()
    call random_number(A)
    call random_number(u)
    y = matmul(A, u)
end subroutine Generate_Data
!!---------------------------------------------------------------------------
subroutine Straightforward_Elimination()
    integer :: i, j, k

    elimination_start = omp_get_wtime()

    do k = 1, n-1
        do i = k+1, n
            a(i, k) = a(i, k) / a(k, k)
        end do

        do j = k+1, n
            do i = k+1, n
                a(i, j) = a(i, j) - a(i, k) * a(k, j)
            end do
        end do
    end do

    elimination_finish = omp_get_wtime()
end subroutine Straightforward_Elimination
!!---------------------------------------------------------------------------
subroutine Fortran9x_Elimination()
    integer :: k

    elimination_start = omp_get_wtime()

    do k = 1, n-1
        a(k+1: n, k) = a(k+1: n, k) / a(k, k)

        a(k+1: n, k+1: n) = a(k+1: n, k+1: n) - &
                matmul(a(k+1: n, k: k), a(k: k, k+1: n))
    end do

    elimination_finish = omp_get_wtime()
end subroutine Fortran9x_Elimination
```

```
!!---------------------------------------------------------------------------
subroutine BLAS_Elimination()
    integer :: k

    elimination_start = omp_get_wtime()

    do k = 1, n-1
        ! x = a*x
        call sscal(n-k, 1.0 / a(k, k), a(k+1, k), 1)

        ! A := alpha*x*y'+ A
        call sger(n-k, n-k, -1.0, &
                a(k+1, k), 1, &
                a(k, k+1), n, &
                a(k+1, k+1), n)
    end do

    elimination_finish = omp_get_wtime()
end subroutine BLAS_Elimination
!!---------------------------------------------------------------------------
subroutine Straightforward_Backsubstition()
    integer :: i, j

    backsubstition_start = omp_get_wtime()

    ! L x = f  =>  x = L \ f
    do i = 1, n
        do j = 1, i-1
            y(i) = y(i) - a(i, j) * y(j)
        end do
    end do

    ! U y = x  =>  y = U \ x
    do i = n, 1, -1
        do j = i+1, n
            y(i) = y(i) - a(i, j) * y(j)
        end do

        y(i) = y(i) / a(i, i)
    end do

    backsubstition_finish = omp_get_wtime()
end subroutine Straightforward_Backsubstition
!!---------------------------------------------------------------------------
subroutine Fortran9x_Backsubstition()
    integer :: i

    backsubstition_start = omp_get_wtime()

    ! L x = f  =>  x = L \ f
    do i = 1, n
        y(i) = y(i) - dot_product(a(i, 1: i-1), y(1: i-1))
    end do

    ! U y = x  =>  y = U \ x
    do i = n, 1, -1
        y(i) = y(i) - dot_product(a(i, i+1: n), y(i+1: n))
        y(i) = y(i) / a(i, i)
    end do

    backsubstition_finish = omp_get_wtime()
end subroutine Fortran9x_Backsubstition
!!---------------------------------------------------------------------------
subroutine BLAS_Backsubstition()
    backsubstition_start = omp_get_wtime()
    call strsv('L', 'N', 'U', n, a, n, y, 1)
    call strsv('U', 'N', 'N', n, a, n, y, 1)
    backsubstition_finish = omp_get_wtime()
end subroutine BLAS_Backsubstition
!!---------------------------------------------------------------------------
end program ge
```

ESSL is included by adding **`-lessl`** on the link step. To compile the source code *essl_demo.f90*, use the following command:

```
xlf_r essl_demo.f90 -qsmp -O2 -lessl -o essl_demo
```

The result is shown in Example 5-2.

Example 5-2 Execution time result with -lessl

```
$ xlf_r essl_demo.f90 -qsmp -O2 -lessl -o essl_demo
** ge   === End of Compilation 1 ===
1501-510  Compilation successful for file essl_demo.f90.
$ ./essl_demo
 Norms: 0.9980875254 0.1525531709
 Times: 0.143321037292480469 0.156497955322265625E-02
 Norms: 0.9980875254 0.1188478172
 Times: 4.03329610824584961 0.550985336303710938E-03
 Norms: 0.9980875254 0.3463452756
 Times: 0.572490692138671875E-02 0.529289245605468750E-04
```

When you compare the execution-time results, you can see that ESSL helps to dramatically reduce the execution time from `0.143321037292480469` to `0.572490692138671875E-02`.

XL Fortran is shipped with the BLAS library in the `libxlopt` library for high-performance computing (HPC). By using XL Fortran, the optimizer automatically uses the ESSL routines instead of Fortran 90 intrinsic procedures by using the `-qessl` option. For example, compile the source code that is shown in Example 5-1 on page 149 with the following command:

```
xlf_r essl_demo.f90 -qsmp -O2 -lessl -qessl -o essl_demo2
```

The result is shown in Example 5-3.

Example 5-3 Execution-time result with -lessl and -qessl

```
$ xlf_r essl_demo.f90 -qsmp -O2 -lessl -qessl -o essl_demo2
** ge   === End of Compilation 1 ===
1501-510  Compilation successful for file essl_demo.f90.
$ ./essl_demo2
 Norms: 0.9980875254 0.1207416207
 Times: 0.131722927093505859 0.122308731079101562E-02
 Norms: 0.9980875254 0.1376869082
```

```
Times: 0.125740051269531250 0.118207931518554688E-02
Norms: 0.9980875254 0.1039237007
Times: 0.562596321105957031E-02 0.510215759277343750E-04
```

Compare the results of the execution time in Example 5-2 on page 153 and Example 5-3 on page 153. The `-qessl` option helps optimize the execution time of the Fortran intrinsic procedures functions from `4.03329610824584961` to `0.125740051269531250`.

5.2.2 MASS

The XL Fortran compiler is shipped with a set of Mathematical Acceleration Subsystem (MASS) libraries for high-performance mathematical computing.

The MASS libraries consist of a library of scalar Fortran routines, a set of vector libraries that are tuned for specific architectures, and a set of single-instruction, multiple-data (SIMD) libraries that are tuned for specific architectures. The functions that are contained in both scalar and vector libraries are automatically called at certain levels of optimization, but you can also call them explicitly in your programs.

Example 5-4 shows Fortran example source code to use the MASS library.

Example 5-4 Fortran example source code to use a MASS library

```
program test_mass
  implicit none
  include 'massv.include'
  integer, parameter :: n = 1024 * 1024 * 1024
  integer, parameter :: wp = kind(0.0d0)
  integer :: i
  real(wp), dimension(:), allocatable :: x, s, c
  real :: t1, t2

  allocate(x(1: n), s(1: n), c(1: n))

  call random_number(x)
  call cpu_time(t1)
  do i = 1, n
    s(i) = sin(x(i))
    c(i) = cos(x(i))
  end do
  call cpu_time(t2)
  print *, "Naive implementation: ", t2 - t1

  call cpu_time(t1)
  call vsincos(s, c, x, n)
  call cpu_time(t2)
  print *, "MASS: ", t2 - t1

  deallocate(x, s, c)
end program test_mass
```

To compile an application that calls the functions in the scalar, SIMD, or vector MASS libraries, specify one or more of the following library names on the `-l` linker option.

For example, to compile the source code `test_mass.F90`, use the command:

```
xlf -O3 -qarch=pwr8 test_mass.F90 -lmass -lmassvp8
```

Example 5-5 shows the result. The execution time is reduced from `10.63999939 seconds` to `5.609998703` seconds.

Example 5-5 Showing the result of the command execution

```
$xlf -O3 -qarch=pwr8 test_mass.F90 -lmass -lmassvp8
** test_mass   === End of Compilation 1 ===
1501-510  Compilation successful for file test_mass.F90.
$ ./a.out
 Naive implementation:  10.63999939
 MASS:  5.609998703
```

5.3 IBM Parallel Environment Developer Edition

IBM Parallel Environment Developer Edition (PE DE) is an ecosystem of tools for parallel application development in C, C++, and Fortran. PE DE offers environments and tools that cover all of the steps in the development of parallel applications, including the following steps:

- Code programming
- Build
- Launch
- Debug
- Performance analysis
- Tuning

PE DE consists of the following components:

- The PE DE Workbench provides an integrated development environment (IDE) that is based on Eclipse Parallel Tools Platform (PTP).
- The High Performance Computing Toolkit (HPCT) provides a rich set of profile and trace tools. It also includes a visual analysis tool that is called *hpctView*.

The PE DE Workbench

The workbench provides an IDE for Power scale-out/Linux servers that is based on Eclipse PTP. It supports the development of C, C++, and Fortran parallel applications with the Message Passing Interface (MPI), Open MP, and OpenSHEM APIs.

The Fortran application code editors feature the Eclipse Photran. The C/C++ application code editors feature the C/C++ Development Tools (CDT) projects. The features include the following examples:

- Syntax highlighting
- Content assistant
- Outline view
- Place markers for compiler error messages

These editors are also extended with the PTP Parallel Language Development Tools (PLDT), which provide assistants and static analysis tools to help edit of MPI, OpenMP, and OpenSHEM code.

For more information about Photran, see the *Fortran Development User Guide* at the following website:

http://help.eclipse.org/luna/index.jsp

For more information about CDT, see the *C/C++ Development User Guide* at the following website:

http://help.eclipse.org/luna/index.jsp

For more information about PTP PLDT, select **Parallel Development User Guide** → **Parallel Language Development Tools (PLDT)** at the following website:

http://help.eclipse.org/luna/index.jsp

PE DE is installed at the developer workstation. Application coding is performed locally within the Eclipse workbench. Other tasks that are required to be performed on the server side (usually, the HPC login node), for example, build, run, debug, and profile, are actually performed remotely. Therefore, PE DE implements the remote development working model that is illustrated in Figure 5-1.

PE DE uses a synchronized project type to keep local and remote copies of the working directory updated so that code programming is not 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.

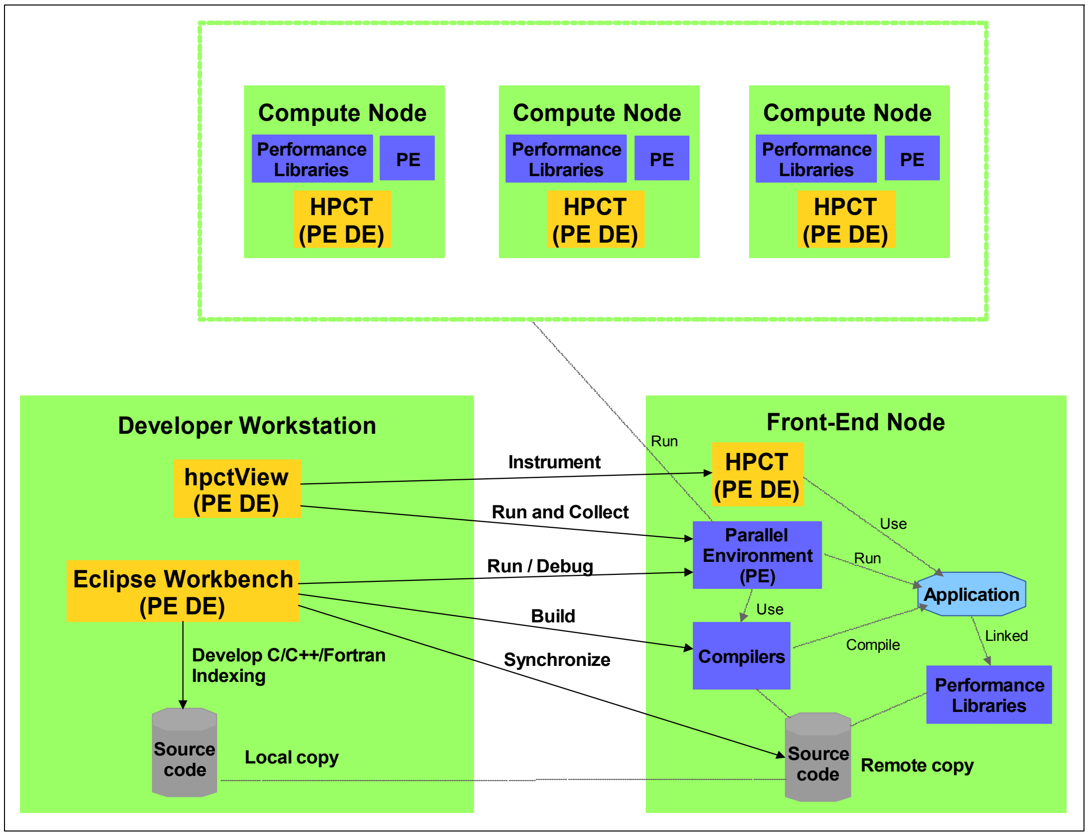


Figure 5-1 PE DE overview

Parallel applications that are written in C, C++, and Fortran can be built from within the PE DE environment easily. PE DE already provides the integration for both XL and GNU compilers to build makefile projects that are managed by Eclipse. Eclipse CDT non-managed makefile or GNU autotool projects are supported, too. In 5.1, “Compilers” on page 144, several options are available to help you better explore the compile features that target the POWER8 processor.

PE DE includes an Eclipse PTP scalable parallel debugger, which is able to launch and remote debug parallel applications by using several types of job schedulers. The debugger provides specific debugging features for parallel applications that distinguish it from the Eclipse debugger for serial applications. For more information about the PE DE parallel debugger, see 7.4, “Debugging tools” on page 188.

For more information about how to set up projects, and build and run parallel applications, see the Eclipse PTP help manual, *Parallel Development User Guide*, at the following website:

`http://help.eclipse.org/luna/index.jsp`

Table 5-4 contains a summary of the supported platforms, integrated software levels, and base components that are bundled within PE DE, version 2.1.

Table 5-4 PE DE V2.1 supported component matrix

Area	Software
Development workstation	► Microsoft Windows 7 64 bit ► Mac OS X 10.9 (Mavericks) or later ► Linux x86 64 bit
Parallel run time	IBM Parallel Runtime Edition 2.1.0 or later
Workload manager and job scheduler	IBM Platform Load Sharing Facility (LSF) 9.1.1 or later
Target operating system	Ubuntu 14.04.01 little endian for Power Systems
Compiler	► IBM XL C 13.1.1 or later ► IBM XL Fortran 15.1.1 ► GNU C/C++/Fortran 4.9
Base Eclipse bundles	► Platform 4.4 ► CDT 8.4 ► Photran 9.0 ► PTP 8.1

The IBM HPC Toolkit

You can tune code to improve performance by using a set of profiling and tracing tools that spot problem areas and pain points in the code. You can use these tools to identify potential improvements that take advantage of the strengths of POWER8. The tools are powered by the IBM HPC Toolkit (HPCT) and integrate well with the PE DE user interface. The following tools are available:

- Call graph analysis
- MPI profiling tracing
- OpenMP tracing
- I/O tracing
- Hardware Performance Monitor profiling

Accompanying the toolkit, *hpctView* is a tool that provides an environment to visualize collected profile and trace data. It also includes facilities to prepare the parallel application with instrumentation and then to launch the parallel application to gather data.

The HPCT profiling and tuning tools are described in 7.2.3, "IBM HPC Toolkit" on page 182.

For more information and installation details about HPCT, see the IBM Knowledge Center website. Select **Cluster software** → **Parallel Environment Developer Edition** → **Parallel Environment Developer Edition 2.1.0** → **PE Developer Edition 2.1** → **PE Developer Edition** → **PE Developer Edition HPC Toolkit: Installation and Usage Guide**:

http://www.ibm.com/support/knowledgecenter

5.4 IBM POWER8 features

Many capabilities are unique for IBM POWER architecture. This section outlines how to use these capabilities in application code.

5.4.1 AltiVec

The POWER architecture provides vector and vector-scalar operations through the VMX and VSX instruction sets, which are specified on Power Instruction Set Architecture (ISA) 2.07. These instruction sets are the foundations of the GNU GCC and IBM XL implementation of scalar code optimizations and the AltiVec API specification.

Implementation with GNU GCC

The instruction sets are used by GNU/GCC C to implement the AltiVec API specification with modifications. The modifications 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 with the `-maltivec` option at least. 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:

- Added 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 single-instruction, multiple-data (SIMD) instructions for the hardware.

Table 5-5 shows vector data type extensions as implemented by GCC. Vector types are signed by default, unless an otherwise `unsigned` keyword is specified. The only expectation 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 5-5 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 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:

http://gcc.gnu.org/onlinedocs/gcc-4.9.0/gcc/PowerPC-AltiVec_002fVSX-Built-in-Functions.html

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.

Similarly with 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 5-5 on page 159.

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 the XL C/C++ vector support and other topics, such as vector initialization and the `vec_step` operator, see the following page of the IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.1/com.ibm.xlcpp1311.lelinux.doc/language_ref/altivec_exts_both.html

A comprehensive list of built-in functions for vector operations is available at the following website:

http://www.ibm.com/support/knowledgecenter/SSXVZZ_13.1.1/com.ibm.xlcpp1311.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 too. They must be only used within vectors.

Vectors are automatically aligned to 16 bytes, but exceptions apply. For more information about vector types, see the following website:

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.1/com.ibm.xlf1511.lelinux.doc/language_ref/vectordatatype.html

For the list of vector intrinsic procedures, see the following website:

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.1/com.ibm.xlf1511.lelinux.doc/language_ref/vmxintrinsics.html

5.4.2 Using decimal point floating unit

Binary floating point datatypes in most programming languages (for example, `float` and `double` in C/C++, and `REAL` in Fortran) fulfill most of the needs for technical computing workloads. However, several problems have stricter demands on the precision of non-integer computations that cannot be satisfied with binary floating point arithmetic. For example, to comply with legal requirements, the results of calculations in financial, commercial, and trading applications often need to match the results that can be calculated manually[1].

Example 5-6 lists a source code example of a simple program that demonstrates the effect of floating point rounding errors.

Example 5-6 Source code bin-vs-dec.c to demonstrate the floating point rounding errors

```
#include <stdio.h>

int main() {
    double b1 = 0.1, b2 = 0.2;
    printf("b1 + b2 ? %d\n", (b1 + b2) == 0.3);

    _Decimal64 d1 = 0.1dd, d2=0.2dd;
    printf("d1 + d2 ? %d\n", (d1 + d2) == 0.3dd);

    return 0;
}
```

When you compile and run the program, you see that binary floating point arithmetic does not provide the expected results:

```
$> gcc bin-vs-dec.c
$> ./a.out
b1 + b2 ? 0
d1 + d2 ? 1
```

The result is unexpected because most floating point numbers cannot be represented by using a finite number of binary digits. The following example shows the result if we expand two simple decimal numbers over powers of two:

$0.75 = 0.5 + 0.25 = 1/2^1 + 1/2^2 = 0.11_2,$
$0.3 = 0.25 + 0.03125 + 0.015625 + \ldots = 1/2^2 + 1/2^5 + 1/2^6 + \ldots = 0.010011\ldots_2,$

We immediately see that 0.75 can be represented precisely; however, 0.3 requires an infinite number of binary digits.

To mitigate the inherent deficiency of binary floating point datatypes, you can choose decimal floating point datatypes to perform precision-sensitive computations. Example 5-6 uses the `_Decimal64` datatype for this purpose.

Typically, decimal floating point arithmetic is emulated in software. The IBM POWER8 processor (with its predecessors, the IBM POWER6® and the IBM POWER7 processors) core has a built-in decimal floating point (DFP) unit that is designed to speed up operations with decimal datatypes[2].

[1] For more details, see “Frequently Asked Questions” at http://speleotrove.com/decimal/.
[2] The recent IBM z™ Systems processors (z10™, z196, z12, and z13™) also implement decimal floating point arithmetic fully in hardware.

The efficiency of the hardware implementation heavily relies on the work of Mike Cowlishaw, a retired IBM Fellow, who developed a decimal floating point format that is known as *densely packed decimal* (DPD) encoding. DPD encoding is part of the IEEE Standard for Floating-Point Arithmetic (IEEE 754-2008).

For more information about how to take advantage of decimal floating point arithmetic on POWER processors, see *Performance Optimization and Tuning Techniques for IBM Processors, including IBM POWER8*, SG24-8171.

5.4.3 Memory model and storage synchronization considerations

One of the most important concurrent programming concepts of the multi-core age is the shared memory consistency model. Most of the modern programming languages used eventually specialized memory models as part of the standards (C11, C++11, and Java 5.0). The familiarity with this concept is important to write correct and efficient code.

If you are a programmer who uses POSIX threads, you can skip this paragraph unless you protect *every* access to *every* shared variable by using POSIX thread synchronization routines.

OpenMP programmers need to be familiar with the *Memory model* section of the OpenMP specifications that are published by the OpenMP Architecture Review Board[3] and need to augment their knowledge with the content of this paragraph.

The memory model that is implied by the MPI remote memory access operations relates to the current paragraph to a lesser extent. Nevertheless, you need to ensure that your understanding of the *Memory model* section of “MPI: A Message-Passing Interface Standard”, which is published by Message Passing Interface Forum[4], is consistent with the following content.

Concepts, such as “memory ordering”, “memory barriers”, and “atomicity” are relevant after a program uses any form of access to shared memory or thread synchronization. Regardless of the target computer architecture, a software developer needs to understand the respective specifications to write correct code.

When the interaction between the threads becomes a performance bottleneck, lock-free algorithms are sometimes used. The correctness of lock-free algorithms is established by careful reasoning on the order in which threads update and see the values of memory locations. A compiler and a processor are free to rearrange memory accesses if it does not break the sequential semantics.

The POWER8 processor provides extended features that a programmer can use to the benefit of lock-free algorithms. To take full advantage of POWER8 multithreading capabilities, a software developer needs to be aware that the POWER architecture specifies a so-called “weak memory consistency” model. The weak memory consistency implies that a processor is free to reorder any combination of reads and writes that carry no sequential dependency. Therefore, a greater number of C11/C++11 memory ordering specifications have hardware support on POWER. So to fully use the architecture, the weakest possible C11/C++11 memory ordering can be used in lock-free algorithms. For the details, see the Storage model chapter of the *Power Instruction Set Architecture* document[5].

[3] “The OpenMP API specification for parallel programming”, http://openmp.org
[4] “Message Passing Interface (MPI) Forum”, http://www.mpi-forum.org
[5] Power.org, http://www.power.org

The most portable way to write multithreaded programs is to use the capabilities of the standard language library:

- The C11 revision of the C programming language enables support of multithreaded programming through the `<stdatomic.h>` and `<threads.h>` header files.
- The C++11 revision of the C++ programming language enables support of multithreaded programming by using `<atomic>` and `<thread>` headers.

Several of the most common techniques to consider when you develop multithreaded applications in C and C++ are described:

- Never use the `volatile` keyword as a way to specify the variable to use for thread synchronization. The `volatile` keyword does not guarantee the atomicity of memory access. It also does not imply any memory barriers. Think of the `volatile` keyword as a mechanism to interact with the hardware, signals, and interruptions.
- Generally, atomicity of the operation does not guarantee a particular memory ordering unless explicitly specified. The C11/C++11 `atomic_fetch_add_explicit()` and related functions accept the memory ordering that you want as an argument. For example, the `memory_order_relaxed` ordering can be used for updating counters that are not used for synchronization. In contrast, the `atomic_fetch_add()` function has only sequential consistency and therefore enforces memory barriers.
- To establish runtime memory synchronization ordering regardless of any expressions, use the C11/C++11 `atomic_thread_fence()` function.
- Using only the compile-time memory barrier (such as GCC `asm volatile("" ::: "memory");`) does not prevent runtime memory reordering by a processor.

The development of lock-free algorithms is a complex topic regardless of a target platform. We strongly advise that you examine openly available code, for example, Linux operating system kernel sources, before you write production-level code.

5.4.4 Controlling prefetching with the Data Stream Control Register

Prefetch is a well-known strategy that can be used by either the processor (hardware-detected) or application (software-controlled) to attenuate data transfer latency from memory and between the various cache levels. The POWER8 prefetch engine is implemented by the Load-Store Unit, which is able to maintain three kinds of data streams:

- Load stream: Detect sequential access patterns and feed cache lines into L1, L2, and L3 levels ahead of current memory demands.
- Store stream: Detect sequential store patterns.
- Stride-*N*: Detect non-sequential but strided access patterns.

Prefetch uses a confirmation mechanism to determine whether an application increases or decreases subsequent accesses to cache lines so that it controls the buildup of the data stream (from initialization until it achieves steady state) and its depth.

The Data Stream Control Register (DSCR) is a Special-Purpose Register (SPR) design to provide a means to supply information and controls to the POWER data prefetch mechanism. It is also used to change software-controlled prefetch settings. The register has 64 bits and its layout, as specified in Power ISA version 2.07, has the following fields:

- Bit 39: Software transient enable
- Bit 40: Hardware transient enable
- Bit 41: Store transient enable
- Bit 42: Load transient enable

- Bit 43: Software count enable
- Bit 44: Hardware count enable
- Bits 45 - 54: Unit count
- Bits 55 - 57: Depth attainment urgency
- Bit 58: Load Stream disable
- Bit 59: Stride-N stream detect
- Bit 60: SSE
- Bit 61 - 63: Default prefetch depth

If the register enabled any of its transient bits (39 - 42), these bits give hints to the prefetch engine that accesses are likely to be short. The count bits (43 - 54) are used to change the default number of units in either the software or hardware streams. The depth attainment urgency field specifies how quickly the hardware-detected stream's depth can be reached. Its 3 bits accept values (in decimal) 0 - 7 that indicate one of the available urgency profiles (in order): *default*, *not urgent*, *least urgent*, *less urgent*, *medium*, *urgent*, *more urgent,* and *most urgent*. Bits 58, 59, and 60 control stride-*N* load and store streams. Last, the prefetch stream depth field (3 bits) accepts values (in decimal) 0 - 7 also, which correspond to one of the available profiles (in order): *default*, *none*, *shallowest*, *shallow*, *medium*, *deep*, *deeper,* and *deepest*.

Table 5-6 shows the action that is taken by several values (in decimal) that are assigned to DSCR.

Table 5-6 Effects of several values that are specified to DSCR

DSCR value (decimal)	Action	Description
0	Apply default setting	All bits zeroed.
1	Disable hardware-detected streams	Prefetch stream depth is set to *none*.
2 - 7	Change prefetch stream depth	2=shallowest, 3=shallow, 4=medium, 5=deep, 6=deeper, and 7=deepest.
32	Disable hardware-detected load streams	Set bit 58 (Load Stream Disable).
64 * *N*	Change depth attainment urgency	Where *N* = 1 means not urgent, *N* = 2 means least urgent, *N* = 3 means less urgent, *N* = 4 means medium urgent, *N* = 5 means urgent, *N* = 6 means more urgent, and *N* = 7 means most urgent.

The POWER8 prefetch engine is powerful but its default settings cannot show the best results for all applications because their storage access patterns vary. However, the POWER8 prefetcher is extremely configurable. You can fine-tune the engine functions to the best match according to the application's characteristics. The following tips can be handy when you experiment with different prefetch settings:

- Turn on stride-*N* streams when the application presents strided access patterns.
- Performance gains are possible by disabling the hardware-detected streams when the application uses software-controlled prefetches heavily.
- Applications with a high ratio of stream mispredicts likely benefit from a less urgent depth attainment profile. Higher urgency profiles benefit streams with short-to-medium depth.

Higher-level interfaces to ease the handling of the DSCR register are available other than inserting inline assembly in your application. These interfaces are described.

Data Stream Control Library

The `libpaf-dsc` (Data Stream Control Library) library provides C functions and macro handlers for dscr values. The following operations are examples:

- Check prefetch support
- Turn on/turn off prefetching
- Set different profiles
- Change data stream depth

The library is part of the IBM Power Architecture® Facilities (PAF) project that is hosted at the following website:

http://github.com/paflib/paflib

The library is also distributed within the IBM Advance Toolchain for Linux on POWER.

DSCR support in the IBM XL compiler

The IBM XL compiler provides DSCR build flags and built-in functions that target either C/C++ or Fortran applications.

Using the `dscr` suboption of the **`-qprefetch`** flag causes the compiler to set dscr with a specified value. But it is valid only when `-mcpu=pwr8` is set and the optimization level is equal or greater than `-O2`.

For the complete documentation of the DSCR built-in functions for Fortran, see the following website and select **Language Reference** → **Hardware-specific directives** → **PREFETCH**:

http://www.ibm.com/support/knowledgecenter/SSAT4T_15.1.1/com.ibm.compilers.linux.doc/welcome.html

Toggling DSCR on a process basis

Use the **`ppc64_cpu --dscr=`*`<value>`*** command-line option to set DSCR with a `<value>` so that child processes will inherit it. The **`-p`** ***`<pid>`*** argument can be supplied to apply the change for only the process that is running with the specified process identifier (PID).

Note: If a process sets DSCR on its own, the command has no effect.

The command **`ppc64_cpu --dscr=1`** turns off all prefetch. See Table 5-6 on page 164 for several values for DSCR.

5.5 Migration considerations

You must consider the following questions before you start a code migration to the IBM POWER architecture from a non-POWER architecture:

- Does the code work correctly on the non-POWER platform?
- What type of operating system was originally used?
- What Linux distribution was originally used?
- Does the code rely on hardware accelerators, such as graphics processing units (GPU) or Field Programmable Gate Arrays (FPGA)?
- Does the code rely on non-Linux specific features or system libraries?
- Does the code depend on software libraries that are not part of a Linux distribution?

- Does the code use static linkage or shared libraries?
- What programming language is the code written in?
- Does the code use machine-specific instructions (inline assembly blocks or intrinsics)?
- Does the code have memory-endianness dependence?
- Does the code perform binary file or network input/output operations?
- What type of threading does the code use (vanilla pthreads or C11/C++11 threads)?
- Does the code protect access to shared variables with synchronization routines?
- Does the code use lock-free techniques?
- Does the code rely on the shared memory consistency model of the original architecture?
- Does the code rely on the sign of the C/C++ `char` datatype in arithmetic operations?
- Does the code use a compiler macro to distinguish between different target architectures?

The list is not complete. Its intention is to provide general ideas to start with.

6

Running applications

This chapter describes practical workload scenarios about how to run different applications on the IBM Power System S824L cluster. In this book, Platform Load Sharing Facility (LSF) works as workload management platform to execute these jobs. Typical IBM high-performance computing (HPC) solution users run both serial and parallel jobs, including Message Passing Interface (MPI), OpenMP, and Hybrid (MPI + OpenMP) applications on the cluster.

This chapter contains the following topics:

6.1 Running serial applications

To run applications in the HPC cluster, users can use the **bsub** command to submit a job to the Platform LSF job queue. Users can run the **bsub** command with a spool file or interactively by using a script or the command line.

6.1.1 Run batch jobs by using a job file

Users can submit to the Platform LSF cluster by using a job file. A *job file* is a batch script that describes how you want to run your application by using a Platform LSF keyword statement. The **bsub** command has many options to submit the job. If the users do not specify any options, the job is submitted to the default queue (usually the queue normal) that is configured by the Platform LSF administrator.

Example 6-1 is a sample Platform LSF job file. It defines the output files and the application for this job.

Example 6-1 Sample Platform LSF job file: myserial.lsf

```
#!/bin/sh
#BSUB -o %J.log -e %J.err
#BSUB -J serial

myserial_prog
```

Example 6-2 shows the command to submit the job file `myserial.lsf` to Platform LSF. The application that a user wants to run is `myserial_prog`. It can be your own executable, a software program, a system command, or an executable script file. The job name `serial` is the job name. %J is a unique job ID number that is assigned by Platform LSF. It is always a good idea to include `%J` in the output file names to make the file names unique for different jobs. The standard output of `myserial_prog` will be stored in the file `%J.log`. The error messages will be stored in the file `%J.err`. If a user does not specify a queue, the job goes to the default queue `normal`.

Example 6-2 Submitting the job myjob.lsf to Platform LSF

```
$ bsub < myserial.lsf
Job <399> is submitted to default queue <normal>.
```

In Example 6-2, `399` is the job ID that is assigned to this job, and `normal` is the name of the default job queue.

The job remains pending until all conditions for its execution are met. Each queue has execution conditions that apply to all jobs in the queue, and users can specify additional conditions when they submit the jobs.

6.1.2 Run the jobs by using command lines

A user can also run the **bsub** command by using the command line only. The following example submits a serial job to queue `small` by using the command line:

```
bsub -o %J.log -e %J.err -q small 'myserial_prog'
```

Users can also build a job file one line at a time by running the **bsub** command without specifying a job to submit. The **bsub** command starts an interactive session and reads command lines from the standard input and submits them as a single batch job. The **bsub** command prompts `bsub>` for each line. Example 6-3 submits a job that sets the output file as `%J.out` and the job name as `cmdline`.

Example 6-3 bsub one line at a time

```
$ bsub
bsub> #BSUB -o %J.out
bsub> #BSUB -J cmdline
bsub> myserial_prog
bsub> Job <422> is submitted to default queue <normal>.
```

6.2 Running parallel applications

By using the **bsub** commands, users can also specify an execution host or a range of hosts, a queue, start and termination times, and many other job options. The following command submits a parallel application that requests four CPUs to the job queue `medium`. The job output and any errors will be saved to the files `%J.out` and `%J.err`.

```
bsub -n 4 -q medium -o %J.out -e %J.err 'mypara_prog'
```

Run IBM Parallel Environment jobs

Users can submit IBM Parallel Environment (PE) Runtime Edition (IBM PE Runtime Edition) jobs through the **bsub** command. Example 6-4 runs an application with IBM PE by using job file `poejob.lsf`. The job file sets the job name as `parajob`, and the job queue as `medium`. The `mypoe_prog` application runs with PE, and the job outputs and errors will be saved to the files `%J.out` and `%J.err`.

Example 6-4 poejob.lsf file

```
#!/bin/sh
#BSUB -q medium
#BSUB -J parajob
#BSUB -o %J.out
#BSUB -e %J.out

poe mypoe_prog -resd poe -procs 2 -ilevel 4
```

Then, use the **bsub** command to submit the script, and the result is shown in Example 6-5.

Example 6-5 Submit PE job

```
$ bsub < poejob.lsf
Job <413> is submitted to queue <medium>.
$ bjobs 413
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
413     weili   RUN   medium     c656f6n05.p c656f6n05.p parajob    Dec 10 16:31
```

Run MPICH2 jobs

MPICH2 integrates with Platform LSF through the Hydra process manager. Example 6-6 shows a job script that asks for 16 slots in total (eight slots per host), redirects stdout and stderr to files, and uses **mpiexec.hydra** to run the application `mpich2_prog` with MPICH2.

Example 6-6 MPICH2 job file: mpich2job.lsf

```
#!/bin/sh
#BSUB -n 16
#BSUB -e mpich2_%J.err
#BSUB -o mpich2_%J.out
#BSUB -R "span[ptile=8]"

mpiexec.hydra mpich2_prog
```

Use the following command to submit a job file `mpich2.lsf` to Platform LSF:

```
bsub < mpich2.lsf
```

Run OpenMP jobs

With Platform LSF, you can start parallel jobs that use OpenMP to communicate between processes on shared-memory machines and MPI to communicate across networked and non-shared memory machines.

You can set the environment variable **OMP_NUM_THREADS** manually, or let Platform LSF set **OMP_NUM_THREADS** automatically with an affinity scheduling resource requirement. To enable the Platform LSF affinity scheduling feature, users need to ensure that the Platform LSF administrator configured Platform LSF to load the affinity scheduler plug-in in the `lsb.modules` file and that the Platform LSF administrator also configured individual hosts on the `lsb.hosts` file to enable affinity scheduling on the hosts.

Example 6-7 shows an `openmp` program source code.

Example 6-7 openmp example: omp_hello.c

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

int main (int argc, char *argv[])
{
int nthreads, tid;

#pragma omp parallel private(nthreads, tid)
  {

  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }
```

```
   }  /* All threads join master thread and disband */

}
```

Use the following command to compile `omp_hello.c`:

```
gcc -fopenmp omp_hello.c -o omp_hello
```

Example 6-8 is an openmp job file script `openmp.lsf`. To run the application `omp_hello`, this job script requests four slots on one host and exports **`OMP_NUM_THREADS`** manually.

Example 6-8   openmp job file openmp.lsf

```
#!/bin/sh
#BSUB -n 4
#BSUB -J openmp
#BSUB -o omp_%J.out -e omp_%J.err
#BSUB -R "span[hosts=1]"
export OMP_NUM_THREADS=4

omp_hello
```

Use the following command to submit a job file `openmp.lsf` to Platform LSF:

```
bsub < openmp.lsf
```

Example 6-9 is a job file that requests one slot and binds to four cores in the same host. Because an affinity scheduling resource requirement is specified, Platform LSF sets **`OMP_NUM_THREADS`** to (`num_processor_unit * num_subtask`). In this example, `num_processor_unit` is `4`, and `num_subtask` is `1`.

Example 6-9   openmp2.lsf

```
#! /bin/sh
#BSUB -R "affinity[core(4)]"
#BSUB -o omp_%J.out -e omp_%J.err
#BSUB -J openmp2

omp_hello
```

Use the following command to submit `openmp2.lsf` to Platform LSF:

```
bsub < openmp2.lsf
```

When the job is running, users can use the **`bjobs -l -aff`** command to see the detail affinity information for this job. Example 6-10 is the output of the job file `openmp2.lsf`, which includes the affinity information.

Example 6-10   Obtaining query affinity information by using the bjobs -l -aff command

```
$ bjobs -l -aff 443

Job <443>, User <weili>, Project <default>, Status <RUN>, Queue <normal>, Comma
                     nd <#! /bin/sh;#BSUB -R "affinity[core(4)]";#BSUB -o %J.ou
                     t -e %J.err;/share/hello;sleep 1000>
Mon Dec 15 10:00:39: Submitted from host <c656f6n05.pok.stglabs.ibm.com>, CWD <
                     /share>, Output File <443.out>, Error File <443.err>, Requ
                     ested Resources <affinity[core(4)]>;
```

```
Mon Dec 15 10:00:40: Started 1 Task(s) on Host(s) <c656f6n05.pok.stglabs.ibm.co
                     m>, Allocated 1 Slot(s) on Host(s) <c656f6n05.pok.stglabs.
                     ibm.com>, Execution Home </home/weili>, Execution CWD </sh
                     are>;
Mon Dec 15 10:03:43: Resource usage collected.
                     MEM: 6 Mbytes;  SWAP: 0 Mbytes;  NTHREAD: 5
                     PGID: 98043;  PIDs: 98043 98044 98048 98053


 MEMORY USAGE:
 MAX MEM: 13 Mbytes;  AVG MEM: 7 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] affinity[core(4)*1]
 Effective: select[type == local] order[r15s:pg] affinity[core(4)*1]

 AFFINITY:
                     CPU BINDING                          MEMORY BINDING
                     ------------------------             --------------------
 HOST                TYPE   LEVEL  EXCL   IDS             POL   NUMA SIZE
 c656f6n05.pok.stgla core   -      -      /0/32           -     -    -
                                          /0/40
                                          /0/96
                                          /0/104
```

Run hybrid (MPI + OpenMP) jobs

Example 6-11 shows an example job file for a hybrid (MPI + OpenMP) job. This job script requests that two hosts start four IBM PE tasks in total. Each host starts two tasks, and each host binds each task to two cores. The **`OMP_NUM_THREADS`** environment variable is set to `(num_processor_unit * num_subtask)`. In Example 6-11, `num_processor_unit` is `2`, and `num_subtask` is `1`.

Example 6-11 Hybrid job file: hybrid.lsf

```
#!/bin/sh
#BSUB -n 4
#BSUB -J Hybrid
#BSUB -o %J.out -e %J.err
#BSUB -R "affinity[core(2)]"
#BSUB -R "span[hosts=2]"

poe hybrid_prog
```

Use the following command to submit `hybrid.lsf` to Platform LSF:

```
bsub < bybrid.lsf
```

6.3 Managing workloads

Users can modify the pending and running workloads by using Platform LSF commands.

6.3.1 Modify pending jobs

If the submitted jobs are pending (the **bjobs** command shows the job in the PEND state), users can modify the job submission parameters. Users can also modify entire job arrays or the individual elements of a job array.

Replace the job command

To replace the job command file, run the following command:

```
bmod -Z "new_command"
```

Change job parameter

To change a specific job parameter, run **bmod -b**.

The specified options replace the submitted options. For example, to change the start time of job 135 to 5:00 am, run the following command:

```
bmod -b 5:00 135
```

Reset to its default submitted value

To reset an option to its default submitted value (to undo a **bmod** command), append the **n** character to the option name and do not include an optional value. For example, to reset the start time for job 123 back to its default value, run the following command:

```
bmod -bn 123
```

6.3.2 Modify running jobs

After the submitted job is running, users can modify several job options, including resource reservation, CPU limit, memory limit, swap limit, and other options.

Only the job owner or a Platform LSF administrator can modify a running job. Ensure that `LSB_MOD_ALL_JOBS` is set in `lsf.conf`. Additionally, several parameters need to be set for each type of modification type. To modify the CPU limit of running jobs, `LSB_JOB_CPULIMIT=Y` must be defined in `lsf.conf`. To modify the memory limit of running jobs, `LSB_JOB_MEMLIMIT=Y` must be defined in `lsf.conf`.

To modify the name of job error file for a running job, you must use **bsub -e** or **bmod -e** to specify an error file before the job starts running.

To modify the running job, run **bmod** with the appropriate option:

- CPU limit option:

  ```
  -c [hour:]minute[/host_name | /host_model] | -cn
  ```

- Memory limit option:

  ```
  -M mem_limit | -Mn
  ```

- Rerunnable jobs option:

  ```
  -r | -rn
  ```

- Standard error file name option:

  `-e error_file | -en`

- Standard output file name option:

  `-o output_file | -on`

- Run limit option:

  `-W run_limit[/host_name | /host_model] | -Wn`

6.3.3 Canceling the jobs

Users can cancel a job from running or pending by using the **`bkill`** command to cancel it. The following example cancels the job with ID `432`:

```
bkill 432
```

If a job cannot be canceled in the operating system, the user can also use the **`bkill -r`** command to force the removal of the job from Platform LSF.

In certain scenarios, users need to suspend or stop a job that is already running. To suspend a job, you must be an administrator or the user who submitted the job. You can use the **`bstop`** command to suspend the job. The following example suspends the job with ID `1500`:

```
bstop 1500
```

When you stop a job, it is suspended until you choose to resume it. You must be the same user who suspended the job. If your job was suspended by the administrator, you cannot resume it. Only the administrator or `root` can resume the job for you. The following command resumes the job with ID `1500`:

```
bresume 1500
```

7

Tuning and debugging applications

This chapter provides information about how to tune and debug applications.

The following topics are described in this chapter:

- General analysis, optimization, and tuning strategy
- Profiling and tracing tools
- Runtime tuning
- Debugging tools

7.1 General analysis, optimization, and tuning strategy

Many strategies can be used to analyze and tune a high-performance computing (HPC) application for performance, either at system or application levels. From an application point of view, several well-known strategies can be considered:

- Optimization that is driven by compiler options
- Optimization of the compiler that is based on feedback-directed optimization
- Performance library tuning
- Use of “white-box” techniques, such as profiling and tracing, that often involve changes in source code
- Analysis of the hardware performance counters

Several of these strategies are discussed in other sections of this publication. See 5.1, “Compilers” on page 144, 5.2, “Performance libraries” on page 148, and 5.4, “IBM POWER8 features” on page 158.

Performance Optimization and Tuning Techniques for IBM Processors, including IBM POWER8, SG24-8171, is also an excellent reference on this topic, although it is focused on multi-threaded applications.

Section 7.2, “Profiling and tracing tools” on page 176 covers profiling and tracing parallel applications on POWER8 and the analysis that is provided with the help of the hardware performance counters.

Hardware performance analysis

Modern multi-core processors usually provide a special unit to store hardware events. This information is useful to understand the processor’s internal behavior on the program’s run time. Analysis of these units produce information that reveals the program performance from the processor’s point of view. The analysis shows “bubbles” or abnormalities in the execution pipeline, processing units where the processor stalls, and performance measurements, such as cycles per instruction.

The POWER8 processor includes a built-in Performance Monitor Unit (PMU) that consists of six Performance Monitor Counters (PMC) in which four PMCs (PMC1 - PMC4) are programmable. PMC5 counts instructions that are completed. And, PMC6 counts cycles.

Hundreds of hardware events are exposed by the POWER PMU. These events’ data can be accessed with help of Oprofile, perf, and the IBM HPC Toolkit tools, which are introduced in 7.2, “Profiling and tracing tools” on page 176.

7.2 Profiling and tracing tools

The Linux development environment has many tools for profiling and tracing parallel applications. Table 7-1 on page 177 lists suggested tools for various HPC programming models that are supported in POWER8.

Table 7-1 Profile and trace tools for HPC in POWER8

Programming model or pattern	Tools
Message Passing Interface (MPI)	MPI profile and trace I/O trace HPM gprof Call Graph analysis
OpenMP	OpenMP profiling perf oprofile gprof
Pthreads	perf oprofile gprof

7.2.1 gprof

The GNU Profiler (**gprof**) is the most used tool to find hotspots in Linux applications, no matter whether the program is serial (single or multi-threaded) or multi-program (MPI, for example). Therefore, this section shows high-level details only.

Its input is the call graph file (`gmon.out`) that is generated when a program that was compiled with the **-pg** flag is executed, and then, it can display profile information in two formats:

- The flat profile shows the following information:
  - Time that is spent in each method
  - Count of the number of times that each method was called
- The call graph shows the following information for each method:
  - Caller methods
  - Caller methods with a count of the number of times and an estimate of time spent

When it is used with parallel programs, one profile file is created for each task and is named `gmon.out.taskid`. The files can be loaded in **gprof** individually or more than one file at one time. However, if multiple files are loaded at one time, the tool adds the profile information of each file together. Use the following reference to learn how to use **gprof** with the IBM Parallel Environment (PE) Runtime Edition:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.1.0/com.ibm.cluster.pe.v2r1.pe100.doc/am102_profappl.htm

For more information about **gprof**, see the GNU Binutils website:

http://www.gnu.org/software/binutils

7.2.2 Oprofile and perf

Oprofile and perf are open source suites of tools that are designed to explore information that is provided by the PMU of the processors and to support POWER8 profiling. Both suites of tools rely on the Linux Kernel `perf_events` application binary interface (ABI) and work similarly in that they accept event arguments to be profiled in either system-wide mode or user application-execution mode.

The Ubuntu operating system provides packages to install perf and Oprofile in the system. The latest version of Oprofile also is included in the performance package of the IBM Advance Toolchain for Linux on Power.

The heart of Oprofile profiling is the **`operf`** command, which saves collected information in the `./oprofile_data` folder for later post-processing. A list of events to be profiled is passed with `--events` (or `-e`) `event1_name:count[,event2_name:count...]`, where *eventn_name* is the symbolic name of the event and *count* is the minimum sample count. The **`operf`** command has several settings to fine-tune that data, for instance, `--separate-thread` to sample per thread groups, `--separate-cpu` to sample per CPU, `--system-wide` for a system-wide profile, and `--callgraph` to generate a call graph.

Use **`opreport`** to report the data that is gathered by Oprofile or **`opannotate`** to show a source or an assembled, annotated view. Example 7-1 shows a simple Oprofile session where a Fortran application with 20 threads was profiled for completion stalls and completion run instruction events.

Example 7-1 Profiling a Fortran application

```
$ operf -e PM_CMPLU_STALL:10000,PM_RUN_INST_CMPL:100000 ./mg.C.x


 NAS Parallel Benchmarks (NPB3.3-OMP) - MG Benchmark

<... Output Omitted ...>

 * * * ATTENTION: The kernel lost 9705144 samples. * * *
Decrease the sampling rate to eliminate (or reduce) lost samples.

WARNING: Lost samples detected! See
/home/wainersm/NPB3.3.1/NPB3.3-OMP/bin/oprofile_data/samples/operf.log for
details.
Lowering the sampling rate might reduce or eliminate lost samples.
See the '--events' option description in the operf man page for help.

Profiling done.
$ opreport 2>/dev/null
CPU: ppc64 POWER8, speed 3690 MHz (estimated)
Counted PM_CMPLU_STALL events (Completion stall.) with a unit mask of 0x00 (No
unit mask) count 10000
Counted PM_RUN_INST_CMPL events (Run_Instructions.) with a unit mask of 0x00 (No
unit mask) count 100000
PM_CMPLU_STALL...|PM_RUN_INST_CM...|
  samples|      %|  samples|      %|
------------------------------------
  1746868 100.000    750175 100.000 mg.C.x
   PM_CMPLU_STALL...|PM_RUN_INST_CM...|
     samples|      %|  samples|      %|
    ------------------------------------
       970550 55.5594    428732 57.1509 mg.C.x
       738897 42.2984    273345 36.4375 kallsyms
        34677  1.9851     47507  6.3328 libxlsmp.so.1
         2111  0.1208       494  0.0659 libpthread-2.19.so
          401  0.0230        74  0.0099 libc-2.19.so
          189  0.0108        22  0.0029 ld-2.19.so
         24  0.0014        0       0 [stack] (tgid:53923
range:0x3ffffc120000-0x3ffffc15ffff)
          7 4.0e-04        1 1.3e-04 libxlf90_r.so.1
          6 3.4e-04        0       0 nf_conntrack
          3 1.7e-04        0       0 ip_tables
          2 1.1e-04        0       0 bnx2x
          1 5.7e-05        0       0 libgcc_s.so.1
```

The POWER8 PMU has hundreds of events. The available symbolic event names for profiling with Oprofile are listed with the **ophelp** command as shown in Example 7-2.

Example 7-2 Oprofile: List of available POWER8 events

```
$ ophelp

oprofile: available events for CPU type "ppc64 POWER8"

This processor type is fully supported by operf; opcontrol timer mode might be
available.
See Power ISA 2.07 at https://www.power.org/

For architectures using unit masks, you might be able to specify
unit masks by name.  See the 'opcontrol' or 'operf' man page for more details.

CYCLES: (counter: 0)
        Cycles (min count: 100000)
PM_1PLUS_PPC_CMPL: (counter: 0)
        one or more ppc instructions finished (min count: 100000)
PM_1PLUS_PPC_DISP: (counter: 3)
        Cycles at least one Instr Dispatched (min count: 100000)
PM_ANY_THRD_RUN_CYC: (counter: 0)
        One of threads in run_cycles (min count: 100000)
PM_BR_MPRED_CMPL: (counter: 3)
        Number of Branch Mispredicts (min count: 10000)
PM_BR_TAKEN_CMPL: (counter: 1)
        New event for Branch Taken (min count: 10000)
PM_CYC: (counter: 0)
        Cycles (min count: 100000)
PM_DATA_FROM_L2MISS: (counter: 1)
        Demand LD - L2 Miss (not L2 hit) (min count: 10000)
<<< Output omitted>>>
```

Like Oprofile, perf provides commands to record profiling information about events and to report them in human-readable format. However, only a few symbolic event names can be used. (Use **perf list** to see them.) All other event names must be passed in raw hex code with the -e argument.

Use the **perf record** and **perf report** pair of commands to record and report workflow. The tool also offers several subtools that are focused on specialized kinds of analyses:

- **perf lock**: To profile Linux kernel lock events
- **perf trace**: To trace system events

The **perf stat** subtool is a simple events counter. (No profiling data is gathered.) Its counterpart in Oprofile is the **ocount** command (available since version 0.9.9 of the suite tool).

Both perf and Oprofile can profile an application with the `CYCLES` event to identify hotspots. They behave similarly to the **`gprof`** flat profile.

Many other events or derived metrics are available for POWER8. The correlating cycles, instructions, and stalled cycle metrics are useful to measure the processor's efficiency in running an application. These metrics are calculated in percentages and use Oprofile symbolic event names, as shown in these examples:

- Run cycles per instruction (CPI)

  Formula: `(PM_RUN_CYC / PM_RUN_INST_CMPL) * 100`
- Stall CPI

  Formula: `(PM_CMPLU_STALL / PM_RUN_INST_CMPL) * 100`
- Stall ratio

  Formula: `(PM_CMPLU_STALL / PM_RUN_CYC) * 100`

A high ratio of stalled cycles can represent a serious performance issue. You can perform further analysis with Oprofile or perf to identify the processor facility cycles that are wasted:

- Stalls due to Instruction Fetch Unit (IFU), either Branch or CR units

  Formula: `(PM_CMPLU_STALL_BRU_CRU / PM_CMPLU_STALL) * 100`
- Stalls due Fixed-point Unit (FXU)

  Formula: `(PM_CMPLU_STALL_FXU / PM_CMPLU_STALL) * 100`
- Stalls due Vector-and-Scalar Unit (VSU)

  Formula: `(PM_CMPLU_STALL_VSU / PM_CMPLU_STALL) * 100`
- Stalls due Load Store Unit (LSU)

  Formula: `(PM_CMPLU_STALL_LSU / PM_CMPLU_STALL) * 100`
- Stalls due reject (load hit store)

  Formula: `(PM_CMPLU_STALL_NTCG_FLUSH / PM_CMPLU_STALL) * 100`

Next, drill down on the units on which most of the stalls occur. Break them down into more fine-grained stall events until you identify the problem. It is beyond the scope of this book to explain all of the stall events. We suggest that you review them by using the **`ophelp`** command.

You can perform a similar analysis of data misses on L1, L2, and L3 caches. The load and store miss ratio on each level is measured with the following formulas:

- L1 store miss

  Formula: `(PM_ST_MISS_L1 / PM_ST_FIN) * 100`
- L1 load miss

  Formula: `(PM_LD_MISS_L1 / PM_LD_REF_L1) * 100`
- L2 store miss

  Formula: `(PM_L2_ST_MISS / PM_L2_ST) * 100`
- L2 load miss

  Formula: `(PM_L2_LD_MISS / PM_L2_LD) * 100`
- L3 load miss

  Formula: `(PM_DATA_FROM_L3MISS / PM_DATA_FROM_MEM) * 100`

The memory affinity of an application on a Non-Uniform Memory Access (NUMA) system consists of data access patterns of either local (same chip controller), remote (same NUMA node), or distant (another NUMA node) attached memory. You need to investigate this area because distant memory access takes higher latency. The metric formulas of memory affinity are obtained by these formulas:

- Memory locality

  Formula: `PM_DATA_FROM_LMEM / (PM_DATA_FROM_LMEM + PM_DATA_FROM_RMEM + PM_DATA_FROM_DMEM) * 100`
- Local to remote and distant accesses ratio

  Formula: `PM_DATA_FROM_LMEM / (PM_DATA_FROM_RMEM + PM_DATA_FROM_DMEM)`
- Local to remote accesses ratio

  Formula: `PM_DATA_FROM_LMEM / PM_DATA_FROM_RMEM`
- Local to distant accesses ratio

  Formula: `PM_DATA_FROM_LMEM / PM_DATA_FROM_DMEM`

In Example 7-3, the **ocount** command from the Advance Toolchain 8.0 was used to gather local, remote, and distance memory access statistics of a Fortran application (*NAS Parallel Benchmarks*) with 20 threads. By default, the memory locality for the application was calculated in 64.69%. In the next run, with the use of thread binding, the percentage of locality reached up to 96.1%.

Example 7-3 ocount: Calculate memory locality statistics in two scenarios

```
$ export PATH=/opt/at8.0/bin:$PATH
$ export OMP_NUM_THREADS=20
$ ocount -e PM_DATA_FROM_LMEM,PM_DATA_FROM_RMEM,PM_DATA_FROM_DMEM ./mg.C.x


 NAS Parallel Benchmarks (NPB3.3-OMP) - MG Benchmark

<... output omitted ...>

Events were actively counted for 5.0 seconds.
Event counts (actual) for /home/wainersm/NPB3.3.1/NPB3.3-OMP/bin/mg.C.x:
   Event                   Count                    % time counted
   PM_DATA_FROM_DMEM       8,393,647                100.00
   PM_DATA_FROM_LMEM       17,274,296               100.00
   PM_DATA_FROM_RMEM       1,033,889                100.00
$ export XLSMPOPTS=PROCS=8,10,12,14
$ ocount -e PM_DATA_FROM_LMEM,PM_DATA_FROM_RMEM,PM_DATA_FROM_DMEM ./mg.C.x


 NAS Parallel Benchmarks (NPB3.3-OMP) - MG Benchmark

<... output omitted ...>

Events were actively counted for 2.2 seconds.
Event counts (actual) for /home/wainersm/NPB3.3.1/NPB3.3-OMP/bin/mg.C.x:
   Event                   Count                    % time counted
   PM_DATA_FROM_DMEM       420,742                  100.00
   PM_DATA_FROM_LMEM       19,033,385               100.00
   PM_DATA_FROM_RMEM       350,558                  100.00
```

7.2.3 IBM HPC Toolkit

IBM HPC Toolkit (HPCT) provides profiling and trace tools for MPI and multi-threaded programming models. HPCT gathers rich data about the parallel application behavior in execution time. Therefore, we suggest that you use these tools to obtain the initial performance measurements for your application, find hotspots, and also bottlenecks. The tools are listed:

- Hardware Performance Monitoring (HPM) tool for performance analysis that is based on hardware event counters
- MPI profiling and trace tool for performance analysis that is based on MPI events
- OpenMP profiling and trace tool for performance analysis of openMP primitives
- I/O profiling tool for performance analysis of parallel application I/O activities
- Call graph analysis

A few steps are required to accomplish profiling and tracing:

1. Instrumentation

   Data will be collected by using trace points that can be inserted in the application executable.

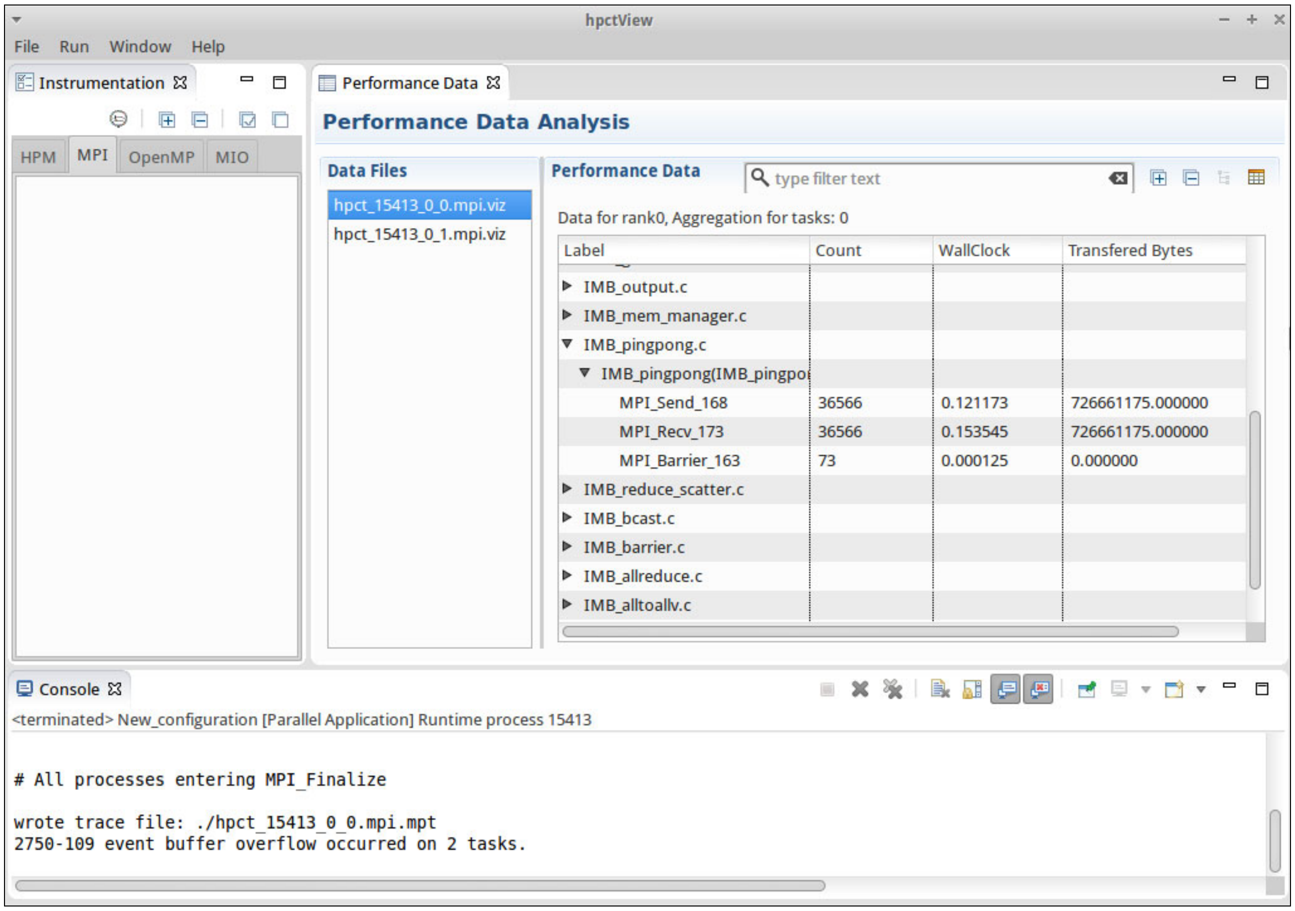
2. Run the application

   You run the application to gather and save data in text log files.

3. Visualize results

   Import log files on either hpctView or the HPCT plug-in of the IBM Parallel Environment Developer Edition (PE DE) Workbench to visualize profile and tracing information that is presented at several levels of detail. The tools facilitate browsing and the manipulation of the collected data.

Although both *hpctView* and *Workbench HPCT plug-in* implement graphically the instrument-run-visualize workflow, hpctView is the suggested tool to use. Figure 7-1 on page 183 shows the hpctView tool compartments: Tools for instrumentation on the left panel, the Performance Data Analysis panel where produced data can be loaded for visual analysis, and the Run button at the top tools bar.



Data for rank0, Aggregation for tasks: 0

Label	Count	WallClock	Transfered Bytes
▶ IMB_output.c			
▶ IMB_mem_manager.c			
▼ IMB_pingpong.c			
▼ IMB_pingpong(IMB_pingpo			
MPI_Send_168	36566	0.121173	726661175.000000
MPI_Recv_173	36566	0.153545	726661175.000000
MPI_Barrier_163	73	0.000125	0.000000
▶ IMB_reduce_scatter.c			
▶ IMB_bcast.c			
▶ IMB_barrier.c			
▶ IMB_allreduce.c			
▶ IMB_alltoallv.c			

<terminated> New_configuration [Parallel Application] Runtime process 15413

```
# All processes entering MPI_Finalize

wrote trace file: ./hpct_15413_0_0.mpi.mpt
2750-109 event buffer overflow occurred on 2 tasks.
```

Figure 7-1 hpctView

HPCT allows trace points to any enclosed block of execution or marked methods within the parallel application. Instrumentation can occur as either the compiled binary file or as an application source code with the use of an API library.

Important: Do not mix different tools in a single instrumentation because they can interfere with each other's analysis in an unpredictable way.

The HPC Toolkit tools are briefly detailed in the next sections. For more information, see the guide to getting started at the following website and select **Cluster software** → **Parallel Environment Developer Edition** → **Parallel Environment Developer Edition 2.1.0**:

http://www.ibm.com/support/knowledgecenter

Call graph analysis

The use of **gprof** as shown in 7.2.1, "gprof" on page 177 to profile MPI applications generates one profile file for each task that is created. To visualize collected information usefully, *hpctView* offers the call graph analysis tool. The tool imports **gprof**-formatted files and displays the profile and call graph information for the aggregated tasks.

MPI profile and tracing

Understanding MPI application behavior in run time to identify bottlenecks or inefficient communication patterns, for instance, is a tough task that involves gathering and processing a large amount of information. Therefore, HPCT features an MPI profile and tracing tool to ease the performance analysis in this context. The tool can be used to identify and break down the following problems:

- Highly frequent communication
- Unwanted, totaled transferred data
- Unbalanced workload

The profiler collects MPI application data to report on MPI function usage with aggregated values for each task and files. hpctView is then able to show the following information:

- Number of times that MPI functions are executed
- Total wall time that is spent in each function
- Total of transferred bytes

Use the tracing tool for an overview of the MPI function calls for a certain period. hpctView can display colorful function calls as trace points, which are separated by task ranks, in a timeline. hpctView offers many ways to zoom in and out of the chart, filter unwanted MPI functions, and drill down to the source code that relates to a specific trace point. More options are available to facilitate the browsing of the information that was produced.

I/O profiling and tracing

Certain parallel applications, for instance in the bioinformatics area, require the intensive use of I/O operations because their solutions consume large amounts of data. Moreover, due to the distributed process of MPI applications, analysis from the application's point of view is hard to achieve by using traditional Linux administration tools. For example, **`iostat`** is meant for monitoring within a single system. Therefore, HPC Toolkit offers a specialized tool, which is called the MPI I/O (MIO) Tool, for I/O profiling and tracing. This tool can be handy to identify and break down several performance problems. The tool can gather information about I/O system calls (syscalls) that are executed at each file access that is performed by the MPI tasks.

The basic hpctView visualization data display for each accessed file includes the following information:

- Call frequency of each I/O syscall (`fnctl`, `open`, `read`, `write`, and `close`)
- Accumulated time in each operation

These syscalls can be more detailed. hpctView can show the following information, depending on the operation:

- Cumulative bytes that are requested
- Cumulative bytes that are delivered
- Minimum request size (bytes)
- Maximum request size (bytes)
- Rate (bytes)
- Suspend wait count
- Suspend wait time
- Forward seeks average
- Backward seeks average

Switching to the MIO Data view, hpctView shows the trace in a tree hierarchy (for each file and application) with I/O operations in leaf nodes. Detailed information about each operation over time is displayed in a tabular or graph mode.

Hardware Performance Monitor (HPM) profiling

HPC Toolkit provides the Hardware Performance Monitor (HPM), which is a user application-level facility to explore POWER8 PMU data for parallel applications. It gathers and combines data from multiple MPI tasks at one time, including different hardware performance counters for each task.

With the tookit, you can profile any of the hundreds of hardware events that are available by POWER8 PMU. The HPM tool also provides 31 metrics that are derived from performance events, such as utilization rate, millions of instructions per second (MIPS), and memory load traffic.

The **`hpccount`** and **`hpcstat`** command-line tools use POWER8 performance counters to gather performance data. They can gather the measurements of groups of events of your application and produce system-wide statistics.

By default, performance events are profiled for the application execution as a whole. By using the API that is provided by the `libhpc` library, you can change the HPM tool behavior by specifying the blocks in source code that can be profiled.

7.3 Runtime tuning

To fine-tune the execution of OpenMP, pthreads, and distributed memory applications, users can decide to set certain environment variables. In certain cases, the system administrator needs to change the configuration files of the operating system. Execution of Java applications can be tuned by passing a number of arguments to the Java virtual machine (JVM).

7.3.1 Controlling the execution of multithreaded applications

The technical computing applications that run on a computing node typically consist of multiple threads. The runtime environment has multiple options to support the runtime fine-tuning of multithreaded programs. First, we first show how to control the OpenMP workload by setting certain environment variables. Then, we explain the tools that a user can use to retrieve or set a process’s affinity at run time and control the NUMA policy for processes or shared memory.

Running OpenMP applications

A user can control the execution of OpenMP applications by setting the number of environment variables. In this section, we cover only a view of the environment variables that are specific for technical computing workloads. The environment variables that are prefixed with `OMP_` are defined by the OpenMP standard.

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, static distribution fits well in most cases. If an application does not specify the scheduling policy, 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 be run for 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, you need to 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 `stderr`. If the user sets the value to `TRUE`, the OpenMP version number and initial values of ICVs are published. The `VERBOSE` value instructs the OpenMP run time to display the values of vendor-specific variables, also. 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 possibilities for automatic load balancing. But for technical computing workloads, the fixed position of threads within the server is typically preferred.

The numbers of 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 chip of the first socket. The first logical processor of the second core has the index 8. The numbering of the logical processors of the second chip of the first socket starts when we finish with the numbering of the logical processors of the first chip. We start numbering the logical processors of the second socket only after we finish the numbering of the logical processors of the first chip.

For the OpenMP application that is compiled with IBM Xpertise Library (XL) compilers, you need to use the **`XLSMPOPTS`** environment variable. This environment has many suboptions, and only a few of them are for thread binding.

The first option is to specify the starting logical processor number for the binding first thread of an application and an increment for the subsequent threads. For the example, the following value of **`XLSMPOPTS`** instructs the OpenMP runtime environment to bind OpenMP threads to logical processors 40, 44, 48, and so on, up to the last available processor:

```
export XLSMPOPTS=startproc=40:stride=8
```

A user can also specify a list of logical processors to use for thread binding. For example, to use only even-numbered logical processors of a processor's second core, you need to specify this value of **`XLSMPOPTS`**:

```
export XLSMPOPTS=PROCS=8,10,12,14
```

For more details about this variable, see the "XLSMPOPTS" section of the online manuals:

- 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`

For the OpenMP application that is compiled with GNU Compiler Collection (GCC) compilers, you need to assign a list of the logical processors that you want to the **GOMP_CPU_AFFINITY** environment variable, as you do for the PROCS suboption of the IBM XL **XLSMPOPTS** environment variable. For more information about the **GOMP_CPU_AFFINITY** environment variable, see the corresponding section of the GCC manual.

The OpenMP 3.1 revision introduced the **OMP_PROC_BIND** environment variable. Then, 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.

For more details about the performance impact of thread binding and simple code that you can use to generate your own binding map, see Appendix D, “Applications and performance” on page 263.

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 affinity for highly loaded threads only. In this circumstance, you need to identify the process identifiers (PIDs) of highly loaded running threads (for example, by examining the output of the **top -H** command), and perform binding only on those threads.

Knowing the PID, you can use the **taskset** command to retrieve the affinity of the corresponding entity (a thread or a process).

For more details and a description of the syntax, see the **<command>** manual pages:

```
$ man <command>
```

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, we 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 5-core or 6-core chip, and the POWER8 processor consists of two nodes in the terminology of the **numactl** command. To discover the indexes of nodes and estimate the memory access penalty, run **numactl** with the **-H** argument. Example 7-4 shows the corresponding output for a 20-core IBM Power System S824L server that is running Ubuntu 14.04 little endian (LE).

Example 7-4 The numactl -H command (truncated) output on a 20-core IBM Power System S824L

```
$ numactl -H
available: 4 nodes (0-1,16-17)
< ... output omitted ... >
node distances:
node   0   1  16  17
  0:  10  20  40  40
  1:  20  10  40  40
 16:  40  40  10  20
 17:  40  40  20  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 significant 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.

For more details and a description of the syntax, see the **`<command>`** manual pages:

```
$ man <command>
```

7.4 Debugging tools

The Linux development ecosystem has many tools for application debugging and analysis. The POWER8 processor is supported by most of them. This section describes the most commonly used tools.

7.4.1 Parallel debugger

The GNU debugger (GDB) is a well-known debugger that is suitable for either single-threaded or multiple-threaded serial applications. However, with MPI applications, a parallel debugger is preferable because of its ability to attach to several jobs and treat processes as a single entity. For more information about the GDB, see this website:

http://www.gnu.org/software/gdb

IBM PE DE Parallel Debugger

The IBM Parallel Environment Development Edition (PE DE) Eclipse Workbench is described in 5.3, “IBM Parallel Environment Developer Edition” on page 155. It comes with the Eclipse Parallel Tools Project (PTP) parallel debugger, which extends the default Eclipse debugger to combine the information of many jobs, processes, and threads into a single debugging viewer.

PE DE debugger presents the visualization of parallel applications as a collection of jobs, processes, and threads. In addition to viewing thread parallel applications as a single entity, you can browse information at three levels of detail to inspect applications from many aspects when you look for a bug.

Any arbitrary set of processes within a job can be rearranged and managed collectively so that you can fine-tune operations that target a specific subset of processes. Also, a different type of breakpoint, which is called a *parallel breakpoint*, can be applied in these collections of processes.

For more information about the Eclipse PTP parallel debugger, go to this website and select **Parallel Development User Guide** → **Parallel debugging**:

http://help.eclipse.org

Parallel Debugger for the IBM Parallel Environment

The Parallel Debugger (PDB) for the IBM Parallel Environment is a command-line debugger that comes with the IBM Parallel Environment (PE) and operates in a similar manner to many instances of GBD debugging multiple programs. The debugging session can be initiated by either launching the parallel application with the **pdb** command or attaching it to a running job.

PDB's features include multiple consoles, message control, thread stack information for each task, and controls to reduce the task information that is generated. Use PDB to debug large-scale jobs.

For more information about PDB, see the following website:

http://www.ibm.com/support/knowledgecenter/SSFK3V_2.1.0/com.ibm.cluster.pe.v2r1.pe100.doc/am102_usingpdb.htm

7.4.2 Application analyzers

This section provides information about application analyzers.

Valgrind

Valgrind is an open source framework to ease the dynamic instrumentation of applications by building powerful analysis tools. Its suite of tools is rich and provides a thread error detector that is called *Helgrind* to identify a wide range of problems within POSIX thread applications:

- Mistakes in using the Pthreads API
- Potential deadlocks
- Data race conditions

For more information, see this website:

http://valgrind.org

Helgrind is executed with the **valgrind --tool=helgrind** command as shown in Example 7-5. Valgrind can be installed through the *valgrind* package that is provided with Ubuntu 14.04.

Example 7-5 Inspecting thread errors with the Helgrind tool

```
$ valgrind --tool=helgrind --history-level=none ./uts-pthread -T 8 -t 1 -a 3 -d 15
-b 4 -r 19
==17114== Helgrind, a thread error detector
==17114== Copyright (C) 2007-2013, and GNU GPL'd, by OpenWorks LLP et al.
==17114== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==17114== Command: ./uts-pthread -T 8 -t 1 -a 3 -d 15 -b 4 -r 19
==17114==
UTS - Unbalanced Tree Search 2.1 (PThreads)
Tree type:  1 (Geometric)
Tree shape parameters:
  root branching factor b_0 = 4.0, root seed = 19
  GEO parameters: gen_mx = 15, shape function = 3 (Fixed branching factor)
Random number generator: SHA-1 (state size = 20B)
Compute granularity: 1
Execution strategy:  Parallel search using 8 threads
   Load balance by work stealing, chunk size = 20 nodes
  CBarrier Interval: 1
   Polling Interval: 1

==17114== ---Thread-Announcement------------------------------------------
```

```
==17114==
==17114== Thread #3 was created
==17114==    at 0x42C9E9C: clone (clone.S:77)
==17114==    by 0x41784DB: pthread_create@@GLIBC_2.17 (createthread.c:81)
==17114==    by 0x407A537: pthread_create@* (hg_intercepts.c:300)
==17114==    by 0x10003EFF: pthread_main (in /home/wainersm/uts-1.1/uts-pthread)
==17114==    by 0x41C2A7F: generic_start_main.isra.0 (libc-start.c:289)
==17114==    by 0x41C2C93: (below main) (libc-start.c:80)
==17114==
==17114== ----------------------------------------------------------------
==17114==
==17114== Possible data race during read of size 4 at 0x43800B4 by thread #3
==17114== Locks held: none
==17114==    at 0x100032D0: parTreeSearch (in /home/wainersm/uts-1.1/uts-pthread)
==17114==    by 0x100034CB: pthread_spawn_search (in
/home/wainersm/uts-1.1/uts-pthread)
==17114==    by 0x40796C7: mythread_wrapper (hg_intercepts.c:233)
==17114==    by 0x4177CAB: start_thread (pthread_create.c:312)
==17114==    by 0x42C9EDF: clone (clone.S:96)
==17114==
==17114== Address 0x43800B4 is 4 bytes inside a block of size 160 alloc'd
==17114==    at 0x407378C: malloc (vg_replace_malloc.c:291)
==17114==    by 0x10003CAB: pthread_main (in /home/wainersm/uts-1.1/uts-pthread)
==17114==    by 0x41C2A7F: generic_start_main.isra.0 (libc-start.c:289)
==17114==    by 0x41C2C93: (below main) (libc-start.c:80)
==17114==
==17114== ----------------------------------------------------------------
< ... output omitted ... >
```

8

NVIDIA CUDA on IBM POWER8

The exploitation of general-purpose computing on graphics processing units (GPUs) and modern multi-core processors in a single heterogeneous parallel system proves to be highly efficient for running several technical computing workloads. This approach applies to a wide range of areas, such as chemistry, bioinformatics, molecular biology, engineering, and big data analytics.

The IBM Power System S824L and the NVIDIA Tesla K40 GPU, combined with the latest IBM POWER8 processor, provide a unique technology platform for high-performance computing (HPC). With this platform, you can develop applications that use Compute Unified Device Architecture (CUDA): a parallel computing platform and programming model that was created by NVIDIA and implemented by the GPUs.

This chapter describes the installation of the system, and the development of C/C++ and Java applications that use the NVIDIA CUDA platform for IBM POWER8. The following topics are covered:

- Advantages of NVIDIA on POWER8
- Software stack
- System monitoring
- Application development
- Tuning and debugging

For more information, see this website:

`http://docs.nvidia.com/cuda/cuda-gdb`

8.1 Advantages of NVIDIA on POWER8

The IBM and NVIDIA partnership was announced in November 2013 to integrate IBM Power Systems with NVIDIA GPUs and to enable applications and workloads that are accelerated by GPUs. The goal is to deliver higher performance and better energy efficiency to companies and data centers.

This collaboration produced its initial results in 2014:

- The announcement of the first IBM POWER8 system (IBM Power System S824L) that featured NVIDIA Tesla GPUs
- The release of CUDA 5.5 for POWER8
- The availability of several applications and tools to use GPU acceleration and CUDA, for example, IBM Xpertise Library (XL) C/C++ compilers, IBM Java, and others

More applications, middleware, and workloads from various fields announced upcoming support for GPU acceleration on IBM Power Systems.

The computational capability that is provided by the combination of NVIDIA Tesla GPUs and IBM POWER8 systems enable workloads from scientific, technical, and HPC to run on data center hardware. In most cases, these workloads were run on supercomputing hardware. This computational capability is built on top of massively parallel and multithreaded cores with NVIDIA Tesla GPUs and IBM POWER8 processors, where processor-intensive operations were offloaded to GPUs and coupled with the system's high memory-hierarchy bandwidth and I/O throughput.

An overview of the NVIDIA Tesla GPU is provided in 8.1.1, "NVIDIA Tesla K40 GPU" on page 192. For more information, see the following website:

http://nvidia.com/tesla

For more information about the IBM POWER8 processor and systems, see 8.1.1, "NVIDIA Tesla K40 GPU" on page 192.

Moreover, the development and portability of GPU-accelerated applications for IBM Power Systems with NVIDIA GPUs are made easier with the availability of CUDA and little-endian mode on POWER8. CUDA and little-endian mode increase the commonality with other popular architectures and the growing ecosystem that is built around the OpenPOWER Foundation. For more information about OpenPOWER, see the following website:

http://openpowerfoundation.org

IBM Power Systems with NVIDIA GPUs provide a computational powerhouse for running applications and workloads from several scientific domains, and for processing massive amounts of data, with easier application development and portability. All of this power builds on the strong ecosystems of the NVIDIA CUDA architecture and the OpenPOWER Foundation.

8.1.1 NVIDIA Tesla K40 GPU

The NVIDIA Tesla K40 GPU is based on Kepler architecture version GK110B that supports the CUDA compute capability 3.5. It can deliver 1.43 Tflop/s (peak) performance for double-precision floating point operations and 4.29 Tflop/s for single-precision.

The Kepler GK110B architecture highlights the following improvements over previous generations:

- Streaming Multiprocessor (SMX): Features a new generation of Streaming Multiprocessor (SM)
- Dynamic parallelism: Ability to launch nested CUDA kernels
- Hyper-Q: Allows several CPU threads or processes to dispatch CUDA kernels concurrently

As a result, the NVIDIA Tesla K40 GPU can be logically viewed:

- Fifteen SMX multiprocessors, each with the following components:
  - 192 single-precision CUDA cores
  - 64 double-precision units
  - 32 special function units (SFU)
  - 32 load/store units
  - 16 texture filtering units
- Four warp schedulers and eight instruction dispatch units per SMX
- Six 64-bit memory controllers for each GPU

Each thread can address up to 255 registers. The single and double-precision arithmetic units fully comply with the IEEE 754-2008 standard, including an implementation of fused multiply-add (FMA).

A *warp* is a group of 32 parallel threads that are scheduled to run in an SMX multiprocessor. The maximum number of warps for each SMX is 64. The SMX can issue and execute four warps simultaneously. Figure 8-1 on page 194 shows the organization among multiprocessors, warps, and threads with the Tesla K40 GPU.

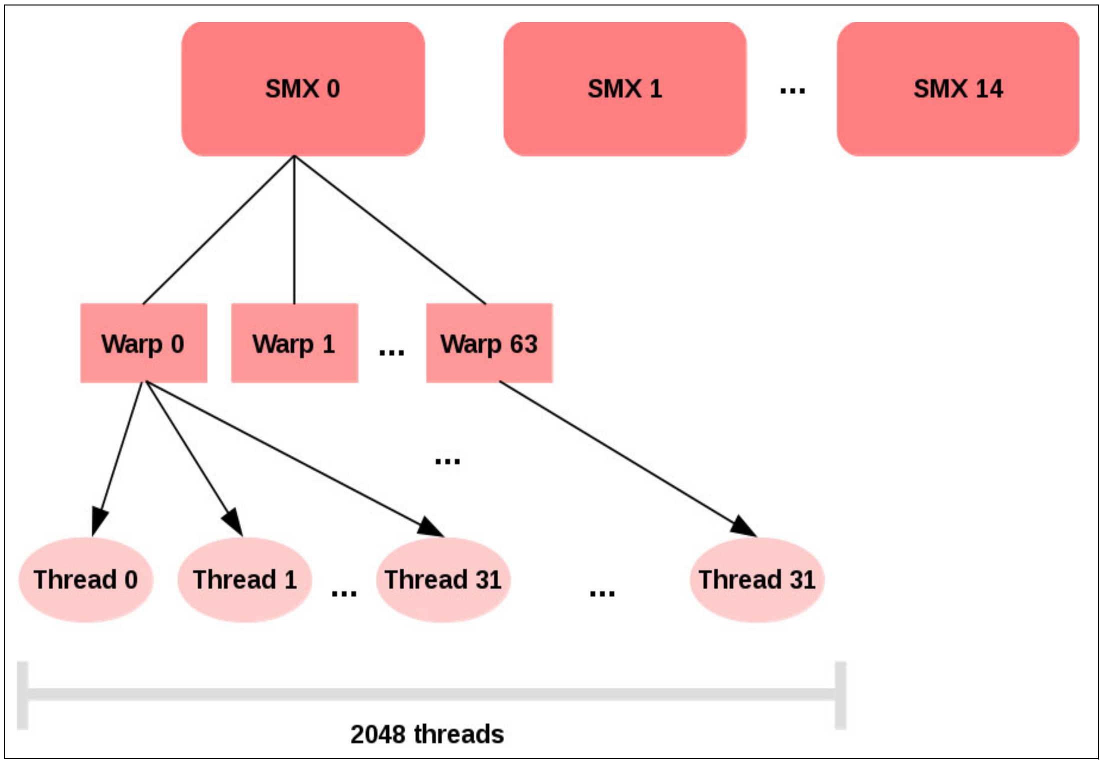


Figure 8-1 NVIDIA Tesla K40 GPU warps organization

The memory hierarchy is shown in Figure 8-2 on page 195. The memory hierarchy is divided into the following levels:

- 64 KB configurable shared memory and L1 cache for each multiprocessor.

  Three configurations are available:

  – 48 KB shared memory and 16 KB L1 cache
  – 16 KB shared memory and 48 KB L1 cache
  – 32 KB shared memory and 32 KB L1 cache

  The shared memory is accessed in words of 8 bytes.

- 48 KB read-only data cache for each multiprocessor.
- 1536 KB L2 cache.
- 12 GB dynamic random access memory (DRAM) (GDDR5).

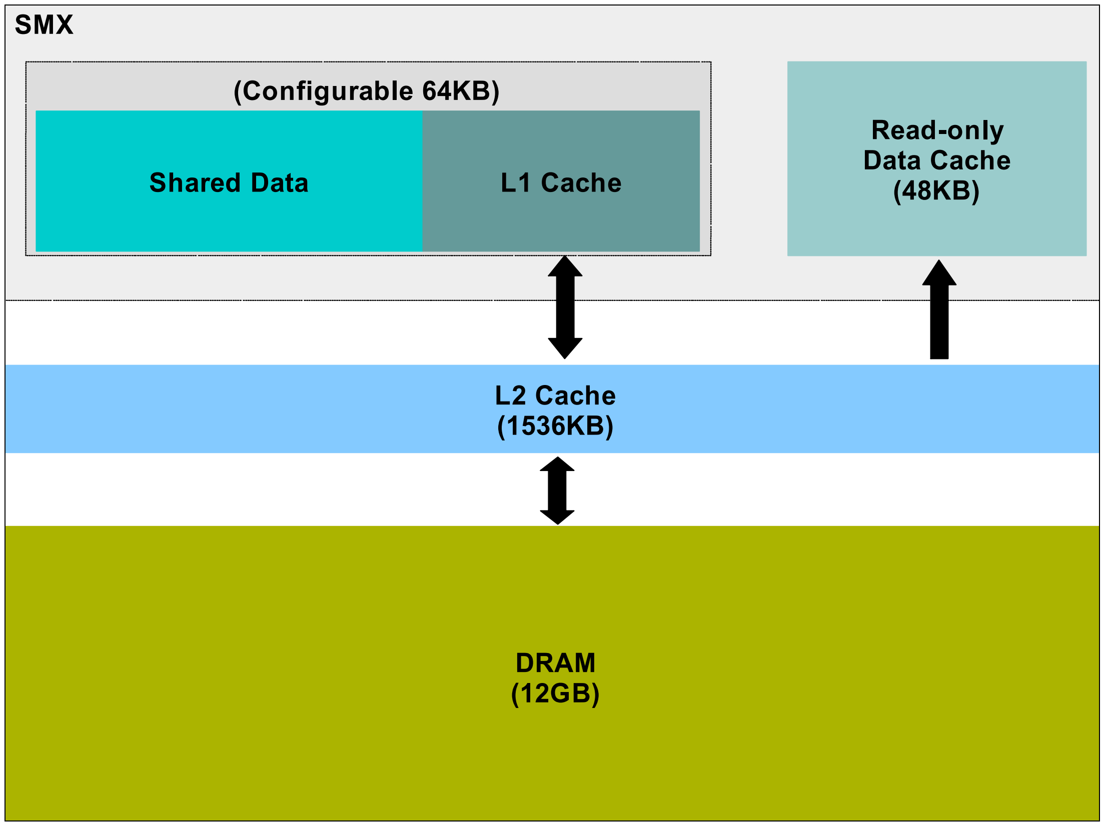


Figure 8-2 NVIDIA Tesla K40 GPU memory hierarchy

Memory errors in the shared memory, L1/L2 cache, and DRAM are remedied with the use of an error correction code (ECC) implementation of a single-error correction or double-error detection (SECDED) scheme. The read-only data cache uses parity checking to detect problems and a single-error correction mechanism, which consists of reload data from the L2 cache.

The Hyper-Q technology, which is also known as *CUDA Streams*, is controlled by hardware that offers up to 32 connections to the work queues.

The NVIDIA GPU Boost feature is designed to accelerate applications by taking advantage of the power headroom on the graphical card. A user application cannot take full advantage of a 235 W Tesla K40 power budget. With the NVIDIA GPU Boost feature, users can change the default SMX clock frequency to higher values, while the power consumption is maintained under the allowed limit. The processor clock options vary from 324 MHz up to 875 MHz. The default value is 745 MHz. The memory clock frequency is mostly kept at 3 GHz in any profile, except for the lowest 324 MHz processor clock frequency option as shown in Table 8-1 on page 196. Because 745 MHz is the base processor clock frequency, the 810 MHz and 857 MHz processor clock frequency options are used to boost performance.

Table 8-1 Tesla K40 GPU supported clocks

Memory clock (MHz)	Processor clock (MHz)
3004	875
3004	810
3004	745
3004	666
324	324

The detailed rationale and information about NVIDIA Tesla K40 Boost are available in the *NVIDIA GPU Boost for Tesla K40 Passive and Active Boards - Application Note* at this website:

http://www.nvidia.com/object/tesla_product_literature.html

One ×16 Peripheral Component Interconnect (PCI) Express Gen3 slot is used to connect a GPU card to the host system.

The Tesla K40 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 8-1 shows the output of the *deviceQuery* application that is included in the CUDA toolkit samples. (See 8.4.7, “NVIDIA CUDA Toolkit code samples” on page 238.) The command provides more information about the NVIDIA Tesla K40 GPU of the IBM Power System S824L server, which is discussed throughout this book.

Example 8-1 Output of the deviceQuery application (partial output omitted)

```
$ ./deviceQuery
<...>
Detected 2 CUDA Capable device(s)

Device 0: "Tesla K40m"
  CUDA Driver Version / Runtime Version          6.5 / 5.5
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 11520 MBytes (12079136768 bytes)
  (15) Multiprocessors, (192) CUDA Cores/MP:     2880 CUDA Cores
  GPU Clock rate:                                745 MHz (0.75 GHz)
  Memory Clock rate:                             3004 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:   2 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

Device 1: "Tesla K40m"
<... output omitted ...>
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
> Peer access from Tesla K40m (GPU0) -> Tesla K40m (GPU1) : No
> Peer access from Tesla K40m (GPU1) -> Tesla K40m (GPU0) : No

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 6.5, CUDA Runtime Version = 5.5,
NumDevs = 2, Device0 = Tesla K40m, Device1 = Tesla K40m
Result = PASS
```

8.2 Software stack

This section describes the installation of the software stack for running CUDA applications on POWER8 with the following components:

- CUDA: NVIDIA CUDA Toolkit v5.5 (CUDA 5.5) for POWER8
- Operating system: Ubuntu Server 14.10 in non-virtualized mode
- C/C++ compilers: GCC[1] 4.9 or IBM XL C/C++ compilers v13.1.1
- Java: IBM SDK, Java Technology Edition 7.1-2.0[2]

Note: The instructions that are provided in this chapter do not use the xCAT functionality to document the required steps for users or scenarios that do not require or use xCAT. For similar instructions that are based on xCAT, see Chapter 3, “Software deployment and configuration” on page 25.

xCAT does not support the NVIDIA GPU configuration on POWER8 at the time of this writing. Therefore, the steps in this chapter are required.

[1] This GCC is the Ubuntu Server GCC. The Advance Toolchain GCC is not supported by CUDA at the time of this writing.
[2] Version 7 release 1, service refresh 2, Fix Pack 0.

Note: NVIDIA announced an upcoming release of CUDA 7.0 as of this writing. You can expect similarity with the content that is described in this book with changes where appropriate (for example, version numbers in command lines, and program output).

The supported operating systems for CUDA 7.0 on POWER8 now include Ubuntu Server 14.04.2 Long Term Support (LTS), in addition to Ubuntu Server 14.10 (both in non-virtualized mode).

This chapter also describes the installation of the cross-platform development packages to run a development environment on an x86-64 computer with CUDA applications that run on the POWER8 system (remotely) with the following components:

- CUDA: NVIDIA CUDA Toolkit v5.5 (CUDA 5.5) for POWER8 (cross-platform on x86-64)
- Integrated development environment (IDE): NVIDIA NSight Eclipse Edition version 5.5.0
- Operating system: Ubuntu 14.04 LTS

Additionally, this chapter describes the configuration of CPU Frequency Scaling.

Parts of this section (for example, performing the operating system installation) include instructions with Intelligent Platform Management Interface (IPMI) commands. The IPMI commands and functionality are described in 3.3, "Intelligent Platform Management Interface (IPMI)" on page 28.

8.2.1 Ubuntu Server

The supported operating system for CUDA 5.5 on POWER8 is Ubuntu Server 14.10 in non-virtualized mode.

Note: The supported operating systems for CUDA 7.0 on POWER8 now include Ubuntu Server 14.04.2 LTS, in addition to Ubuntu Server 14.10 (both in non-virtualized mode).

To install Ubuntu Server on the IBM Power System S824L, you can choose one of the following methods:

- CD/DVD installation
- Network installation (netboot)

Both methods use an IPMI console session to perform the installation process.

Petitboot (system bootloader) is used to boot the installation media. The basic instructions for operating it are listed:

- To move the selection between elements, use the up/down arrows or Tab/Shift-Tab.
- To select an entry, confirm ("click") a button, mark a check box, move the selection to it, and press Enter or the Spacebar.
- To return from a window (discard changes), press Esc.
- To enter long text strings, you can use the "paste" feature of your terminal.
- The bottom lines display shortcut keys/help.

Power on and open the console session

The following steps are used to power on and open the console session:

1. Power on (or power cycle, if already on) the system:

   ```
   $ ipmitool -I lanplus -H fsp-address -P ipmi-password power on # or cycle
   ```

2. Open a console session:

   ```
   $ ipmitool -I lanplus -H fsp-address -P ipmi-password sol activate
   ```

3. Wait a few minutes for the Petitboot window.

CD/DVD installation

The following steps consist of downloading an Ubuntu Server installation ISO image, burning it to a CD/DVD and booting the CD/DVD:

1. Download the Ubuntu Server installation image (`ubuntu-`*`version`*`-server-ppc64el.iso`), which is at the following website[3] and select **version** → **release**:

   http://cdimage.ubuntu.com/releases

   Or, download the Ubuntu Server installation image (`ubuntu-`*`version`*`-server-ppc64el.iso`) from the following website and select **Download** → **Server** → **POWER8** → **Ubuntu Server *version***:

   http://ubuntu.com

2. Burn (write) the ISO image to a CD/DVD.

   The general instructions are available at this website:

   http://help.ubuntu.com/community/BurningIsoHowto

   Example 8-2 shows how to burn the ISO image to a CD/DVD in the command-line interface.

   Example 8-2 Burning an ISO to CD/DVD by using the wodim command-line program

   ```
   $ dmesg | grep writer # check the CD/DVD writer device (usually /dev/sr0)
   $ sudo apt-get install wodim
   $ sudo wodim -v dev=/dev/sr0 ubuntu-version-server-ppc64el.iso
   ```

3. Insert the CD/DVD into the system's optical drive.
4. In the Petitboot window, select **Rescan devices**.
5. Wait a few seconds for a new entry (from the Ubuntu Server CD/DVD).
6. Select the new entry.
7. Wait for the installer to start, and proceed with the Ubuntu Server installation.

Network installation (netboot)

The following steps consist of configuring and testing the network, and booting the network installation files.

[3] Direct link (version 14.10 "Utopic Unicorn"):
http://cdimage.ubuntu.com/releases/14.10/release/ubuntu-14.10-server-ppc64el.iso

Network configuration

To perform the network configuration:

1. In the Petitboot window, select **System Configuration** to go to the Petitboot System Configuration window.
2. From the Network list, select one of the following network configuration options:
   - Dynamic Host Configuration Protocol (DHCP) on all active interfaces
   - DHCP on a specific interface
   - Static IP configuration
3. Provide the network settings, if any, in the required fields under the Network list:

   Note: The domain name services (DNS) server field is required if the DNS servers are not provided by DHCP.

   - DHCP on all active interfaces: **None**
   - DHCP on a specific interface: **Device**
   - Static IP configuration: **Device, IP/mask, Gateway, DNS servers**
4. Select **OK**.

Network configuration test

Use the following steps to test the network configuration:

1. In the Petitboot window, select **Exit to shell** to go to the Petitboot shell.
2. Test the network. (See Example 8-3 for a successful `ping` test.)

   Example 8-3 Network configuration test in the Petitboot shell

```
# ping -c1 ubuntu.com
PING ubuntu.com (91.189.94.40): 56 data bytes
64 bytes from 91.189.94.40: seq=0 ttl=47 time=73.004 ms

--- ubuntu.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 73.004/73.004/73.004 ms
```

3. Enter the `exit` command to return to the Petitboot window.

```
# exit
```

If the network configuration test succeeds, proceed to the next steps. Otherwise, repeat the network configuration and test steps, and verify your network settings.

Tip: You can use normal Linux network commands (for example, `ping`, `ip`, `ifconfig`, and `route`) in the Petitboot shell to debug network configuration issues.

New entry for netboot files

The netboot files are the kernel (`vmlinux` file) and initial RAM disk (`initrd.gz` file). You will need the URL (link address) for both files, which can be obtained at the following website[4]. Select ***version*** → **ppc64el** → **ubuntu-installer** → **ppc64el**:

http://cdimage.ubuntu.com/netboot/

Or, go to this website and select **Download** → **Server** → **POWER8** → **Netboot image** ***version*** → **ubuntu-installer** → **ppc64el**:

http://ubuntu.com

Follow these steps:

1. Copy the URLs for the kernel (`vmlinux` file) and initial RAM disk (`initrd.gz` file).
2. In the Petitboot window, press the n key to create a new entry. (Pressing the n key takes you to the Petitboot Option Editor window.)
3. From the Device list, select **Specify paths/URLs manually**.
4. In the Kernel field, enter the URL for `vmlinux`:

   http://ports.ubuntu.com/ubuntu-ports/dists/utopic-updates/main/installer-ppc64el/current/images/netboot/ubuntu-installer/ppc64el/vmlinux
5. In the Initrd field, enter the URL for `initrd.gz`:

   http://ports.ubuntu.com/ubuntu-ports/dists/utopic-updates/main/installer-ppc64el/current/images/netboot/ubuntu-installer/ppc64el/initrd.gz
6. Optional: If you have a `preseed`[5] file (which is used for automatic installation), enter the related options in the Boot arguments field.
7. Select **OK** to return to the Petitboot window.
8. Select **User item 1** (new entry).
9. The bottom line changes to the following information:

   ```
   Info: Booting <url>
   ```
10. Wait for the files to download.
11. The bottom lines change to the following information:

   ```
   The system is going down NOW!
   Sent SIGTERM to all processes
   Sent SIGKILL to all processes
   ```
12. Wait for the installer to start, and proceed with the Ubuntu Server installation.

[4] Direct link (version 14.10 “Utopic Unicorn”): http://ports.ubuntu.com/ubuntu-ports/dists/utopic-updates/main/installer-ppc64el/current/images/netboot/ubuntu-installer/ppc64el

[5] For more information, check the *Ubuntu Installation Guide*, Appendix B, “Automating the installation using preseeding”.

Ubuntu Server installer

The Ubuntu Server installation process is standard, regardless of the GPU cards or IBM Power System S824L. See the Ubuntu Server installation documentation for instructions:

- In the Overview, go to this website and select **Ubuntu *version*** → **Ubuntu Server Guide** → **Installation**:

  http://help.ubuntu.com

- In the Ubuntu Installation Guide, go to this website and select **Ubuntu *version*** → **Installing Ubuntu** → **IBM/Motorola PowerPC**[6]:

  http://help.ubuntu.com

During the installation, in the Software selection window, select **OpenSSH Server** to access the system through Secure Shell (SSH) on the first boot.

After the installation finishes, the system reboots. Next, verify and configure the autoboot settings.

Autoboot configuration

By default, Petitboot automatically boots (*autoboot*) from any disk or network interface within 10 seconds. This time is reasonable if you install only one operating system, but you can change or configure it depending on your preferences:

1. In the Petitboot window, select **System configuration** to go to the Petitboot System Configuration window.
2. From the Autoboot list, select one of the following autoboot options:
   - Do not autoboot (wait for user selection)
   - Autoboot from any disk/network device (default)
   - Only Autoboot from a specific disk/network device
3. Provide values for the required fields, if any, under the Autoboot list:
   - Only Autoboot from a specific disk/network device: Select the device from the disk/net list

     Note: You can check the disk devices with the operating systems that are detected in the Petitboot window.

4. Select **OK**.

   Petitboot does not automatically boot any entry now because of user interaction. This time, it is necessary to boot the Ubuntu Server manually.
5. Select **Ubuntu**.
6. Wait for the operating system to boot (login prompt).

Verifying the network connectivity

The network settings are inherited from the installation process. Therefore, networking is working now. You can verify the network connectivity with the following commands.

[6] This documentation is targeted at PowerPC desktop systems, but it provides general instructions.

You can verify the network connectivity with the following commands from another system:

- Ping the system:

  ```
  $ ping -c1 system-address
  ```

- Open an SSH connection:

  ```
  $ ssh user@system-address
  ```

If the ping and SSH connection tests succeed, proceed to the next steps. Otherwise, verify and configure the system's network settings, and restart the network interface (Example 8-4).

Example 8-4 The network configuration file, its manual page, and a network interface restart

```
$ sudo nano /etc/network/interfaces
$ man interfaces
$ sudo ifdown ethX
$ sudo ifup ethX
```

Close the console session

If you can open SSH connections to the system, you can perform the next steps by using SSH, which is more convenient. It is a preferred practice to close the IPMI console session.

8.2.2 CUDA toolkit

To install the CUDA toolkit, follow these steps:

1. Verify that the GPU cards are detected. (Example 8-5 shows two GPU cards.)

   Example 8-5 Verifying the GPU cards with the lspci command

   ```
   $ lspci | grep -i nvidia
   0002:01:00.0 3D controller: NVIDIA Corporation GK110BGL [Tesla K40m] (rev a1)
   0006:01:00.0 3D controller: NVIDIA Corporation GK110BGL [Tesla K40m] (rev a1)
   ```

2. Install the basic development packages (CUDA toolkit dependencies):

   ```
   $ sudo apt-get install build-essential
   ```

3. Download the DEB package of the CUDA repository for Ubuntu 14.10 on POWER8. The DEB package is available in the Downloads section of the NVIDIA CUDA Zone website. Select **Ubuntu 14.10 DEB**:

   http://developer.nvidia.com/cuda-downloads-power8

   Note: On future CUDA releases, the files that are mentioned will be available in the normal NVIDIA CUDA downloads page, under the Linux POWER8 tab:

   http://developer.nvidia.com/cuda-downloads

   Use this command to download the DEB package, plus the website:

   ```
   $ wget
   ```

   http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1410/ppc64el/cuda-repo-ubuntu1410_5.5-54_ppc64el.deb

4. Install the DEB package:

   ```
   $ sudo dpkg -i cuda-repo-ubuntu1410_5.5-54_ppc64el.deb
   ```

   This command inserts the CUDA repository in the package manager configuration file (Example 8-6).

   Example 8-6 Package manager configuration file for the CUDA repository

   ```
   $ cat /etc/apt/sources.list.d/cuda.list
   deb http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1410/ppc64el /
   ```

5. Update the package manager's definitions:

   ```
   $ sudo apt-get update
   ```

6. Install the CUDA toolkit by using the *cuda* meta-package (Example 8-7).

   With the *cuda* meta-package, the appropriate packages of the CUDA toolkit version are available for the architecture and their dependencies.

   Among other steps, the installation builds and configures the kernel modules for the NVIDIA graphics driver, which requires the basic development tools that were installed previously.

   The support for Dynamic Kernel Module Support (DKMS) is installed by default (specified as a dependency of the `nvidia-340` package), so that the kernel modules are automatically rebuilt on kernel upgrades, if required.

   Example 8-7 Installation of the cuda meta-package

   ```
   $ sudo apt-get install cuda
   <...>
   The following NEW packages will be installed:
     acpid cuda cuda-5-5-power8 cuda-command-line-tools-5-5-power8 <...>
     cuda-cusparse-5-5-power8 cuda-cusparse-dev-5-5-power8 <...>
     cuda-misc-headers-5-5-power8 cuda-npp-5-5-power8 <...>
     dkms libcuda1-340 nvidia-340 nvidia-340-dev nvidia-340-uvm
   <...>
   Do you want to continue? [Y/n] y
   <...>
   Loading new nvidia-340-340.50 DKMS files...
   First Installation: checking all kernels...
   Building only for 3.16.0-28-generic
   Building for architecture ppc64el
   Building initial module for 3.16.0-28-generic
   Done.
   <...>
   DKMS: install completed.
   <...>
   ********************************************************************************
   **
   Reboot your computer and verify that the nvidia graphics driver is loaded.
   If the driver fails to load, use the NVIDIA graphics driver .run installer
   to get into a stable state.
   ********************************************************************************
   **
   <...>
   ```

7. Configure the search paths for the CUDA commands and libraries. The setting is available to all users and persistent across reboots (Example 8-8).

Example 8-8 Configuration of search paths for CUDA commands and libraries

```
$ echo 'export PATH=$PATH:/usr/local/cuda-5.5-power8/bin' | sudo tee
/etc/profile.d/cuda.sh
$ echo /usr/local/cuda-5.5-power8/lib64 | sudo tee /etc/ld.so.conf.d/cuda.conf
$ sudo ldconfig
```

Note: The changes to the command search path environment variable (**`PATH`**) do not affect the current shell. (However, the shared library-related changes affect the current shell.) To apply the changes to the current shell, run the following command:

```
$ source /etc/profile.d/cuda.sh
```

If the shared library search path is not configured correctly, CUDA applications that link to the CUDA shared libraries fail, as shown in Example 8-9.

Example 8-9 Failure of CUDA applications with misconfigured shared library search path

```
<application>: error while loading shared libraries: <cuda library>.so.x.x:
cannot open shared object file: No such file or directory
```

Note: Refer to the *Linux Getting Started Guide*, which is also known as the *NVIDIA CUDA Getting Started Guide for Linux*, at the following website. Select **Installing CUDA Development Tools** → **Install the NVIDIA CUDA Toolkit** → **Package Manager Installation** → **Environment Setup** for a different approach:

http://developer.nvidia.com/cuda-downloads

The following export entries are useful if you do not have superuser privileges in the system. The downside is that these entries are not available to all users and they are not permanent across reboots. Also, the shared library search path can be inadvertently overridden when you build or run non-system-wide installed software.

```
$ export PATH=$PATH:/usr/local/cuda-5.5-power8/bin
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-5.5-power8/lib64
```

To make this setting permanent across reboots for certain users, they must add these lines to their `~/.bashrc` file.

8. Reboot the system:

```
$ sudo reboot
```

9. After you reboot the system, verify that the *nvidia* module is loaded and that the GPU devices are available (Example 8-10).

Note: The kernel message "`nvidia: module verification failed`" is not an error. The message refers to loading modules with a proprietary license. (See the lines before the message.)

Example 8-10 Verification of nvidia module and GPU status

```
$ dmesg | grep -i nvidia
[<...>] nvidia: module license 'NVIDIA' taints kernel.
[<...>] nvidia: module license 'NVIDIA' taints kernel.
```

```
[<...>] nvidia: module verification failed: signature and/or  required key
missing - tainting kernel
[<...>] nvidia 0002:01:00.0: enabling device (0140 -> 0142)
[<...>] nvidia 0006:01:00.0: enabling device (0140 -> 0142)
[<...>] [drm] Initialized nvidia-drm 0.0.0 20130102 for 0002:01:00.0 on minor 0
[<...>] [drm] Initialized nvidia-drm 0.0.0 20130102 for 0006:01:00.0 on minor 1
[<...>] NVRM: loading NVIDIA UNIX ppc64le Kernel Module  340.50 <...>

$ lsmod | grep nvidia
nvidia              13941291  0
drm                   382761  2 nvidia

$ nvidia-smi --list-gpus
GPU 0: Tesla K40m (UUID: GPU-9c8168e0-dc96-678f-902f-3997a6f2dfc5)
GPU 1: Tesla K40m (UUID: GPU-6d42c48f-2d92-c254-9703-7218aa15fa40)
```

10. Perform a simple test with the CUDA sample *simpleCUFFT*, which uses CUDA libraries and GCC (default compiler), as shown in Example 8-11[7].

Example 8-11 Building and running simpleCUFFT with GCC

```
$ cp -r /usr/local/cuda-5.5-power8/samples/ ~/cuda-samples
$ cd ~/cuda-samples/7_CUDALibraries/simpleCUFFT/
$ make
/usr/local/cuda-5.5-power8/bin/nvcc -ccbin g++ -I../../common/inc  -m64
-gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,code=sm_30
-gencode arch=compute_35,code=\"sm_35,compute_35\" -o simpleCUFFT.o -c
simpleCUFFT.cu
/usr/local/cuda-5.5-power8/bin/nvcc -ccbin g++   -m64        -o simpleCUFFT
simpleCUFFT.o  -lcufft
mkdir -p ../../bin/ppc64le/linux/release
cp simpleCUFFT ../../bin/ppc64le/linux/release

$ ../../bin/ppc64le/linux/release/simpleCUFFT
[simpleCUFFT] is starting...
GPU Device 0: "Tesla K40m" with compute capability 3.5

Transforming signal cufftExecC2C
Launching ComplexPointwiseMulAndScale<<< >>>
Transforming signal back cufftExecC2C
```

For more information, see the *Linux Getting Started Guide*, which is also known as the *NVIDIA CUDA Getting Started Guide for Linux*, which is available in the Downloads section of the NVIDIA CUDA Zone website:

http://developer.nvidia.com/cuda-downloads

8.2.3 IBM XL C/C++ compiler

To install the IBM XL C/C++ compiler, perform the following steps:

1. Install the basic development packages (IBM XL C/C++ compiler dependencies):

```
$ sudo apt-get install build-essential
```

[7] Shell linebreaks were added for clarity.

2. Install the DEB packages that are provided with your installation media (Example 8-12 on page 207).

Example 8-12 Installation of the XL C/C++ Compiler packages

```
$ cd <path to IBM XL C/C++ deb-packages>

$ sudo dpkg -i xlc*.deb libxlc*.deb libxlmass*.deb libxlsmp*.deb # or *.deb
<...>
Setting up xlc-license.13.1.1 (13.1.1.0-141105) ...
Setting up libxlc (13.1.1.0-141105) ...
Setting up libxlc-devel.13.1.1 (13.1.1.0-141105) ...
Setting up libxlmass-devel.8.1.0 (8.1.0.0-141027) ...
Setting up libxlsmp (4.1.0.0-141010) ...
Setting up libxlsmp-devel.4.1.0 (4.1.0.0-141010) ...
Setting up xlc.13.1.1 (13.1.1.0-141105) ...
Run 'sudo /opt/ibm/xlC/13.1.1/bin/xlc_configure' to review the license and
configure the compiler.
```

3. To review the license and configure the compiler, enter this command:

   `$ sudo /opt/ibm/xlC/13.1.1/bin/xlc_configure`

   For non-interactive installations, see Example 8-13.

Example 8-13 Reviewing the license and configuring the IBM XL C/C++ compiler non-interactively

```
$ echo 1 | sudo /opt/ibm/xlC/13.1.1/bin/xlc_configure
<...>
International License Agreement <...>
<...>
Press Enter to continue viewing the license agreement, or, Enter "1" to accept
the agreement, "2" to decline it or "99" to go back to the previous
screen, "3" Print.
[INPUT] ==> 1
<...>
update-alternatives: using /opt/ibm/xlC/13.1.1/bin/xlc to provide /usr/bin/xlc
(xlc) in auto mode
<...>
INFORMATIONAL: GCC version used in
"/opt/ibm/xlC/13.1.1/etc/xlc.cfg.ubuntu.14.04.gcc.4.8.2" -- "4.8.2"
INFORMATIONAL: /opt/ibm/xlC/13.1.1/bin/xlc_configure completed successfully
```

4. Perform a simple test with the CUDA sample *simpleCUFFT*, which uses CUDA libraries and IBM XL C++ (by using the `-ccbin xlC` option for **nvcc**, which can be defined in CUDA samples with the make variable `GCC`), as shown in Example 8-14.

   Consider these points:

   – The compiler warning is not an error; it refers to analysis of the source code.
   – The output of running *simpleCUFFT* is the same as the output that is shown in Example 8-11 on page 206[8].

Example 8-14 Building and running simpleCUFFT with IBM XL C++

```
Note: this step may have been performed in the other example (GCC).
$ cp -r /usr/local/cuda-5.5-power8/samples/ ~/cuda-samples
$ cd ~/cuda-samples/7_CUDALibraries/simpleCUFFT/
```

[8] Shell linebreaks were added for clarity.

```
$ make clean
rm -f simpleCUFFT.o simpleCUFFT
rm -rf ../../bin/ppc64le/linux/release/simpleCUFFT

$ make GCC=xlC
/usr/local/cuda-5.5-power8/bin/nvcc -ccbin xlC -I../../common/inc  -m64
-gencode arch=compute_20,code=sm_20 -gencode arch=compute_30,code=sm_30
-gencode arch=compute_35,code=\"sm_35,compute_35\" -o simpleCUFFT.o -c
simpleCUFFT.cu
../../common/inc/helper_cuda.h:493:9: warning: 4 enumeration values not handled
in switch: 'CUFFT_INCOMPLETE_PARAMETER_LIST', 'CUFFT_INVALID_DEVICE',
      'CUFFT_PARSE_ERROR'... [-Wswitch]
switch (error)
        ^
1 warning generated.
/usr/local/cuda-5.5-power8/bin/nvcc -ccbin xlC   -m64        -o simpleCUFFT
simpleCUFFT.o  -lcufft
mkdir -p ../../bin/ppc64le/linux/release
cp simpleCUFFT ../../bin/ppc64le/linux/release

$ ../../bin/ppc64le/linux/release/simpleCUFFT
<...>
```

For more information, see the IBM Knowledge Center website and select **Rational** → **C and C++ Compilers** → **XL C/C++ for Linux** → **XL C/C++ for Linux *version*** → **Installation Guide:**

http://www.ibm.com/support/knowledgecenter

8.2.4 Java

To install the IBM Developer Kit for Linux, Java Technology Edition, perform the following steps:

1. Download the IBM Developer Kit for Linux, Java Technology Edition, which is available at the following website. In row Linux, select **Downloads** → **Java SE Version 7**. In row 64-bit IBM POWER (LE), select **Download Now**:

   http://www.ibm.com/developerworks/java/jdk/

   Note: Download the SDK file `ibm-java-ppc64le-sdk-<version>.bin`. (This file is not the SDK in the `archive` format, and it is not the Java Runtime Environment (JRE) only because JRE is included in the SDK.)

2. Make the file executable:

   ```
   $ chmod +x ibm-java-ppc64le-sdk-7.1-2.0.bin
   ```

3. Perform the installation with one of the following methods:

   – Interactively to review the license and options:

   ```
   $ sudo ./ibm-java-ppc64le-sdk-7.1-2.0.bin
   ```

   – Non-interactively to use the default options:

   ```
   $ sudo ./ibm-java-ppc64le-sdk-7.1-2.0.bin -i silent
   ```

4. Configure the environment and command search path for Java (Example 8-15 on page 209)[9].

Example 8-15 Configuring the environment and command search path for Java

```
$ cat <<"EOF" | sudo tee /etc/profile.d/ibm-java.sh
export JAVA_HOME=/opt/ibm/java-ppc64le-71
export PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin
EOF
```

Note: The changes to the shell environment (**`PATH`**) do not affect the current shell. To apply the changes to the current shell, run:

```
$ source /etc/profile.d/ibm-java.sh
```

5. Verify the Java virtual machine (JVM) and Java Compiler commands (Example 8-16).

Example 8-16 Verifying the Java virtual machine and Java Compiler commands

```
$ java -version
java version "1.7.0"
Java(TM) SE Runtime Environment (build pxl6470_27sr2-20141101_01(SR2))
IBM J9 VM (build 2.7, JRE 1.7.0 Linux ppc64le-64 Compressed References
20141031_220034 (JIT enabled, AOT enabled)
J9VM - R27_Java727_SR2_20141031_1950_B220034
JIT  - tr.r13.java_20141003_74587.02
GC   - R27_Java727_SR2_20141031_1950_B220034_CMPRSS
J9CL - 20141031_220034)
JCL - 20141004_01 based on Oracle 7u71-b13

$ javac -version
javac 1.7.0-internal
```

6. Perform a simple test that uses the *com.ibm.gpu* classes (that are discussed in "CUDA and IBM Java" on page 223) to list the available GPUs. (Example 8-17 shows two GPUs.)

   Notes about the **`java`** command options:

   – *-Xmso512k*: Java requires an operating system thread stack size of at least 300 KB for CUDA functionality (otherwise, an exception is triggered), as of this writing.

   – *-Dcom.ibm.gpu.enable*: Enables CUDA functionality.

   – *-Dcom.ibm.gpu.verbose*: Verbose logging of the CUDA functionality.

Example 8-17 Java application with the com.ibm.gpu classes to list the available GPUs

```
$ cat > ListGPUs.java <<EOF
import com.ibm.gpu.*;

public class ListGPUs {
   public static void main(String[] args) throws Exception {
      System.out.println(CUDAManager.getInstance().getCUDADevices());
   }
}
EOF

$ javac ListGPUs.java
```

[9] The first EOF delimiter is quoted ("EOF") to avoid variable expansion.

```
$ java -Xmso512k -Dcom.ibm.gpu.enable ListGPUs
[com.ibm.gpu.CUDADevice@5530ad1, com.ibm.gpu.CUDADevice@611a71b1]

$ java -Xmso512k -Dcom.ibm.gpu.enable -Dcom.ibm.gpu.verbose ListGPUs
[IBM GPU]: [<...>]: System property com.ibm.gpu.enable=
[IBM GPU]: [<...>]: Enabling sort on the GPU
[IBM GPU]: [<...>]: Discovered 2 devices
[IBM GPU]: [<...>]: Providing identifiers of discovered CUDA devices, found: 2
devices
[IBM GPU]: [<...>]: Discovered devices have the following identifier(s):
0, 1
[com.ibm.gpu.CUDADevice@5530ad1, com.ibm.gpu.CUDADevice@611a71b1
```

For more information about the development of Java applications that use GPUs, see the IBM Knowledge Center website and select **WebSphere®** → **IBM SDK, Java Technology Edition** → **IBM SDK, Java Technology Edition *version*** → **Developing Java applications** → **Writing Java applications that use a graphics processing unit**:

http://www.ibm.com/support/knowledgecenter

For more information about the installation and configuration of the IBM JDK, see the IBM Knowledge Center website and select **WebSphere** → **IBM SDK, Java Technology Edition** → **IBM SDK, Java Technology Edition *version*** → **Linux User Guide for IBM SDK, Java Technology Edition, *version*** → **Installing and configuring the SDK and Runtime Environment**:

http://www.ibm.com/support/knowledgecenter

Also, see the IBM developerWorks pages for IBM Developer Kit for Linux, Java Technology Edition:

http://www.ibm.com/developerworks/java/jdk/linux

Note: The terms “IBM Developer Kit for Linux, Java Technology Edition” and “IBM SDK, Java Technology Edition” are used interchangeably in this section.

8.2.5 CPU frequency scaling

Linux provides CPU frequency scaling with the `cpufreq` mechanism. It can adjust the processors’ clock frequency over time, according to certain policies, which are called *governors*.

The default scaling governor in Ubuntu Server is `ondemand`, which dynamically adjusts CPU frequency according to system load, on a processor-core level. This function enables power-savings without compromising performance. A core’s frequency is maintained low when idle, and increased under load.

On highly used HPC systems, the preferable scaling governor is `performance`, which statically sets CPU frequency to the highest available frequency. This function ensures that a core is clocked at maximal frequency at all times, and eliminates any frequency-scaling overhead and jitter.

The `cpufreq` settings are per-CPU (hardware thread) files that are placed in this directory:

/sys/devices/system/cpu/cpu<*number*>/cpufreq/

Although the `cpufreq` settings are exposed *per-CPU* (or *per-hardware thread*), they are actually *per-core* on POWER8 (that is, equal among all hardware threads in a core). This design is reflected in the `related_cpus` file, as seen in Example 8-18, which shows that CPU (or hardware thread) 0 shares settings with other CPUs (hardware threads) of its core (its eight hardware threads, in SMT8 mode).

Example 8-18 The related cpus file

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/related_cpus
0 1 2 3 4 5 6 7
```

For example, the current scaling governor can be displayed or changed by using the `scaling_governor` file, and the available scaling governors are displayed in the `scaling_available_governors` file (Example 8-19).

Example 8-19 Accessing the scaling_governor and scaling_available_governors files

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
ondemand

$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
conservative userspace powersave ondemand performance

$ echo performance | sudo tee
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance

$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
performance
```

Set the scaling governor to performance on all CPUs

The `cpufrequtils` package provides tools for managing `cpufreq` settings on all CPUs/hardware threads (system-wide) and for making changes persistent across reboots. Follow these steps:

1. Install `cpufrequtils`:

   ```
   $ sudo apt-get install cpufrequtils
   ```

2. Add the line `GOVERNOR=“performance”` to the file `/etc/default/cpufrequtils` to make the setting persistent across reboots. You can either use a text editor or the following command:

   ```
   $ echo ‘GOVERNOR=“performance”’ | sudo tee -a /etc/default/cpufrequtils
   ```

3. Disable the system’s default `ondemand` governor setting:

   ```
   $ sudo update-rc.d ondemand disable
   ```

4. Activate the `performance` governor without rebooting the system:

   ```
   $ sudo service cpufrequtils start
   ```

You can use the **`cpufreq-info`** command to obtain the status of the `cpufreq` settings. See Example 8-20 on page 212 to verify the number of CPUs/hardware threads that are running with a certain governor.

Example 8-20 Number of CPUs with the specific scaling governor

```
$ cpufreq-info | grep 'The governor' | uniq -c
   160           The governor "performance" <...>
```

8.2.6 CUDA cross-platform development

The CUDA cross-platform development tools run in an x86-64 computer, and the CUDA applications run on the POWER8 system (remotely). The supported operating system is Ubuntu 14.04 LTS for amd64.

Note: Installation instructions for Ubuntu 14.04 LTS for amd64 are not covered in this book. See the *Ubuntu Installation Guide* at the following website. Select **Ubuntu 14.04 LTS** → **Installing Ubuntu** → **AMD64 & Intel EM64T**:

http://help.ubuntu.com

You can download Ubuntu 14.04 LTS for amd64 at the following website. Select **Download** → **Desktop** → **Ubuntu 14.04(*.x*) LTS** → **Flavour: 64-bit** → **Download**:

http://ubuntu.com

To install the cross-platform development packages, you are required to add the target system's architecture as a foreign architecture in the package manager. This task is a requirement for the CUDA packages.

For CUDA cross-platform development in POWER8 systems, add the `ppc64el` architecture (the architecture string for the 64-bit PowerPC instruction-set in little-endian mode in Ubuntu and Debian distributions) as a foreign architecture for the package manager (`dpkg`). This step does not interfere with the primary architecture of the package manager, which is still `amd64` (similarly, for the 64-bit version of the x86 instruction-set). You can see that `i386` (similarly, for the x86 instruction-set) is already a foreign architecture by default, which enables the installation of 32-bit packages (Example 8-21).

Example 8-21 Adding ppc64el as a foreign architecture in Ubuntu 14.04 LTS for amd64

```
Primary architecture:
$ dpkg --print-architecture
amd64

Foreign architectures:
$ dpkg --print-foreign-architectures
i386

$ sudo dpkg --add-architecture ppc64el

$ dpkg --print-foreign-architectures
i386
ppc64el
```

Proceed to install the CUDA repository package (which inserts the CUDA repository in the package manager configuration files), and update the package manager's definitions (Example 8-22 on page 213).

Example 8-22 Installing the CUDA repository package

```
$ wget
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1404/x86_64/cuda-rep
o-ubuntu1404_5.5-54_amd64.deb

$ sudo dpkg -i cuda-repo-ubuntu1404_5.5-54_amd64.deb

$ cat /etc/apt/sources.list.d/cuda.list
deb http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1404/x86_64 /

$ sudo apt-get update
<...>
```

Note: During the **`apt-get update`** command, several warnings are displayed for files that are "not found", for example:

```
W: Failed to fetch <...>/binary-ppc64el/Packages  404  Not Found [IP: <...>]
```

You can safely ignore these warnings. This warning happens as a result of the foreign-architecture setting that is combined with a different URL that is used for Ubuntu servers that host `ppc64el` binary packages (that is, `ports.ubuntu.com`), at the time that this publication was written.

Finally, install the `cuda-cross-ppc64el` meta-package and the `nvidia-nsight` package (Example 8-23).

Example 8-23 Installing the cuda-cross-ppc64el meta-package and the nvidia-nsight package

```
$ sudo apt-get install cuda-cross-ppc64el
<...>
The following NEW packages will be installed:
<...>
  cuda-core-5-5-power8 cuda-cross-ppc64el cuda-cross-ppc64el-5-5-power8
<...>
Do you want to continue? [Y/n] y
WARNING: The following packages cannot be authenticated!
<...>
Install these packages without verification? [y/N] y
<...>
Setting up cuda-cross-ppc64el (5.5-54) ...
*** LICENSE AGREEMENT ***
<...>

$ sudo apt-get install nvidia-nsight
<...>
The following NEW packages will be installed:
  nvidia-cuda-doc nvidia-nsight
<...>
Do you want to continue? [Y/n] y
<...>
```

For more information, see the *Linux Getting Started Guide*, which is also known as the *NVIDIA CUDA Getting Started Guide for Linux*, at the following website. Select **Installing CUDA Development Tools** → **Install the NVIDIA CUDA Toolkit Package** → **Manager Installation** → **Available packages**:

http://developer.nvidia.com/cuda-downloads

8.3 System monitoring

The **nvidia-smi** tool, which is provided by NVIDIA, is used to manage and monitor the activities of the GPU devices. In Ubuntu 14.10, the **nvidia-smi** tool is included with the nvidia-340 DEB package. Because **nvidia-smi** has many options, the topic is only introduced in this section. We suggest that you learn more about the **nvidia-smi** tool at the following website:

http://developer.nvidia.com/nvidia-system-management-interface

Querying the state and properties of devices

Run **nvidia-smi** to show a summary of all of the GPUs that are attached to the system. Use **nvidia-smi -q** to display the current state and properties of the devices (Example 8-24). An individual device can be accessed with the `-i` *<id>* option, where *<id>* is the identification number of the card.

Example 8-24 The nvidia-smi tool: Displaying devices' states with the -q option (partial output omitted)

```
$ sudo nvidia-smi -q

==============NVSMI LOG==============

Timestamp                           : Thu Nov 27 10:32:44 2014
Driver Version                      : 340.50

Attached GPUs                       : 2
GPU 0002:01:00.0
    Product Name                    : Tesla K40m
    Product Brand                   : Tesla
    Display Mode                    : Disabled
    Display Active                  : Disabled
    Persistence Mode                : Disabled
    Accounting Mode                 : Disabled
    Accounting Mode Buffer Size     : 128
    Driver Model
        Current                     : N/A
        Pending                     : N/A
    Serial Number                   : 0324114015717
    GPU UUID                        : GPU-9c8168e0-dc96-678f-902f-3997a6f2dfc5
    Minor Number                    : 0
    VBIOS Version                   : 80.80.3E.00.01
    MultiGPU Board                  : No
    Board ID                        : 0x20100
    Inforom Version
        Image Version               : 2081.0202.01.04
        OEM Object                  : 1.1
        ECC Object                  : 3.0
        Power Management Object     : N/A
```

```
GPU Operation Mode
    Current                     : N/A
    Pending                     : N/A
PCI
    Bus                         : 0x01
    Device                      : 0x00
    Domain                      : 0x0002
    Device Id                   : 0x102310DE
    Bus Id                      : 0002:01:00.0
    Sub System Id               : 0x097E10DE
    GPU Link Info
        PCIe Generation
            Max                 : 3
            Current             : 3
        Link Width
            Max                 : 16x
            Current             : 16x
    Bridge Chip
        Type                    : N/A
        Firmware                : N/A
Fan Speed                       : N/A
Performance State               : P0
Clocks Throttle Reasons
    Idle                        : Not Active
    Applications Clocks Setting : 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            : 39 C
        GPU Shutdown Temp           : 95 C
        GPU Slowdown Temp           : 90 C
    Power Readings
        Power Management            : Supported
        Power Draw                  : 63.16 W
        Power Limit                 : 235.00 W
        Default Power Limit         : 235.00 W
        Enforced Power Limit        : 235.00 W
        Min Power Limit             : 180.00 W
        Max Power Limit             : 235.00 W
    Clocks
        Graphics                    : 745 MHz
        SM                          : 745 MHz
        Memory                      : 3004 MHz
    Applications Clocks
        Graphics                    : 745 MHz
        Memory                      : 3004 MHz
    Default Applications Clocks
        Graphics                    : 745 MHz
        Memory                      : 3004 MHz
    Max Clocks
        Graphics                    : 875 MHz
        SM                          : 875 MHz
        Memory                      : 3004 MHz
    Clock Policy
        Auto Boost                  : N/A
        Auto Boost Default          : N/A
    Compute Processes               : None
```

```
GPU 0006:01:00.0
    Product Name                        : Tesla K40m
    Product Brand                       : Tesla
<... output omitted ...>
```

Query one or more of the device properties by adding the `-d` *`<property_name>`* option, where *`property_name`* can assume the values that are shown in Table 8-2.

Table 8-2 The nvidia-smi tool: Query the names of the properties

Property	Description
MEMORY	Show memory usage statistics
UTILIZATION	Show utilization statistics of GPU, memory, encoder, and decoder subsystems
ECC	Show status of ECC and error statistics
TEMPERATURE	Display GPU temperature
POWER	Show power management status
CLOCK	Show clock rate of graphics, Streaming Multiprocessor (SM), memory, and applications; display usage statistics
SUPPORTED_CLOCKS	Display available combinations to GPU and memory clock boost
COMPUTE	Display current compute mode
PIDS	List running applications
PERFORMANCE	Show performance statistics
PAGE_RETIREMENT	Display number of retired pages
ACCOUNTING	Display current accounting mode

Managing devices

Several device settings can be enabled (`1`) and disabled (`0`) by passing `1` (one) or `0` (zero) to the management options of the **`nvidia-smi`** command. For example, use option `-e` to set/unset error correction code (ECC) support and use `-p` to reset ECC errors.

To save the current properties between driver unloads, use the `-e 1` option to turn on persistence mode. This option loads the driver even though no compute application is executing.

The GPU compute mode is set with the `-c` option. The allowed values are `0/DEFAULT`, `1/EXCLUSIVE_THREAD`, `2/PROHIBITED`, and `3/EXCLUSIVE_PROCESS`.

To boost performance, you can use other **`nvidia-smi`** options to manage memory and graphics clock rates for computer applications. Use option `-ac` to set clock values, `-rac` to reset to base clocks, and `-acp` to permit non-root users' clock changes. You can also control the auto boost mechanism by using the `--auto-boost-default`, `--auto-boost-default-force`, and `--auto-boost-permission` options.

8.4 Application development

The IBM POWER8 chip is a highly parallel microprocessor with strong memory subsystem and high floating-point performance.

Moreover, POWER8 is the first IBM POWER architecture processor that supports little-endian memory access mode in virtualized and non-virtualized (or bare-metal) mode. At the time that this publication was written, the IBM Power System S824L included the Ubuntu Linux operating system distribution that supports little-endian.

To fully use the computational power of the IBM Power System S824L, you need to harness the computing capabilities of the GPU that is included with the system. The GPU acceleration can be used to offload highly parallel operations and boost Java performance.

The application developer has access to the GPU accelerator through the NVIDIA CUDA parallel computing platform and programming model. In theory, it is possible to develop applications that take advantage of the benefits of GPU even without the knowledge of the CUDA programming model. The CUDA toolkit includes a set of libraries that are tuned for NVIDIA GPUs.

In the area of technical computing, the NVIDIA CUDA infrastructure is known to application developers. For more information, see the NVIDIA website:

`http://developer.nvidia.com/cuda-zone`

In the area of enterprise application development, IBM made significant steps toward the integration of CUDA into a Java environment. Currently, IBM JDK includes Java functions that directly take advantage of the GPU. The CUDA4J application programming interface (API) makes it possible to invoke external CUDA kernels directly from the Java code. These topics are covered in 8.4.5, "CUDA and IBM Java" on page 223.

The CUDA toolkit includes the Nsight Eclipse Edition integrated development environment (IDE) for CUDA applications. For an explanation of how to create, edit, build, and launch projects from within the environment, see 8.4.6, "Nsight" on page 227.

This publication emphasizes only those CUDA programming features that are specific for POWER8 server systems. The manuals that relate to your system are available in the `<cuda_installation_dir>/doc/` directory of your server or cross-platform development computer.

8.4.1 CUDA paradigm

CUDA is a parallel computing platform and programming model by NVIDIA. Remember the following points about CUDA:

- CUDA is about throughput, not about latency
- CUDA is for highly data-parallel applications

A POWER8 server that is equipped with GPU accelerators constitutes a perfectly balanced system in a way. The primary beneficiaries of this union are hybrid applications: the applications that have both highly data-parallel parts and latency-critical pieces that require large memory bandwidth.

The most favorable workload for GPU is where you need to execute the same sequence of operations on each element of a large data set. Traditionally, these data sets are image and video processing applications, and graphics rendering. The CUDA programming model also proved to be useful in the following areas:

- Computational biology
- Chemistry
- Physics
- Molecular dynamics
- Bioinformatics
- Material science
- Computational fluid dynamics
- Structural mechanics
- Computer-aided design
- Electronic design automation
- Seismic processing and interpretation
- Reservoir simulation
- Financial analytics
- Data mining

The CUDA application typically adheres to the following execution model:

1. Data is copied from main memory into GPU memory.
2. The CPU instructs GPU to schedule processing.
3. The GPU computing cores execute a specified GPU kernel function.
4. Data is copied back to the main memory from GPU memory.

To better use the computing resources of CPU and GPU, it is better to overlap memory copy and computation operations by using CUDA asynchronous memory transfers and CUDA execution streams.

In CUDA for C/C++, the source files that need to be compiled with the CUDA compiler can have the `.cu` file extension. The source files that do not contain CUDA statements have the usual `.c` or `.cpp` file extension.

8.4.2 Compilers

The CUDA application includes parts that are executed solely on a CPU (host code) and pieces that are run solely on a GPU (device code). The NVIDIA CUDA Compiler Driver (NVCC) takes the source code and forwards the host code functions to the program that is specified with the `-ccbin` option as a host code compiler.

Based on the CUDA programming model, both GCC and the IBM XL C/C++ compiler can be used for host code compilation. The following sections demonstrate how to compile CUDA applications by using GCC and the IBM XL C/C++ compilers.

NVIDIA CUDA Compiler Driver

The NVCC is a compiler that is intended for use with CUDA. It is based on the widely used LLVM (formerly Low Level Virtual Machine) open source compiler infrastructure. CUDA codes run on both the CPU and GPU. NVCC separates these two parts and sends the host source code (the part of the code that runs on the CPU) to a C/C++ compiler, and sends the device source code (the part that runs on the GPU) to NVIDIA compilers/assemblers.

Any source file that contains CUDA language extensions (`.cu`) must be compiled with NVCC. NVCC is a compiler driver that simplifies the process of compiling C/C++ code.

NVCC provides simple and familiar command-line options and executes them by starting the collection of tools that implement the different compilation stages. For more information about NVCC, see the NVIDIA website:

http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc

IBM XL C/C++ compiler as the host compiler

The IBM XL C/C++ compiler v13.1.1 is the first version to support CUDA for POWER8 on little-endian Linux.

In this section, the CUDA sample project *simplePrintf* program is used as an example. (See 8.4.7, "NVIDIA CUDA Toolkit code samples" on page 238.) The project is at this location:

`<cuda_installation_dir>/samples/0_Simple/simplePrintf`

To compile the example with the IBM XL C/C++ compiler as the host compiler, copy the source code files to a directory where you have file read, write, and execute permission. Run the **`make GCC=xlC`** command. The output program is copied to the `release` directory (Example 8-25).

Example 8-25 Building the simplePrintf sample

```
$ cp -r /usr/local/cuda-5.5-power8/samples/ ~/cuda-samples
$ cd ~/cuda-samples/0_Simple/simplePrintf/
$ make GCC=xlC
<...>
cp simplePrintf ../../bin/ppc64le/linux/release
```

The following command shows how to compile the example by manually starting an **`nvcc`** compilation command:

```
$ nvcc -ccbin xlC -I<cuda_installation_dir>/samples/common/inc -m64 \
    -gencode arch=compute_35,code=sm_35 simplePrintf.cu -o simplePrintf
```

Example 8-26 shows the output of executing the *simplePrintf* program.

Example 8-26 The output of the CUDA application example (simplePrintf)

```
GPU Device 0: "Tesla K40m" with compute capability 3.5

Device 0: "Tesla K40m" with Compute 3.5 capability
printf() is called. Output:

[0, 0]:         Value is:10
[0, 1]:         Value is:10
[0, 2]:         Value is:10
<... output omitted ...>
[2, 7]:         Value is:10
```

The IBM Power System S824L server supports NVIDIA Tesla K40 GPUs that are based on the NVIDIA Kepler architecture GK110B. For application binary compatibility, set the **`nvcc`** `code=sm_35` option. To enable full support for the Kepler architecture, the `arch=compute_35` option is also required. For more information, see the following website:

http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/#application-compatibility

GCC compiler as the host compiler

To compile the CUDA *clock* program sample with **gcc** as the host compiler, go to the source code directory and run this command:

```
$ make
```

The sample code can also be compiled with **nvcc** on the command line as shown:

```
$ nvcc -ccbin g++ -I<cuda_installation_dir>/samples/common/inc -m64 \
    -gencode arch=compute_35,code=sm_35 -o clock.o -c clock.cu
```

The output of executing the *clock* program is shown in Example 8-27.

Example 8-27 The output of the CUDA application example (clock)

```
CUDA Clock sample
GPU Device 0: "Tesla K40m" with compute capability 3.5

Total clocks = 654801
```

8.4.3 Running CUDA applications

You can run a CUDA application from the command line as a normal application. No special action is required. But in certain cases, you need to provide a path to the shared libraries. (See the configuration steps that are discussed in 8.2.2, “CUDA toolkit” on page 203.) Or, you can control the execution by setting environment variables.

Environment variables

The execution of CUDA applications can be controlled by setting several environment variables. An important environment variable to highlight is the **CUDA_VISIBLE_DEVICES** environment variable. This variable helps to control the devices that your application uses. CUDA applications only see those devices whose index is given in the sequence that is assigned to the variable. The devices are also enumerated in the order of the sequence.

The *deviceQuery* example from the CUDA toolkit code samples helps to illustrate the effect that the **CUDA_VISIBLE_DEVICES** environment variable has on the devices that are available for an application. (See 8.4.7, “NVIDIA CUDA Toolkit code samples” on page 238.) For a system with two GPU cards, the result of the *deviceQuery* execution is listed in Example 8-28 for various values that are assigned to the **CUDA_VISIBLE_DEVICES** environment variable.

Example 8-28 The CUDA_VISIBLE_DEVICES environment variable effect on device enumeration

```
$ unset CUDA_VISIBLE_DEVICES
$ ./deviceQuery | grep PCI
  Device PCI Domain ID / Bus ID / location ID:   2 / 1 / 0
  Device PCI Domain ID / Bus ID / location ID:   6 / 1 / 0
$ CUDA_VISIBLE_DEVICES=0,1 ./deviceQuery | grep PCI
  Device PCI Domain ID / Bus ID / location ID:   2 / 1 / 0
  Device PCI Domain ID / Bus ID / location ID:   6 / 1 / 0
$ CUDA_VISIBLE_DEVICES=1,0 ./deviceQuery | grep PCI
  Device PCI Domain ID / Bus ID / location ID:   6 / 1 / 0
  Device PCI Domain ID / Bus ID / location ID:   2 / 1 / 0
$ CUDA_VISIBLE_DEVICES=0 ./deviceQuery | grep PCI
  Device PCI Domain ID / Bus ID / location ID:   2 / 1 / 0
$ CUDA_VISIBLE_DEVICES=1 ./deviceQuery | grep PCI
  Device PCI Domain ID / Bus ID / location ID:   6 / 1 / 0
```

8.4.4 GPU accelerated libraries

The CUDA toolkit includes a set of highly optimized GPU accelerated libraries. We suggest that you use routines from these libraries instead of writing your own implementation of standard functions.

As part of the CUDA toolkit, you can use the following APIs and libraries:

CUDA math API	This module contains a set of regular mathematical functions to use in device code (trigonometric and hyperbolic functions, error functions, logarithmic and exponential functions, rounding functions, and so on).
CUBLAS	The CUBLAS library implements Basic Linear Algebra Subprograms (BLAS) on top of CUDA. You can use BLAS level 3 functions if you want to fully use the computational power of your GPU.
CUFFT	The CUFFT library contains a set of routines for computing discrete fast fourier transform (FFT). It consists of two separate libraries: CUFFT and CUFFTW. The CUFFT library uses GPU to compute FFT for complex-valued and real-valued data sets. The CUFFTW library provides the FFTW3 API to facilitate porting of existing applications that use the FFTW[10] library.
CURAND	The CURAND library provides random number generation facilities. It consists of two parts: a library on the host (CPU) side and a library on the device (GPU) side. The host-side library is used to generate a set of random numbers in host or device memory for later use. The device-side library defines random number generation functions to be used by user-written kernels. The latter approach implies that random data is immediately consumed by user kernels, and that global memory copying operations are not required.
CUSPARSE	This module includes subroutines that implement basic linear algebra operations with sparse vectors and matrixes. The library supports several storage formats for sparse matrixes.
NPP	The focus of the NVIDIA Performance Primitives (NPP) library is digital signal processing. The library contains a set of routines to handle imaging and video data.
Thrust	Thrust is a C++ template library for CUDA, which was inspired by the C++ Standard Template Library (STL). The library provides a collection of common data parallel primitives (transformations, reductions, prefix-sums, reordering, and sorting). The application developer creates complex algorithms that are based on these primitives, and thrust automatically tries to choose the most efficient implementation.

You can access user guides for these libraries in the “CUDA API References” section of the CUDA toolkit documentation. (See 8.4, “Application development” on page 218.)

Compilation and linking

When you write code that uses CUDA toolkit libraries, include the corresponding header files and link with the related libraries. For example, if your program *cufft_sample.cu* depends on the CUFFT library, compile it with the following command:

```
$ nvcc -ccbin xlC -m64 cufft_sample.cu -o cufft_sample -lcufft
```

[10] See the FFTW home page at this website: http://fftw.org

For the names of the header files and the libraries, check the “CUDA API References” section of the CUDA Toolkit Documentation. (See 8.4, “Application development” on page 218.)

Sometimes, your environment cannot contain full paths to the compilers and options for the header files and libraries. In this case, explicitly specify them on the command line:

- `<cuda_installation_dir>/bin/nvcc`
- `<ibmxl_installation_dir>/bin/xlC` or `<gcc_installation_dir>/bin/g++`
- `-I<cuda_installation_dir>/include`
- `-L<cuda_installation_dir>/lib64`

8.4.5 CUDA and IBM Java

Many features of the IBM POWER8 processor were designed to work with big data and analytics workloads. Many data processing applications are currently written in the Java programming language. If you need to accelerate such applications, offload the most computationally intensive parts to hardware accelerators. However, it is unrealistic to completely refactor such applications into CUDA C/C++.

To provide a better experience for Java users and to use the capabilities of GPU-accelerated POWER8 servers, IBM SDK, Java Technology Edition started to include support for offloading computations to the GPU directly from the Java code. IBM SDK, Java Technology Edition support for GPU started from version 7 release 1, service refresh 2, fix pack 0 (7.1-2.0).

The application developer has several options for using GPU on POWER8 from the Java code:

- Let the Java virtual machine (JVM) decide when to offload processing to a GPU.
- Use the *com.ibm.gpu* classes to offload specific tasks.
- Use the CUDA4J API to specify in the application exactly when to use the GPU.

As usual, your Java program can target a specific GPU if you set the **`CUDA_VISIBLE_DEVICES`** environment variable, as described in “Environment variables” on page 221.

If you experience any runtime issues and want to trace operations that occur with Java applications that use GPU, check the “GPU problem determination” subsection of the problem determination section of the *Linux User Guide for IBM SDK, Java Technology Edition* at the following website:

http://www.ibm.com/support/knowledgecenter/SSYKE2

Relying on the JVM logic to offload processing to a GPU

Certain Java functions are particularly useful for GPU processing. One class of such functions is array processing routines. For example, a data array can be sorted on a GPU faster than on a CPU. However, to benefit from this type of processing, the array must be of a sufficient size. This size is required to justify the time of data movement between the CPU and the GPU.

JVM, which is shipped with IBM SDK, Java Technology Edition, is able to automatically offload certain Java functions. This offload happens when the JVM expects that the speed of data processing at the GPU outweighs the cost of data movement from main memory to the GPU. With this option, you can use the GPU processing power without needing to change the source code. The ability to offload computations to a GPU is controlled by setting the system property when you start JVM by using the command line.

Example 8-29 lists simple Java code that uses the *sort()* function from the *java.util.Arrays* package. This code runs without any changes on any platform that supports Java.

Example 8-29 Source file of a program that uses the sort() function from the java.util.Arrays package

```
import java.util.Arrays;
import java.util.Random;

public class BuiltInSort {
  public static void main(String[] args) {
    int N = 128*1024*1024;
    int[] toSort = new int[N];
    Random rnd = new Random();
    for (int i = 0; i < N; ++i) {
      toSort[i] = rnd.nextInt();
    }
    long startTime = System.nanoTime();
    Arrays.sort(toSort);
    long estimatedTime = System.nanoTime() - startTime;
    System.out.println("CPU\t" + N + '\t' + estimatedTime * 1e-9 + " seconds");
  }
}
```

Example 8-30 shows how to compile the code and run it by supplying various options to the JVM that is included with IBM SDK, Java Technology Edition.

Example 8-30 Compiling and running the program that uses the sort() function (partial output omitted)

```
$ javac BuiltInSort.java

$ java -Xmso4m -Xmx12g -Dcom.ibm.gpu.verbose  BuiltInSort
CPU     134217728       16.67377956 seconds

$ java -Xmso4m -Xmx12g -Dcom.ibm.gpu.verbose -Dcom.ibm.gpu.disable BuiltInSort
[IBM GPU]: [<...>]: System property com.ibm.gpu.disable=
[IBM GPU]: [<...>]: Disabling sort on the GPU
CPU     134217728       15.439728421000002 seconds

$ java -Xmso4m -Xmx12g -Dcom.ibm.gpu.verbose -Dcom.ibm.gpu.enable=sort BuiltInSort
[IBM GPU]: [<...>]: System property com.ibm.gpu.enable=sort
[IBM GPU]: [<...>]: Enabling sort on the GPU
[IBM GPU]: [<...>]: Discovered 2 devices
[IBM GPU]: [<...>]: Providing identifiers of discovered CUDA devices, found: 2
devices
[IBM GPU]: [<...>: Discovered devices have the following identifier(s):
0, 1
[IBM GPU]: [<...>]: Acquired device: 0
[IBM GPU]: [<...>]: Using device: 0 to sort int array of length: 134217728
[IBM GPU]: [<...>]: Sorted ints on device: 0 successfully
[IBM GPU]: [<...>]: Released device: 0
CPU     134217728       4.600549149 seconds
```

The `-Xmso` runtime option sets the initial stack size for operating system threads. The `-Xmx` option specifies the maximum size of the memory allocation pool (heap).

The system properties have the following meanings:

- *-Dcom.ibm.gpu.enable*. This system property controls the type of processing that can be offloaded by JVM to a GPU.
- *-Dcom.ibm.gpu.disable*. This option enables JVM to turn off the automatic GPU offloading capability.
- *-Dcom.ibm.gpu.verbose*. This option enables a detailed logging of how JVM handles GPU processing.

The code was run on an IBM Power System S824L with 3.42 GHz cores and NVIDIA Tesla K40 GPU accelerators. The processing times that are seen in the output show that, by default, JVM does not offload the work to be performed by the *sort()* function to the GPU. When this feature is enabled, JVM discovers that the data processing requirements meet a specific threshold and offloads the sorting task to the GPU.

For more information about this feature, see "Running Java applications of the Linux User Guide for IBM SDK, Java Technology Edition" in *Enabling application processing on a graphics processing unit* at the IBM Knowledge Center:

http://www.ibm.com/support/knowledgecenter/SSYKE2

The com.ibm.gpu application programming interface

You can use the *com.ibm.gpu* classes to explicitly specify in the code when you want to use GPU for certain tasks. Example 8-31 shows how to start a sorting routine that will run on a GPU. You can compile this code with the usual **javac** command, and execute the resulting Java byte code by using ordinary **java** commands.

Example 8-31 Source file of a program that uses com.ibm.gpu classes

```
import java.util.Random;

import com.ibm.gpu.GPUConfigurationException;
import com.ibm.gpu.GPUSortException;
import com.ibm.gpu.Maths;

public class GpuSort {
  public static void main(String[] args) {
    int N = 128*1024*1024;
    int[] toSort = new int[N];
    Random rnd = new Random();
    for (int i = 0; i < N; ++i) {
      toSort[i] = rnd.nextInt();
    }
    long startTime = System.nanoTime();
    try {
      Maths.sortArray(toSort);
    } catch (GPUSortException e) {
      e.printStackTrace();
    } catch (GPUConfigurationException e) {
      e.printStackTrace();
    }
    long estimatedTime = System.nanoTime() - startTime;
    System.out.println("GPU\t" + N + '\t' + estimatedTime * 1e-9 + " seconds");
  }
}
```

Figure 8-3 shows how the performance of a sorting function that is built into Java compares to the performance of a sorting function that is performed in a GPU. Not only is the performance of the GPU sorting function higher, but the routine is more scalable than a Java native sorting routine.

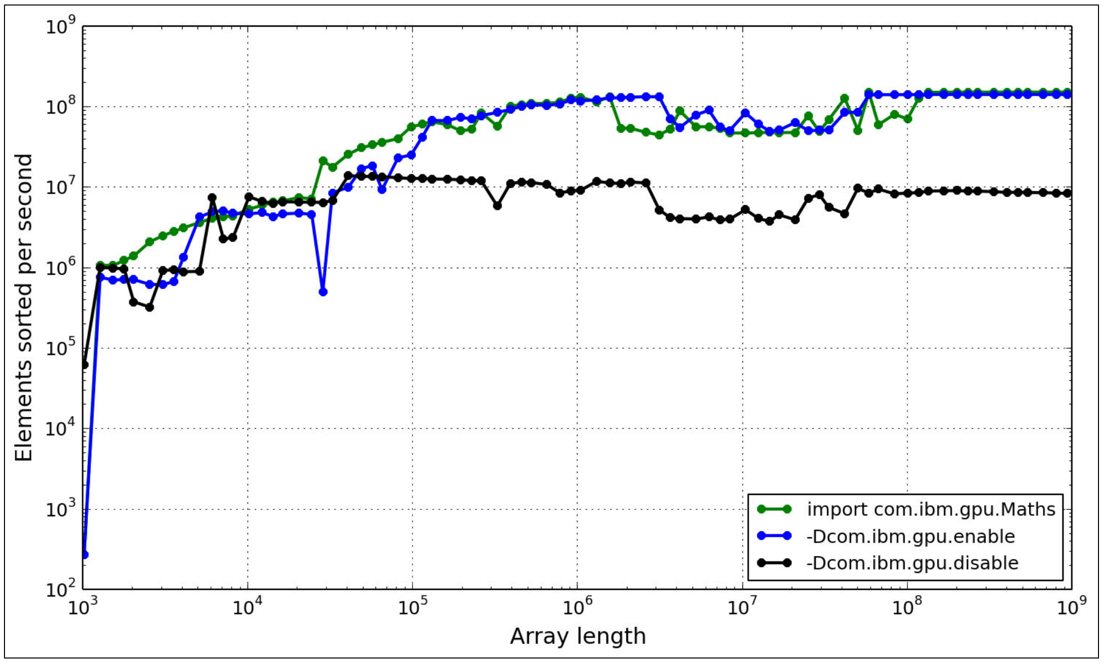


Figure 8-3 The performance of the Java sort() function when it is executed on a CPU and a GPU

In addition, with the *com.ibm.gpu* classes, you can enable verbose output programmatically. You can also provide a device identifier with the function arguments, if you want to target a specific GPU device.

For more information about the *com.ibm.gpu* classes, see "The ibm.com.gpu application programming interface" in the *Linux User Guide for IBM SDK, Java Technology Edition* at the IBM Knowledge Center website:

http://www.ibm.com/support/knowledgecenter/SSYKE2

The CUDA4J application programming interface

You can improve the performance of your Java applications by offloading certain processing functions from a CPU to a GPU. So far, this publication described the functions that are supplied as part of IBM JDK packages only.

CUDA4J is an API that is offered by IBM for developing applications in which you can specify exactly when and how to use the GPU for application processing.

With the CUDA4J API, you can develop applications that can move data between the Java heap and memory buffers on the GPU. On the GPU, kernels that are written in the C programming language can be started to process that data, and the results can be moved back into the Java heap under the control of your Java application. The CUDA4J API provides classes for setting up buffers that allow data transfer between Java memory and device memory.

For information about *com.ibm.cuda* classes that supply the CUDA4J API, see "CUDA4J" in the *Linux User Guide for IBM SDK, Java Technology Edition* at the following website:

http://www.ibm.com/support/knowledgecenter/SSYKE2

8.4.6 Nsight

Nsight Eclipse Edition is a CUDA development environment that is provided by NVIDIA as part of the CUDA toolkit's cross-platform development packages. Nsight Eclipse Edition 6.0.0 is included with CUDA 5.5 for POWER8, and it is built on Eclipse Platform 3.8.1 and CDT (C/C++ Development Tools). It includes added-value specific features for easing the GPU development cycle of coding, building, launching, debugging, and analyzing.

Note: Nsight runs on a x86-64 computer, not on the target POWER8 system.

This section explains the basics of creating, editing, building, and launching projects from within the environment. For more information about Nsight for Eclipse Edition, see this website:

http://developer.nvidia.com/nsight-eclipse-edition

Note: You will notice the differences in version numbers between the CUDA 5.5 toolkit and certain components (for example, Nsight and CUDA debugger) in screen captures and examples. These differences are mostly addressed in the CUDA 5.5-54 release and are not in CUDA 7.0.

After the CUDA cross-platform development packages are installed, and the environment variables are configured (8.2, "Software stack" on page 197) on your personal development computer, Nsight can be opened with the **`nsight`** command as shown in Example 8-32.

Example 8-32 Start Nsight development environment

```
$ nsight &
```

Create projects

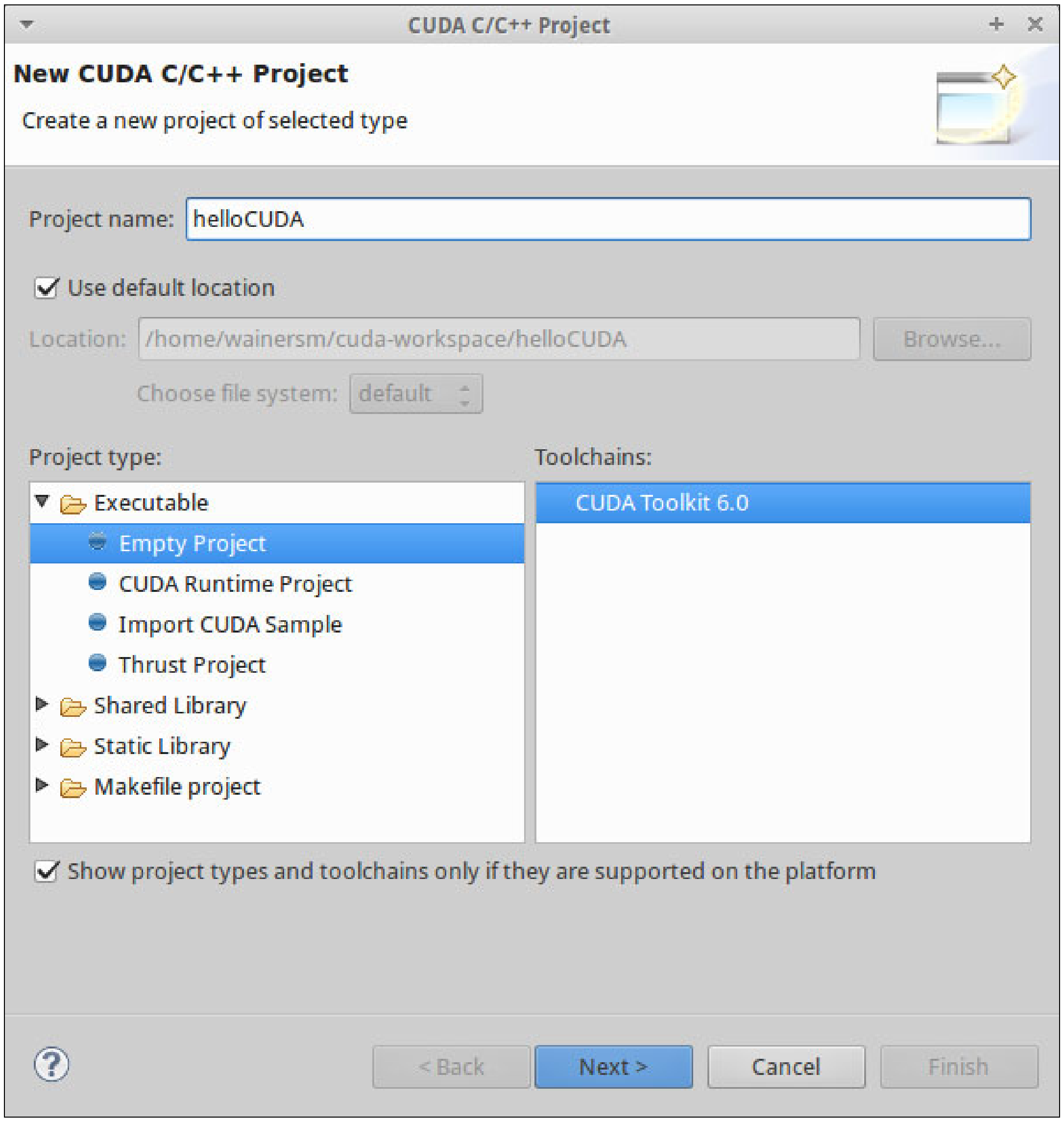
With Nsight, you can create and develop projects that are either local (with or without cross-compiling) or remote. However, the suggested mode in this book is to develop remotely. The environment generates a synchronized project where remote source files are mirrored locally so that the code editor does not show interruptions, due to network latency, when you are typing. To enable synchronization operations, you must install and configure the Git versioning tool in the remote machine, as shown in Example 8-33. You must also install the CUDA toolkit for POWER8 in the remote target machine (8.2, "Software stack" on page 197).

Example 8-33 Install and configure Git in Ubuntu

```
$ sudo apt-get install git
$ git config --global user.name "Wainer dos Santos Moschetta"
$ git config --global user.email "wainersm@br.ibm.com"
```

Follow these steps to create a remote synchronized project:

1. At the menu bar, click **File** → **New** → **CUDA C/C++ Project**. The New Project window opens, as shown in Figure 8-4 on page 228.


CUDA C/C++ Project

New CUDA C/C++ Project

Create a new project of selected type

Project name: helloCUDA

☑ Use default location

Location: /home/wainersm/cuda-workspace/helloCUDA

Choose file system: default

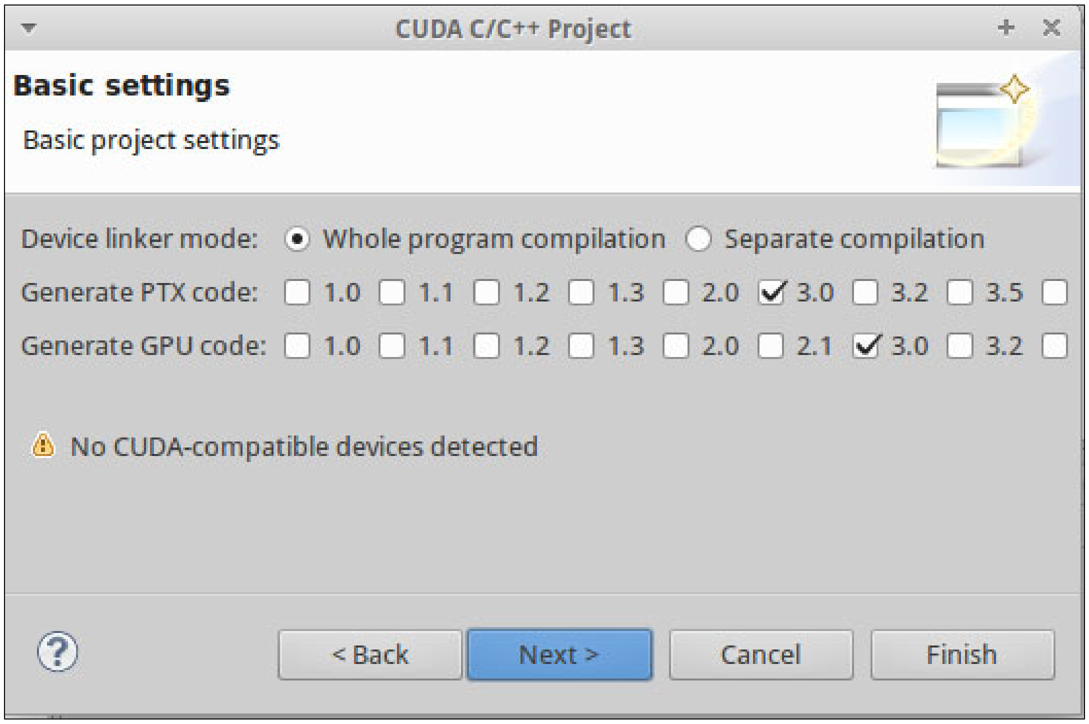
Project type:
- Executable
  - Empty Project
  - CUDA Runtime Project
  - Import CUDA Sample
  - Thrust Project
- Shared Library
- Static Library
- Makefile project

Toolchains:
- CUDA Toolkit 6.0

☑ Show project types and toolchains only if they are supported on the platform

Figure 8-4 Nsight new CUDA C/C++ project window

2. In the Project type area (left side of panel), select **Executable** → **Empty Project** to create a makefile project that is managed by Eclipse, then click **Next**. The Basic settings window opens, as shown in Figure 8-5 on page 229. On the Basic settings window, select the device linker mode that can be taken at compilation time, and also the compatibility levels of the generated executable. See “Compilers” on page 219 for available options for GPU on POWER8. These options can be changed later in the project’s Properties window.



Basic settings

Basic project settings

Device linker mode: [● Whole program compilation] ○ Separate compilation

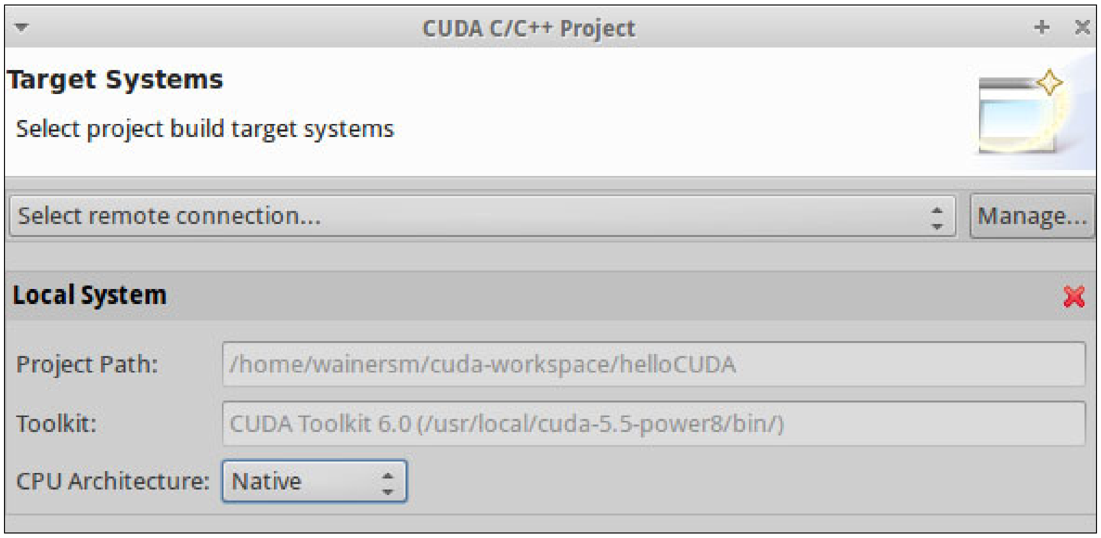
Generate PTX code: ☐ 1.0 ☐ 1.1 ☐ 1.2 ☐ 1.3 ☐ 2.0 [✓ 3.0] ☐ 3.2 ☐ 3.5 ☐

Generate GPU code: ☐ 1.0 ☐ 1.1 ☐ 1.2 ☐ 1.3 ☐ 2.0 ☐ 2.1 [✓ 3.0] ☐ 3.2 ☐

No CUDA-compatible devices detected

Figure 8-5 Basic settings for the new CUDA C/C++ project

3. Click **Next** to open the Target Systems window as shown in Figure 8-6. If a connection to the compute machine exists, select an option in the drop-down list. Otherwise, click **Manage** to add a connection to the machine. The Remote Connections window opens (see Figure 8-7 on page 230). Then, click **Finish**.



Target Systems

Select project build target systems

Select remote connection...

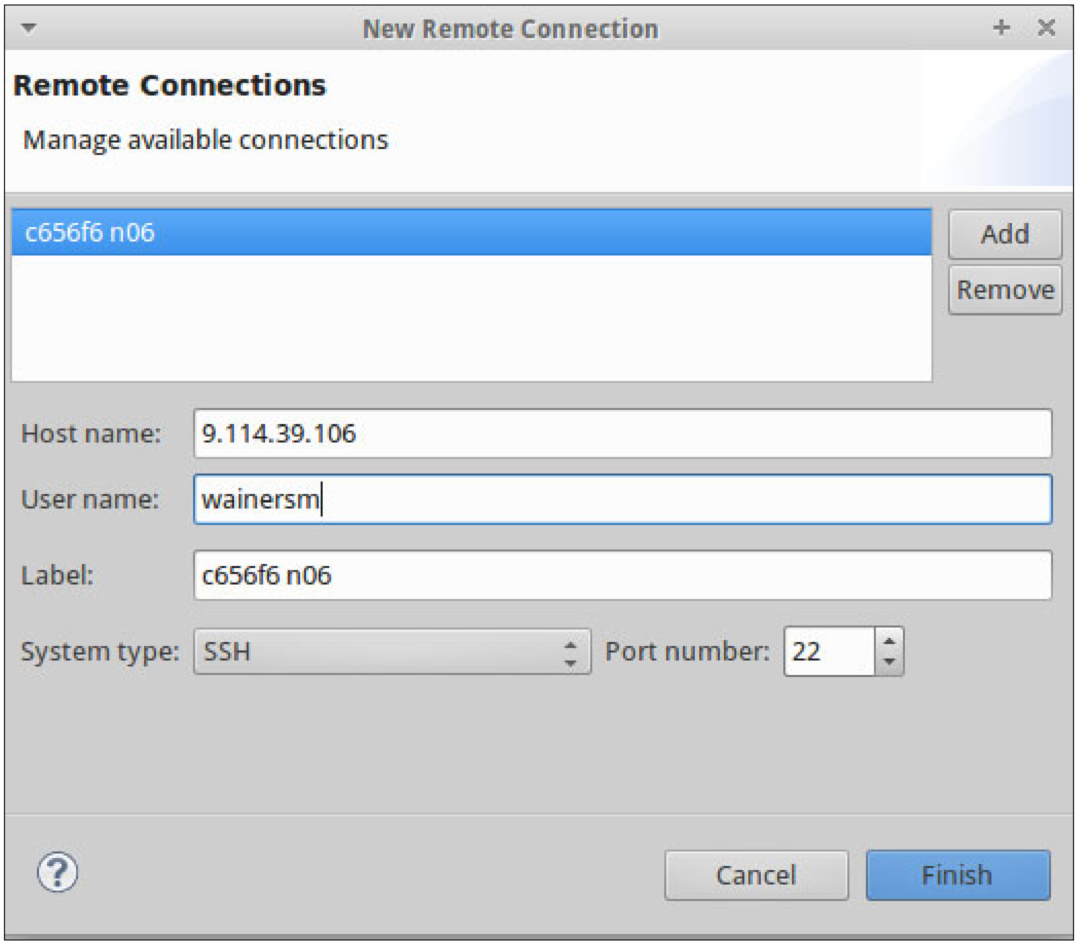
Local System

Project Path: /home/wainersm/cuda-workspace/helloCUDA

Toolkit: CUDA Toolkit 6.0 (/usr/local/cuda-5.5-power8/bin/)

CPU Architecture: Native

Figure 8-6 Initial target systems for the new CUDA C/C++ project



c656f6 n06

Host name: 9.114.39.106

User name: wainersm

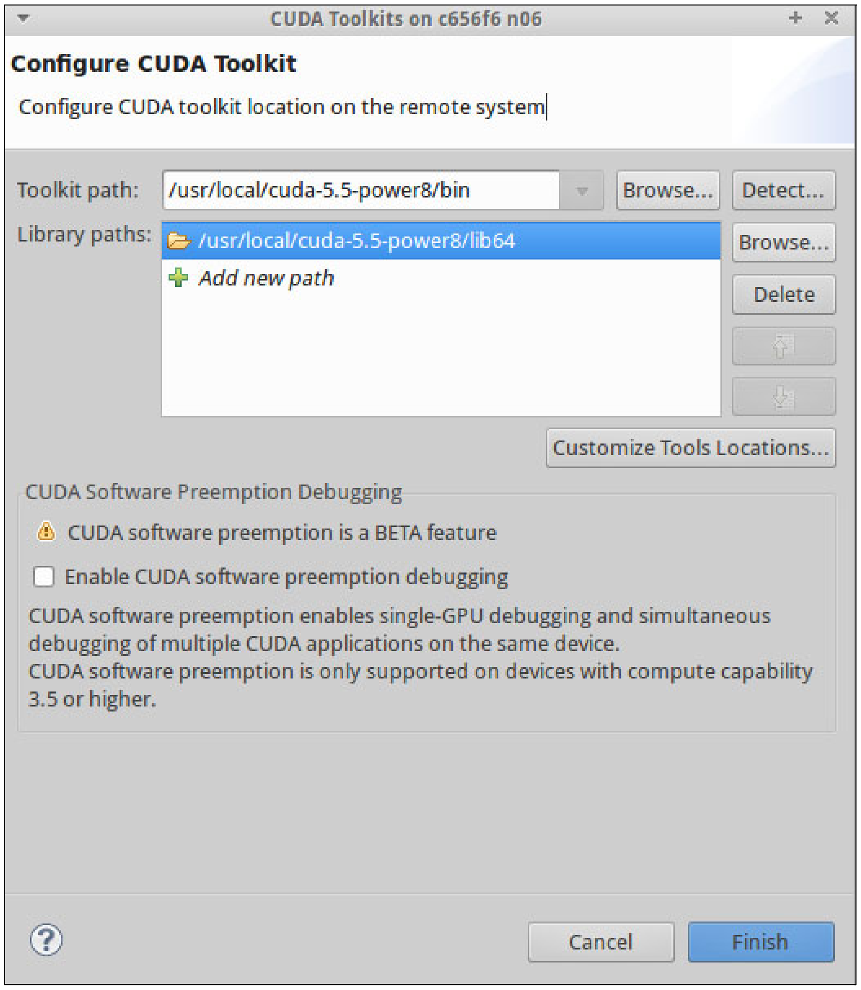
Label: c656f6 n06

System type: SSH

Port number: 22

Figure 8-7 New connection configuration window

4. The next step is to configure the location of the CUDA toolkit and libraries in the remote machine. Click **Manage** at the connection box, and the window that is shown in Figure 8-8 on page 231 opens. Set the location of the CUDA toolkit (`<`*`cuda_installation_dir`*`>/bin/`) and libraries (`<`*`cuda_installation_dir`*`>/lib64/`), and then, click **Finish**.


CUDA Toolkits on c656f6 n06

Configure CUDA Toolkit

Configure CUDA toolkit location on the remote system

Toolkit path: /usr/local/cuda-5.5-power8/bin

Library paths: /usr/local/cuda-5.5-power8/lib64

Add new path

CUDA Software Preemption Debugging

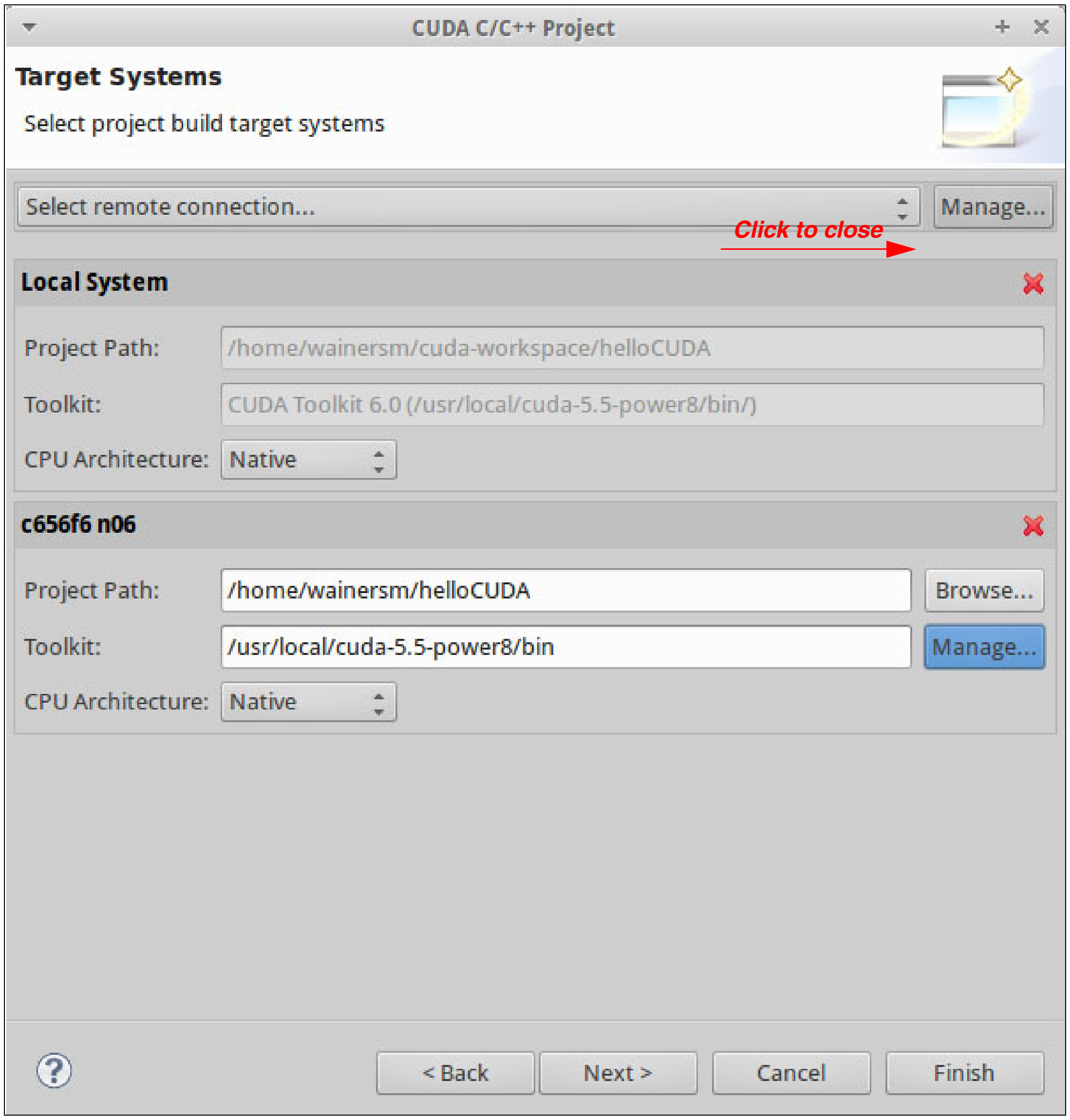
CUDA software preemption is a BETA feature

☐ Enable CUDA software preemption debugging

CUDA software preemption enables single-GPU debugging and simultaneous debugging of multiple CUDA applications on the same device.
CUDA software preemption is only supported on devices with compute capability 3.5 or higher.

Figure 8-8 Setting the CUDA toolkit and library paths on the remote machine

5. Until now, the remote connection was set correctly with a local connection that exists, by default. To avoid triggering builds locally, delete the local configuration as shown with the red arrow in Figure 8-9 on page 232.



Target Systems

Select project build target systems

Select remote connection... Manage...

Local System

Project Path: /home/wainersm/cuda-workspace/helloCUDA

Toolkit: CUDA Toolkit 6.0 (/usr/local/cuda-5.5-power8/bin/)

CPU Architecture: Native

c656f6 n06

Project Path: /home/wainersm/helloCUDA Browse...

Toolkit: /usr/local/cuda-5.5-power8/bin Manage...

CPU Architecture: Native

Figure 8-9 Target systems for the new CUDA C/C++ project

6. Click **Finish** to generate the project in the Eclipse workspace.

The result of the preceding steps is a project that has a development cycle of build, run, and debug in the remote GPU-enabled machine.

To create a sample application, click **File** → **New** → **Source file**, and choose the CUDA C Bitreverse Application template.

Building projects

Eclipse C/C++ Development Tools (CDT) fully control the build process of the project that was created in Figure 8-9. In simple terms, it generates several makefiles from the project settings to build the source code on the remote side. It is possible to generate other kinds of projects, for instance, makefile or shell script, where the developer controls the construction process, but those projects are not covered in this publication.

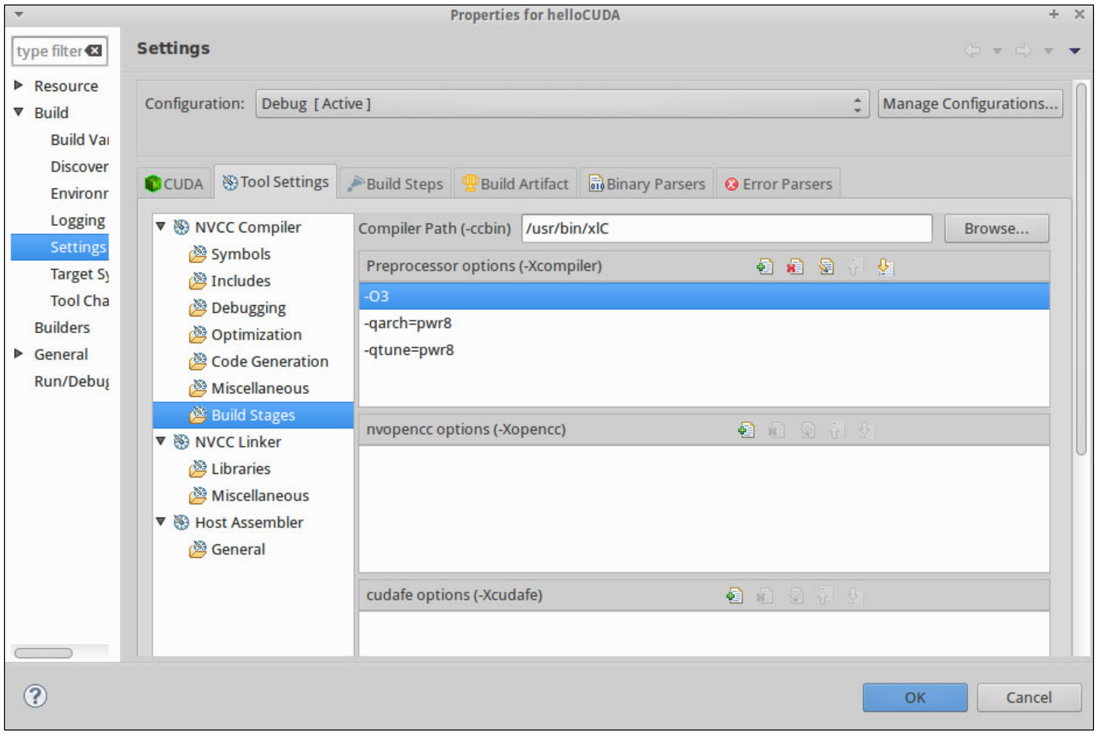
To build the project, either click the hammer icon in the main toolbar, or select **Project** → **Build Project**.

Nsight uses the preset options of the nvcc compiler from a location that was defined in the remote connection setup (Figure 8-9 on page 232). Nsight chooses the gcc or the g++ compiler. Both compilers are in the system `PATH` variable.

Several properties in the project build settings change the default behavior. Click **Project** → **Properties** on the main toolbar, expand **Build** on the left, and click **Settings** to see most of the properties.

To change the back-end compiler, follow these steps:

1. From the Build → Settings window, switch to the **Tool Settings** tab.
2. On the left panel, expand the **NVCC Compiler** and click **Build Stages**. Several properties for controlling the nvcc build stages are shown in Figure 8-10.


Properties for helloCUDA

Settings

Configuration: Debug [Active]

Compiler Path (-ccbin) /usr/bin/xlC

Preprocessor options (-Xcompiler)

-O3
-qarch=pwr8
-qtune=pwr8

nvopencc options (-Xopencc)

cudafe options (-Xcudafe)

Figure 8-10 Building settings for NVCC stages

3. Enter the path to the compiler back end in the Compiler Path (-ccbin) field. Click **OK**.

Figure 8-10 shows the configuration for setting the `xlC` back-end compiler and passing flags **`-O3`**, **`-qarch=pwr8`**, and **`-qtune=pwr8`** (Preprocessor options field).

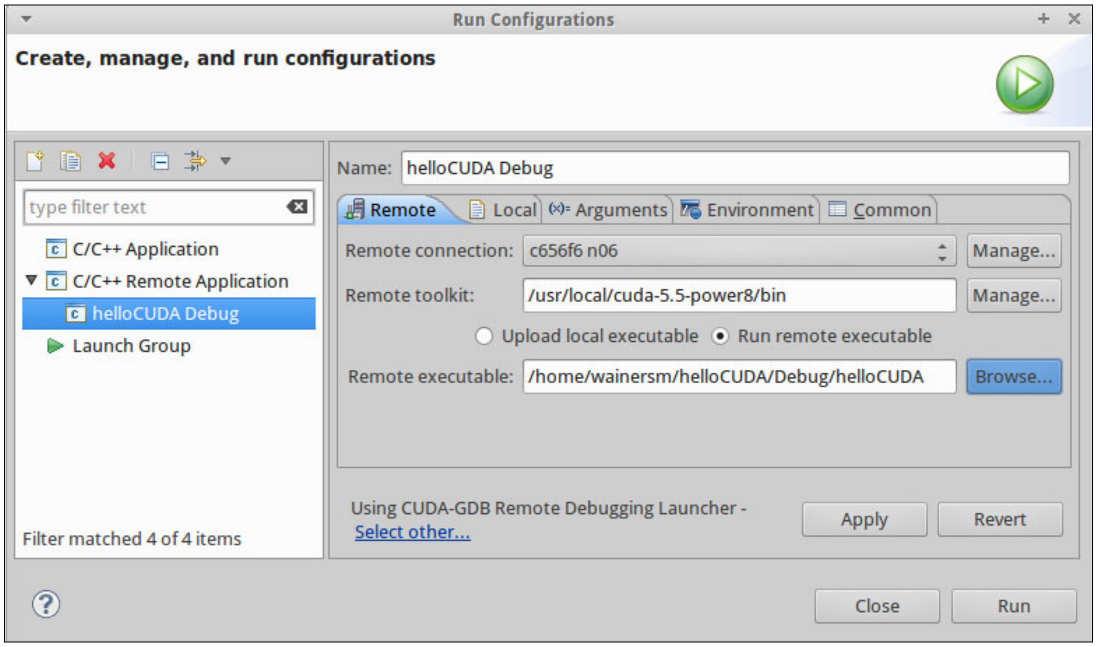
To change the CUDA runtime library linkage, follow these steps:

1. From the build Settings window, switch to the **Tool Settings** tab.
2. On the left panel, expand the **NVCC Linker** and click **Libraries**.
3. Set the CUDA Runtime Library drop-box list to either None, Shared, or Static. The default value is Static.

Launching applications

By using the project configuration that was previously set, you can run the CUDA application in the remote GPU-enabled machine. To run the application, follow these steps:

1. Click **Run** → **Run Configurations** on the main toolbar.
2. Double-click **C/C++ Remote Application**. The Run Configurations window opens, as shown in Figure 8-11.



Name: helloCUDA Debug

Remote connection: c656f6 n06

Remote toolkit: /usr/local/cuda-5.5-power8/bin

Upload local executable [✓ Run remote executable]

Remote executable: /home/wainersm/helloCUDA/Debug/helloCUDA

Using CUDA-GDB Remote Debugging Launcher - Select other...

Figure 8-11 Run Configurations window

3. On the **Remote** tab, change the Remote connection drop-down menu from Local to the label of the connection.
4. Click **Run remote executable**.
5. Click **Browse** to select the path to the executable at the remote machine. Use the Arguments and Environment tabs to set any application-specific properties.
6. Click **Run**.

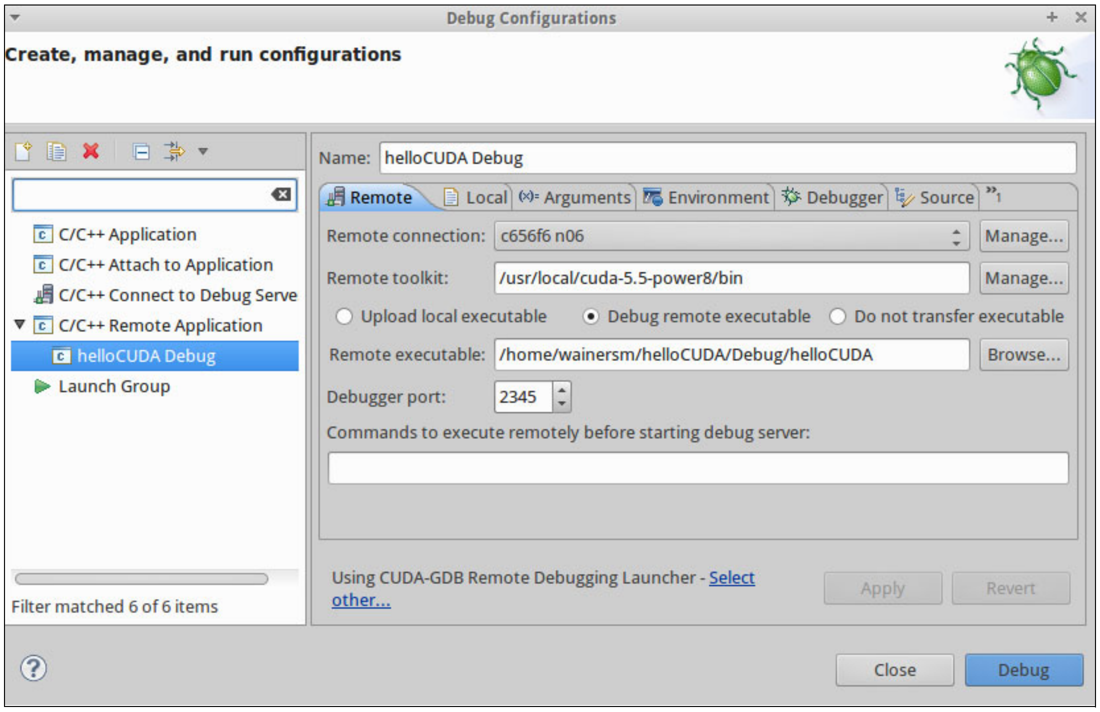
The executable output is shown in the Eclipse console on the bottom panel.

Debug applications

You can also identify problems in the CUDA executables from within the Eclipse environment. Nsight provides a graphical debugger that starts the `cuda-gdbserver` at the remote machine and then connects. You can start a debug session. Nsight can also connect to an already running `cuda-gdbserver`, but this topic is not covered in this publication.

To begin a debug session, follow these steps:

1. Click **Run** → **Debug Configurations** in the main toolbar.
2. Double-click **C/C++ Remote Application**. The Create, manage, and run configurations window opens (Figure 8-12 on page 235). Use the **Remote** tab to specify connection details.


Debug Configurations

Create, manage, and run configurations

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Connect to Debug Serve
- C/C++ Remote Application
  - helloCUDA Debug
- Launch Group

Filter matched 6 of 6 items

Name: helloCUDA Debug

Remote | Local | Arguments | Environment | Debugger | Source

Remote connection: c656f6 n06

Remote toolkit: /usr/local/cuda-5.5-power8/bin

( ) Upload local executable (●) Debug remote executable ( ) Do not transfer executable

Remote executable: /home/wainersm/helloCUDA/Debug/helloCUDA

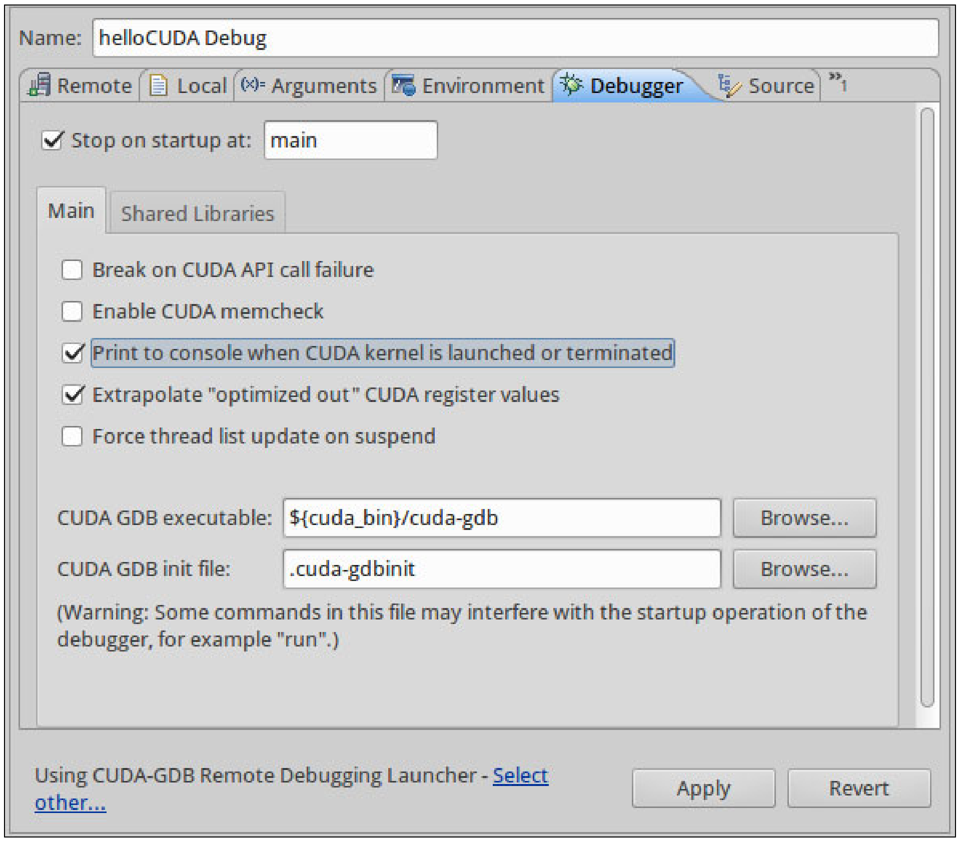
Debugger port: 2345

Commands to execute remotely before starting debug server:

Using CUDA-GDB Remote Debugging Launcher - Select other...

Figure 8-12 Remote tab of the Debug Configurations window

3. Switch to the **Debugger** tab and complete more configuration details for the session. By default, the debugger halts at the main method but also at any breakpoint that was set in the host and kernel code. Figure 8-13 on page 236 is an example of a default debugger configuration.


Name: helloCUDA Debug

☑ Stop on startup at: main

☐ Break on CUDA API call failure
☐ Enable CUDA memcheck
☑ Print to console when CUDA kernel is launched or terminated
☑ Extrapolate "optimized out" CUDA register values
☐ Force thread list update on suspend

CUDA GDB executable: ${cuda_bin}/cuda-gdb

CUDA GDB init file: .cuda-gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

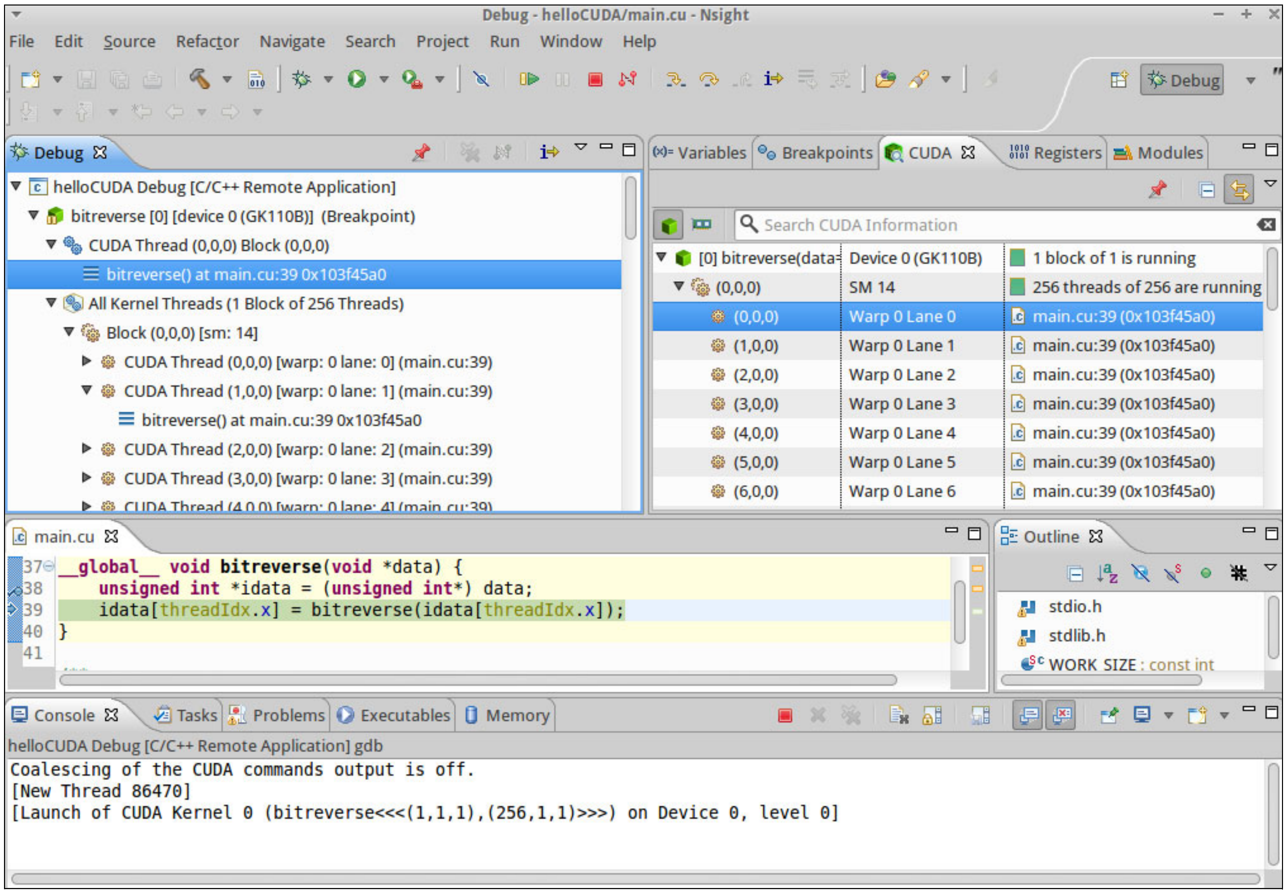
Using CUDA-GDB Remote Debugging Launcher - Select other...

Figure 8-13 Debugger tab on the Debug Configuration window

4. Use the Arguments and Environment tabs to set any application-specific properties. Click **Debug** to begin the session.

After a connection to the remote cuda-gdbserver is established, the Eclipse debugger opens. The Eclipse debugger provides information about the host source code and kernel source code so that you can debug them. Figure 8-14 on page 237 shows the following important debug information:

- Number of threads
- Threads for each block and grid
- Threads for each warp and lanes within the GPU device
- Breakpoints with source code
- GDB console output


Debug - helloCUDA/main.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Debug
Variables Breakpoints CUDA Registers Modules
Search CUDA Information
main.cu
Outline
Console Tasks Problems Executables Memory

```
helloCUDA Debug [C/C++ Remote Application]
  bitreverse [0] [device 0 (GK110B)] (Breakpoint)
    CUDA Thread (0,0,0) Block (0,0,0)
      bitreverse() at main.cu:39 0x103f45a0
    All Kernel Threads (1 Block of 256 Threads)
      Block (0,0,0) [sm: 14]
        CUDA Thread (0,0,0) [warp: 0 lane: 0] (main.cu:39)
        CUDA Thread (1,0,0) [warp: 0 lane: 1] (main.cu:39)
          bitreverse() at main.cu:39 0x103f45a0
        CUDA Thread (2,0,0) [warp: 0 lane: 2] (main.cu:39)
        CUDA Thread (3,0,0) [warp: 0 lane: 3] (main.cu:39)
        CUDA Thread (4,0,0) [warp: 0 lane: 4] (main.cu:39)
```

[0] bitreverse(data=	Device 0 (GK110B)	1 block of 1 is running
(0,0,0)	SM 14	256 threads of 256 are running
(0,0,0)	Warp 0 Lane 0	main.cu:39 (0x103f45a0)
(1,0,0)	Warp 0 Lane 1	main.cu:39 (0x103f45a0)
(2,0,0)	Warp 0 Lane 2	main.cu:39 (0x103f45a0)
(3,0,0)	Warp 0 Lane 3	main.cu:39 (0x103f45a0)
(4,0,0)	Warp 0 Lane 4	main.cu:39 (0x103f45a0)
(5,0,0)	Warp 0 Lane 5	main.cu:39 (0x103f45a0)
(6,0,0)	Warp 0 Lane 6	main.cu:39 (0x103f45a0)

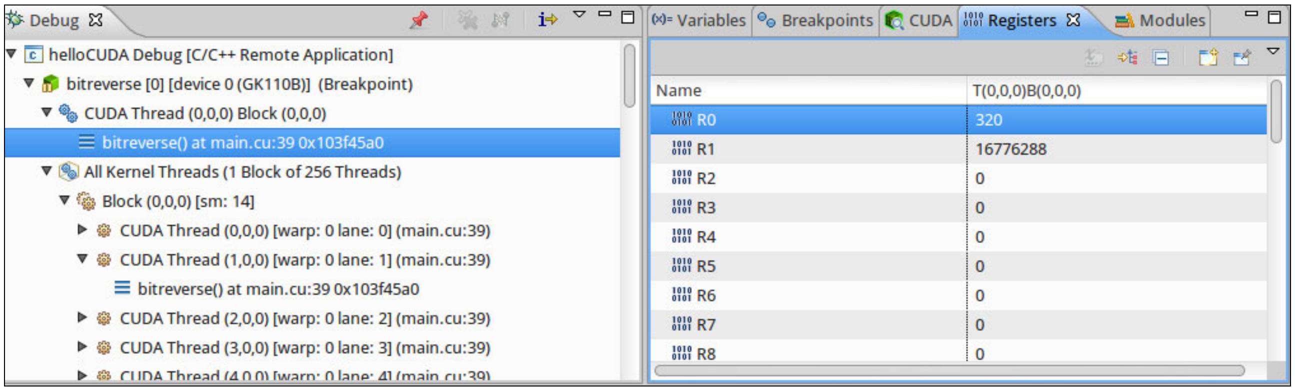
```
__global__ void bitreverse(void *data) {
    unsigned int *idata = (unsigned int*) data;
    idata[threadIdx.x] = bitreverse(idata[threadIdx.x]);
}
```

```
stdio.h
stdlib.h
WORK_SIZE : const int
```

```
helloCUDA Debug [C/C++ Remote Application] gdb
Coalescing of the CUDA commands output is off.
[New Thread 86470]
[Launch of CUDA Kernel 0 (bitreverse<<<(1,1,1),(256,1,1)>>>) on Device 0, level 0]
```

Figure 8-14 Debug window main view

You can see the values of the GPU registers by switching to the **Registers** tab, as shown in Figure 8-15.

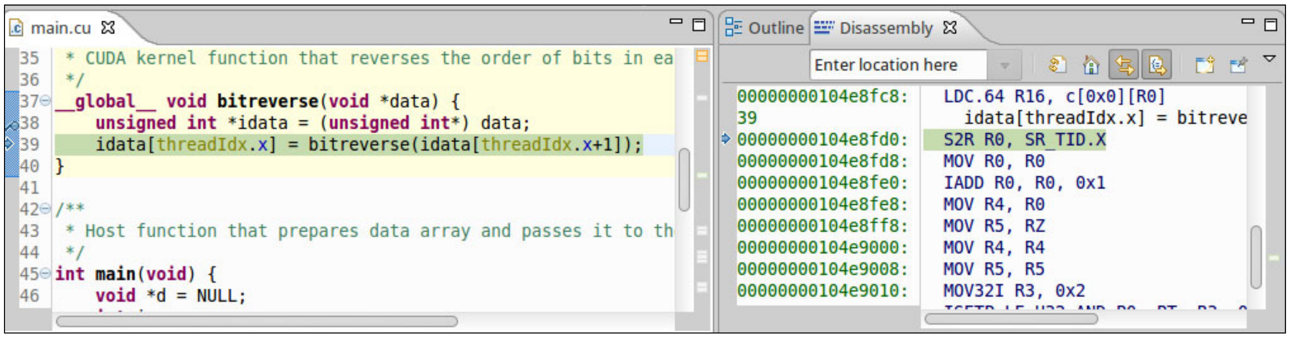


```
helloCUDA Debug [C/C++ Remote Application]
  bitreverse [0] [device 0 (GK110B)] (Breakpoint)
    CUDA Thread (0,0,0) Block (0,0,0)
      bitreverse() at main.cu:39 0x103f45a0
    All Kernel Threads (1 Block of 256 Threads)
      Block (0,0,0) [sm: 14]
        CUDA Thread (0,0,0) [warp: 0 lane: 0] (main.cu:39)
        CUDA Thread (1,0,0) [warp: 0 lane: 1] (main.cu:39)
          bitreverse() at main.cu:39 0x103f45a0
        CUDA Thread (2,0,0) [warp: 0 lane: 2] (main.cu:39)
        CUDA Thread (3,0,0) [warp: 0 lane: 3] (main.cu:39)
        CUDA Thread (4,0,0) [warp: 0 lane: 4] (main.cu:39)
```

Name	T(0,0,0)B(0,0,0)
R0	320
R1	16776288
R2	0
R3	0
R4	0
R5	0
R6	0
R7	0
R8	0

Figure 8-15 GPU Registers tab on the Debug main view

Click the **Disassembly** tab to see the view for disassembling code, as shown in Figure 8-16 on page 238.



```
 * CUDA kernel function that reverses the order of bits in ea
 */
__global__ void bitreverse(void *data) {
    unsigned int *idata = (unsigned int*) data;
    idata[threadIdx.x] = bitreverse(idata[threadIdx.x+1]);
}

/**
 * Host function that prepares data array and passes it to th
 */
int main(void) {
    void *d = NULL;
```

```
00000000104e8fc8:   LDC.64 R16, c[0x0][R0]
39                    idata[threadIdx.x] = bitreve
00000000104e8fd0:   S2R R0, SR_TID.X
00000000104e8fd8:   MOV R0, R0
00000000104e8fe0:   IADD R0, R0, 0x1
00000000104e8fe8:   MOV R4, R0
00000000104e8ff8:   MOV R5, RZ
00000000104e9000:   MOV R4, R4
00000000104e9008:   MOV R5, R5
00000000104e9010:   MOV32I R3, 0x2
```

Figure 8-16 Code Disassembly tab on the Debug main view

8.4.7 NVIDIA CUDA Toolkit code samples

The NVIDIA CUDA toolkit includes a set of code samples that cover many of the CUDA features (8.4.1, "CUDA paradigm" on page 218) and GPU accelerated libraries (8.4.4, "GPU accelerated libraries" on page 222).

To start with the examples, copy them to a directory to which you have write access, for example:

```
$ cp -r <cuda_installation_dir>/samples/ $HOME/cuda-samples
```

To build an individual example, run the **make** command in its directory. To build all of the examples at one time, execute **make** in the top-level directory that contains the code samples.

By default, the samples are compiled with the **g++** compiler. To enforce the use of the IBM XL C/C++ compiler, explicitly set the value for the ***GCC*** variable:

```
$ make GCC=xlC
```

You can clean the object directories and binary files by running the following command:

```
$ make clean
```

8.5 Tuning and debugging

A CUDA program interleaves portions of execution code that run sometimes on a CPU (host) and at other times on a GPU (device) while being concomitant (associated or related) to each other.

Running on different computational hardware (the host versus a device) means that CUDA execution environments must be understood before any optimization efforts occur because the environments provide distinct purposes and computational resources (threading model, memory capabilities, and so on) with a notable influence on performance. For an overview of GPU hardware, see 8.1.1, "NVIDIA Tesla K40 GPU" on page 192. Extensive documentation is available for application optimization for POWER8. The essential information is in *Performance Optimization and Tuning Techniques for IBM Processors, including IBM POWER8*, SG24-8171.

Heterogeneous parallel programming paradigms, such as CUDA, offer many possible scenarios for the application. The characterization of the application is important for determining the techniques and strategies to use.

For example, you can have applications with the following characteristics:

- Runs completely on the CPU side. No GPU operation is in use.
- Runs GPU portions by using CUDA-accelerated third-party libraries only. No internal algorithms that use CUDA kernels are executed.
- Runs heavily on the CPU but certain computations are offloaded to the GPU to speed them up.
- Runs almost entirely on the GPU. The host CPU is used for data feeding and solution consolidation only.

Therefore, understanding the application behavior in run time is often the first step toward performance optimization. Here, profiling and tracing tools play fundamental roles because they are handy for application characterization. Profiling and tracing tools are used to show hotspots and bottlenecks and to discover inefficiencies in the use of computational resources. This section introduces the main profiling tools for CUDA C/C++.

The broad use of performance-driven programming techniques and strategies can be used to take advantage of GPU as much as possible. We suggest that you read the *CUDA C Best Practices Guide*, which is available at the following site:

http://docs.nvidia.com/cuda/cuda-c-best-practices-guide

8.5.1 CUDA profiling tools

The CUDA 5.5 toolkit for POWER8 includes the **nvprof** command-line profiling tool. The **nvprof** tool can be used to identify performance hotspots and analyze behavior in areas of potential performance improvement.

By default, the **nvprof** tool shows the total time that is spent in each application kernel, the number of calls, and the average time for each call.

Example 8-34 shows the output of the **nvprof** tool for the *quasirandomGenerator* application. (See 8.4.7, "NVIDIA CUDA Toolkit code samples" on page 238.) The application spends about 35% of the time copying data from the device to the host and 64% of the time in kernel execution (*quasirandomGeneratorKernel* is responsible for about 48% of this number).

Example 8-34 The nvprof tool that is used to identify hotspots in a CUDA application

```
$ nvprof ./4_Finance/quasirandomGenerator/quasirandomGenerator
./4_Finance/quasirandomGenerator/quasirandomGenerator Starting...

==8071== NVPROF is profiling process 8071, command:
./4_Finance/quasirandomGenerator/quasirandomGenerator
GPU Device 0: "Tesla K40m" with compute capability 3.5

Allocating GPU memory...
Allocating CPU memory...

<... output omitted ...>

Shutting down...
==8071== Profiling application: ./4_Finance/quasirandomGenerator/quasirandomGenerator
==8071== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 47.89%  9.1456ms        21  435.51us  431.30us  438.27us  quasirandomGeneratorKernel(float*,
unsigned int, unsigned int)
```

```
 34.89%  6.6623ms         2  3.3312ms  3.3154ms  3.3469ms  [CUDA memcpy DtoH]
 16.55%  3.1612ms        21  150.53us  147.74us  151.97us  inverseCNDKernel(float*, unsigned
int*, unsigned int)
  0.66%  126.59us         2  63.296us  62.816us  63.776us  [CUDA memset]
  0.01%  1.6320us         1  1.6320us  1.6320us  1.6320us  [CUDA memcpy HtoD]
```

The tool generates predefined metrics about GPU hardware usage, including occupancy, utility, and throughput. These metrics are generated from hardware event data that is collected while the CUDA application runs. Use the `--query-events` **`nvprof`** option to list all events that are available for profiling. Use the `--query-metrics` **`nvprof`** option to list all metrics that are available for profiling.

A good starting point is to check the overall GPU's capacity usage by the application. For example, the achieved_occupancy metric can be used. See Table 8-3.

All utilization metrics can be used together to find a hotspot in the hardware subsystems that the application mostly uses. Use the metrics in Table 8-3 to understand the application's behavior from a GPU internals perspective.

Table 8-3 The nvprof tool utilization metrics

Metric	**Description**
achieved_occupancy	Ratio of the average active warps for each active cycle to the maximum number of warps that is supported on a multiprocessor.
l1_shared_utilization	The utilization level of the L1/shared memory relative to peak utilization.
l2_utilization	The utilization level of the L2 cache relative to the peak utilization.
tex_utilization	The utilization level of the texture cache relative to the peak utilization.
dram_utilization	The utilization level of the device memory relative to the peak utilization.
sysmem_utilization	The utilization level of the system memory relative to the peak utilization.
ldst_fu_utilization	The utilization level of the multiprocessor function units that execute load and store instructions.
alu_fu_utilization	The utilization level of the multiprocessor function units that execute integer and floating-point arithmetic instructions.
cf_fu_utilization	The utilization level of the multiprocessor function units that execute control-flow instructions.
tex_fu_utilization	The utilization level of the multiprocessor function units that execute texture instructions.
issue_slot_utilization	Percentage of issue slots that issued at least one instruction, averaged across all cycles.

Example 8-35 on page 241 contains the output (highlights in bold) of the **`nvprof`** tool when it is profiled with the *quasirandomGenerator* sample application. The output shows the GPU utilization that uses all of the metrics in Table 8-3. Example 8-35 on page 241 shows that the *inverseCNDKernel* had a medium utilization of the device memory and the arithmetic function units. The *quasirandomGeneratorKernel* kernel used a maximum amount of the arithmetic function unit, but it kept low or no (idle) utilization of the other units.

Example 8-35 The nvprof tool: quasirandomGenerator application output of utilization metrics (adjusted to fit the page)

```
$ nvprof --metrics dram_utilization,l1_shared_utilization,l2_utilization,\
tex_utilization,sysmem_utilization,ldst_fu_utilization,alu_fu_utilization,\
cf_fu_utilization,tex_fu_utilization,issue_slot_utilization \
./4_Finance/quasirandomGenerator/quasirandomGenerator
./4_Finance/quasirandomGenerator/quasirandomGenerator Starting...

==7171== NVPROF is profiling process 7171, command:
./4_Finance/quasirandomGenerator/quasirandomGenerator
GPU Device 0: "Tesla K40m" with compute capability 3.5

Allocating GPU memory...
Allocating CPU memory...

<... output omitted ...>

Shutting down...
==7171== Profiling application: ./4_Finance/quasirandomGenerator/quasirandomGenerator
==7171== Profiling result:
==7171== Metric result:
Invocations           Metric Name             Metric Description       Min       Max       Avg
Device "Tesla K40m (0)"
        Kernel: inverseCNDKernel(float*, unsigned int*, unsigned int)
         21 issue_slot_utilization         Issue Slot Utilization    60.98%    61.94%    61.55%
         21  l1_shared_utilization    L1/Shared Memory Utilization   Low (1)   Low (1)   Low (1)
         21         l2_utilization              L2 Cache Utilization   Low (2)   Low (2)   Low (2)
         21        tex_utilization       Texture Cache Utilization  Idle (0)  Idle (0)  Idle (0)
         21       dram_utilization       Device Memory Utilization   Mid (4)   Mid (4)   Mid (4)
         21     sysmem_utilization       System Memory Utilization  Idle (0)   Low (1)  Idle (0)
         21    ldst_fu_utilization  Load/Store Function Unit Utili   Low (1)   Low (1)   Low (1)
         21     alu_fu_utilization  Arithmetic Function Unit Utili   Mid (4)   Mid (4)   Mid (4)
         21      cf_fu_utilization  Control-Flow Function Unit Uti   Low (2)   Low (2)   Low (2)
         21     tex_fu_utilization  Texture Function Unit Utilizat  Idle (0)  Idle (0)  Idle (0)
        Kernel: quasirandomGeneratorKernel(float*, unsigned int, unsigned int)
         21 issue_slot_utilization         Issue Slot Utilization    51.74%    51.97%    51.91%
         21  l1_shared_utilization    L1/Shared Memory Utilization   Low (1)   Low (1)   Low (1)
         21         l2_utilization              L2 Cache Utilization   Low (1)   Low (1)   Low (1)
         21        tex_utilization       Texture Cache Utilization  Idle (0)  Idle (0)  Idle (0)
         21       dram_utilization       Device Memory Utilization   Low (2)   Low (2)   Low (2)
         21     sysmem_utilization       System Memory Utilization  Idle (0)   Low (1)  Idle (0)
         21    ldst_fu_utilization  Load/Store Function Unit Utili   Low (1)   Low (1)   Low (1)
         21     alu_fu_utilization  Arithmetic Function Unit Utili  Max (10)  Max (10)  Max (10)
         21      cf_fu_utilization  Control-Flow Function Unit Uti   Low (1)   Low (1)   Low (1)
         21     tex_fu_utilization  Texture Function Unit Utilizat  Idle (0)  Idle (0)  Idle (0)
```

The results in Example 8-35 show that it can be beneficial to lower the use of the function unit by the *quasirandomGeneratorKernel* kernel because it was the most accessed unit. Therefore, Example 8-36 on page 242 shows the output of the **nvprof** tool when it profiles the application to generate metrics of a single and a double-precision floating point operation. Example 8-36 on page 242 shows in bold that the kernel ran only a single-precision floating point multiply operation.

Example 8-36 The nvprof tool: quasirandomGenerator application output of arithmetic unit metrics (adjusted to fit page)

```
$ nvprof --kernels ::quasirandomGeneratorKernel:1 --metrics \
flops_sp_special,flops_dp_fma,flops_dp_mul,flops_dp_add,flops_sp_fma,flops_sp_mul,flops_sp_add \
./4_Finance/quasirandomGenerator/quasirandomGenerator
./4_Finance/quasirandomGenerator/quasirandomGenerator Starting...

==8079== NVPROF is profiling process 8079, command:
./4_Finance/quasirandomGenerator/quasirandomGenerator
GPU Device 0: "Tesla K40m" with compute capability 3.5

Allocating GPU memory...
Allocating CPU memory...

<... output omitted ...>

Shutting down...
==8079== Profiling application: ./4_Finance/quasirandomGenerator/quasirandomGenerator
==8079== Profiling result:
==8079== Metric result:
Invocations           Metric Name             Metric Description       Min       Max       Avg
Device "Tesla K40m (0)"
        Kernel: quasirandomGeneratorKernel(float*, unsigned int, unsigned int)
          1          flops_sp_add              FLOPS(Single Add)         0         0         0
          1          flops_sp_mul              FLOPS(Single Mul)   3145728   3145728   3145728
          1          flops_sp_fma              FLOPS(Single FMA)         0         0         0
          1          flops_dp_add              FLOPS(Double Add)         0         0         0
          1          flops_dp_mul              FLOPS(Double Mul)         0         0         0
          1          flops_dp_fma              FLOPS(Double FMA)         0         0         0
          1      flops_sp_special          FLOPS(Single Special)         0         0         0
```

Hardware events can be used to expand performance analysis, although the **nvprof**-provided metrics are useful for identifying the area of potential improvements.

8.5.2 CUDA debug tools

The CUDA 5.5 toolkit for POWER8 provides a set of tools for debugging that can help locate issues in parallel applications:

- CUDA binary utilities
- CUDA Memcheck
- CUDA GDB

CUDA binary utilities

The CUDA binary utilities include the **cuobjdump** and **nvdisasm** tools, which are designed to inspect binary objects (**cuobjdump**) and disassemble them (**nvdisasm**).

Use the **cuobjdump** tool to extract and present, in human-readable format, information about both host and cubin object files. The **cuobjdump** tool has many options for gathering information about only certain segments of the object file, for instance, any Executable and Linking Format (ELF) header. Example 8-37 on page 243 shows the standard output of the **cuobjdump** tool for an executable file.

Example 8-37 The cuobjdump tool's standard output

```
$ cuobjdump helloCUDA/Debug/helloCUDA

Fatbin elf code:
================
arch = sm_20
code version = [1,7]
producer = <unknown>
host = linux
compile_size = 64bit
identifier = ./main.o

Fatbin elf code:
================
arch = sm_30
code version = [1,7]
producer = cuda
host = linux
compile_size = 64bit
has debug info
compressed
identifier = ../main.cu

Fatbin ptx code:
================
arch = sm_30
code version = [3,2]
producer = cuda
host = linux
compile_size = 64bit
has debug info
compressed
identifier = ../main.cu
ptxasOptions =  -g --dont-merge-basicblocks --return-at-end
```

Use the **`nvdisasm`** tool for disassembling cubin files and showing control flow. Notice that the tool does not allow the host code to disassemble the files.

CUDA memcheck

The **`cuda-memcheck`** is a suite of tools that are designed to identify memory-related issues in CUDA applications. You can execute the tool in different modes by using the tool option, and by passing in either of the following values:

- `memcheck`:

  Analyzes the correctness of memory accesses, including checking allocations on device heap and detecting memory leaks

- `racecheck`:

  Checks the race conditions on accesses to shared memory

The **`memcheck`** output in Example 8-38 on page 244 shows an error that was caused by a thread (ID=`255` in block 0) that tried to access 4 bytes out of the shared memory bounds. It also detected a problem with the CUDA API call to the *cudaThreadSynchronize* function.

Example 8-38 The cuda-memcheck tool: Memory access checking

```
cuda-memcheck helloCUDA/Debug/helloCUDA
========= CUDA-MEMCHECK
========= Invalid __global__ read of size 4
=========     at 0x000000e8 in /home/wainersm/helloCUDA/Debug/../main.cu:39:bitreverse(void*)
=========     by thread (255,0,0) in block (0,0,0)
=========     Address 0x43046c0400 is out of bounds
=========     Saved host backtrace up to driver entry point at kernel launch time
=========     Host Frame:/usr/lib/powerpc64le-linux-gnu/libcuda.so (cuLaunchKernel + 0x24c)
[0x152cfc]
=========     Host Frame:/usr/local/cuda-5.5-power8/lib64/libcudart.so.5.5 [0xa7f8]
=========     Host Frame:/usr/local/cuda-5.5-power8/lib64/libcudart.so.5.5 (cudaLaunch + 0x184)
[0x39c04]
=========     Host Frame:helloCUDA/Debug/helloCUDA [0x13cc]
=========     Host Frame:helloCUDA/Debug/helloCUDA [0xba0]
=========     Host Frame:/lib/powerpc64le-linux-gnu/libc.so.6 [0x24e80]
=========     Host Frame:/lib/powerpc64le-linux-gnu/libc.so.6 (__libc_start_main + 0xc4)
[0x25074]
=========
========= Program hit error 30 on CUDA API call to cudaThreadSynchronize
Error unknown error at line 59 in file ../main.cu
=========     Saved host backtrace up to driver entry point at error
=========     Host Frame:/usr/lib/powerpc64le-linux-gnu/libcuda.so [0x36c39c]
=========     Host Frame:/usr/local/cuda-5.5-power8/lib64/libcudart.so.5.5
(cudaThreadSynchronize + 0x19c) [0x3427c]
=========     Host Frame:helloCUDA/Debug/helloCUDA [0xba4]
=========     Host Frame:/lib/powerpc64le-linux-gnu/libc.so.6 [0x24e80]
=========     Host Frame:/lib/powerpc64le-linux-gnu/libc.so.6 (__libc_start_main + 0xc4)
[0x25074]
=========
========= ERROR SUMMARY: 2 errors
```

CUDA GDB

The **cuda-gdb** command-line debugger, which is provided by the CUDA toolkit 5.5, is based on GNU GDB 7.2 implementation. Use the **cuda-gdb** command-line debugger to debug both host and device portions of code.

The **cuda-gdb** debugger includes the same **gdb** command set, but it also implements the specific CUDA commands, **info cuda**, **cuda**, and **set cuda**, which are operations that show information about activities (**info cuda**), set focus on context (**cuda**), and set debugger configuration (**set cuda**).

Example 8-39 shows the use of ordinary **gdb** and CUDA-specific commands in the same debug session. One breakpoint is set in the *bitreverse* kernel and halts execution. Context information for devices, threads, and kernels is accessed with the **info cuda** *<option>* commands. The thread focus is switched with the **cuda thread** command. The **info frame** command, for example, is triggered to demonstrate conventional **gdb** commands.

Example 8-39 The cuda-gdb debugger: A simple walk-through session

```
$ cuda-gdb ./helloCUDA/Debug/helloCUDA
NVIDIA (R) CUDA Debugger
6.0 release
<... output omitted ...>
Reading symbols from /home/wainersm/helloCUDA/Debug/helloCUDA...done.
```

```
(cuda-gdb) start
Temporary breakpoint 1 at 0x10000acc: file ../main.cu, line 46.
Starting program: /home/wainersm/helloCUDA/Debug/helloCUDA
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/powerpc64le-linux-gnu/libthread_db.so.1".

Temporary breakpoint 1, main () at ../main.cu:46
46    void *d = NULL;
(cuda-gdb) info stack
#0  main () at ../main.cu:46
(cuda-gdb) next
45 int main(void) {
(cuda-gdb) next
46    void *d = NULL;
(cuda-gdb) next
51       idata[i] = (unsigned int) i;
(cuda-gdb) next
53    CUDA_CHECK_RETURN(cudaMalloc((void**) &d, sizeof(int) * WORK_SIZE));
(cuda-gdb) next
[New Thread 0x3fffb6b7f180 (LWP 6018)]
54    CUDA_CHECK_RETURN(
(cuda-gdb) next
57    bitreverse<<<1, WORK_SIZE, WORK_SIZE * sizeof(int)>>>(d);
(cuda-gdb) info frame
Stack level 0, frame at 0x3fffffff100:
 pc = 0x10000b8c in main (../main.cu:57); saved pc 0x3fffb7d74e80
 source language c++.
 Arglist at 0x3fffffffe820, args:
 Locals at 0x3fffffffe820, Previous frame's sp is 0x3fffffff100
 Saved registers:
  r22 at 0x3fffffff0b0, r23 at 0x3fffffff0b8, r24 at 0x3fffffff0c0, r25 at
0x3fffffff0c8, r26 at 0x3fffffff0d0,
  r27 at 0x3fffffff0d8, r28 at 0x3fffffff0e0, r29 at 0x3fffffff0e8, r30 at
0x3fffffff0f0, r31 at 0x3fffffff0f8,
  pc at 0x3fffffff110, lr at 0x3fffffff110
(cuda-gdb) info cuda devices
  Dev Description SM Type SMs Warps/SM Lanes/Warp Max Regs/Lane Active SMs Mask
    0      GK110B   sm_35  15       64         32           256 0x00000000
    1      GK110B   sm_35  15       64         32           256 0x00000000
(cuda-gdb) info cuda kernels
No CUDA kernels.
(cuda-gdb) info breakpoints
No breakpoints or watchpoints.
(cuda-gdb) break main.cu:38
Breakpoint 5 at 0x103f42a0: file ../main.cu, line 38.
(cuda-gdb) list 38
33
34 /**
35  * CUDA kernel function that reverses the order of bits in each element of the
array.
36  */
37 __global__ void bitreverse(void *data) {
38    unsigned int *idata = (unsigned int*) data;
39    idata[0] = bitreverse(idata[threadIdx.x]);
40 }
41
42 /**
(cuda-gdb) info breakpoints
Num     Type           Disp Enb Address            What
5       breakpoint     keep y   0x00000000103f42a0 in bitreverse(void*) at
../main.cu:38
(cuda-gdb) continue
Continuing.
[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device
0, sm 14, warp 0, lane 0]

Breakpoint 5, bitreverse<<<(1,1,1),(256,1,1)>>> (data=0x43046c0000) at
../main.cu:39
39    idata[0] = bitreverse(idata[threadIdx.x]);
(cuda-gdb) next
40 }
(cuda-gdb) info cuda threads
  BlockIdx ThreadIdx To BlockIdx ThreadIdx Count         Virtual PC   Filename
Line
Kernel 0
*  (0,0,0)   (0,0,0)     (0,0,0)  (31,0,0)    32 0x00000000103f43a0 ../main.cu
40
   (0,0,0)  (32,0,0)     (0,0,0) (255,0,0)   224 0x00000000103f42a0 ../main.cu
39
(cuda-gdb) cuda thread 10
[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (10,0,0), device
0, sm 14, warp 0, lane 10]
40 }
(cuda-gdb) cuda thread
thread (10,0,0)
(cuda-gdb) info cuda devices
  Dev Description SM Type SMs Warps/SM Lanes/Warp Max Regs/Lane Active SMs Mask
*   0      GK110B   sm_35  15       64         32           256 0x00004000
    1      GK110B   sm_35  15       64         32           256 0x00000000
(cuda-gdb) cuda sm
sm 14
(cuda-gdb) continue
Continuing.
Input value: 0, device output: 254, host output: 0
Input value: 1, device output: 1, host output: 128
Input value: 2, device output: 2, host output: 64
Input value: 3, device output: 3, host output: 192
<... output omitted ...>
[Thread 0x3fffb6b7f180 (LWP 6018) exited]
[Inferior 1 (process 5985) exited normally]
(cuda-gdb)
The program is not being run.
```

Remote debugging is possible by using the **cuda-gdbserver** tool, which is also delivered with the CUDA toolkit 5.5. The **cuda-gdbserver** tool behaves just like the **gdb-server** tool. The **cuda-gdbserver** tool allows sessions to be handled over a Secure Shell (SSH) or File Transfer Protocol (FTP) connection.

Another alternative to debugging CUDA applications is to use the Nsight debugger, which is integrated with the **`cuda-gdb`** graphically. The Nsight debugger is covered in 8.4.6, “Nsight” on page 227.

The reference documentation of the **`cuda-gdb`** debugger is available at the following site:

`http://docs.nvidia.com/cuda/cuda-gdb`

A

Problem determination

This appendix provides problem determination guidance.

The following topics are described:

- IBM XL SMP Runtime error 1587-120
- Problems with the Intelligent Platform Management Interface (IPMI) Tool
- Incorrect IPMI password
- Network traffic impact
- IPMI server hangs

IBM XL SMP Runtime error 1587-120

The runtime error 1587-120 indicates a problem in the IBM Xpertise Library (XL) symmetric multiprocessor (SMP) runtime environment. Typically, the error occurs when a user incorrectly specifies the number of OpenMP threads and their affinity. For example, a user can specify a stride that does not conform to the current simultaneous multithreading (SMT) mode.

At the time that this publication was written, this error occurred when a system administrator switched from a lower SMT mode to a higher SMT mode on Ubuntu Server installations with the *cgmanager* service enabled. For example, the system administrator can change the SMT mode from SMT4 to SMT8 by using the **`ppc64_cpu`** command. If a user starts a program with two or more OpenMP threads, the use of either of the following statements before a program is run results in the 1587-120 error:

```
export XLSMPOPTS=PROCS=0,4
export XLSMPOPTS=STARTPROC=0:STRIDE=4
```

The error occurs because the *cpuset* control groups (*cgroups*) are not automatically updated with the CPU hotplug operations. Applications are forced into `cpuset cgroups` by the `cgmanager` service, which is used for the Linux Containers (LXC) functionality (therefore not required in high-performance computing (HPC) environments). This problem occurs in the Ubuntu Server 14.04 Long Term Support (LTS) series and Ubuntu Server 14.10. Later versions of the Ubuntu Server distribution will include a new kernel feature, which addresses the root cause with the `cpuset cgroups`, or fixes to the cgmanager service.

To return the system back to a normal operation, the system administrator can disable the cgmanager service with the following commands:

```
$ sudo service cgmanager stop
$ sudo update-rc.d cgmanager disable
```

For more information, see the following bug report in the Ubuntu Launchpad at the following website:

http://bugs.launchpad.net/bugs/1392176

Problems with the Intelligent Platform Management Interface (IPMI) Tool

This section describes problems that you might encounter when you use the **`ipmitool`** and suggests steps to identify and address them.

Several error messages that are produced by the **`ipmitool`** do not contain enough detail for you to identify a particular problem or distinguish between separate problems. In this case, increasing verbosity (one or more `-v` options) provides more details about the operations that were performed.

Many problems with the **`ipmitool`** are caused by certain network traffic conditions. The IPMI `lanplus`[1] interface uses the Remote Management Control Protocol+ (RMCP+), which is based on User Datagram Protocol (UDP), and therefore more sensitive to network congestion and packet loss. It is usually beneficial to execute the **`ipmitool`** in the same local network (LAN) as the system's flexible service processor (FSP) or within as few network hops as possible.

[1] Short for IPMI v2.0 RMCP+ LAN interface.

Incorrect IPMI password

To determine whether the IPMI password is incorrect, increase the verbosity (one level **-v**) and check for message "`RAKP 2 HMAC is invalid`" (Example A-1).

Example A-1 ipmitool: incorrect IPMI password

```
$ ipmitool -I lanplus -H fsp-address -P incorrect-ipmi-password command
Error: Unable to establish IPMI v2 / RCMP+ session

$ ipmitool -I lanplus -H fsp-address -P incorrect-ipmi-password -v command
> RAKP 2 HMAC is invalid
Error: Unable to establish IPMI v2 / RCMP+ session
```

Network traffic impact

The "`Insufficient resources for session`" message (Example A-2) when you initiate an **ipmitool** operation can indicate that the **ipmitool** operation was affected by network congestion or packet loss.

Example A-2 ipmitool: Insufficient resources for session

```
$ ipmitool -I lanplus -H fsp-address -P ipmi-password command
Error in open session response message : insufficient resources for session

Error: Unable to establish IPMI v2 / RCMP+ session
```

Confirm whether the error was caused by network congestion/packet loss by using more verbosity (two levels or **-vv**). Check for multiple "`Sending IPMI command payload`" messages with no response. Example A-3 shows four attempts to open a session and the messages. Run the **ipmitool** with fewer network hops to the system's FSP to minimize or eliminate this problem.

Example A-3 ipmitool: Send payload retries (output shortened)

```
$ ipmitool -I lanplus -H fsp-address -P ipmi-password -vv command
>> Sending IPMI command payload
<...>

>> Sending IPMI command payload
<...>

>> Sending IPMI command payload
<...>

>> Sending IPMI command payload
<...>

>> SENDING AN OPEN SESSION REQUEST

<<OPEN SESSION RESPONSE
<...>
```

```
IPMIv2 / RMCP+ SESSION OPENED SUCCESSFULLY
<...>
```

Another effect of network traffic is instability during active console sessions (SOLs). The **ipmitool** program automatically performs several connection retries with certain retry intervals. However, the problematic network condition can last longer than the total retry period, which causes **ipmitool** to disconnect. You can increase the SOL retry interval (`100` milliseconds, by default), as shown in Example A-4 (`500` milliseconds).

Example A-4 ipmitool: Increase SOL retry interval

```
$ ipmitool -I lanplus -H fsp-address -P ipmi-password sol set retry-interval 50
```

IPMI server hangs

In rare occasions, the IPMI daemon in the system's FSP experiences a problem. This error might be reported as the "`Insufficient resources for session`" error message (Example A-2 on page 251) or the "`Get Auth Capabilities error`" message (Example A-5), depending on the stage of the operation when the problem occurs.

Example A-5 ipmitool: Get Auth Capabilities error

```
$ ipmitool -I lanplus -H fsp-address -P ipmi-password -v command
Get Auth Capabilities error
Error issuing Get Channel Authentication Capabilities request
Error: Unable to establish IPMI v2 / RCMP+ session
<...>
```

In this situation, you can reset the FSP in the Advanced System Management interface (ASMI). For instructions, see 3.2, "OPAL firmware and the ASM interface" on page 27.

B

Useful commands

This appendix provides useful commands that help the user to configure and use xCAT, Platform Load Sharing Facility (LSF), and InfiniBand.

This appendix contains the following sections:

- xCAT commands
- Platform LSF commands
- InfiniBand commands

xCAT commands

This section describes useful xCAT commands that can be used on IBM Power System S824L on Ubuntu 14.04 LTS (little endian) as shown in Table B-1.

Table B-1 xCAT commands

Command (Linux)	**Description**
nodestat	Display the running status of a node range.
rpower	Remote power control of nodes. Also displays information about the status of the nodes.
rcons	Use the command to monitor the console and watch the installation process.
makedhcp	Run the **`makedhcp -n`** command to create a new Dynamic Host Configuration Protocol (DHCP) configuration file and add the network definitions.
makedns	Set up domain name services (DNS) from the entries in `/etc/hosts`.
mkdef	Use the command to create the initial node definition for the two compute nodes.
chdef	Change xCAT data object definitions.
lsdef	Display xCAT object definitions that are stored in the xCAT database.
nodels	List the nodes, and their attributes, from the xCAT database.
rmdef	Remove xCAT data object definitions.
tabdump	Display an xCAT database table in comma-separated value (CSV) format.
tabedit	View an xCAT database table in an editor and make changes.

For more information about xCAT commands, see the following website:

`http://sourceforge.net/projects/xcat`

Platform LSF commands

This section describes useful Platform LSF commands that are based on IBM Platform LSF v9.1.3, as shown in Table B-2.

Table B-2 Platform LSF commands

Command	**Description**
bsub	Submit a job to LSF by running the specified command and its arguments.
bhist	Display historical information about jobs.

Command	Description
bjobs	Display and filter information about LSF jobs. Specify one or more job IDs (and, optionally, an array index list) to display information about specific jobs (and job arrays).
bqueues	Display information about queues.
bmod	Modify the job submission options of a job.
bkill	Send signals to kill, suspend, or resume unfinished jobs.
bstop	Suspend unfinished jobs.
bresume	Resume one or more suspended jobs.
badmin	Administer LSF.
bhosts	Display hosts and their static and dynamic resources.
busers	Display information about users and user groups.
lsload	Display load information for hosts.
lsid	Display the current LSF version number, cluster name, and master host name.
lshosts	Display hosts and their static resource information.
lsclusters	Display configuration information about LSF clusters.

For more information about Platform LSF commands, see this website:

http://www.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_kc_cmd_ref.dita

InfiniBand commands

Table B-3 contains a few useful commands to obtain information about the InfiniBand (IB) subnet and card adapter and the subnet of a high-performance computing (HPC) cluster. For a comprehensive list of commands that are available for Mellanox InfiniBand solutions on IBM Power System S824L, see the Mellanox OpenFabrics Enterprise Distribution for Linux (MLNX_OFED) website and open the *Mellanox OFED for Linux User Manual* version 2.3:

http://www.mellanox.com/page/products_dyn?product_family=26

Table B-3 Useful InfiniBand commands

Command	Description
ibstat	Display the IB driver information of the local host.
ibhosts	List all IB hosts in the topology.
ibnetdiscover	Scan the IB subnet and display the topology.
ibdiagnet	Run a complete diagnostic test in the IB subnet.
smpquery <*property*>	Query the IB subnet management properties, where <*property*> can be `nodeinfo`, `nodedesc`, `portinfo`, `switchinfo`, `pkeytable`, `sl2vltable`, `vlarbitration`, `guidinfo`, and `mlnxexportinfo`.

C

IBM Tivoli Workload Scheduler LoadLeveler to IBM Platform Load Sharing Facility migration

This appendix provides information for IBM Tivoli® Workload Scheduler LoadLeveler® (LoadLeveler) administrators and users about how to migrate their clusters to IBM Platform Load Sharing Facility (LSF).

LoadLeveler to Platform LSF migration

LoadLever and LSF are both workload managers, but only LSF supports the IBM Power System S824L on the Ubuntu Server 14.04 LTS (little-endian) operating system. This appendix provides the information about how to migrate a LoadLeveler cluster to an LSF cluster.

The migration from LoadLeveler includes two parts:

- Migrate the cluster configuration
- Migrate the users and jobs from LoadLeveler to LSF

Information about mapping LoadLeveler concepts, components, and commands to LSF is also provided to help users who are familiar with LoadLeveler to use LSF as their workload manager.

For additional information about IBM Tivoli Workload Scheduler LoadLeveler, see this website:

`http://www.ibm.com/software/tivoli/products/scheduler-loadleveler/`

For a complete description of IBM Platform LSF, see *Administering IBM Platform LSF,* SC22-5346, and *IBM Platform LSF Configuration Reference,* SC22-5350.

For more details about migration from LoadLeveler to LSF, see the *IBM LoadLeveler to IBM Platform LSF Migration Guide,* REDP-5048:

`http://www.redbooks.ibm.com/abstracts/redp5048.html?Open`

User scenario

To demonstrate LoadLeveler to LSF migration, this appendix provides an example to simulate the user scenario, including planning, mapping, and migration implementation.

Planning

In a production environment, a workload manager is widely used in areas, such as Electronic Design Automation (EDA), Industrial Manufacturing, Government/Education, Life Sciences, and Oil and Gas. The jobs for different areas have different characteristics, so the workload manager needs to be configured differently for overall performance. For example, to an integrated circuit (IC) design company, improving the use of computing resources is essential to its productivity. These jobs are usually short, serial jobs, but often many of them are submitted in a short time so they affect the entire system throughput.

However, in the government and education areas, many parallel jobs run across hundreds of computing nodes with many threads on each node and the entire job lasts for days, so special requirements exist for the scalability of the workload manager.

Meanwhile, in certain production environments, the jobs and data are automatically generated by other software, such as a pipeline, so the interface is defined for the scripts, such as job submission and query, to keep the environment working smoothly. It is important to map the commands and job definitions to ensure that the system works as expected.

Typically, three phases are required to plan the migration. The first phase, assessment, assesses the migration background to ensure that the current environment, such as the characteristics and requirements of the system, is understood. The second phase, timeline, is a pilot implementation phase. The third phase, production, is the production environment migration phase.

Assessment

The current LoadLeveler cluster system background is described to prepare for the migration:

- Two major clusters: Development and production.
- 10K cores in total.
- The LoadLeveler cluster runs on IBM Power Systems.
- The workload has the following characteristics:
  - Jobs can be sequential and parallel (Message Passing Interface (MPI)).
  - Jobs can come in a flow.
  - A job involves multiple (for example, six) job steps, with dependencies among them. The example environment currently uses job control language (JCL) to describe them.
  - A large volume of jobs exists. Normally, 10,000 jobs run each day. Each job can involve multiple steps. Therefore, the LSF job definition might be 10,000 x 6.
  - The current environment uses preemption.

Timeline

After the assessment, we suggest that you plan a pilot implementation in a small development environment to help understand the environment and to transfer skills. Then, plan a production build and get the system up and running. Then, conduct a user training session to ensure that both the new cluster and the users are ready.

Migration plan phase one (assessment) consists of these tasks:

- Analyze and design:
  - Assess the current LoadLeveler implementations (setup, configurations, policies, scripting, and applications).
  - Understand the business objectives, requirements, and service-level agreements (SLAs).
  - Proposal solution design:
    - LSF cluster design (selecting the products and add-ons, cluster layout and failover, and infrastructure)
    - LSF configuration (such as queue policies), scripting (such as submission and query), monitoring, and other necessary solutions to migrate applications
  - Proposed implementation plan.
- Deliverable:
  - Document the suggested LSF solution proposals.
  - Document the suggested LSF implementation plan for phase two.

Migration plan phase two (pilot implementation) consists of these tasks:

- Build and test:
  - Implement based on the proposal and the plan from phase one.
  - Cover any cases that are not specified by the phase one proposal and plan.
  - Design detailed test case.
  - Execute test case.
- Deliverable:
  - Successful completion of all test cases.
  - Proven functional, working LSF implementation.
  - Ready to build the production environment.

Migration plan phase three (production environment built, up, and running, and user training completed) consists of these tasks:

- Build out and test the production environment based on the pilot implementation:
  - Perform performance, scalability, and stress tests.
  - Ensure that monitoring and alarm systems are in place and ready.
  - Test and ensure the ability to meet the SLA.
  - Ensure operational readiness.
  - Pass the user acceptance testing (UAT).
- Deliverable

  Fully tested functional and operational production environment
- LSF training:
  - LSF basic configuration and administration training completed.
  - LSF advanced configuration and administration training completed.

Mapping

The migration includes two types of mapping: the system-level mapping and the user-level mapping. The system-level mapping is for LoadLeveler or LSF cluster administrators, including the configuration changes and the workload scheduler commands mapping. The user-level mapping is for cluster users, including job file terminology, job submission, and job query.

For more information, see the *IBM LoadLeveler to IBM Platform LSF Migration Guide,* REDP-5048.

The mapping is summarized into the following areas:

- System-level mapping:
  - Cluster setup
  - Mapping resources
  - Machine groups
  - Job management (classes and queues)
  - Job accounting and control
- User-level mapping:
  - Job commands
  - Job states
  - Job I/O and status notification
  - Job resource requirements and preferences
  - Resource limits

- Job files
- Job environment variables
- User interaction

Migration

After the mapping, generate the cluster configuration files and start the cluster migration. LSF is installed in the same node where LoadLeveler is installed but LoadLeveler and LSF must use different ports.

Migration implementation

The following list describes the migration implementation steps:

1. Send a notice to all cluster users. Drain all of the queues (classes) of the LoadLeveler to prevent any user from submitting more jobs. Wait until all current jobs are in completed status. Back up the data.
2. Stop the LoadLeveler cluster.
3. Install the LSF and plug-ins. Configure the LSF and environment for the expected functions of the previous system.
4. Start the LSF cluster.
5. Optional: Test the compatibility with other software and test the script interface.

Then, the tester can test the LSF cluster by running LSF commands and submitting test jobs. User training and job migration are also important as part of the migration.

Value-add of migration to LSF

LSF is a powerful workload scheduler and offers many mature features that can boost your productivity and increase resource utilization. As a value-add step of the migration to LSF, cluster administrators and users can take advantage of and differentiate LSF functions to help realize the following benefits:

- Improve cluster utilization.
- Improve the administrator's ability to monitor and address job and hardware issues.
- Improve control over how jobs are dispatched and how to consume resources for the job.
- Increased flexibility for the developer in defining how their job will be scheduled and executed.

Depending on the specific system requirements, the LSF migration can include the use of the LSF add-on products.

The following list shows the LSF add-on products:

- IBM Platform License Scheduler
- IBM Platform Session Scheduler
- IBM Platform Reporting, Tracking, and Monitoring (RTM)
- IBM Platform Analytics
- IBM Platform Application Center
- IBM Platform Process Manager
- IBM Platform Dynamic Cluster
- IBM Platform Data Manager

Optional add-ons extend Platform LSF to provide a complete set of workload management capabilities to address HPC needs.

D

Applications and performance

Although this book generally targets system administrators and application developers, this appendix covers two topics that mostly relate to the users of IBM POWER8 high-performance computing (HPC) solution servers:

- Application software

  Examples of application software, such as bioinformatics and molecular dynamics packages
- NAS Parallel Benchmarks

  Suggested practices for tuning applications (on an example of the network-attached storage (NAS) Parallel Benchmarks suite)
- Sample code for environment variables’ thread affinity strings

Application software

In this section, we list examples of software packages from the following application domains:

- Bioinformatics
- Molecular dynamics simulations

We also provide basic guidance about the compilation and execution of several of these applications.

The content in "Molecular dynamics" on page 265 originally appeared in the IBM Redpaper publication *NVIDIA CUDA on IBM POWER8: Technical Overview, Software Installation, and Application*, REDP-5169.

Bioinformatics

Personal healthcare is a rapidly growing area these days. 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 is 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 can ideally be equipped with multiple fast processors, large 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 that is 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 BioBuilds suite. It is a no-charge, available collection of open source bioinformatics tools. The package is pre-built and optimized for the IBM Power platform that runs on Linux. It also includes supporting libraries for the tools.

Table D-1 on page 265 lists the set of available tools as of the BioBuilds 2015.04 release. In addition, BioBuilds 2015.04 includes the iRODS application in a beta version.

Table D-1 List of bioinformatics available tools in the BioBuilds 2015.04 release

Bioinformatics tools as of the BioBuilds 2015.04 release			
ALLPATHS-LG	Cufflinks	Mothur	SOAP3-DP[a]
bedtools	FastQC	PICARD	SOAPaligner/soap2
bfast	FASTA	PLINK	SOAPDenovo
BioConductor Base	HMMER	TMAP	tabix
BLAST (NCBI)	HTSeq	RNAStar	TopHat
Bowtie	iSAAC	Samtools	Trinity
BWA	IGV	SHRiMP	Velvet/Oases

a. Requires a POWER8 server that is equipped with a graphics processor unit (GPU)

Because genome sequencing is a compute-intensive task, the sequencing can gain a large performance benefit by using an accelerator. The SOAP3-DP tool that is listed in Table D-1 is an example of an open source bioinformatics application that is enabled with a GPU.

For more information about the BioBuilds collection, see the following link:

http://www.ibm.com/partnerworld/gsd/solutiondetails.do?solution=51837

BALSA

BALSA is another example of an application that uses the computational power of 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

Molecular dynamics

Many existing application programs can use the computational power of NVIDIA GPUs. Systems that are equipped with GPUs are efficient in accelerating compute-intensive applications. In this subsection, GROMACS and LAMMPS are examples of how to install, configure, and run Compute Unified Device Architecture (CUDA) applications on a POWER8 server that is accelerated by a GPU.

GROMACS

GROningen MAchine for Chemical Simulations (GROMACS) is a molecular dynamics package that is primarily designed for simulations of proteins, lipids, and nucleic acids. For more information about GROMACS, see the following website:

http://www.gromacs.org

Installation of GROMACS

If the CMake tool and the Fastest Fourier Transform in the West (FFTW) library are not available in the server, request the system administrator to install the *CMake*, *libfftw3-3*, and *libfftw3-dev* packages by using the following command:

```
$ sudo apt-get install cmake libfftw3-3 libfftw3-dev
```

Before the installation, download and unpack the GROMACS source code from the GROMACS website. Example D-1 shows the required commands to install GROMACS version 4.6.7.

Example D-1 The commands to install GROMACS

```
$ mkdir -p $HOME/install/gromacs467
$ tar xfz gromacs-4.6.7.tar.gz
$ cd gromacs-4.6.7
$ mkdir build
$ cd build/
$ PREFIX=$HOME/local/gromacs/4.6.7
$ cmake .. -DGMX_GPU=ON -DGMX_PREFER_STATIC_LIBS=ON -DCMAKE_INSTALL_PREFIX=$PREFIX
$ make
$ make install
$ source $PREFIX/bin/GMXRC
```

Running GROMACS simulations

Several test data sets are available at the GROMACS website. In this section, the `water_GMX50_bare.tar.gz` data set serves as an example. You can download the data set by following the following link:

ftp://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz

Three types of input files are in the `water_GMX50_bar.tar.gz` data set:

- The `.mdp` file defines the parameters for running the simulation.
- The `.top` file extension stands for topology.
- The `.gro` file is a fixed-column coordinate file format. (This format was first used in the GROMOS simulation package.)

To run the simulation, you need to generate a `.tpr` file. The `.tpr` file extension stands for the portable binary run input file. The `.tpr` file contains the starting structure of the simulation (coordinates and velocities), the molecular topology, and all of the simulation parameters. The `.tpr` file is generated by the **`grompp`** command and then executed by the **`mdrun`** command to run the simulation. Example D-2 shows how to prepare the data set for the simulation.

Example D-2 Commands to prepare a GROMACS simulation data set

```
$ tar zxf water_GMX50_bare.tar.gz
$ cd water-cut1.0_GMX50_bare/1536
$ grompp -f pme.mdp
```

The final step is to run the simulation. Run the following command in the data set directory:

```
$ mdrun -s topol.tpr -npme 0 -resethway -noconfout -nb gpu -nsteps 1000 -v
```

The output of the command contains the GPU information for the run as shown in Example D-3.

Example D-3 Output of a GROMACS simulation run

```
<... output omitted ...>
Reading file topol.tpr, VERSION 4.6.7 (single precision)
Changing nstlist from 10 to 40, rlist from 1 to 1.101

Overriding nsteps with value passed on the command line: 1000 steps, 2.000 ps
```

```
Using 2 MPI threads
Using 80 OpenMP threads per tMPI thread

2 GPUs detected:
  #0: NVIDIA Tesla K40m, compute cap.: 3.5, ECC: yes, stat: compatible
  #1: NVIDIA Tesla K40m, compute cap.: 3.5, ECC: yes, stat: compatible

2 GPUs auto-selected for this run.
Mapping of GPUs to the 2 PP ranks in this node: #0, #1

starting mdrun 'Water'
1000 steps,      2.0 ps.
step   80: timed with pme grid 200 200 200, coulomb cutoff 1.000: 1591.3 M-cycles
step  160: timed with pme grid 168 168 168, coulomb cutoff 1.187: 1333.3 M-cycles
step  240: timed with pme grid 144 144 144, coulomb cutoff 1.384: 1655.6 M-cycles
<... output omitted ...>
```

Support for the message passing interface (MPI) was not enabled at the time of compilation. However, GROMACS has its internal thread-based implementation of MPI. For this reason, **mdrun** reports the usage of MPI when it is running on a two-socket IBM Power System S824L testbed. For more information about the available options for acceleration and parallelization of GROMACS, see the following web page:

http://www.gromacs.org/Documentation/Acceleration_and_parallelization

LAMMPS

Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is classical molecular dynamics simulation code that runs efficiently on parallel computers. LAMMPS runs on single processors or in parallel by using message-passing techniques and a spatial-decomposition of the simulation domain. For more information about LAMMPS, see the following website:

http://lammps.sandia.gov

Installation of LAMMPS

If the CMake tool and the FFTW library are not available in the server, ask the system administrator to install the *CMake*, *libfftw3-3*, and *libfftw3-dev* packages by using the following command:

```
$ sudo apt-get install cmake libfftw3-3 libfftw3-dev
```

Before the installation, download and unpack the LAMMPS source code from the LAMMPS website:

http://lammps.sandia.gov/download.html#tar

Example D-4 shows the detailed commands that can be used to deploy LAMMPS (stable version, 30 Oct 2014).

Example D-4 Deploying LAMMPS

```
$ tar xzf lammps.tar.gz
$ cd ./lammps-30Oct14/
$ export CUDA_HOME=/usr/local/cuda
$ export PATH=$PATH:/usr/local/cuda/bin
$ cd src/
$ cp MAKE/Makefile.serial MAKE/MINE/Makefile.auto
$ python Make.py -p cuda -cuda arch=20 -o cuda lib-cuda exe
```

```
Installing packages ...
Action lib-cuda ...
building cuda library ...
Created lib/cuda library
Action exe ...
Created src/MAKE/MINE/Makefile.auto
building serial STUBS library ...
Created src/STUBS/libmpi_stubs.a
Created lmp_cuda in /home/<username>/lammps-30Oct14/src
```

The binary **`lmp_cuda`** is ready for running LAMMPS simulations.

Running LAMMPS simulations

Several data sets for benchmarking purposes are available on the LAMMPS website:

http://lammps.sandia.gov/bench.html

In this section, the `in.lj` data set is used to demonstrate the LAMMPS simulation. Download the data set from the following website:

http://lammps.sandia.gov/inputs/in.lj.txt

After the data set is downloaded, rename the file to `in.lj`. Run the following commands for the simulation:

```
$ <LAMMPS_installation_dir>/src/lmp_cuda -sf cuda -c on -v g 2 -v x 64 -v y 64 -v
z 64 -v t 1000 < in.lj
```

The output of the command is shown in Example D-5. The output contains the GPU information that relates to the run.

Example D-5 Output of a LAMMPS simulation

```
# Using LAMMPS_CUDA
USER-CUDA mode is enabled (../lammps.cpp:475)
Lattice spacing in x,y,z = 1.6796 1.6796 1.6796
Created orthogonal box = (0 0 0) to (33.5919 33.5919 33.5919)
# CUDA: Activate GPU
# Using device 0: Tesla K40m
  1 by 1 by 1 MPI processor grid
Created 32000 atoms
# CUDA: Using precision: Global: 8 X: 8 V: 8 F: 8 PPPM: 8
Setting up run ...
# CUDA: VerletCuda::setup: Upload data...
Test TpA
Test BpA

# CUDA: Timing of parallelization layout with 10 loops:
# CUDA: BpA TpA
 0.052649 0.011521
# CUDA: Total Device Memory usage post setup: 162.921875 MB
<... output omitted ...>
```

NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB)[1] 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 prime examples of the most widespread numerical kernels into their product. We use the OpenMP flavors of NPB benchmarks to achieve the following goals:

- Provide guidance for the choice of compilation parameters
- Show the variation of performance for different simultaneous multithreading (SMT) modes
- Demonstrate the importance of binding threads to logical processors
- Confirm the equivalence of logical processors of a core

For benchmarking, we used a 20-core offering of IBM Power System S824L that is based on the POWER8 processor. The cores run at 3.42 GHz. Each memory slot of the system was populated with a 16 GB RAM module for a total of 256 GB. The server was running the Ubuntu 14.10 (little-endian) operating system in a non-virtualized mode. The Linux kernel version was 3.16.0-23. We used version v15.1.1 of IBM Xpertise Library (XL) Fortran compiler (pre-release) 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 a possible impact 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 that are shown here.

The size of problems in the NPB benchmarking suite is predefined by the developers. We opted for the benchmarks of class C. The volume of memory that is required for this class exceeds the total amount of all cache memory of both sockets. We did not change the source code of the NPB suite. We controlled the code generation by setting compilation parameters in a `make.def` file as shown in Example D-6.

Example D-6 NPB: A sample make.def file for the -O3 -qhot parameter set

```
F77 = xlf_r -qsmp=omp -qnosave
FLINK = $(F77)
FFLAGS = -qarch=auto -qtune=auto -O3 -qhot -qmaxmem=-1 -q64
FLINKFLAGS = $(FFLAGS)
CC = xlc_r -qsmp=omp
CLINK = $(CC)
C_LIB = -lm
CFLAGS = $(FFLAGS)
CLINKFLAGS = $(CFLAGS)
UCC = xlc
BINDIR = ../bin
RAND  = randi8
WTIME = wtime.c
MACHINE = -DIBM
```

It is not feasible to cover all of the phase space of compilation parameters, so we varied only the most important ones. The following parameters were common for all builds:

```
-qsmp=omp -qnosave -qarch=auto -qtune=auto -qmaxmem=-1 -q64
```

[1] "NAS Parallel Benchmarks" at http://www.nas.nasa.gov/publications/npb.html
[2] At the time of this publication, IBM XL Fortran for Linux, V15.1.2 (little-endian distribution) was available.

We considered four sets of compilation parameters. In the addition to the previous compilation parameters, the remaining parameters are listed in Table D-2. The column **Option set name** lists shortcuts that we use to reference each set of compilation parameters that are presented in the **Compilation parameters** column.

Table D-2 NPB: Compilation parameters that were used to build the NPB suite executables

Option set name	**Compilation parameters**
-O3 -qnohot	`-O3 -qnohot`
-O3 -qhot	`-O3 -qhot`
-O5 -qnohot	`-O5 -qnohot`
-O5 -qprefetch	`-O5 -qprefetch=aggressive`

The *-O3 -qnohot* option set proved to be less efficient for all of the tests, so we omit it in our analysis.

We performed all of the runs with the following set:

```
export OMP_DYNAMIC="FALSE"
export OMP_SCHEDULE="static"
```

To establish the affinity of threads, we used a simple program. We list the source code for this program in the “Sample code for environment variables’ thread affinity strings” on page 285.

Note: In several cases, performance results that are presented deviate significantly from the general behavior within the same plot. The plot bars can be ignored because the tests can experience unlucky conditions (operating system jitters, parasitic external workload, and so on).

Choice of an SMT mode

The POWER8 core is able to simultaneously run instructions from up to eight application threads. The number of threads for each core, which are necessary to deliver the optimal performance, can vary for different applications. Compilation parameters and core layout can also affect the timing.

On the bar charts that are shown in Figure D-1 on page 272 through Figure D-7 on page 278, we examine the performance benefits that come from the rational choice of SMT mode for several applications from the NPB suite (*bt.C, cg.C, ep.C, ft.C, lu.C, mg.C,* and *sp.C*). In our measurements, we considered the following information:

- Three sets of compiler options, which are shown in Table D-2
- Fifteen different core layouts with one and two sockets, one and two chips for each socket, and 1-5 cores for each chip

The plots are organized as shown:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a set of compiler options.
- The *x*-axis lists core layouts. For example, the eighth triple ("sockets: 1, chips: 2, cores: 3") designates the benchmarking run where six application threads were bound to cores from two chips (three threads for each chip) within one socket.
- The *y*-axis shows the performance gain (in a percentage) in relationship to a baseline. As a baseline, we choose an SMT mode that is less favorable for the particular application.

The results show that the rational choice of SMT mode affects performance substantially.

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. Our recommendation is for the administrator to continue to run the system in SMT8 mode and to use the physical processor as a unit of hardware resource allocation. By using the physical processor as a unit of hardware resource allocation, we 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 for each core. After the value is identified, that number can be considered when you arrange actual computations.

If possible, we advise the user to recompile the application with the *-qtune=pwr8:smtX* IBM XL compiler option (where *X* is one of 1, 2, 4, or 8, depending on the favorable SMT mode), and repeat the benchmark.

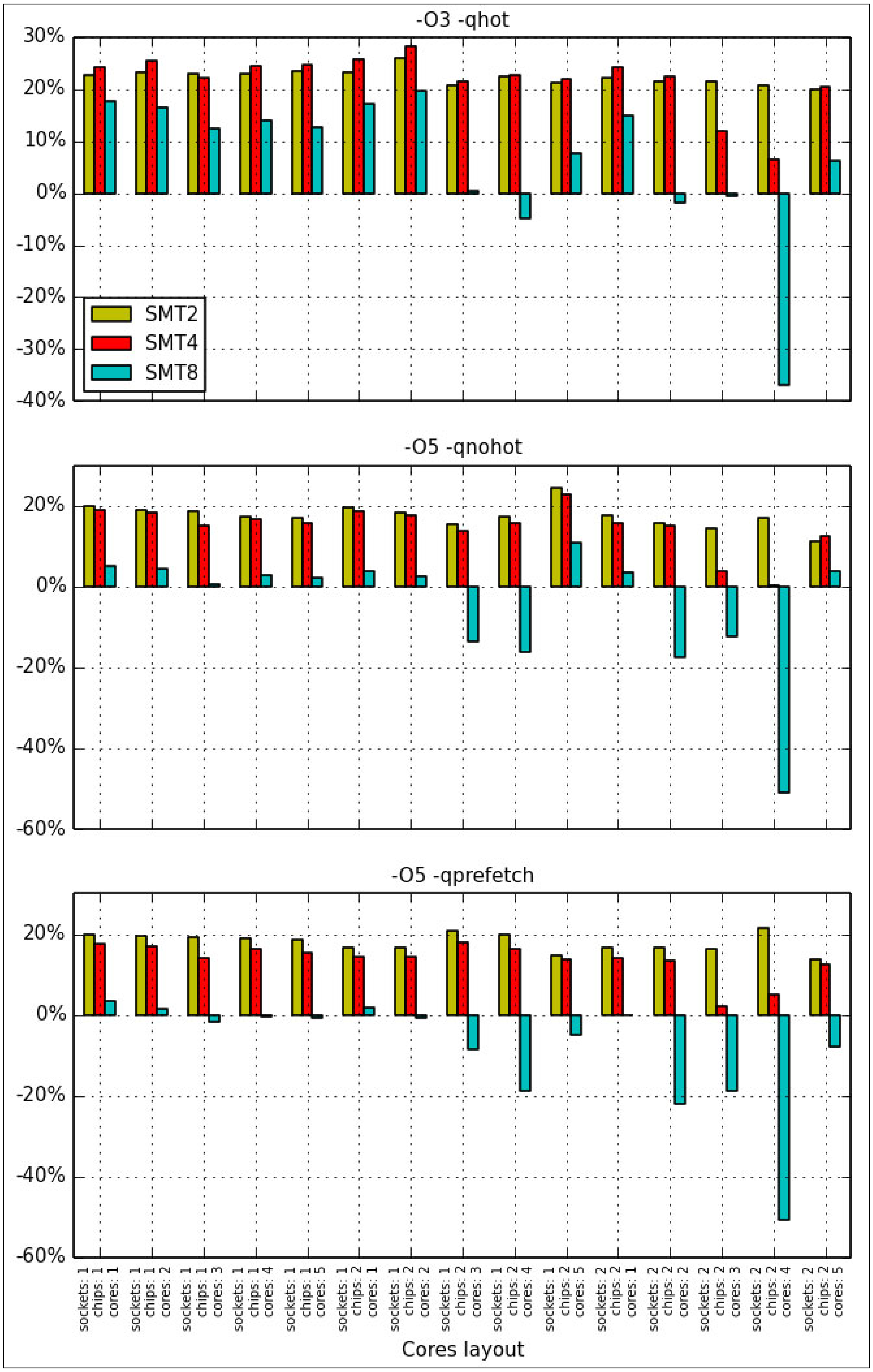


Figure D-1 Performance benefits from the rational choice of SMT mode for the NPB bt.C application

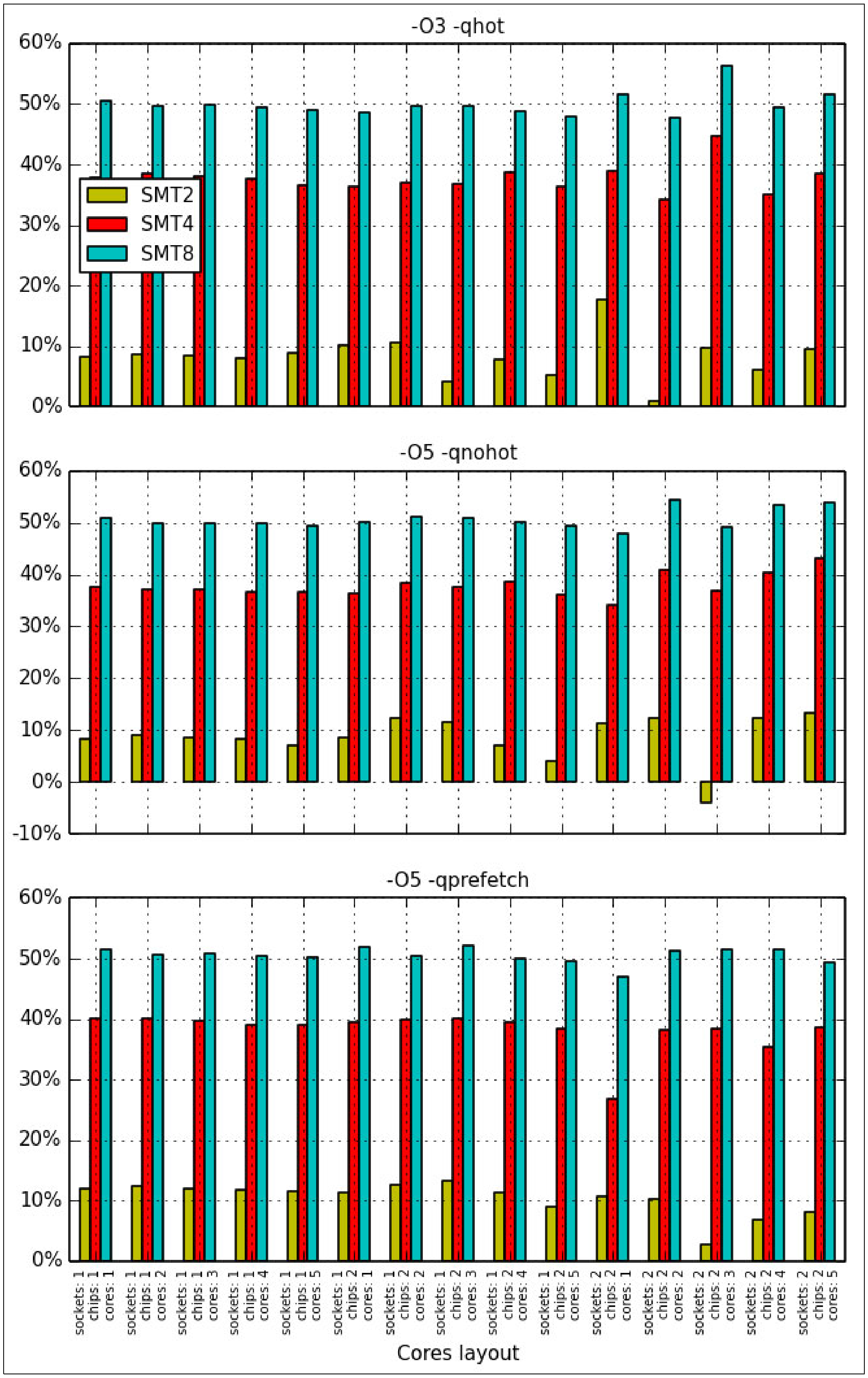


Figure D-2 Performance benefits from the rational choice of SMT mode for the NPB cg.C application

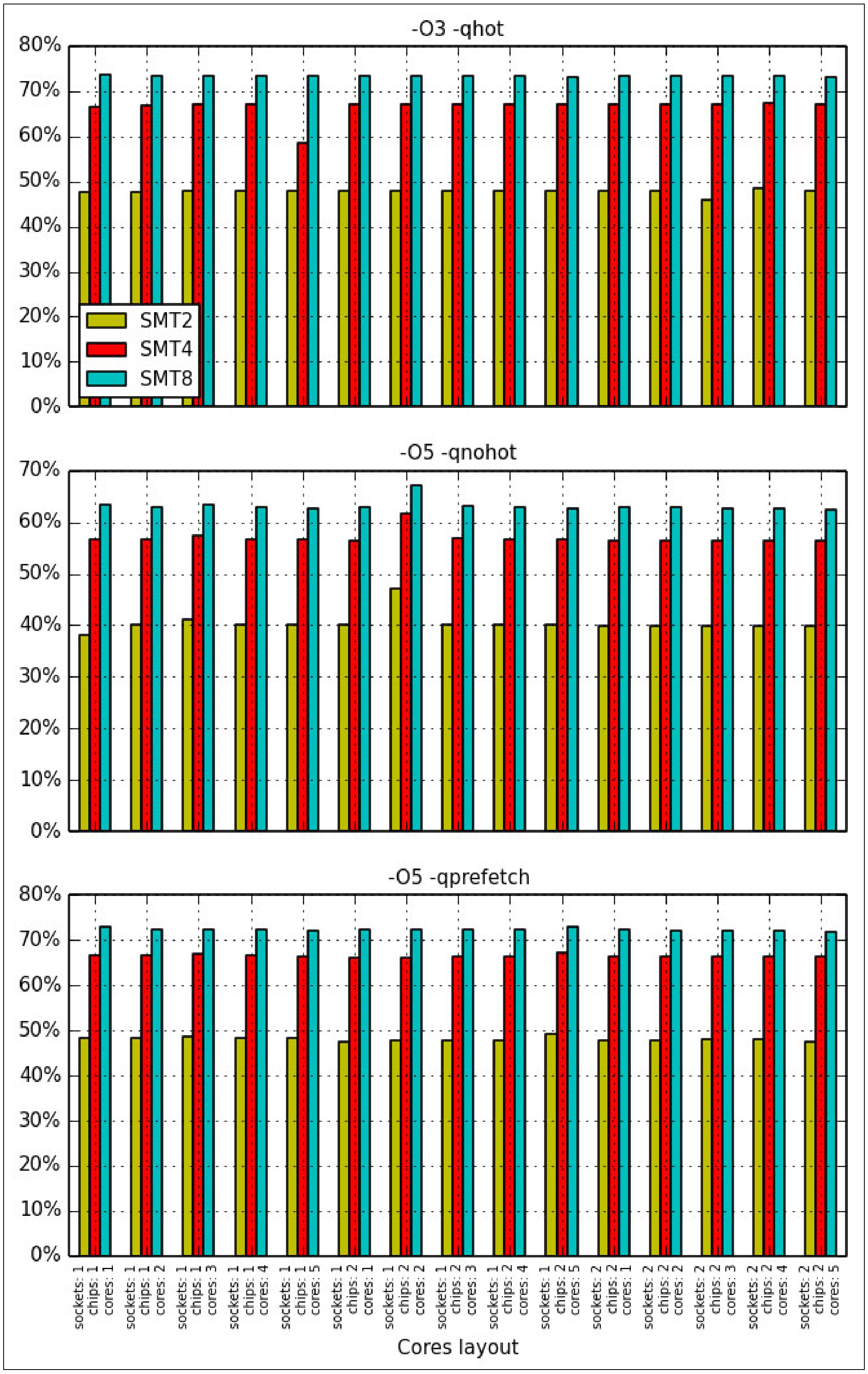


Figure D-3 Performance benefits from the rational choice of SMT mode for the NPB ep.C application

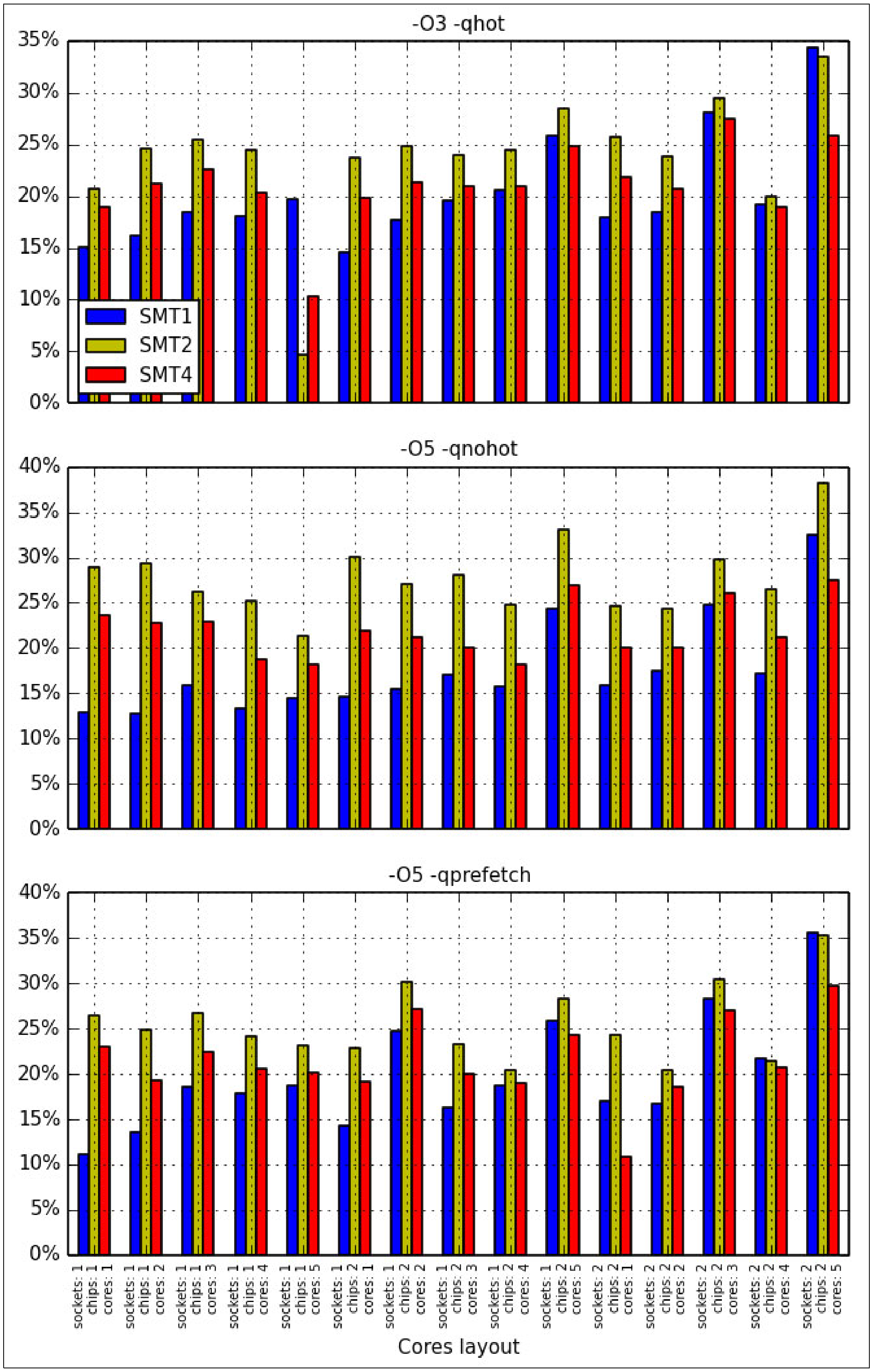


Figure D-4 Performance benefits from the rational choice of SMT mode for the NPB ft.C application

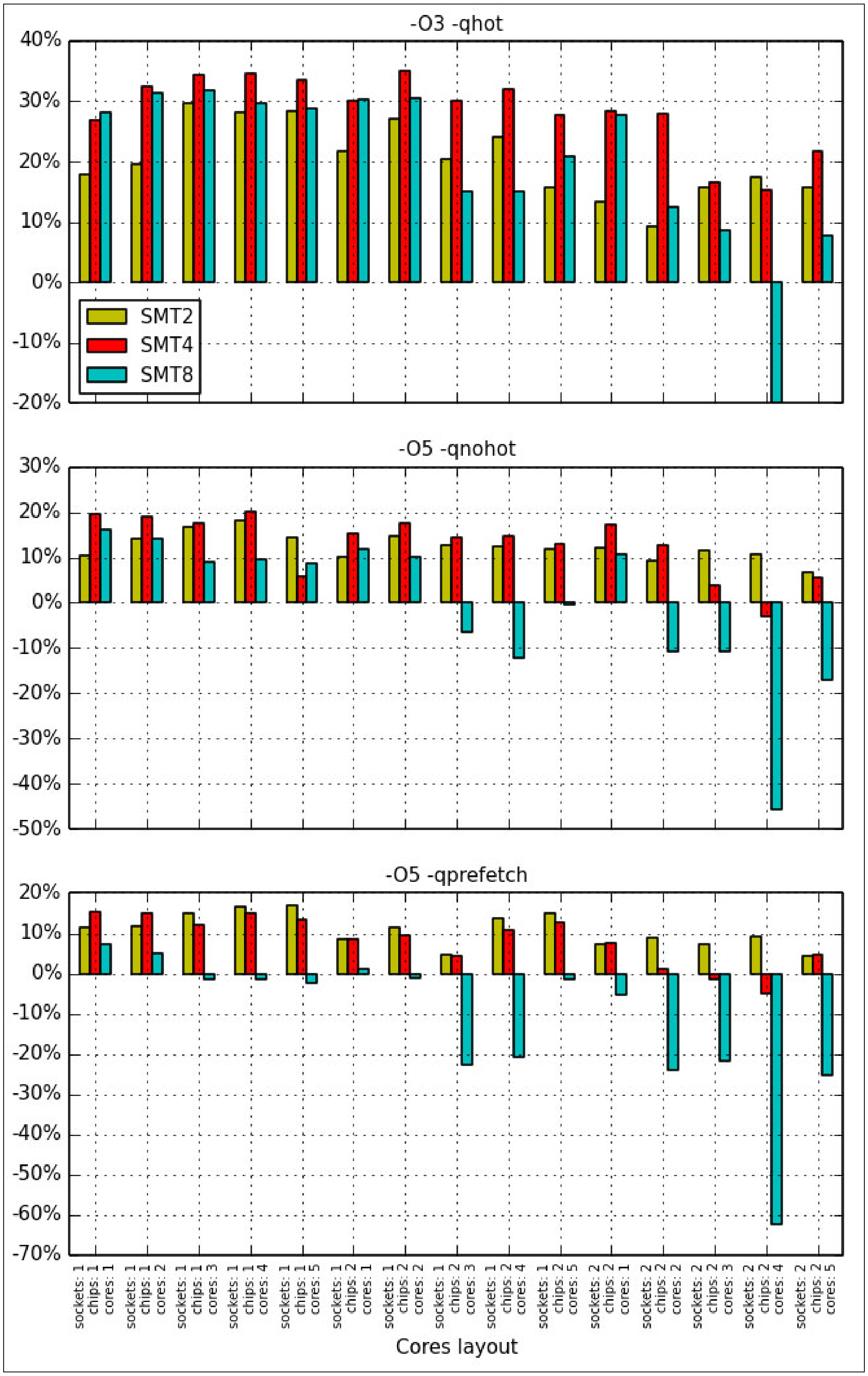


Figure D-5 Performance benefits from the rational choice of SMT mode for the NPB lu.C application

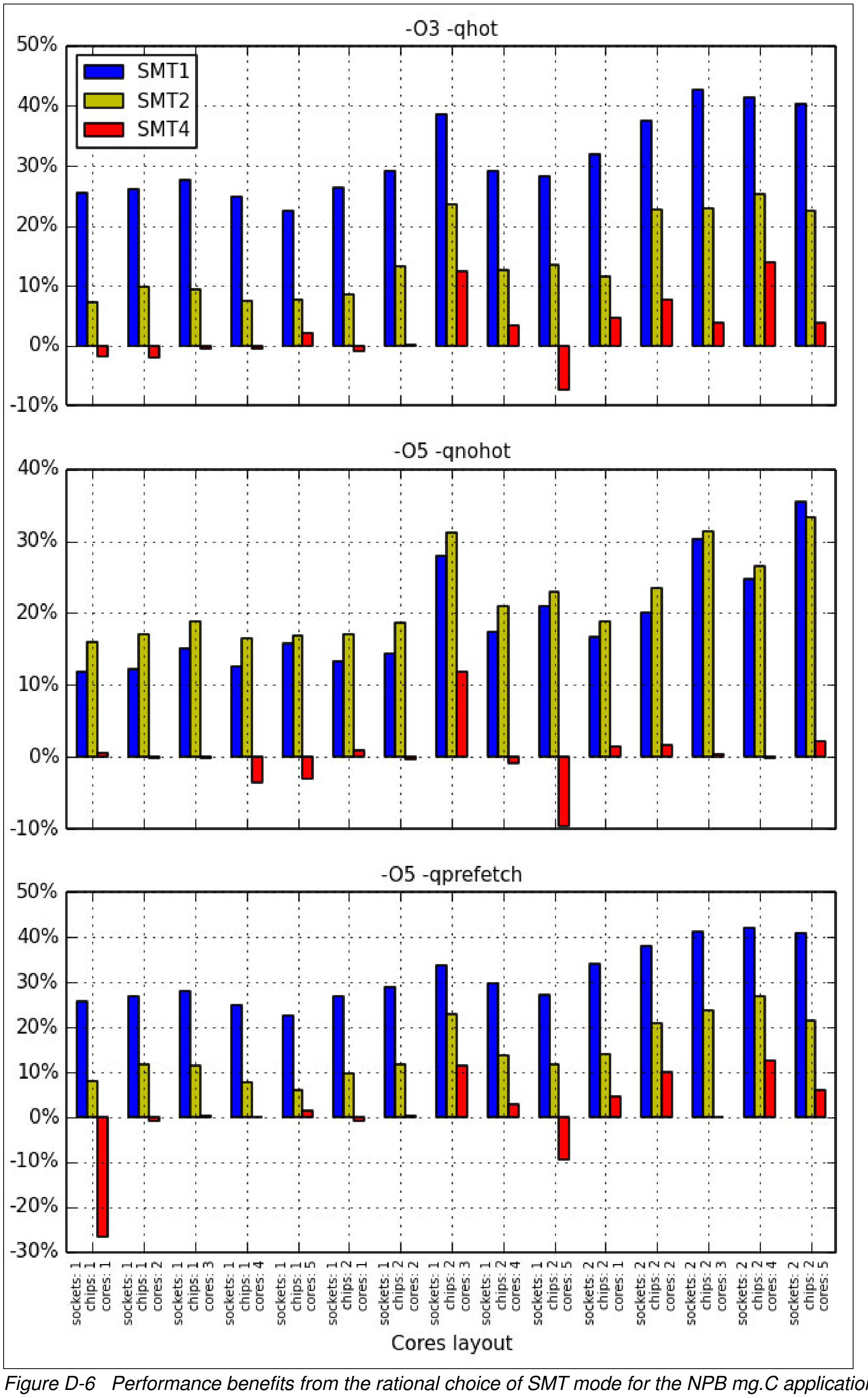


Figure D-6 Performance benefits from the rational choice of SMT mode for the NPB mg.C application

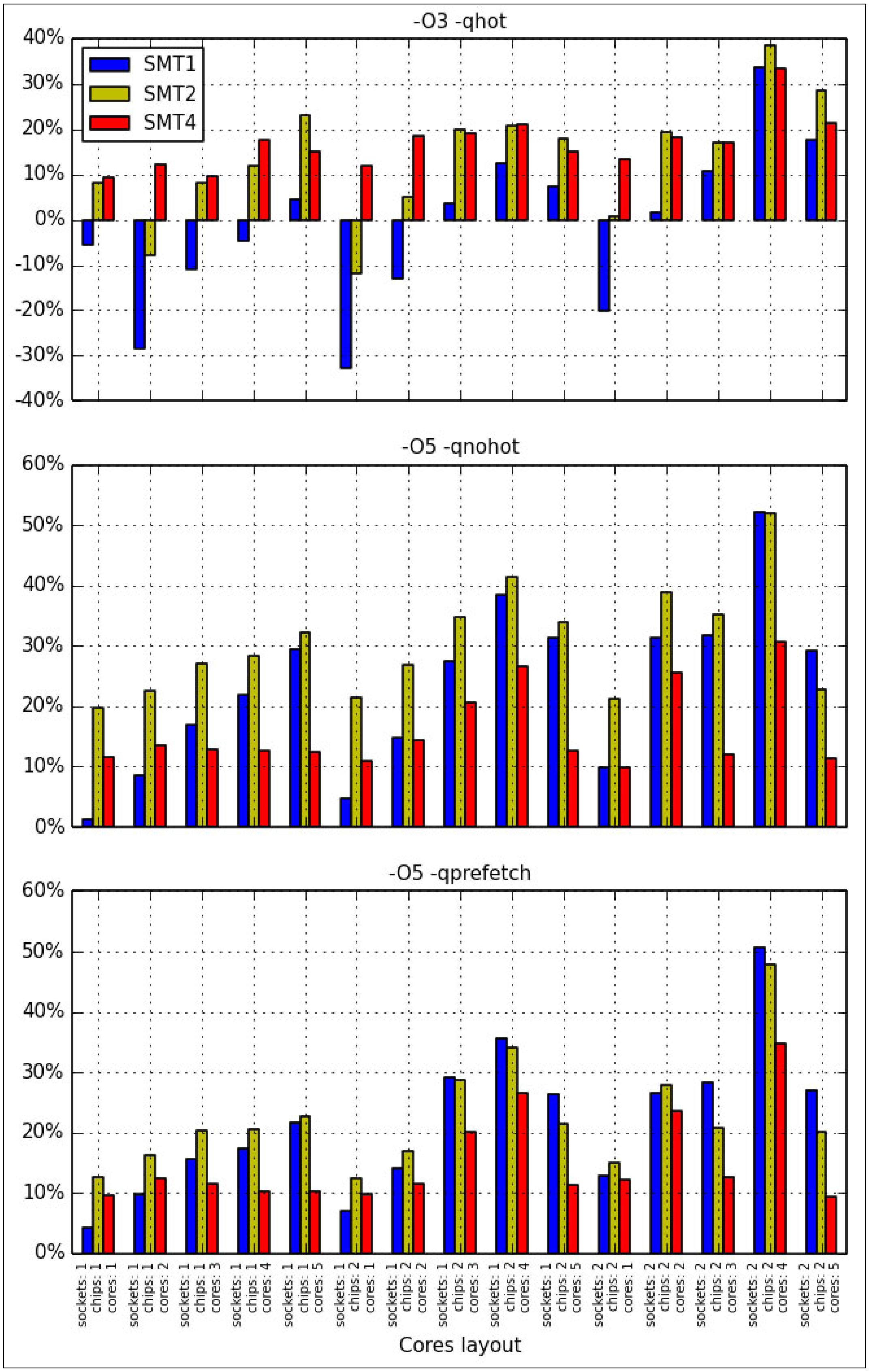


Figure D-7 Performance benefits from the rational choice of SMT mode for the NPB sp.C application

Choice of compilation parameters

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 cannot perform as well for other types of computations. Our advice is to choose a favorable set of compiler options that is based on timing a particular application.

On the bar charts that are shown in Figure D-8 through Figure D-14 on page 282, we 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, lu.C, mg.C,* and *sp.C*) that we examined in “Choice of an SMT mode” on page 270. Again, we considered the three sets of compiler options that are shown in Table D-2 on page 270. The application threads are bound to five cores of one chip within the socket.

The organization of the plots is similar to the scheme that was used in “Choice of an SMT mode” on page 270:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to SMT mode.
- The *x*-axis lists the number of cores that is used.
- The *y*-axis shows the performance gain (in a percentage) in relationship to a baseline. As a baseline, we choose 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, except for *cg.C*.

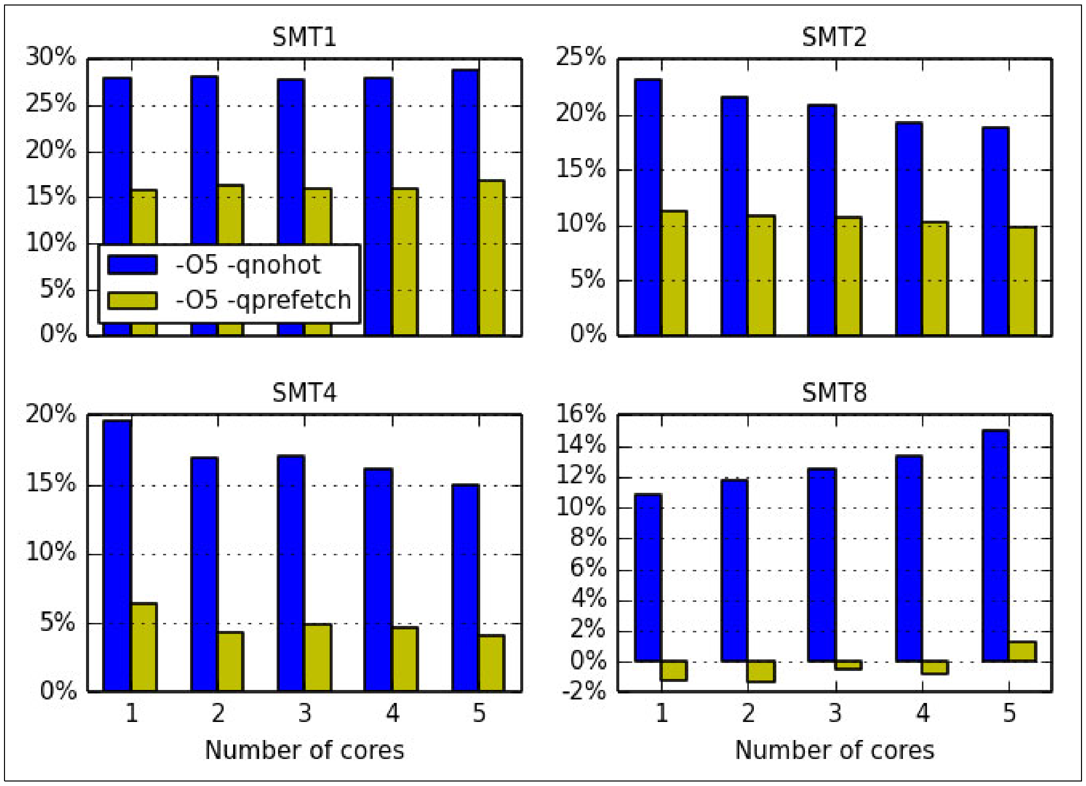


Figure D-8 Performance gain from the rational choice of compiler options for the NPB bt.C test

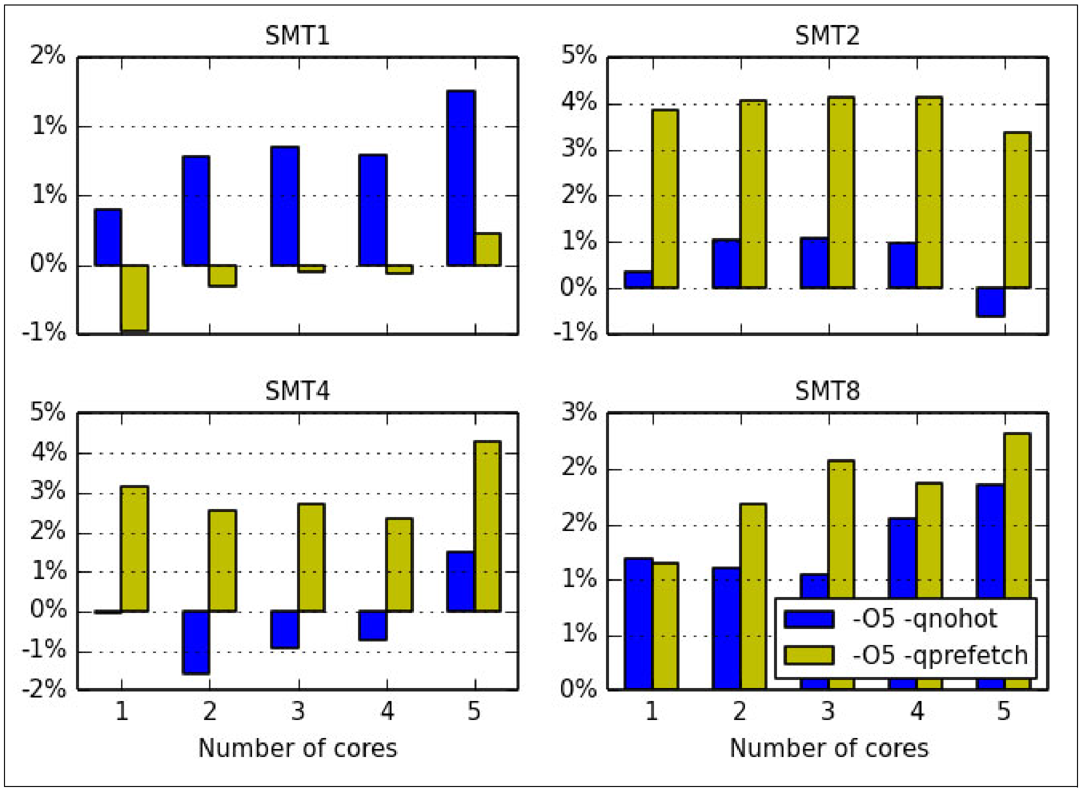


Figure D-9 Performance gain from the rational choice of compiler options for the NPB cg.C test

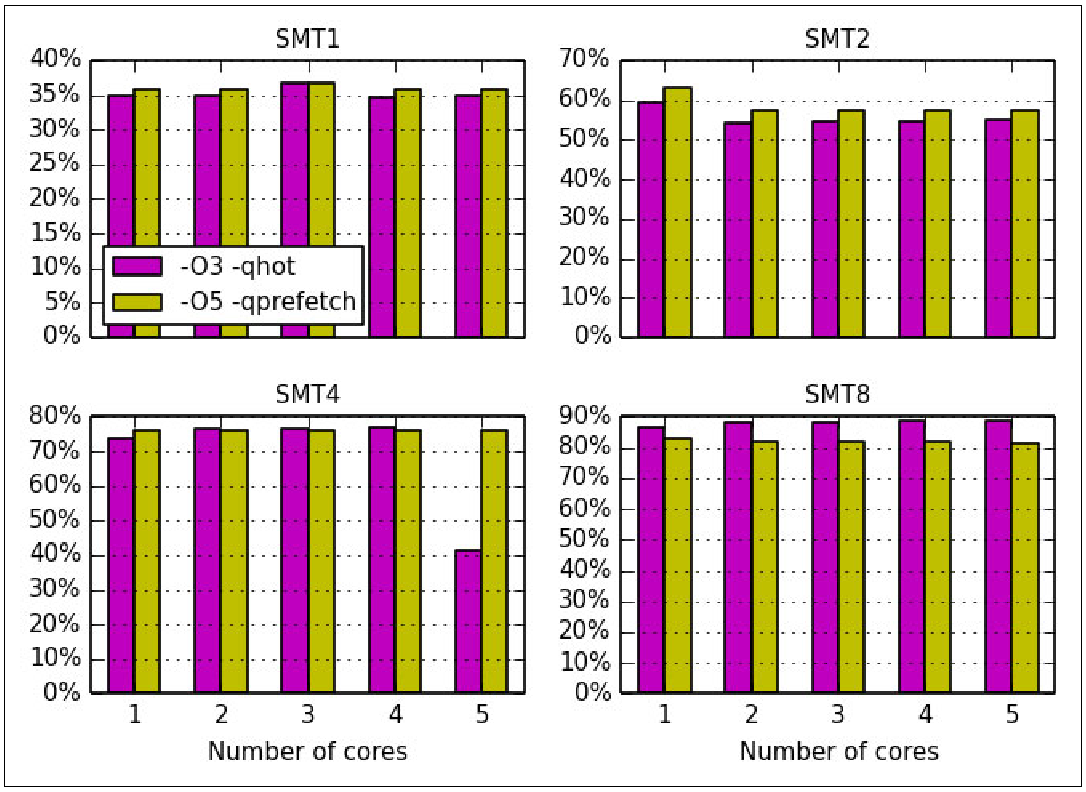


Figure D-10 Performance gain from the rational choice of compiler options for the NPB ep.C test

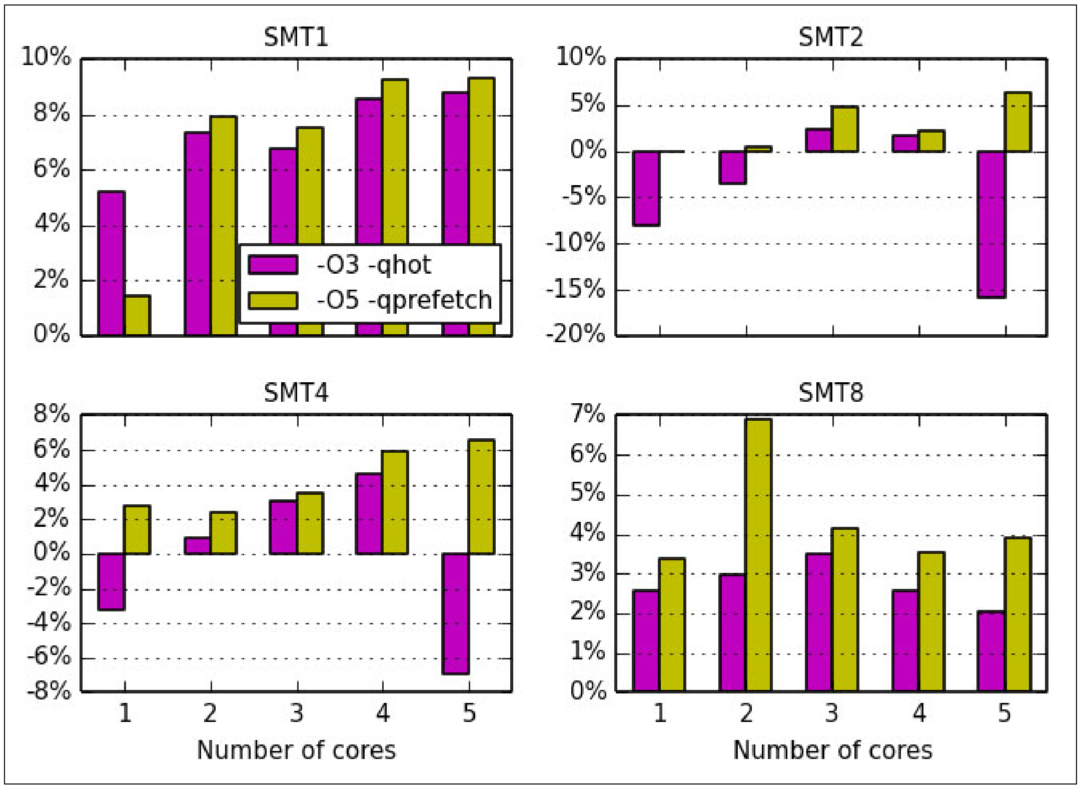


Figure D-11 Performance gain from the rational choice of compiler options for the NPB ft.C test

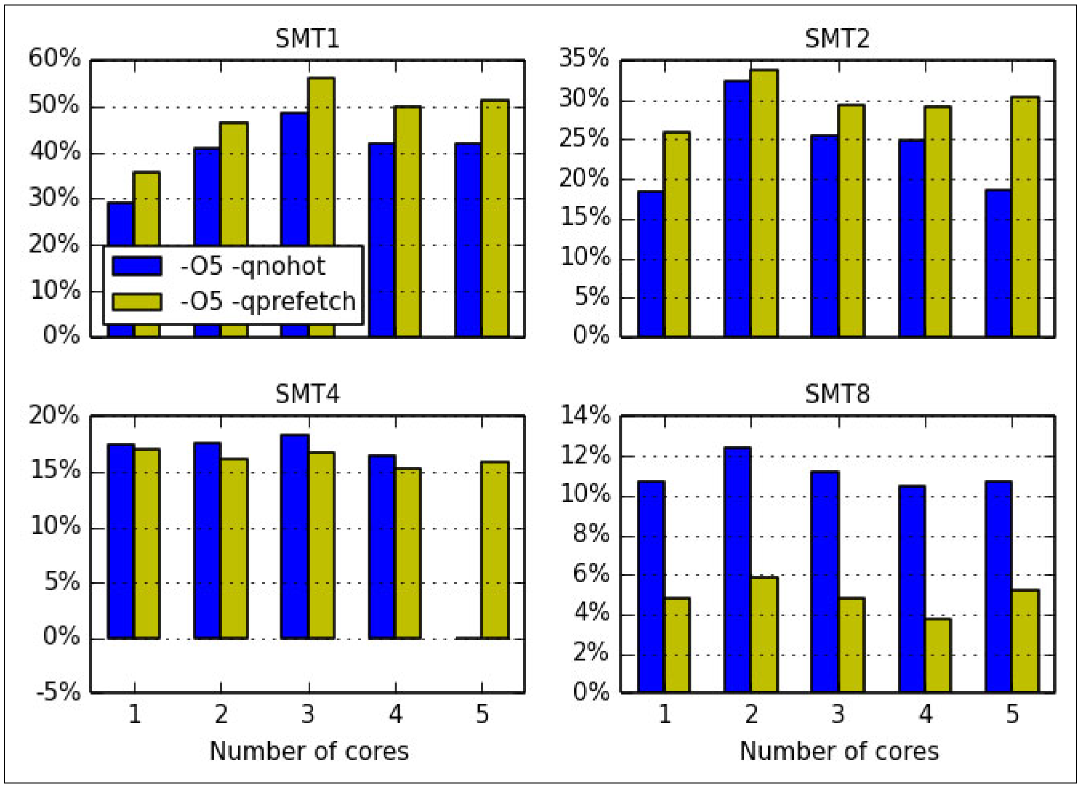


Figure D-12 Performance gain from the rational choice of compiler options for the NPB lu.C test

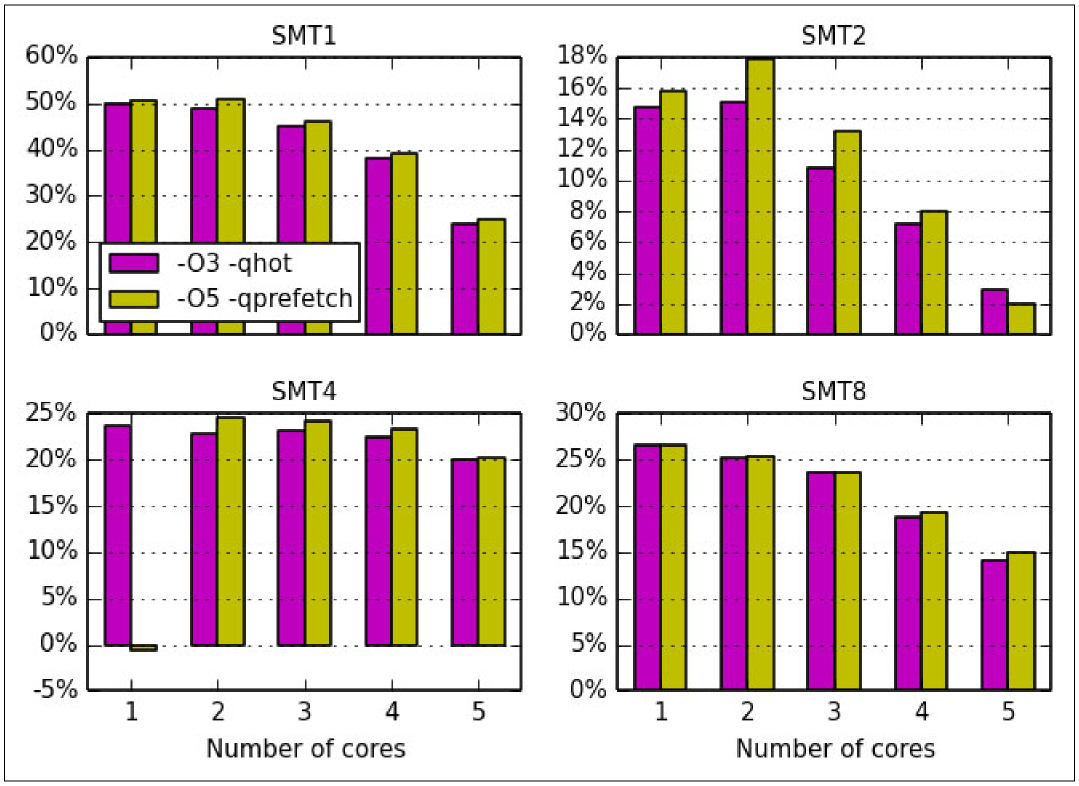


Figure D-13 Performance gain from the rational choice of compiler options for the NPB mg.C test

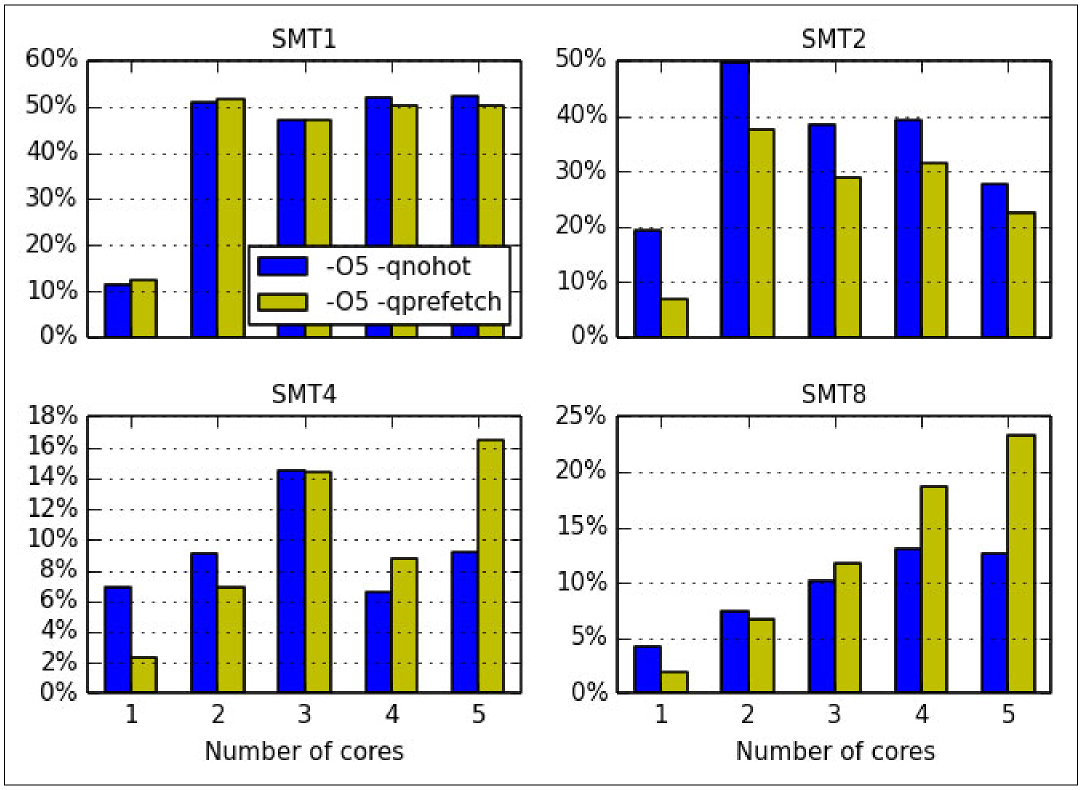


Figure D-14 Performance gain from the rational choice of compiler options for the NPB sp.C test

Importance of threads affinity

The operating system can migrate application threads between logical processors if a user does not explicitly specify thread affinity. As shown in 7.3.1, “Controlling the execution of multithreaded applications” on page 185), a user can specify the binding of application threads to a specific group of logical processors, at the source code level or by setting environment variables.

For technical computing workloads, you typically want to ensure that application threads are bound to logical processors. It often helps to use the POWER architecture memory hierarchy and reduce the overhead that is related to the migration of threads.

To demonstrate the importance of this technique, we chose the $mg.C$ test as an example. The performance of the $mg.C$ application peaks at SMT1. So, for this benchmark, we can expect a penalty if the operating system puts more than one thread for each core.

Figure D-15 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 realized after the assignment of software threads to hardware threads. As SMT mode increases, the operating system can more freely schedule threads, and the effect of thread binding becomes more pronounced for higher values of SMT mode.

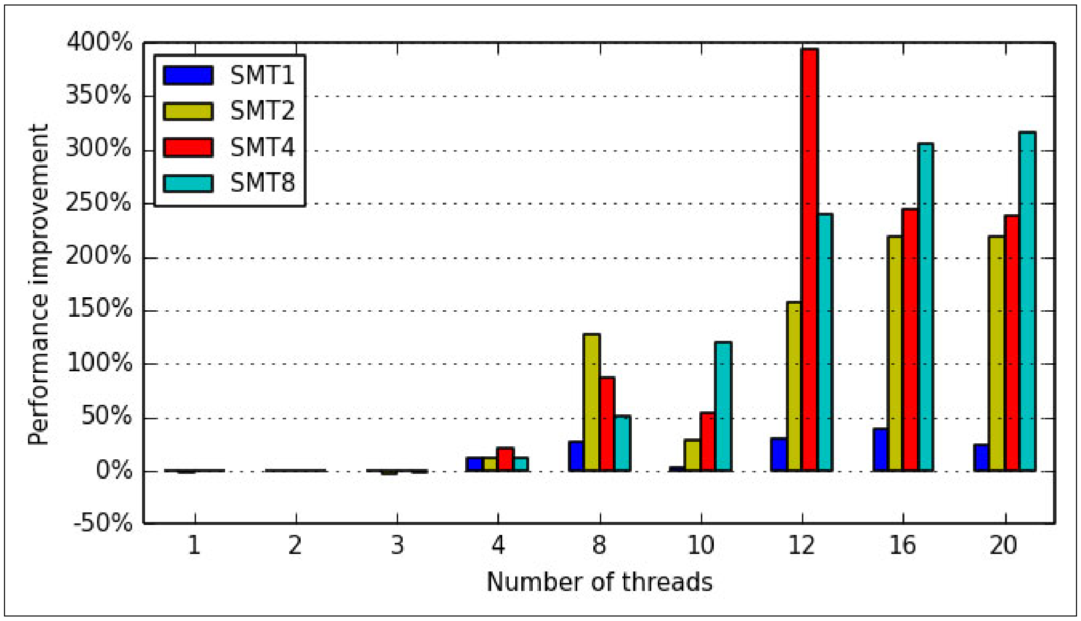


Figure D-15 Performance improvement for an application when thread binding is used

Equivalence of hardware threads

The eight hardware architected threads of the POWER8 core are equivalent if only one or two application threads are running on a core. It does not matter to which hardware thread positions those two threads are bound[3]. As shown in Figure D-16 on page 284 and in Figure D-17 on page 284, the performance variation between logical processors of a core for a single-threaded application and a double-threaded application is ineligible.

[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.

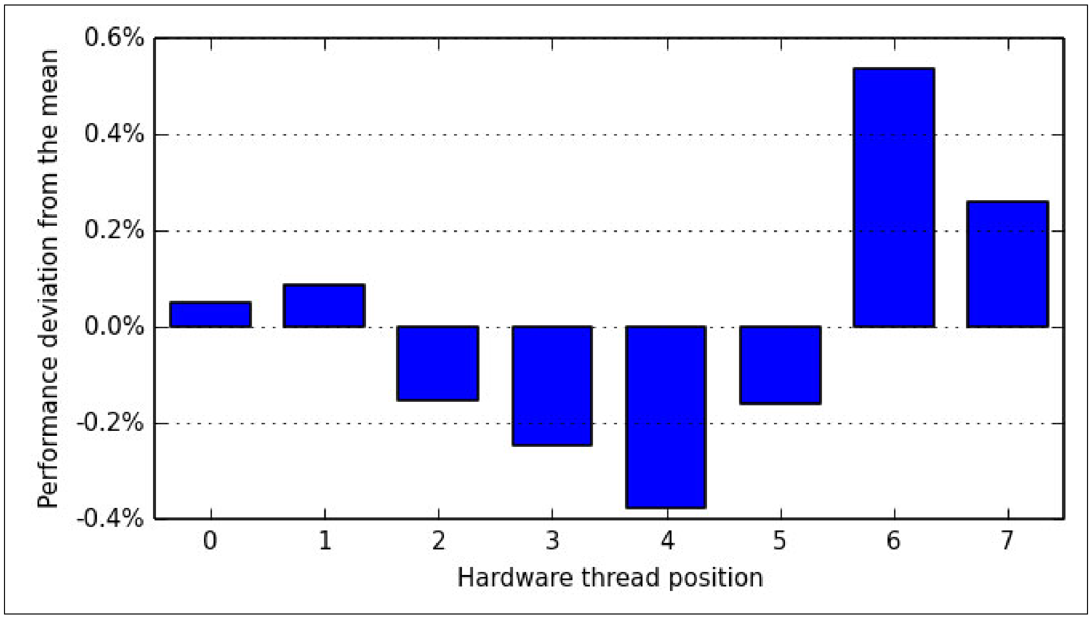


Figure D-16 Performance deviation of a single-threaded application, depending on a thread position

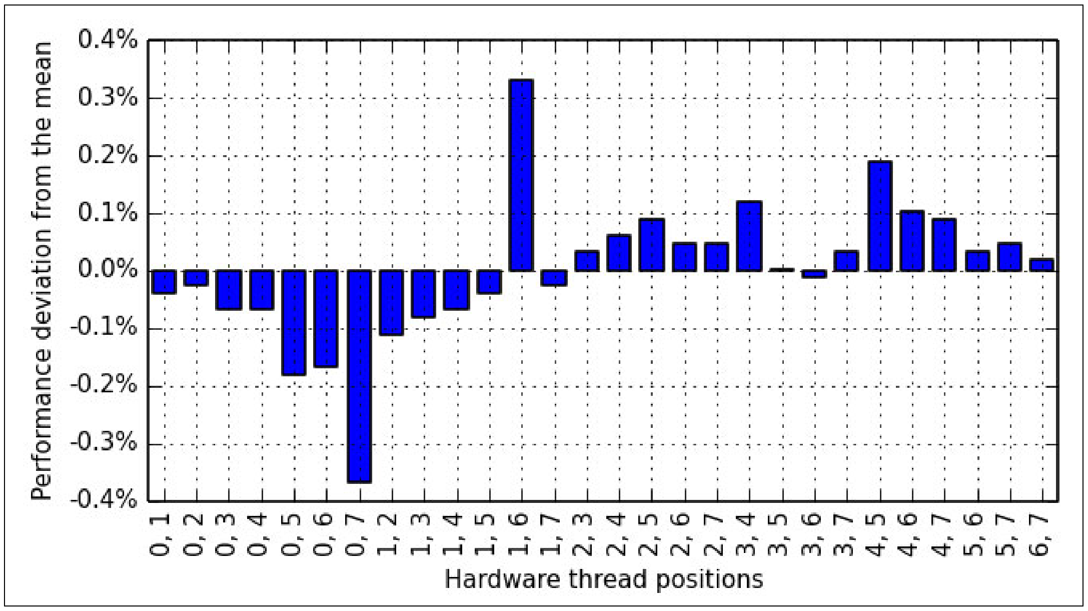


Figure D-17 Performance deviation of a double-threaded application, depending on threads' positions

For a single-threaded application, we chose the $mg.C$ test because its performance peaks at SMT1. We built it with the **-O5 -qprefetch** set of compilation parameters. We used one processor core and performed eight runs, each time we attached the application thread to a different logical processor.

For a double-threaded benchmark, we chose the $ep.C$ test. Because of its inherently parallel nature, we were confident that the two threads carry the minimal algorithmic dependence of each other's execution. We built it with the **-O3 -qhot** set of compilation parameters. We used one processor core and performed 28 runs, each time we attached the application threads to a different pair of logical processors.

Sample code for environment variables' thread affinity strings

Example D-7 provides the source code `threads_map.c` for the program `threads_map` that you can use to generate a string that can be assigned to the environment variables that are responsible for thread binding.

Example D-7 Source code threads_map.c (in C programming language) for the program threads_map

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

#define MAX_TPC 8 // 8 is for the POWER8 processor (max SMT mode is SMT8)
#define MAX_CPC 5 // 5 is for a 20-core system, 6 is for a 24-core system
#define MAX_CPS 2 // two chips per socket
#define MAX_SPS 2 // two sockets per system
#define MAX_THR (MAX_TPC * MAX_CPC * MAX_CPS * MAX_SPS)

static void Print_Map(int sps, int cps, int cpc, 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, chip, socket;

  const int tot = sps * cps * cpc * tpc;
  int cur = 0;

  for (socket = 0; socket < sps; ++socket) {
    for (chip = 0; chip < cps; ++chip) {
      for (core = 0; core < cpc; ++core) {
        for (thread = 0; thread < tpc; ++thread) {
          int shift = socket * MAX_CPS * MAX_CPC * MAX_TPC +
                                  chip * MAX_CPC * MAX_TPC +
                                            core * MAX_TPC;
          shift += base;
          ++cur;
          int c = (cur != tot) ? sep : '\n';
          printf("%d%c", shift + maps[tpc-1][thread], c);
        }
      }
    }
  }
}

static void Print_Usage(char **argv) {
    fprintf(stderr, "Usage: %s "
        "threads_per_core=[1-%d] "
        "cores_per_chip=[1-%d] "
        "chips_per_socket=[1-%d] "
        "sockets_per_system=[1-%d] "
        "base_thread=[0-%d]\n",
        argv[0], MAX_TPC, MAX_CPC, MAX_CPS, MAX_SPS, MAX_THR-1);
}

int main(int argc, char **argv) {
  const int num_args = 5;

  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);
  }

  const int tpc = atoi(argv[1]);
  const int cpc = atoi(argv[2]);
  const int cps = atoi(argv[3]);
  const int sps = atoi(argv[4]);
  const int base = atoi(argv[5]);

  if (tpc < 1 || tpc > MAX_TPC || cpc < 1 || cpc > MAX_CPC ||
      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);
  }

  const int tot = sps * cps * cpc * 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, cpc, tpc, base);

  return EXIT_SUCCESS;
}
```

After you compile the code with the C compiler of your choice, you can use it as shown. The first four arguments specify how many threads, cores, chips, and sockets you want to use. The last argument specifies the first logical processor to use.

If you compiled your application with IBM XL compilers, you need to set the suboption `procs` of the **`XLSMPOPTS`** environment variable. If you compiled your application with the GNU Compiler Collection (GCC), you need to set the **`GOMP_CPU_AFFINITY`** environment variable. You can check the value of the environment variables with the **`echo`** command. Example D-8 shows how to use the `threads_map` program if you want to run your application with 20 OpenMP threads. Use two threads for each core, five cores for each chip, two chips for each socket, only one socket, and bind threads to the second socket.

Example D-8 Use the threads_map application to generate an affinity string

```
$ export XLSMPOPTS=procs="`threads_map 2 5 2 1 80`"
$ export GOMP_CPU_AFFINITY="`threads_map 2 5 2 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
$ echo $GOMP_CPU_AFFINITY
80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
```

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.

- *Optimization and Programming Guide - XLC/C++ for Linux, V13.1,* SC27-4251-01
- *Performance Optimization and Tuning Techniques for IBM Processors, including IBM POWER8,* SG24-8171
- *IBM LoadLeveler to IBM Platform LSF Migration Guide,* REDP-5048
- *NVIDIA CUDA on IBM POWER8: Technical Overview, Software Installation, and Application,* REDP-5169
- *IBM Power System S824L Technical Overview and Introduction*, REDP-5139

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

Other publications

These publications are also relevant as further information sources:

- *Administering IBM Platform LSF*, SC22-5346
- *IBM Platform LSF Configuration Reference*, SC22-5350

Online resources

These websites are also relevant as further information sources:

- OpenPower

  http://openpowerfoundation.org
- Technical documentation for IBM Power System S824L

  http://www.ibm.com/common/ssi/index.wss
- Connect-IB Single and Dual QSFP+ Port PCI Express Gen3 x16 Adapter Card User Manual

  http://www.mellanox.com/page/products_dyn?product_family=142&mtag=connect_ib
- NVIDIA GPU Boost for Tesla K40 Passive and Active Boards - Application Note

  http://www.nvidia.com/object/tesla_product_literature.html

- Linux User Guide for IBM SDK, Java Technology Edition

  http://www.ibm.com/support/knowledgecenter/SSYKE2
- OpenPower

  http://openpowerfoundation.org

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 IBM POWER8

SG24-8263-00

ISBN 0738440930

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8263-00

ISBN 0738440930