Redbooks
ibm.com/redbooks

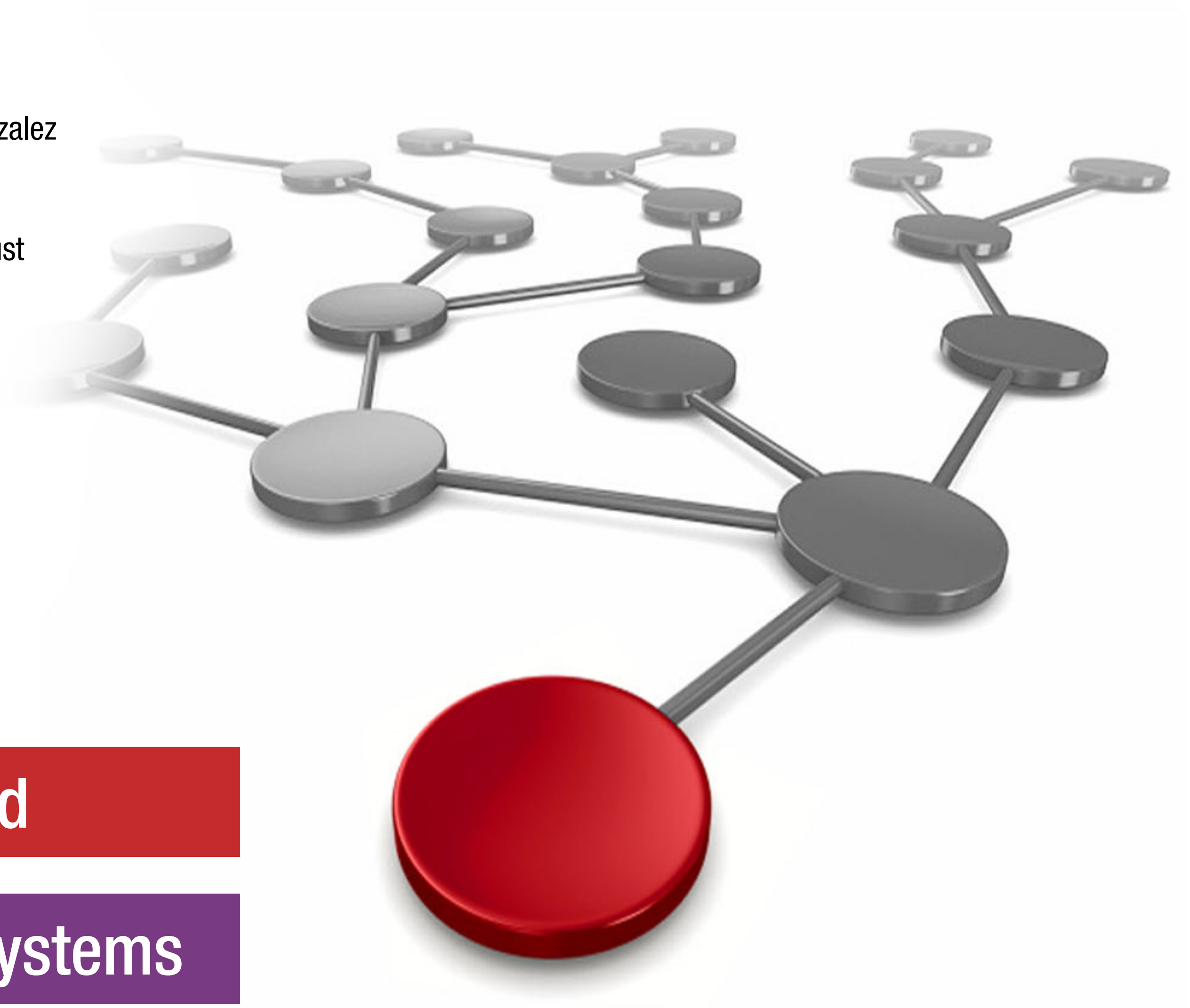
IBM High-Performance Computing Insights with IBM Power System AC922 Clustered Solution

Dino Quintero

Miguel Gomez Gonzalez

Ahmad Y Hussein

Jan-Frode Myklebust



Cloud

Power Systems

IBM

Redbooks

International Technical Support Organization

IBM High-Performance Computing Insights with IBM Power System AC922 Clustered Solution

May 2019

SG24-8422-00

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

First Edition (May 2019)

This edition applies to:

IBM Power System AC922 Firmware OP9-V2.0-2.14
Extreme Cloud Administration Toolkit (xCAT) V2.14.3,
Red Hat Enterprise Linux server 7.5 Alt (RHEL-ALT-7.5-20180315.0-Server-ppc64le-dvd1.iso)
NVIDIA Compute Unified Device Architecture (CUDA) V9.2 (cuda-repo-rhel7-9-2-local-9.2.148-1.ppc64le)
Mellanox MLNX_OFED_LINUX-4.3-4.0.5.1-rhel7.5alternate-ppc64le.iso
IBM XLC xlc-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
IBM XLF xlf-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
AT at12.0
IBM SMPI ibm_smpi-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
IBM Parallel Performance Toolkit ppt-2.4.0-2.tar.bz2
IBM IBM Engineering and Scientific Subroutine Library (IBM ESSL) essl-6.1.0-1-ppc64le
IBM Parallel Engineering and Scientific Subroutine Library (IBM Parallel ESSL) pessl-5.4.0-0-ppc64le
IBM Spectrum Scale 5.0.1.-2
The Portland Group, Inc. (PGI) pgilinux-2018-187-ppc64le.tar.gz
IBM Spectrum Load Sharing Facility (IBM Spectrum LSF) lsf-10.1-build_number.ppc64le_csm.bin
IBM Cluster Systems Management V1.3.0
IBM Burst Buffer V1.3.0

Contents

Notices

Trademarks

Redbooks (logo) ®	IBM Elastic Storage™	Power Systems™
AIX®	IBM Spectrum™	POWER8®
EnergyScale™	IBM Spectrum Scale™	POWER9™
FDPR®	LSF®	PowerLinux™
GPFS™	POWER®	PowerPC®
IBM®	Power Architecture®	Redbooks®

Preface

This IBM® Redbooks® publication documents and addresses topics to set up a complete infrastructure environment and tune the applications to use an IBM POWER9™ hardware architecture with the technical computing software stack.

This publication is driven by a CORAL project solution. It explores, tests, and documents how to implement an IBM High-Performance Computing (HPC) solution on a POWER9 processor-based system by using IBM technical innovations to help solve challenging scientific, technical, and business problems.

This book documents the HPC clustering solution with InfiniBand on IBM Power Systems™ AC922 8335-GTH and 8335-GTX servers with NVIDIA Tesla V100 SXM2 graphics processing units (GPUs) with NVLink, software components, and the IBM Spectrum™ Scale parallel file system.

This solution includes recommendations about the components that are used to provide a cohesive clustering environment that includes job scheduling, parallel application tools, scalable file systems, administration tools, and a high-speed interconnect.

This book is divided in three parts:

- "Planning" on page 1 focuses on the planners of the solution.
- "Deployment" on page 49 focuses on the administrators.
- "Application development" on page 159 focuses on the developers.

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 Solutions Enablement Project Leader and an IBM Level 3 Certified Senior IT Specialist with the International Technical Support Organization in Poughkeepsie, New York. Dino shares his technical computing passion and expertise by leading teams developing content in the areas of enterprise continuous availability, enterprise systems management, HPC, cloud computing, and analytics solutions. He also 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.

Miguel Gomez Gonzalez is an IBM Power Systems Integration engineer based in Mexico with over 11 years experience in Linux and IBM Power Systems technologies. He has extensive experience in implementing several IBM Linux and AIX® clustering solutions. His areas of expertise are Power Systems performance boost, virtualization, high availability, system administration, and test design. Miguel holds a Master in Information Technology Management from ITESM.

Ahmad Y Hussein is an IBM Systems Lab Services lead consultant who is based in Saudi Arabia and covers the Middle East and Africa. He is IBM certified Technical Expert with more than 14 years of experience in designing, leading, and implementing solutions for IBM POWER®, HPC, AIX, and Linux. Ahmad has worked for IBM since 2011. His areas of expertise include availability and performance optimization for IBM Power Systems and HPC solutions. Ahmad holds a M.Sc in Industrial Engineering, and a B.Sc in Electrical and Computing Engineering.

Jan-Frode Myklebust is an IBM Systems Lab Services consultant who is based in Norway and covers the Nordic countries. He has 20 years of experience in HPC, initially as a system administrator at one of Norway's national HPC sites, working with large nonuniform memory access (NUMA) systems spanning whole machine rooms and composed of racks of 32-CPU symmetric multiprocessor (SMP) machines to smaller dual-CPU Linux based clusters. His areas of expertise include IBM Spectrum Scale™ and Linux servers.

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

Wade Wallace
International Technical Support Organization, Austin Center

Robert Blackmore,John Dunham, Robin Hanrahan, Victor Hu, Joan McComb, Serban Maerean, Mark Perez, Fernando Pizzano, Cassandra Qui
IBM Poughkeepsie

Thomas Gooding
IBM Rochester

David Appelhans and Jaime H Moreno
IBM Yorktown Heights

Stanley Graves
IBM Dallas

Majid Ouassir
IBM France

Sean McCombe
IBM Oregon

Shelly Hu
IBM Virginia

Jose Ivan Bobadilla
IBM Mexico

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

Part 1

Planning

This part introduces the solution concept for high-performance computing (HPC), provides a description of the IBM Power System AC922 systems, describes the software stack that can be used in the solution, and provides details about the reference architecture.

The following chapters are included in this part:

- Introduction to IBM high-performance computing
- IBM Power System AC922 server for HPC overview
- Software stack
- Reference architecture

1

Introduction to IBM high-performance computing

The following topics are described in this chapter:

- Overview of HPC
- Reasons for implementing HPC on IBM POWER9 processor-based systems
- Overview of the POWER9 processor-based CORAL Project

1.1 Overview of HPC

What is high-performance computing (HPC)? One simple definition is:

"High-performance computing most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business".[1]

An HPC cluster is a combination of high-performance compute nodes, a low-latency interconnect fabric with high bandwidth, high-performance parallel storage, and system software, which addresses the most challenging requirements for HPC and high-performance data analytics (HPDA).

A typical HPC solution stack is composed of:

- Infrastructure
  - High-performing hardware servers
  - High-speed storage
  - High-speed interconnect
- Management software
- Scheduling software
- Software development environment
- Applications
- Health Insight solution

These solutions can be applied in a wide range of industries. For example, automotive, aerospace, and electronics companies that are focused on electronic design automation (EDA), computer-aided engineering (CAE), big data analytics, and artificial intelligence (AI) benefit greatly from highly optimized computing environments.

Details for these components are explained throughout this book.

1.2 Reasons for implementing HPC on IBM POWER9 processor-based systems

HPC environments have been evolving rapidly for the past few years and expanding to include diverse workloads like big data and AI.

Traditional HPC clusters cannot deliver adequate performance and scalability because of I/O bottlenecks and network latency when moving large data sets. The clusters slow down real-time insights.

The approach of the exascale era, where systems perform an exaFLOP (one billion billion calculations per second), requires innovation, such as POWER9 processor-based systems. IBM introduces high-performance systems with incredible performance and efficiency at affordable acquisition and maintenance prices.

[1] https://insidehpc.com/hpc-basic-training/what-is-hpc/

POWER9 processors were conceived to have new and unique features for technical computing to offer the following advantages:

- Help deliver the highest performance per core.
- Achieve energy efficiency within processor technologies.
- Deliver scalability.
- Provide reliability.

IBM and NVIDIA worked together to integrate IBM Power Systems server with NVIDIA graphics processing units (GPUs) to enable GPU-accelerated applications and workloads.

The computational capability that is provided by the combination of NVIDIA Tesla GPUs and IBM Power Systems servers enables workloads for scientific, technical, and HPC jobs to run on data center hardware. In most cases, these workloads run on supercomputing hardware. This computational capability is built on top of massively parallel and multithreaded cores with NVIDIA Tesla GPUs and IBM Power Architecture® processors, where processor-intensive operations are offloaded to GPUs and coupled with the system's high memory-hierarchy bandwidth and I/O throughput.

IBM also partnered with Mellanox to integrate Mellanox InfiniBand networking solutions into IBM solutions. This enhanced interconnection of different HPC components eliminated I/O bottlenecks.

Another advantage of implementing HPC in POWER9 processor-based systems is the rich infrastructure of software that is built around it to take advantage of its strength. The software stack that is contributed by IBM, OpenPower partners, and open source entities satisfy the needs of technical computing users, administrators, and developers.

In summary, the HPC solutions from IBM have the following attributes:

- They are developed specifically to support compute-intensive big data analytics and AI computing environments.
- They are powered by industry-leading hardware, POWER9 processors, NVIDIA GPUs, and Mellanox connectivity solutions.
- They offer a robust and integrated solution with ease of administration.

1.3 Overview of the POWER9 processor-based CORAL Project

Because IBM was leading the definition of the new POWER processor-based architectures within the OpenPOWER Foundation, IBM learned in 2013 about two contracts to build the next generations of supercomputers, which include the US Department of Energy's Collaboration of Oak Ridge National Labs (ORNL), and Lawrence Livermore National Labs (LLNL), which are collectively known as the CORAL program.

Both CORAL systems (Summit at ORNL and Sierra at LLNL) were the number one and two top-performing supercomputers on Top500.

For example, Summit has a hybrid architecture, and each node contains multiple POWER9 processors and NVIDIA Volta GPUs connected together by NVIDIA NVLink. Each node has over half a terabyte of coherent memory (high-bandwidth memory (HBM2) + DDR4) addressable by all processors and GPUs plus the non-volatile RAM that can be used as a burst buffer (BB) or as extended memory. To provide a high rate of I/O throughput, the nodes are connected in a non-blocking fat-tree by using a dual-rail Mellanox EDR InfiniBand interconnect.

By using Summit, researchers in all fields of science have unprecedented access to solve some of the world's most pressing challenges. For more information about this solution, see Summit.

CORAL was designed as a three interconnected pillars architecture:

- Computing cluster
- High-speed storage cluster
- Cluster-insights cluster

In this architecture:

1. One pillar does not affect the functioning of the other pillar, for example, shutting down or maintaining a pillar does not require other pillars to shut down.
2. The timeline requirements for booting computing clusters are met.
3. There is an effective central solution that handles all clusters event and provides real-time analysis and decisions.

Figure 1-1 shows the main pillars of the POWER9 processor-based CORAL supercomputer solutions.

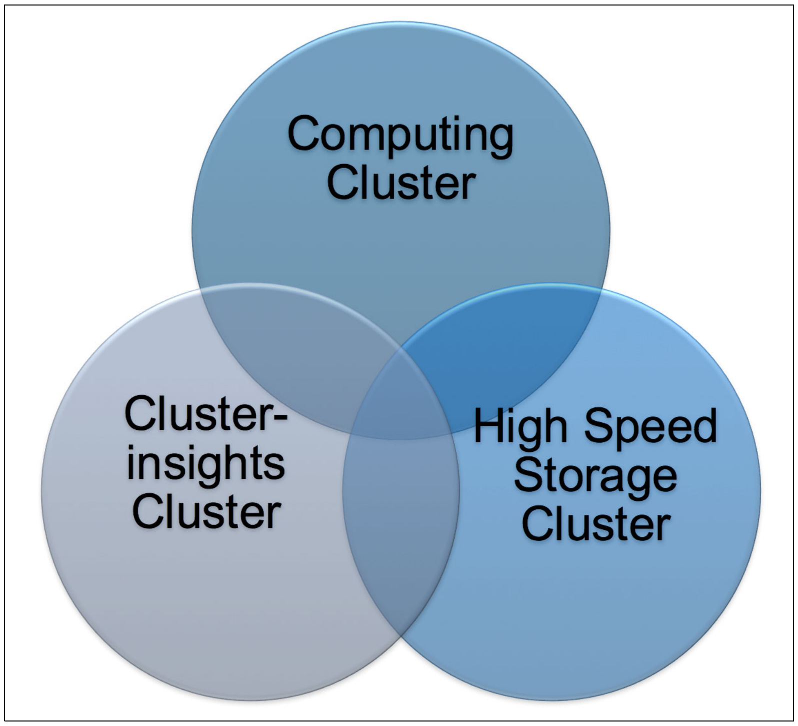


Figure 1-1 Pillars of the HPC cluster

Figure 1-2 shows the details of the components of the CORAL system.

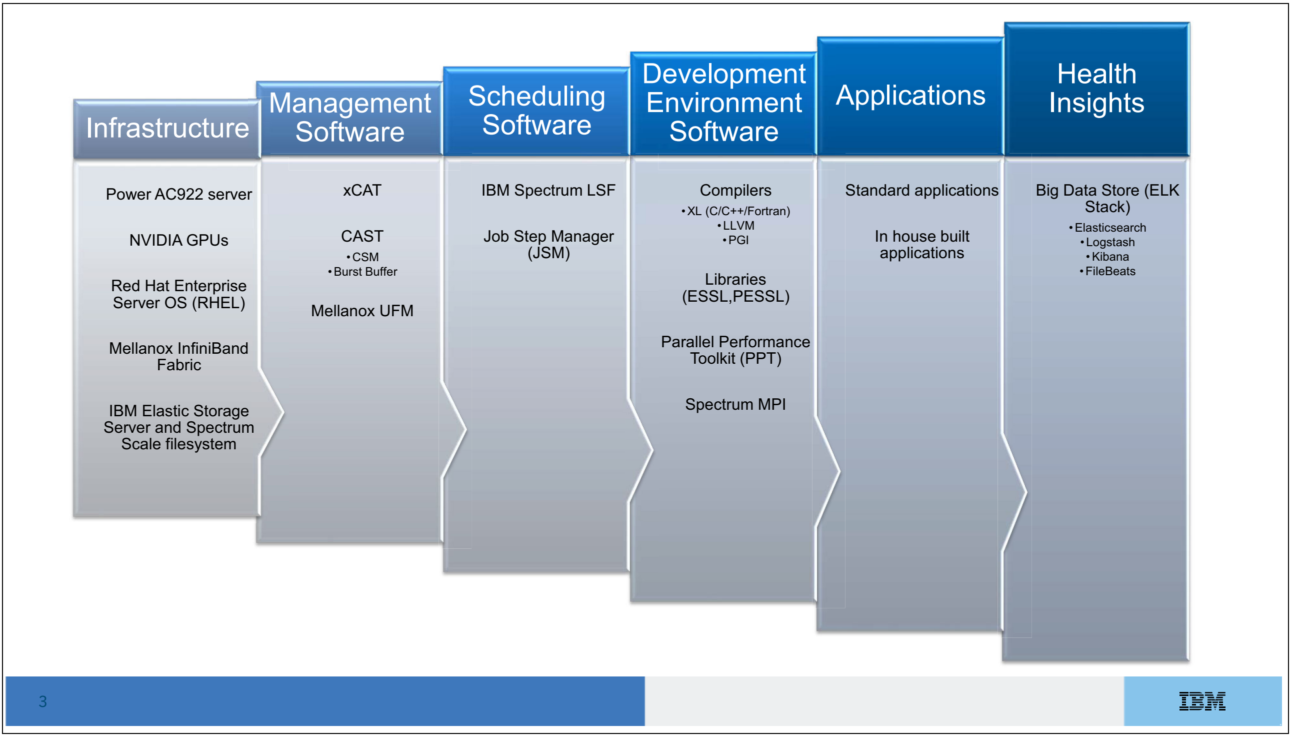


Figure 1-2 Building components of a CORAL HPC cluster

Important: This architecture provides the best performance and cluster utilization efficiency for jobs that require high parallelism and high processor and GPU resources.

Here are the major components that are used to provide a cohesive clustering environment that includes job scheduling, parallel application tools, scalable file systems, administration tools, and a high-speed interconnect:

- Red Hat Enterprise Linux 7.5 for POWER9 processor Little Endian
- Extreme Cluster and Cloud Administration Toolkit (xCAT)
- Cluster Administration and Storage Tools (CAST) (IBM Cluster Systems Management (CSM) and IBM Burst Buffer)
- Big data storage (BDS) (Elasticsearch, Logstash, and Kibana (ELK) stack-based)
- IBM Spectrum Load Sharing Facility (IBM Spectrum LSF®)
- IBM Spectrum Message Passing Interface (IBM Spectrum MPI)
- IBM Engineering and Scientific Subroutine Library (IBM ESSL) and IBM Parallel ESSL
- IBM Xpertise Library (XL) C/C++
- IBM XL Fortran
- LLVM compilers
- The Portland Group, Inc. (PGI) compilers
- IBM Parallel Performance Toolkit
- IBM Spectrum Scale
- IBM Elastic Storage™ Server

- Power AC922 server
- Peripheral Component Interconnect Express (PCIe) Gen3 1.6 TB NVMe flash adapters
- NVIDIA Compute Unified Device Architecture (CUDA) Toolkit
- NVIDIA Data Center GPU Manager (F)
- Tesla V100 SXM2 GPU
- Mellanox OpenFabrics Enterprise Distribution (OFED)
- Mellanox Unified Fabric Manager (UFM)
- Mellanox InfiniBand Switches
- Mellanox CX5 adapter

This book focuses on the computing cluster pillar and provides reference links for the other pillars.

The next chapters describe each of the components in terms of planning, deploying, maintaining, and developing.

2

IBM Power System AC922 server for HPC overview

This chapter introduces the IBM Power System AC922 (Power AC922) server for high-performance computing (HPC).

The following topics are described in this chapter:

- Power AC922 server for HPC
- Functional component description

For more information about IBM Systems hardware and software for HPC, see IBM Power Systems.

2.1 Power AC922 server for HPC

The Power AC922 server is the next generation of IBM POWER processor-based systems, which are designed for deep learning (DL) and artificial intelligence (AI), high-performance data analytics (HPDA), and HPC.

The system is co-designed by IBM and other OpenPOWER Foundation (NVIDIA, Mellanox, and others) members. Two of them are deployed at DOE/SC/Oak Ridge National Laboratory (Summit) and DOE/NNSA/LLNL (Sierra), and they are the most powerful supercomputers on the planet[1].

The system delivers the latest technologies that are available for HPC and improves the movement of data from memory to graphics processing units (GPUs) and vice versa, which enable faster and lower latency data processing. This massive computing capacity is packed into 2Us of rack space. There are special cooling systems to support the largest configurations: One is the air-cooled model 8335-GTH, and the other is the water-cooled model 8335-GTX.

2.2 Functional component description

Among the new technologies in the Power AC922 servers, the most significant are:

- Two IBM POWER9 processors with up to 40 cores for model 8335-GTH or 44 cores for model 8335-GTX, and improved buses
- Up to 2 TB of DDR4 memory with improved speed
- Up to six NVIDIA Tesla V100 (Volta) with NVLink GPUs, with 16 GB or 32 GB HMB2 memory
- Second-generation NVLink technology with 2x throughput compared to the first generation
- Four Peripheral Component Interconnect Express (PCIe) x16 4.0 low-profile (LP) slots. Three are Coherent Accelerator Processor Interface (CAPI)-enabled for future CAPI-enabled devices (a maximum of three CAPI devices can be used concurrently)
- Two 2.5-inch SATA drives for a maximum of 4 TB hard disk drive (HDD) or 7.68 TB of solid-state drive (SSD) storage
- Two integrated USB 3.0 ports
- Two hot-swap and redundant power supplies: (maximum 4-GPU configuration) 2200 W, and 200 - 240 and 277 V AC

Note: At the time of writing, the POWER9 processor-based family of servers are the only ones in the market with PCIe Gen 4 adapters, which have double the performance of PCIe Gen 3 adapters.

[1] https://www.top500.org/system/179397

2.2.1 Power AC922 models

The Power AC922 server is available as two models that can adapt to the different client needs in terms of infrastructure or processing power, as shown in Table 2-1.

Table 2-1 Power AC922 server models

Server model	POWER9 processors	Maximum memory	Maximum GPU cards	Cooling
8335-GTH	2	2 TB	4	Air-cooled
8335-GTX	2	2 TB	6	Water-cooled

Power AC922 8335-GTH model

This summary describes the standard features of the Power AC922 model 8335-GTH server:

- A 19-in. rack-mount (2U) chassis
- Two POWER processor modules:
  - 16-core 2.7 GHz (3.3 GHz turbo) processor module
  - 20-core 2.4 GHz (3.0 GHz turbo) processor module
  - Up to 2048 GB of 2666 MHz DDR4 error-correcting code (ECC) memory
- Two small form factor (SFF) bays for HDDs or SSDs that support:
  - Two 1 TB 7200 RPM NL SATA disk drives
  - Two 2 TB 7200 RPM NL SATA disk drives
  - Two 960 GB SATA SSDs
  - Two 1.92 TB SATA SSDs
  - Two 3.84 TB SATA SSDs
- Integrated SATA Controller
- PCIe Gen4 slots (4x):
  - Two PCIex16 LP CAPI-enabled slot
  - One PCIe x16 4.0 LP CAPI-enabled slot or one PCIe x8 shared 4.0 CAPI-enabled slot
  - One PCIe x4 4.0 LP slot
- NVIDIA Tesla V100 GPU options:
  - Zero, two, or four 16 GB SXM2 NVIDIA Tesla V100 GPUs with NVLink Air-Cooled
  - Zero, two, or four 32 GB SMX2 NVIDIA Tesla V100 GPUs with NVLink Server model POWER9 processors maximum
- Integrated features:
  - IBM EnergyScale™ technology
  - Hot-swap and redundant cooling
  - Two 1 Gb RJ45 ports
  - One front USB 3.0 port for general use
  - One rear USB 3.0 port for general use
  - One system port with RJ45 connector
- Two power supplies (Both are required.)

Power AC922 8335-GTX model

This summary describes the standard features of the Power AC922 model 8335-GTX server:

- 19-in. rack-mount (2U) chassis
- Two POWER9 processor modules:
  - 18-core 3.15 GHz (3.45 GHz turbo) processor module
  - 22-core 2.80 GHz (3.10 GHz turbo) processor module
  - Up to 2048 GB of 2666 MHz DDR4 ECC memory
- Two SFF bays for HDDs or SSDs that support:
  - Two 1 TB 7200 RPM NL SATA disk drives
  - Two 2 TB 7200 RPM NL SATA disk drives
  - Two 960 GB SATA SSDs
  - Two 1.92 TB SATA SSDs
  - Two 3.84 TB SATA SSDs
- Integrated SATA controller
- Four PCIe Gen4 slots:
  - Two PCIe x16 Gen4 LP CAPI-enabled slots
  - One PCIe x8 Shared Gen4 LP CAPI-enabled slot
  - One PCIe x4 Gen4 LP slot
- NVIDIA Tesla V100 GPU options:
  - Four or six 16 GB SXM2 NVIDIA Tesla V100 GPUs with NVLink Water-Cooled
  - Four or six 32 GB SMX2 NVIDIA Tesla V100 GPUs with NVLink Water-Cooled
- Integrated features:
  - IBM EnergyScale technology
  - Hot-swap and redundant cooling
  - Two 1 Gb RJ45 ports
  - One rear USB 3.0 port for general use
  - One system port with RJ45 connector
- Two power supplies (Both are required.)

2.2.2 POWER9 processor

The POWER 9 processor is a superscalar symmetric multiprocessor (SMP) that is designed and optimized for AI, HPC, and data-intensive workloads. The integrated NVLink technology provides enhanced memory, I/O connectivity, and improved processor to processor communication.

The Power AC922 server is a 2-socket system that can have 16 - 22 simultaneous multithreading (SMT)-4 cores.

The POWER9 processor is a single-chip-module-based (SCM) processor that is based on complementary metal–oxide–semiconductor (CMOS) 14-nm technology with 17 metal layers. It is optimized for cloud and data center applications. Within its 68.5 mm × 68.5 mm footprint, it has eight direct-attached memory channels for scale-out configurations. Each DDR4 channel supports up to 2666 Mbps for one DIMM per channel or 2400 Mbps for two DIMMs per channel. Two processors are tightly coupled through two 4-byte 16 Gbps elastic differential interfaces (EDIs) for SMP. There are 34 lanes of PCIe – PEC2 slots: One x16 lane mode or two x8 lanes (bifurcation), or one x8 lane and two x4 lanes.

2.2.3 Memory subsystem

The Power AC922 server is a 2-socket system that supports two POWER9 SCM processors. The server supports a maximum of 16 DDR4 RDIMMs slots in the main system board that is directly connected to the POWER9 processor.

Memory features equate to a single memory DIMM. All memory DIMMs must be populated, and mixing different memory feature codes is not supported.

Table 2-2 shows the available memory options for the Power AC922 server.

Table 2-2 Power AC922 server memory options

Feature code	Description	Minimum/Maximum
#EM60	8 GB DDR4 memory	16/16
#EM61	16 GB DDR4 memory	16/16
#EM63	32 GB DDR4 memory	16/16
#EM64	64 GB DDR memory	16/16
#EM65	128 GB DDR memory	16/16

The supported maximum memory is 2048 GB, which you can achieve by installing 16 memory DIMMs (#EM65). For the Power AC922 server (models 8335-GTH and 8335-GTX), the following requirements apply:

- All the memory DIMMs must be populated.
- Memory features cannot be mixed.

Memory bandwidth

The POWER9 processor has exceptional cache, memory, and interconnect bandwidths. Table 2-3 shows the maximum bandwidth estimates for a single core on the server.

Table 2-3 Bandwidth estimates for a single core on the server

Single core	8335-GTH		8335-GTX	
	3.3 GHz	20 cores @ 3.0 GHZ	18 cores @ 3.26 GHz	22 cores @ 3.07 GHz
L1 (data) cache	158.4 GBps	144 GBps	156.48 GBps	147.36 GBps
L2 cache	158.4 GBps	144 GBps	156.43 GBps	147.36 GBps
L3 cache	211.2 GBps	192 GBps	208.64 GBps	196.48 GBps

Table 2-4 shows the overall bandwidths for the entire Power AC922 server that is populated with two processor modules.

Table 2-4 Power AC922 bandwidth with two processor modules populated

Total bandwidth	**8335-GTH**		**8335-GTX**	
	16 cores @ 3.3 GHZ	**20 cores @ 3.0 GHZ**	**18 cores @ 3.26 GHz**	**22 cores @ 3.07 GHz**
L1 (data cache)	2534.4 GBps	2880 GBps	2816.64 GBps	3241.92 GBps
L2 cache	2534.4 GBps	2880 GBps	2816.64 GBps	3241.92 GBps
L3 cache	3379.2 GBps	3840 GBps	3755.52 GBps	4322.56 GBps
Total memory	280 GBps	280 GBps	280 GBps	280 GBps
SMP interconnect	64 GBps	64 GBps	64 GBps	64 GBps
PCIe interconnect	272 GBps	272 GBps	272 GBps	272 GBps

2.2.4 PCI adapters

The Power AC922 server has four PCIe Gen 4 adapter slots, which double the capacity of the PCIe Gen 3 slots, as shown in Figure 2-1. PCIe Gen 4 provides 32 GB unidirectional and 64 GB bidirectional bandwidth, and it is compatible with PCIe 3 modules.

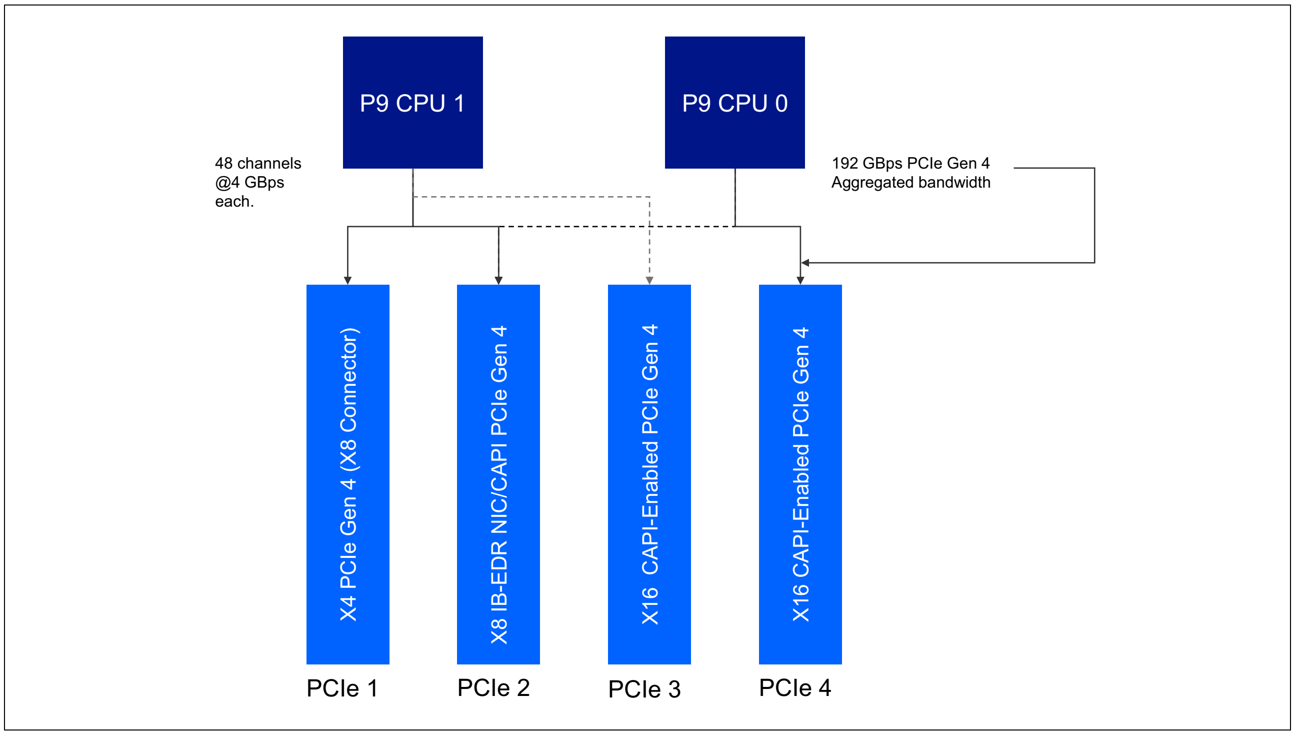

P9 CPU 1
P9 CPU 0
48 channels @4 GBps each.
192 GBps PCIe Gen 4 Aggregated bandwidth
X4 PCIe Gen 4 (X8 Connector)
X8 IB-EDR NIC/CAPI PCIe Gen 4
X16 CAPI-Enabled PCIe Gen 4
X16 CAPI-Enabled PCIe Gen 4
PCIe 1
PCIe 2
PCIe 3
PCIe 4

Figure 2-1 PCI characteristics

Table 2-5 describes the capabilities of each of the four slots.

Table 2-5 PCI Gen 4 slot capabilities

Slot	Description	Card-size	CAPI-capable
Slot 1	PCI Gen 4 x4	Half-height, half-length	No
Slot 2	PCI Gen 4 x8 shared	Half-height, half-length	Yes
Slot 3	PCI Gen 4 x16	Half-height half-length	Yes
Slot 4	PCI Gen 4 x16	Half-height half-length	Yes

The PCIe adapters that are shown in “Local area network adapters” and “CAPI-enabled adapters” on page 13 are required as part of the HPC configuration. Table 2-6 and Table 2-7 show different options that are supported on the Power AC922 servers.

Local area network adapters

Table 2-6 shows the supported adapters for the PCIe slots to connect the AC922 server to a local area network (LAN).

Table 2-6 LAN adapters

Feature code	Description	Maximum	Operating system (OS) support
#EC2R	PCIe3 LP 2-port 10 Gb NIC & ROCE SR/Cu adapter	3	Linux
#EC2T	PCIe3 LP 2-port 25/10 Gb NIC & ROCE SR/Cu adapter	3	Linux
#EL3Z	PCIe2 LP 2-port 10/1 GbE BaseT RJ45 adapter	4	Linux
#EL4M	PCIe2 LP 4-port 1 GbE adapter	4	Linux
#EN0T	PCIe2 LP 4-Port (10 Gb+1 GbE) SR+RJ45 adapter	4	Linux
#EN0V	PCIe2 LP 4-port (10 Gb+1 GbE) Copper SFP+RJ45 adapter	4	Linux

CAPI-enabled adapters

CAPI defines a coherent accelerator interface structure for attaching special processing devices to the POWER9 processor bus. Table 2-7 shows the available CAPI adapters.

Table 2-7 CAPI adapters

Feature code	Description	Maximum	OS support
#EC3L	PCIe3 LP 2-port 100 GbE (NIC & RoCE) QSFP28 adapter x16	2	Linux
#EC62	PCIe4 LP 1-port 100 Gb EDR InfiniBand CAPI adapter	3	Linux
#EC64	PCIe4 LP 2-port 100 Gb EDR InfiniBand CAPI adapter	3	Linux

Compute-intensive accelerators

Compute-intensive accelerators are GPUs that are developed by NVIDIA to off load processor-intensive operations to a GPU accelerator and boost performance. The Power AC922 server can be configured with GPUs that are air-cooled or water-cooled based on the model.

NVIDIA Tesla V100 is the most advanced data center GPU built to accelerate AI, HPC, and graphics. Powered by NVIDIA Volta, the current GPU architecture, Tesla V100 offers the performance of 100 CPUs in a single GPU.

The Tesla V100 includes the following key features:

- Volta architecture

  By pairing Compute Unified Device Architecture (CUDA) cores and Tensor cores within a unified architecture, a single server with Tesla V100 GPUs can replace hundreds of commodity CPU servers for traditional HPC and DL.

- Tensor core

  Equipped with 640 Tensor cores, Tesla V100 delivers 125 teraflops (TFLOPS) of DL performance, that is, 12 Tensor TFLOPS for DL training, and 6 Tensor TFLOPS for DL inference, compared to NVIDIA Pascal GPUs.

- Next generation NVLink

  NVIDIA NVLink in Tesla V100 delivers 2x higher throughput compared to the previous generation. Up to eight Tesla V100 accelerators can be interconnected at up to 300 GBps to unleash the highest application performance possible on a single server.

- Maximum efficiency mode

  The new *maximum efficiency mode* enables data centers to achieve up to 40% higher compute capacity per rack within the existing power budget. In this mode, Tesla V100 runs at peak processing efficiency, providing up to 80% of the performance at half the power consumption.

- High-bandwidth memory (HBM2)

  With a combination of an improved raw bandwidth of 900 GBps and higher DRAM usage efficiency at 95%, Tesla V100 delivers 1.5x higher memory bandwidth over Pascal GPUs, as measured on STREAM.

- Programmability

  The Tesla V100 is designed to simplify programmability. Its new independent thread scheduling enables finer-grain synchronization and improves GPU usage by sharing resources among small jobs.

The Tesla V100 delivers exceptional performance for the most demanding compute applications. It delivers the following performance benefits:

- 7.8 TFLOPS of double-precision floating point (FP64) performance
- 15.7 TFLOPS of single-precision floating point (FP32) performance
- 25 Tensor TFLOPS of mixed-precision

Table 2-8 lists the compute-intensive accelerators for the 8335-GTH air-cooled version of the Power AC922 server.

Table 2-8 Power AC922 8335-GTH model air-cooled accelerators

Feature code	Description	Maximum	OS support
#EC4J	NVIDIA Tesla V100 GPU with NVLink Air-Cooled (16 GB)	4	Linux
#EC4L	NVIDIA Tesla V100 GPU with NVLink Air-Cooled (32 GB)	4	Linux

Table 2-9 lists the compute-intensive accelerators for the 8335-GTX water-cooled version of the Power AC922 server.

Table 2-9 Power AC922 8335-GTX model water-cooled accelerators

Feature code	Description	Maximum	OS support
#EC4H	NVIDIA Tesla V100 GPU with NVLink Water-Cooled (16 GB)	6	Linux
#EC4K	NVIDIA Tesla V100 GPU with NVLink Water-Cooled (32 GB)	6	Linux

Note: The Power AC922 model 8335-GTX server is a water-cooled server and cannot be configured without GPUs. The minimum order is four GPUs.

2.2.5 IBM CAPI2

IBM CAPI2 is the evolution of CAPI that defines a coherent accelerator interface structure for attaching special processing devices to the POWER9 processor bus. As with the original CAPI, CAPI2 can attach accelerators that have coherent shared memory access with the processors in the server and share full virtual address translation with these processors by using standard PCIe Gen4 buses with twice the bandwidth compared to the previous generation.

Applications can have customized functions in Field Programmable Gate Arrays (FPGAs) and queue work requests directly in shared memory queues to the FPGA. Applications can also have customized functions by using the same effective addresses (pointers) that they use for any threads running on a host processor. From a practical perspective, CAPI2 enables a specialized hardware accelerator to be seen as an extra processor in the system with access to the main system memory and coherent communication with other processors in the system, as shown in Figure 2-2.

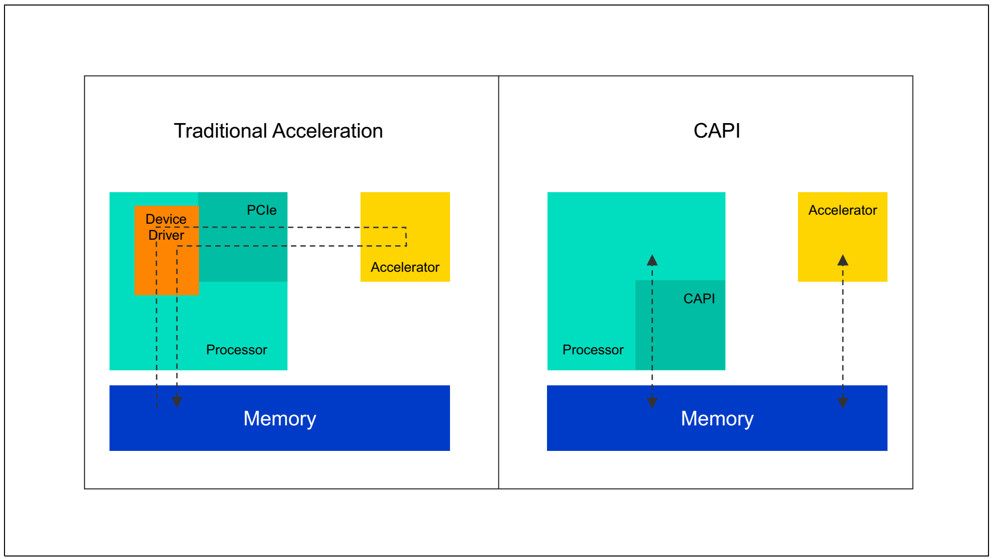


Figure 2-2   CAPI2

The benefits of using CAPI2 include access to shared memory blocks directly from the accelerator, performing memory transfers directly between the accelerator and processor cache, and reducing the code path length between the adapter and the processors. This reduction in the code path length might occur because the adapter is not operating as a traditional I/O device, and there is no device driver layer to perform processing. CAPI2 also presents a simpler programming model.

CAPI2 implements the POWER Service Layer (PSL), which provides address translation and system memory cache for the accelerator functions. The custom processors on the system board, which consist of an FPGA or an ASIC, use this layer to access shared memory regions and cache areas as though they were a processor in the system. This ability enhances the performance of the data access for the device and simplifies the programming effort to use the device. Instead of treating the hardware accelerator as an I/O device, it is treated as a processor, which eliminates the requirement of a device driver to perform communication. It also eliminates the need for direct memory access (DMA) that requires system calls to the OS kernel. By removing these layers, the data transfer operation requires fewer clock cycles in the processor, improving the I/O performance.

The implementation of CAPI2 on the POWER9 processor enables hardware companies to develop solutions for specific application demands. Companies use the performance of the POWER9 processor for general applications and the custom acceleration of specific functions by using a hardware accelerator with a simplified programming model and efficient communication with the processor and memory resources.

For a list of supported CAPI2 adapters, see 2.2.4, “PCI adapters” on page 12.

2.2.6 NVLink 2.0

NVLink 2.0 is the NVIDIA new generation high-speed interconnect technology for GPU-accelerated computing. Supported on SXM2-based Tesla V100 accelerator system boards, NVLink increases performance for both GPU-to-GPU communications and for GPU access to system memory.

Support for the GPU instruction set architecture (ISA) enables programs running on NVLink-connected GPUs to run directly on data in the memory of another GPU and on local memory. GPUs can also perform atomic memory operations on remote GPU memory addresses, enabling much tighter data sharing and improved application scaling.

NVLink 2.0 uses the NVIDIA High-Speed Signaling (NVHS) interconnect. NVHS transmits data over a link that is called NVLink Brick that connects two processors (GPU-to-GPU or GPU-to-CPU). A single NVLink Brick supports up to 50 Gbps of bidirectional bandwidth between the end points. Multiple links can be combined to form *Gangs* for even higher-bandwidth connectivity between processors. The NVLink implementation in Tesla V100 supports up to six links so that a Gang has an aggregate maximum theoretical bidirectional bandwidth of 300 GBps.

Although a traditional NVLink implementation primarily focuses on interconnecting multiple NVIDIA Tesla V100 GPUs, under POWER9 it also connects Tesla V100 GPUs with IBM POWER9 processors to enable direct system memory access and provide GPUs with an extended memory that is orders of magnitude larger than the internal 16 GB memory.

On a Power Systems implementation, NVLink Bricks are always combined to provide the highest bandwidth possible.

Figure 2-3 compares the bandwidth of a POWER9 processor that is connected to two GPUs and three GPUs.

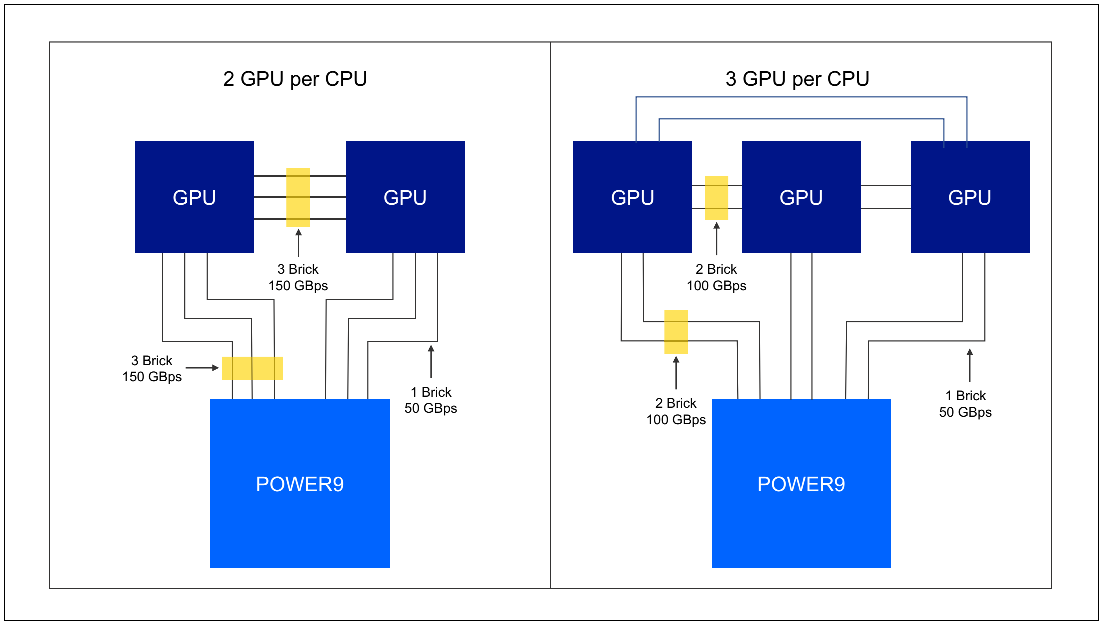


Figure 2-3 Bandwidth comparison with two GPUs and three GPUs

The added value of GPUs that use the NVLink technology is the flexibility in using the Power AC922 server; in addition to traditional HPC simulation, AI and big data workloads are possible by leveraging the current hardware and software stack.

Figure 2-4 shows the added throughput value.

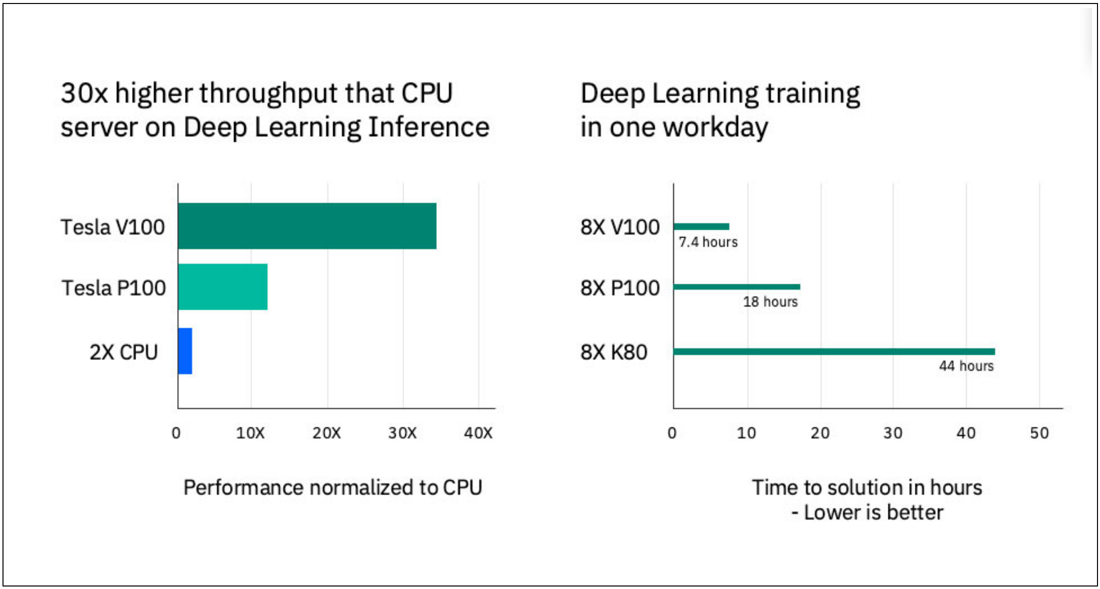


Figure 2-4 Added throughput value

2.2.7 Baseboard management controller

The Power AC922 family of servers features a baseboard management controller (BMC) for out-of-band system management. BMC is an efficient way to do monitoring, management, control, and maintenance. It offers the user the possibility of collecting system event logs through sensors. Users have the possibility to interact through the command-line interface (CLI) (Intelligent Platform Management Interface (IPMI) or OpenBMC) or download the OpenBMC GUI tool.

For more information about BMC, see Managing OpenBMC Systems.

Among the BMC functions are:

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

3

Software stack

This chapter describes the software stack that is used in the implementation of an IBM high-performance computing (HPC) solution with the IBM Power System AC922 (Power AC922) (8335-GTH and 8335-GTX) servers.

The following topics are described in this chapter:

- Red Hat Enterprise Linux
- Device drivers
- Development environment software
- Workload management
- Cluster management software
- IBM Storage and file systems

3.1 Red Hat Enterprise Linux

Red Hat Enterprise Linux is one of the world's leading enterprise Linux platforms. It runs on highly scalable, multi-core systems that support the most demanding workloads. Collaboration between Red Hat and engineers from major hardware vendors ensures that the operating system (OS) uses the newest hardware innovations that are available in processor design, system architecture, and device drivers to improve performance and reduce power utilization.

For more information, see Red Hat Customer Portal.

3.2 Device drivers

This section provides information about the device drivers that are used in this solution.

3.2.1 Mellanox OpenFabrics Enterprise Distribution

This is a single Virtual Protocol Interconnect (VPI) software stack that operates across all Mellanox network adapter solutions. The stack supports specific uplinks and Mellanox InfiniBand Host Channel Adapters (HCAs) to servers.

Mellanox OpenFabrics Enterprise Distribution (OFED) contains many important components, for example:

- Mellanox HCA drivers:
  - Includes both InfiniBand and Ethernet drivers.
- Upper layer protocols (ULPs):
  - For example, IP over InfiniBand.
- Utilities:
  - Diagnostic tools.
  - Performance tests.
- OpenSM: InfiniBand Subnet Manager
- Mellanox Firmware Tools (MFT)
- Message Passing Interface (MPI):
  - The OpenMPI stack supports the InfiniBand, RoCE, and Ethernet interfaces.
  - MPI benchmark tests.

Mellanox OFED supports adaptive routing (AR) management. The AR manager supports the following advanced InfiniBand features:

- Adaptive Routing (AR)
- Fast Link Fault Recovery and Notification (FLFR)

AR enables the switch to select the output port based on the port's load. AR supports two routing modes:

- Free AR: No constraints on the output port selection.
- Bounded AR: The switch does not change the output port during the same transmission burst. This mode minimizes the appearance of out-of-order packets.

The AR manager enables and configures AR mechanism on fabric switches. It scans all the fabric switches, identifies which switches support AR, and configures the AR functions on these switches.

FLFR enables the switch to select the alternative output port if the output port that is provided in the Linear Forwarding Table is not in the Armed or Active state. This mode enables the fastest traffic recovery in case of port failures or disconnects without the intervention of the Subnet Manager.

FLFN enables the switch to report to neighbor switches an alternative output port for the traffic to a specific destination.

The AR Manager is a Subnet Manager plug-in and is installed as a part of the Mellanox OFED installation, and includes AR and FLFR features.

Mellanox OFED also provides support to storage protocols that require high throughput; low latency; and low CPU utilization, for example, NVM Express over Fabrics (NVME-oF).

NVME-oF is a technology specification for networking storage that enables NVMe message-based commands to transfer data between a host computer and a target solid-state disk (SSD) or system over a network such as Ethernet, Fibre Channel, and InfiniBand.

Tunneling NVMe commands through an Remote Direct Memory Access (RDMA) fabric provides high throughput and a low latency. It is an alternative to the SCSi-based storage networking protocols.

NVME-oF Target Offload is an implementation of the new NVME-oF standard target (server) side in hardware. Starting with ConnectX-5 family cards, all regular I/O requests can be processed by the HCA, with the HCA sending I/O requests directly to a real NVMe PCI device by using peer-to-peer PCI communications, which means that except for connection management and error flows, no CPU utilization is observed during NVME-oF traffic.

For more information, see *Mellanox OFED for Linux User Manual*.

Mellanox OFED is required for Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) to function. SHARP improves the performance of MPI operations by offloading collective operations from the processor to the switch network and eliminating the need to send data multiple times between end points.

The SHARP innovative approach decreases the amount of data traversing the network as aggregation nodes are reached, and dramatically reduces the MPI operations time. Implementing collective communication algorithms in the network also has extra benefits, such as freeing valuable CPU resources for computation rather than using them to process communication.

For deployment and configuration details of SHARP, see Mellanox Scalable Hierarchical Aggregator and Reduction Protocol.

3.2.2 NVIDIA CUDA

Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model that was invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

CUDA was developed with two design goals in mind:

- Provide a small set of extensions to standard programming languages like C that enable a straightforward implementation of parallel algorithms. With CUDA C/C++, programmers can focus on the task of parallelization of the algorithms rather than spending time on their implementation.
- Support heterogeneous computation where applications use both the CPU and GPU. Serial portions of applications are run on the CPU, and parallel portions are offloaded to the GPU. As such, CUDA can be incrementally applied to existing applications. The CPU and GPU are treated as separate devices that have their own memory spaces. This configuration also allows simultaneous computation on the CPU and GPU without contention for memory resources

At the time of writing, if you want to use CUDA (10) on a GPU (Tesla GV100) -enabled POWER9 processor-based system, install the following software:

- A supported version of Linux with a gcc compiler and toolchain:
  - Red Hat Enterprise Linux 7.5 IBM Power Little Endian, Kernel 4.14.0, GNU Compiler Collection (GCC) 4.8.5, GLIBC 2.17, The Portland Group, Inc. (PGI) 18.x, XCL 13.1.x, 16.1.x, and “Clang” 6.0.0
  - Ubuntu 18.04.1, Kernel 4.15.0, GCC 7.3.0, GLIBC 2.27, PGI 18.x, XCL 13.1.x, 16.1.x, and “Clang” 6.0.0
- NVIDIA CUDA Toolkit

For Linux, the NVIDIA CUDA Toolkit major components are:

- Compilers: The CUDA-C and CUDA-C++ compilers.
- Development tools:
  - IDEs: **`nsight`**
  - Debuggers: **`cuda-memcheck`**, and **`cuda-gdb`**
  - Profilers: **`nvprof`** and **`nvvp`**
  - Utilities: **`cuobjdump`**, **`nvdisasm`**, and **`gpu-library-advisor`**
- Scientific and utility libraries. For more information, see Toolkit release notes.
- CUDA driver.
- CUDA samples.
- Documentation.
- CUDA-GNU Debugger (GDB) sources.

Note: The NVIDIA driver is installed as part of the CUDA Toolkit installation. This driver is for development purposes and is *not* recommended for use in production with Tesla GPUs. For running CUDA applications in production with Tesla GPUs, download the latest driver for Tesla GPUs from NVIDIA driver downloads.

For more information about CUDA; NVIDIA GPUs; and documentation and installation guides, see NVIDIA.

3.3 Development environment software

This section describes the development environment software for the high-performance cluster solution.

3.3.1 IBM XL compilers

IBM Xpertise Library (XL) C/C++ and IBM XL Fortran compilers are advanced, high-performance compilers that can be used for developing complex, computationally intensive programs, including interlanguage calls with C and Fortran programs. These compilers are derived from a common code base that shares compiler function and optimization technologies for various platforms and programming languages. Programming environments include selected Linux distributions, IBM AIX, and other platforms. The common code base, along with compliance with international programming language standards, helps support consistent compiler performance and ease of code portability across multiple OSs and hardware platforms.

IBM XL compilers help you with:

- Debugging your applications
- Optimizing and tuning application performance
- Selecting language levels and extensions for compatibility with nonstandard features and behaviors that are supported by other Fortran compilers
- Performing many other common tasks that otherwise require changing the source code

Releases of IBM XL compilers for Linux provide full exploitation of the POWER9 processor-based technology.

For more information about IBM XL Fortran, see IBM Knowledge Center.

For more information about IBM XL C/C++, see IBM Knowledge Center.

3.3.2 LLVM

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. The name LLVM itself is not an acronym; it is the full name of the project.

LLVM has many subprojects, which are mentioned at the LLVM website, but one primary subproject is *Clang*.

Clang

This is an LLVM native C, C++, and Objective-C compiler. Clang delivers fast compiles (for example, about 3x faster than GCC when compiling Objective-C code in a debug configuration) and useful error and warning messages, and provides a platform for building source- level tools.

The Clang Static Analyzer is a tool that automatically finds bugs in your code, and is a great example of the sort of tool that can be built by using the Clang front end as a library to parse C and C++ code.

3.3.3 IBM Engineering and Scientific Subroutine Library

IBM Engineering and Scientific Subroutine Library (IBM ESSL) is a state-of-the-art collection of high-performance subroutines that provide a wide range of mathematical functions for many different scientific and engineering applications.

IBM ESSL is tuned for outstanding performance on selected clusters. It is supported on IBM POWER8® and POWER9 processor-based servers and can be used with Fortran, C, and C++ programs operating under the Linux OS.

All IBM ESSL mathematical subroutines are performance-optimized so that utilities can perform general-purpose functions. They are also optimized for nine computational areas, which are:

- Linear algebra subprograms
- Matrix operations
- Linear algebraic equations
- Eigensystem analysis
- Fourier transforms, convolutions and correlations, and related computations
- Sorting and searching
- Interpolation
- Numerical quadrature
- Random number generation

IBM ESSL has the following usability features:

- It has an easy-to-use call interface.
- If your existing application programs use the serial libraries, you only need to relink your program to take advantage of the increased performance of the IBM ESSL Symmetric Multiprocessor (SMP) Libraries.
- It has informative error-handling capabilities, enabling you to calculate auxiliary storage sizes and transform lengths.
- It has online documentation that you can view by using an HTML document browser that you can use with IBM ESSL.

Note: To obtain the preferred performance from POWER9 processor-based servers, most HPC applications that use matrix multiplication extensively must be run in simultaneous multithreading (SMT)-4 mode with four threads per core on POWER9 processor-based SMT-4 servers and in SMT-8 mode with eight threads per core on POWER9 processor-based SMT-8 servers.

For example, in most cases the performance of double precision general matrix multiplication (DGEMM) that is obtained form SMT-4 mode that uses the IBM ESSL SMP Library with four threads that are bound to each core is better than the performance that is obtained by using one thread that is bound to each core or by running the applications in ST mode or SMT-2 mode.

Here is an example environment for a 44-core system:

```
export OMP_PROC_BIND=TRUE
export OMP_WAIT_POLICY=active
export OMP_NUM_THREADS=176
export OMP_PLACES={0:176}
```

To obtain the preferred performance for the IBM POWER8 processor-based servers, most HPC applications that use matrix multiplication extensively must use one thread that is bound to each core.

For more information about IBM ESSL, see IBM Knowledge Center.

3.3.4 IBM Parallel Engineering and Scientific Subroutine Library

IBM Parallel Engineering and Scientific Subroutine Library (IBM Parallel ESSL) is a scalable mathematical subroutine library that supports parallel processing applications on clusters of processor nodes that are connected by a high-performance switch. IBM Parallel ESSL supports the Single Program Multiple Data (SPMD) programming model that uses the MPI library.

IBM Parallel ESSL provides subroutines in the following computational areas:

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- Level 3 PBLAS
- Linear algebraic equations
- Eigensystem analysis and singular value analysis
- Fourier transforms
- Random number generation

For communication, Parallel IBM ESSL includes the Basic Linear Algebra Communications Subprograms (BLACS), which use MPI. For computations, IBM Parallel ESSL uses the IBM ESSL subroutines.

The IBM Parallel ESSL subroutines can be called from 64-bit–environment application programs that are written in Fortran, C, and C++.

IBM Parallel ESSL best performs on clusters of Power AC922 servers that run with or without Tesla V100 with NVLink GPUs and dual-port InfiniBand EDR (ConnectX-5 and Peripheral Component Interconnect Express (PCIe) Gen4) adapters that are interconnected by using EDR InfiniBand switches.

IBM Parallel ESSL supports bare-metal installations of Red Hat Enterprise Linux 7.5 for Power Systems Little Endian.

For more information about IBM Parallel ESSL, see IBM Knowledge Center.

3.3.5 IBM Spectrum MPI

IBM Spectrum Message Passing Interface (IBM Spectrum MPI) is a high-performance, production-quality implementation of the MPI that supports the broadest range of industry standard platforms; interconnects; and OSs to help ensure that parallel applications can run on any platform with accelerated performance. IBM Spectrum MPI is one of the standards for developing scalable and parallel applications.

At the time of writing. IBM Spectrum MPI V10.2 was announced. It is based on OpenMPI V3.0 and implements the full MPI V3.1 standard. IBM Spectrum MPI incorporates advanced CPU affinity features, dynamic selection of network interface libraries, superior workload manager (WLM) integrations, improved performance, and improved RDMA-capable Parallel Active Messaging Interface (PAMI) by using Mellanox OFED on POWER9 processor-based servers in Little Endian mode.

IBM Spectrum MPI also offers an improved collective MPI library that supports the seamless use of GPU memory buffers for the application developer. The library provides advanced logic to select the fastest algorithm of many implementations for each MPI collective operation.

IBM Spectrum MPI supports the following GPU-acceleration technologies:

- NVIDIA GPU Direct RDMA on POWER9 processor-based servers
- CUDA-aware MPI on IBM POWER8 and POWER9 processor-based servers

The IBM Spectrum MPI components are:

- Message passing and collective communication application programming interface (API) subroutine libraries

  Libraries that contain subroutines that help application developers parallelize their code.

- Optimized Collectives Library (libcollectives)

  An optimized collectives library with both one-sided and active messages collectives.

- Parallel Active Messaging Interface

  A messaging API that supports point-to-point communications.

- IBM Spectrum MPI examples

A set of example programs are included in the product package.

IBM Spectrum MPI is a member of the IBM Spectrum Computing family. For more information, see IBM Spectrum Computing.

To access the most recent IBM Spectrum MPI documentation, IBM Knowledge Center.

3.3.6 IBM Parallel Performance Toolkit

IBM Parallel Performance Toolkit is a suite of performance-related tools and libraries that help application tuning. This toolkit is an integrated environment for performance analysis of sequential and parallel applications that uses the MPI and OpenMPI paradigms.

The IBM Parallel Performance Toolkit run time is the server (back-end) component that provides the performance analysis tools. These tools perform the following functions:

- Provide access to hardware performance counters for performing low-level analysis of an application, including analyzing cache usage and floating-point performance.
- Profile and trace an MPI application for analyzing MPI communication patterns and performance problems.
- Profile an application for analyzing performance problems and to help you determine whether an application correctly structures its processing for best performance.
- Profile application I/O for analyzing an application's I/O patterns and whether you can improve the application's I/O performance.
- Profile and trace an Open Multi-Processing (OpenMP) application for analyzing OpenMP thread and OpenMP synchronization behavior to improve OpenMP performance.
- Profile an application's execution for identifying hotspots in the application, and for locating relationships between functions in your application to help you better understand the application's performance.

The IBM Parallel Performance Toolkit provides three types of interfaces:

- An API and shared libraries that are intended to be used by users to insert performance analysis calls into their applications.
- A command-line interface (CLI) that is intended to be used on the login nodes to instrument the user application (**`hpctInst`**).
- A GUI that is intended to be used on a client machine (desktop or notebook) to instrument and run the user application and visualize the trace files that are produced by back-end instrumentation components (**`hpctView`** and the plug-in for Eclipse Parallel Tools Platform (Eclipse PTP)).

With the IBM Parallel Performance Toolkit interfaces, users can do the following tasks:

- Collect performance data from an instrumented application by setting environment variables for the specific analysis tool and then starting the application from the command line.
- Modify the application to add calls to the hardware performance counter tool, rebuild your application linking with the hardware performance counter library, and then run the application to get hardware performance counter data.
- Link the application directly with the MPI profiling and trace library and then run the application to get MPI trace and profiling data.
- Link the application with the I/O profiling and trace library to get I/O profiling and trace information.
- Use a preinstalled OpenMP instrumentation library to get OpenMP profiling and trace information.

For more information about IBM Parallel Performance Toolkit, see IBM Knowledge Center.

3.4 Workload management

This section describes the workload management tools.

3.4.1 IBM Spectrum Load Sharing Facility

This is a powerful workload management platform for demanding and distributed HPC environments. IBM Spectrum Load Sharing Facility (IBM Spectrum LSF) provides a comprehensive set of intelligent and policy-driven scheduling features that you can use to use all of your compute infrastructure resources and ensure optimal application performance.

With IBM Spectrum LSF, you can run direct data staging jobs, which use a burst buffer (for example, an IBM Cluster Administration and Storage Tools (CAST) IBM Burst Buffer (BB)) instead of the cache to stage in and stage out data for data jobs.

Important: A special version of IBM Spectrum LSF is used for the HPC type of clusters in this book, and it is different from the typical IBM Spectrum LSF product and IBM Spectrum LSF suite binary files. It is an RPM installation that provides IBM Spectrum LSF functions that are integrated with IBM Cluster Systems Management (CSM) APIs. Therefore, the typical IBM Spectrum LSF daemons do not run on compute nodes.

3.5 Cluster management software

This section describes the cluster management software that is in the high-performance cluster solution.

3.5.1 Extreme Cluster and Cloud Administration Toolkit

Extreme Cluster and Cloud Administration Toolkit (xCAT) offers a complete systems management solution to manage easily many servers for any type of technical computing workload. xCAT is known for exceptional scaling, a wide variety of supported hardware and OSs, virtualization platforms, and complete setup capabilities.

With xCAT, you can do the following tasks:

- Discover the hardware servers.
- Do remote system management.
- Provision OSs on physical (bare metal) or virtual machines (VMs).
- Provision machines in diskful (stateful) and diskless (stateless) states.
- Install and configure user applications.
- Manage parallel systems.

For more information about xCAT, see xCAT or xCAT documentation.

3.5.2 Cluster Administration and Storage Tools

CAST can enhance the system management of cluster-wide resources. It consists of the open source tools CSM and BB.

Cluster Systems Management

CSM is a systems management tool that is designed for simple and low-cost management of distributed and clustered IBM Power Systems servers.

CSM provides an integrated view of a large cluster and includes discovery and management of system resources; database integration; support of job launches; diagnostics; reliability, availability, and serviceability (RAS) events and actions; and health check.

CSM consists of several major components:

- An SQL database that is shared with xCAT that contains RAS events, job and job step information, system configuration and inventory information, and system and node health information.
- CSM daemons that run on the different node types and the communication infrastructure between these daemons.
- A set of commands that support the RAS query, diagnostics, and system query functions.
- A set of APIs that support the WLM, job step launcher, and BB API.

Integral to CSM is big data storage (BDS), which is required to store the environmental data and provides the capability to correlate RAS events, system environmental data, and job data.

Burst buffer

A *BB* is a cost-effective mechanism that can improve I/O performance for many HPC applications without requiring intermediary hardware. A BB provides a fast storage tier between compute nodes and the traditional parallel file system so that overlapping jobs can stage in and stage out data for various checkpoint and restart, scratch volume, and extended memory I/O workloads.

Here is a summary of the services that are provided by the BB:

- Support for the creation and modifications of job-specific solid-state drive (SSD) file systems.
- Support for moving data between SSD file systems and permanent IBM Spectrum Scale storage.
- A command-line tool that supports the manipulation of BB file systems on all or a subset of the compute nodes in a job that is run from the launch or login node. These commands include normal file system operations, data movement, and control actions.
- An API that can be used by programs that do not have the privileges to control and monitor data movement between the SSD and IBM Spectrum Scale storage. It is intended to support operations like programmatic checkpoint and restart.
- A set of scripts that support the BB stage-in and stage-out phases of an IBM Spectrum LSF job.
- Support for a virtual parallel SSD file system by using IBM Burst Buffer Shared Checkpoint File System (BSCFS).

3.5.3 Mellanox Unified Fabric Manager

Mellanox Unified Fabric Manager (UFM) is a powerful platform for managing large scale-out InfiniBand and Ethernet computing environments. UFM enables data center operators to monitor, efficiently provision, and operate the modern data center fabric. UFM eliminates the complexity of fabric management, provides deep visibility into fabric health and traffic, and optimizes fabric performance.

UFM provides the following features:

- Eliminates fabric congestion and hot spots.
- Simplifies the management of large or complex environments.
- Automates service provisioning on the fabric layer.
- Seamlessly manages workload migration scenarios.
- Tunes your fabric interconnect for highest performance.
- Provides preventive maintenance and “soft degradation” alerts.
- Quickly troubleshoots any connectivity problem.
- Integrates and streamlines fabric information into your IT systems.

For more information about UFM, see Unified Fabric Manager.

3.6 IBM Storage and file systems

This section describes the storage solution and file system solution that is implemented for the clustered solution.

3.6.1 IBM Spectrum Scale

IBM Spectrum Scale is a cluster file system that provides concurrent access to a single file system or set of file systems from multiple nodes. The nodes can be SAN-attached; network-attached; a mixture of SAN-attached and network-attached; or in a shared-nothing cluster configuration, which enables high-performance access to this common set of data to support a scale-out solution or to provide a high availability platform.

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

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

For more information about IBM Spectrum Scale, see IBM Spectrum Scale - Overview and IBM Knowledge Center.

3.6.2 IBM Elastic Storage Server

IBM Elastic Storage Server is an optimized disk storage solution that is bundled with IBM hardware and innovative IBM Spectrum Scale RAID technology (based on erasure coding). IBM Elastic Storage Server can be used to protect hardware failure instead of using data replication and offer better storage efficiency than local storage.

It performs fast background disk rebuilds in minutes without affecting application performance. HDFS Transparency V2.7.0-1 and later allows Hadoop or Spark applications to access the data that is stored in IBM Elastic Storage Server.

For more information, see IBM Knowledge Center.

4

Reference architecture

High-performance computing (HPC) systems are available in various sizes. These solutions can start with a few nodes and scale out to hundreds or even thousands of nodes.

This chapter describes CORAL-like driven HPC systems in two different sizes to illustrate the process of their design and scaling. The cases describe solutions for large clusters, and then describe medium-sized ones by highlighting features, differences, and functions consolidation.

The following topics are described in this chapter:

- Large HPC cluster architecture
- Medium HPC cluster architecture
- Software stack mapping
- Generic sizing of nodes
- Ethernet network layout
- InfiniBand network
- Job launch overview

4.1 Large HPC cluster architecture

HPC clusters of more than 180 nodes are considered large clusters. Figure 4-1 shows the different components with their quantities within a large HPC cluster. These quantities can vary based on the cluster's size. The quantities that are shown can be applied to clusters with more than 4600 compute nodes.

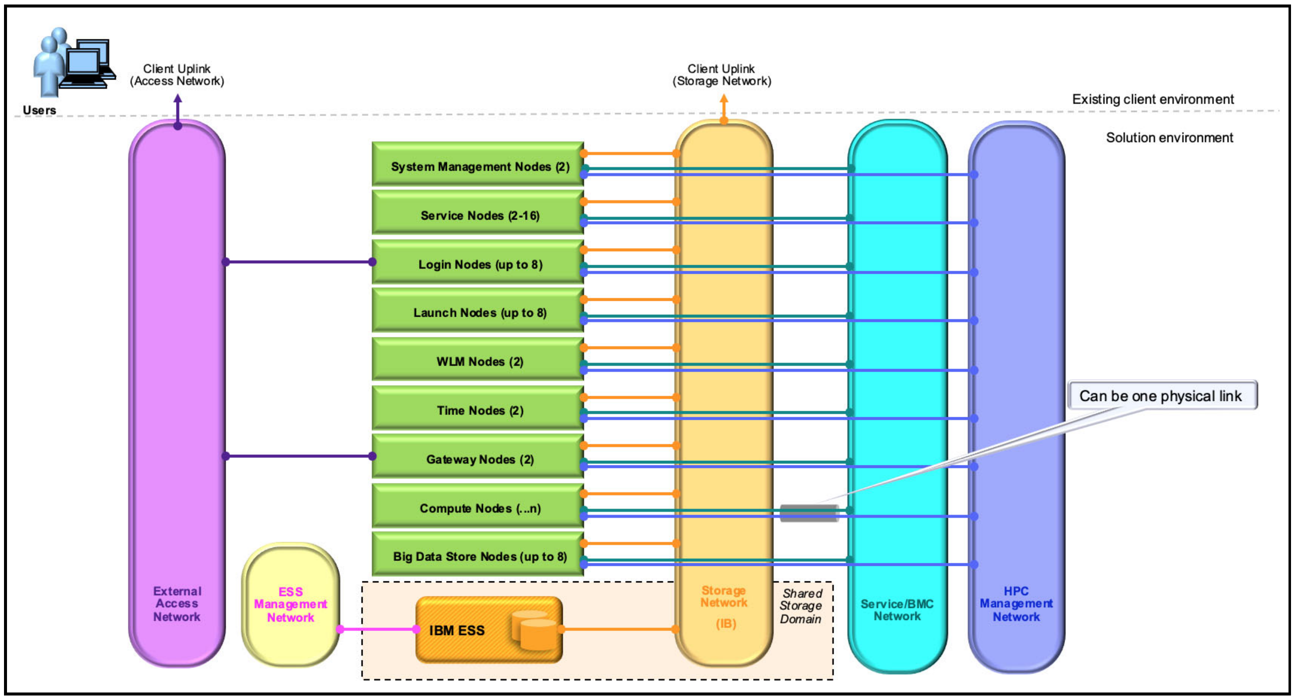


Figure 4-1 Large HPC cluster components

This setup adds many enhancements to support better system utilization, including:

- Supports fast restart of a failed job.
- Supports better problem determination:
  - A diagnostic infrastructure is shipped with the system.
  - There is a reliability, availability, and serviceability (RAS) subsystem to help with fault isolation.
  - Supports collecting environmental data and correlating it back to specific jobs.
- Jitter mitigation:
  - Low-impact access to the file system.
  - Overlap checkpoint data movement with execution with the burst buffer (BB) support.
  - Daemon usage.
- Enhanced job reporting for correlation with the job with events that occur on the system.
- Better resource management:
  - Interface for Power Systems management.
  - Interface for the graphics processing units (GPUs).
- Local solid-state drive (SSD) access for faster scratch I/O.

These enhancements are supported by a collection of more than 60 IBM Cluster Systems Management (CSM) application programming interface (API) calls that support job allocation and tracking, RAS features, system environmental collection, and diagnostics support. These APIs support both privileged and unprivileged interfaces and are intended to enable the functions that are described in previous sections. Major functions include the following ones:

- Interface API:
  - Allocation request.
  - Job request:
    - Nodes, and allocation ID.
  - Job query request.
  - RAS DB API interface.
  - DIAG DB API interface.
  - Job control:
    - Send signal.
    - Store job start information.
- Service agent requests:
  - Collect job completion data.

One important thing is that this setup does not dictate a specific job scheduler. For example, the job scheduling system can be IBM Spectrum Load Sharing Facility (IBM Spectrum LSF); SLURM; or any other scheduler if it provides a scheduler API that can be used by any scheduler to manage jobs on the system, and retrieve job and job step information.

Large clusters also impose several challenges regarding data storage and RAS handling, for example:

- The quantity of data that is stored in a central database can negatively affect system performance.
- The quality and type of the data that is collected cannot be sufficient to manage the system effectively and provide the necessary diagnostic tools.

To overcome these challenges, the solution uses both an SQL database and a big data storage (BDS) to collect all system and RAS information.

The SQL database, which is managed by CSM, contains only information that is required to run the system, which includes the following information:

- Currently active job allocations, jobs, and job steps
- Historical job allocations, jobs, and job steps:
  - Including summary statistics for jobs and job steps
- Node configuration and state for all nodes on the system:
  - Solid-state drive (SSD) partition state and usage
  - GPU state and usage
  - Host Channel Adapter (HCA) state and usage
  - Historical configuration data for each node
- InfiniBand switch configuration and state for all switches on the system and history
- InfiniBand cable information

- RAS events
- Diagnostic run status

The BDS keeps at near-real time the data that is stored in the SQL database. In addition to this data, the BDS stores the following information:

- Environment data on all nodes:
  - Temperature
  - Power
  - GPU usage
  - CPU usage
  - Active memory
- Job details for each task in a job or job step
- Syslogs from all nodes on the system
- IBM GPFS™ RAS data
- Unified Fabric Manager (UFM) RAS data

Section 4.3, “Software stack mapping” on page 41 shows the software components of the HPC cluster and maps them to the hardware components on which they are running.

There are several distinguishing features of the CORAL-like HPC clusters, such as the I/O shipping capability; the SSD support for staging in and staging out data to the node; and the support structure for RAS, data collection, and job support.

Job support includes restricting access to the compute nodes to the user who has a currently active job. It also includes controlling the non-job related processes on the compute node.

To support these capabilities, there are a set of daemons that interoperate among themselves and support the user commands. These daemons use encrypted communication by using x.509 certificates. These daemons, along with Extreme Cluster and Cloud Administration Toolkit (xCAT) support, can provide the sole access to the system by unprivileged users.

Large HPC clusters are composed of the following node types:

- Management nodes

  Up to two highly available management nodes, with xCAT as the core software building block for managing the cluster.

  The management nodes are responsible for the following functions:
  - Administration, management, and monitoring of the system
  - Installation of compute nodes
  - Operating system (OS) distribution management and updates
  - System configuration management
  - Kit management
  - Provisioning templates
  - Stateless and stateful management
  - User logon, compilation, and submission of jobs to the system

  - Acting as a firewall to shield the system from external nodes and networks
  - Acting as a server for many important services, such as DHCP, NFS, DNS, NTP (in small to medium clusters), and HTTP

  Most of these functions are delegated to service nodes so that management nodes are used by CSM. CSM and its APIs provide the following functions:

  - CSM integrator:
    - Integrates individual hardware components POWER9 processors, NVIDIA, Mellanox, NVMe, and so on) into a cluster view to facilitate debugging and troubleshooting activities.
    - Maintains a health list of compute nodes for the scheduler to use.
  - Scalable cluster manager:
    - Design and build thousand of compute nodes with a hierarchy architecture.
  - Jitter-aware design by using the following functions:
    - Lightweight daemons.
    - Core isolation techniques.
  - Data correlation:
    - Manage the PostgreSQL database.
    - Job accounting; job resources; metrics; hardware and software configurations; and logs.

- Service (aggregator) nodes

  These are the *subordinate* servers operating under the management nodes to help system management reduce the load (CPU, and network bandwidth). Service nodes are necessary when managing large clusters.

  Up to 16 service nodes with diskful installation. Each compute node is connected to primary and backup service nodes, as shown in Figure 4-2 on page 39. These service nodes have connections to the base management controller (BMC) for out-of-band control. In addition, there also in an inband connection to daemons on the compute node.

- Workload management nodes

  There are up to eight nodes with diskful installation. These nodes mainly have the job scheduler (IBM Spectrum LSF in this case) that supports the scheduler CSM APIs.

  The scheduler must support:

  - Restart after soft failure
  - Absolute priority scheduling
  - Power management
  - Job steps

- Login nodes

  There are up to eight nodes with diskful installation. This is the place to write, edit, and compile the code; manage data; and submit jobs. Users do not start parallel jobs from a login node or run threaded jobs on a login node. Login nodes are shared resources that are in use by many users simultaneously.

- Launch nodes (LNs)

  There are up to eight nodes with diskful installation, which are the home nodes for job launch. When a batch script (or interactive batch job) starts running, it runs on a LN. All commands within the job script (or the commands run in an interactive job) run on a LN. Like login nodes, these nodes are shared resources, so users must not run multiprocessor or multithreaded programs on LNs. You may run the **`jsrun`** command from LNs.

- Compute nodes

  Most of the HPC cluster nodes are compute nodes with diskless installation, which are where users' parallel jobs run. To access them, run the **`jsrun`** command.

- Time nodes

  There are up to two highly available nodes with diskful installation. Time nodes synchronize the system clocks on all of the compute nodes by using the Precision Time Protocol (PTP), based on the IEEE 1588[1] standard, to keep synchronization accuracy in the submicrosecond range with minimal network and local clock computing resources.

  Synchronization is done over an InfiniBand network for each component in the HPC cluster. InfiniBand network devices must be configured to provide the PTP protocol with the highest quality of service (QoS).

- Gateway nodes

  There are up to two nodes with diskful installation for mounting on an on-demand basis, cluster external file systems to the compute nodes to copy data from or to the cluster environment.

- BDS nodes

  There are up to eight nodes with diskful installation that form an Elasticsearch, Logstash, and Kibana (ELK) stack as a BDS that is integrated with Cluster Administration and Storage Tools (CAST) nodes by way of the `csm-big-data` RPM in the form of suggested configurations and support scripts.

  The recommended installation order of this ELK stack-based solution is:

  a. Elasticsearch

     A distributed analytics and search engine and the core component of the ELK stack. It ingests structured data (typically JSON or key value pairs) and stores the data in distributed index shards.

  b. Kibana

     An open-sourced data visualization tool that is used in the ELK stack.

  c. Logstash

     An open source data processing pipeline that is used in the ELK stack. The core function of this service is to process unstructured data, typically syslogs, and then pass the newly structured text to the Elasticsearch service.

  This installation order minimizes the likelihood of improperly ingested data being stored in Elasticsearch.

> **Note:** The BB server is assumed to be a virtual machine (VM) on each IBM Elastic Storage Server I/O node. An alternative is to have it on a bare metal server.

[1] https://standards.ieee.org/standard/1588-2008.html

Figure 4-2 demonstrates the logical and hierarchy infrastructure connectivity of CSM and xCAT. The management nodes are managing the CSM PostgreSQL DB, and all compute nodes are connected to a primary and backup service nodes.

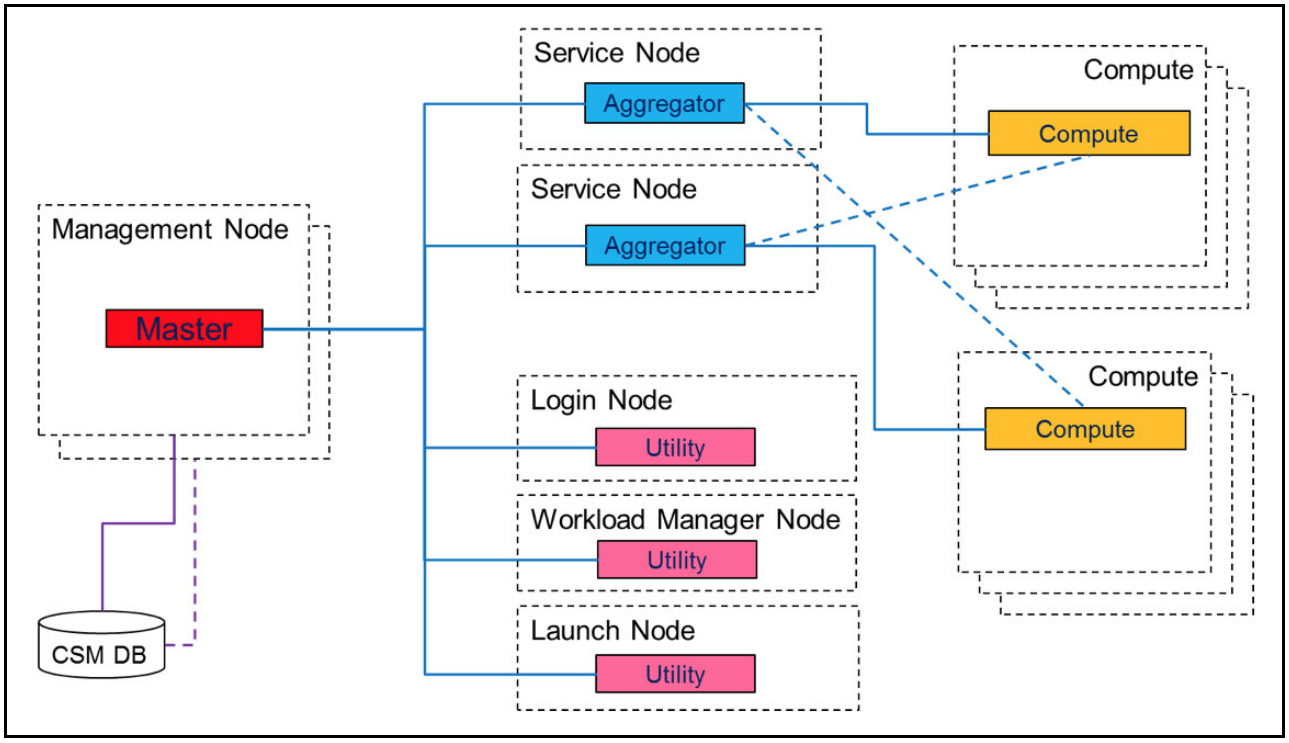


Figure 4-2 The logical and hierarchy infrastructure connectivity of CSM and xCAT

Figure 4-3 demonstrates how CSM uses the lightweight daemons running on every node to integrate and correlate different logs, events, and information from cluster entities that are ingested into the BSD cluster and the CSM DB to get in-depth insights about the cluster.

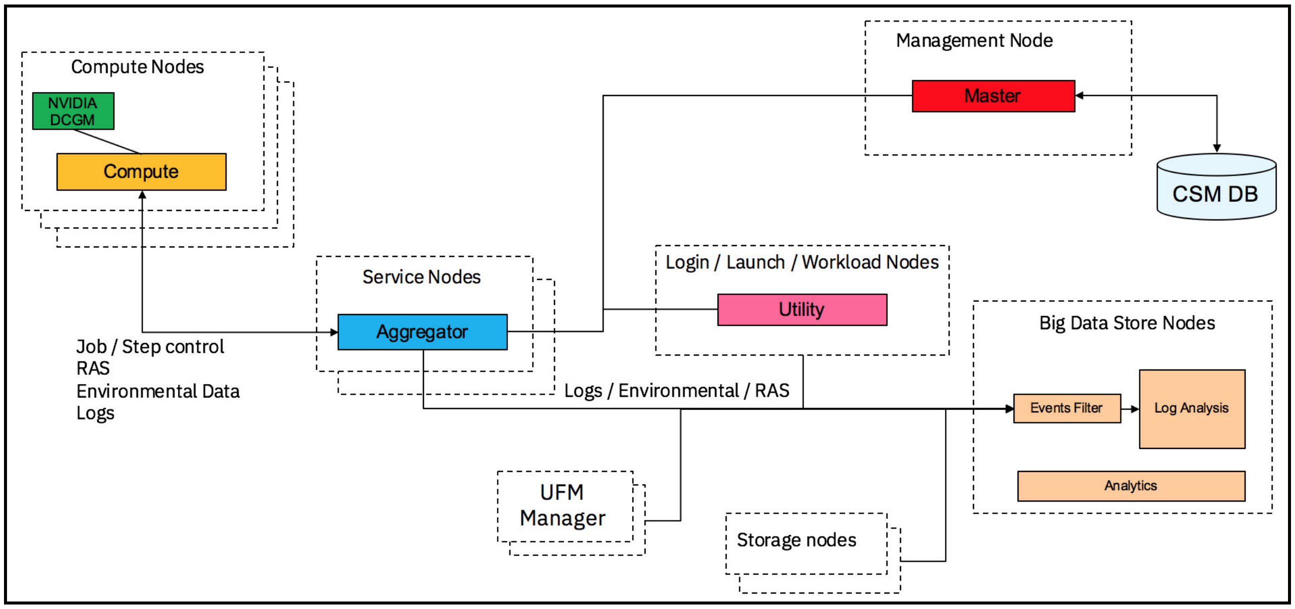


Figure 4-3 Cluster Systems Management architecture

4.2 Medium HPC cluster architecture

Medium clusters are usually considered for clusters with 19 - 180 compute nodes. Depending on the HPC cluster size, different components functions can be consolidated. For example, a cluster with fewer than 80 nodes can easily work without service nodes. Also, gateway nodes are not necessary if there is no need to copy data from or to external computing environments or data lakes. Time servers functions can also be implemented with management nodes if there are less strict time synchronization requirements, which also means that you can use NTP instead of the PTP IEEE protocol.

Figure 4-4 illustrates generic component quantities of a medium-sized cluster, with highlighting of optional components based on a specific client case.

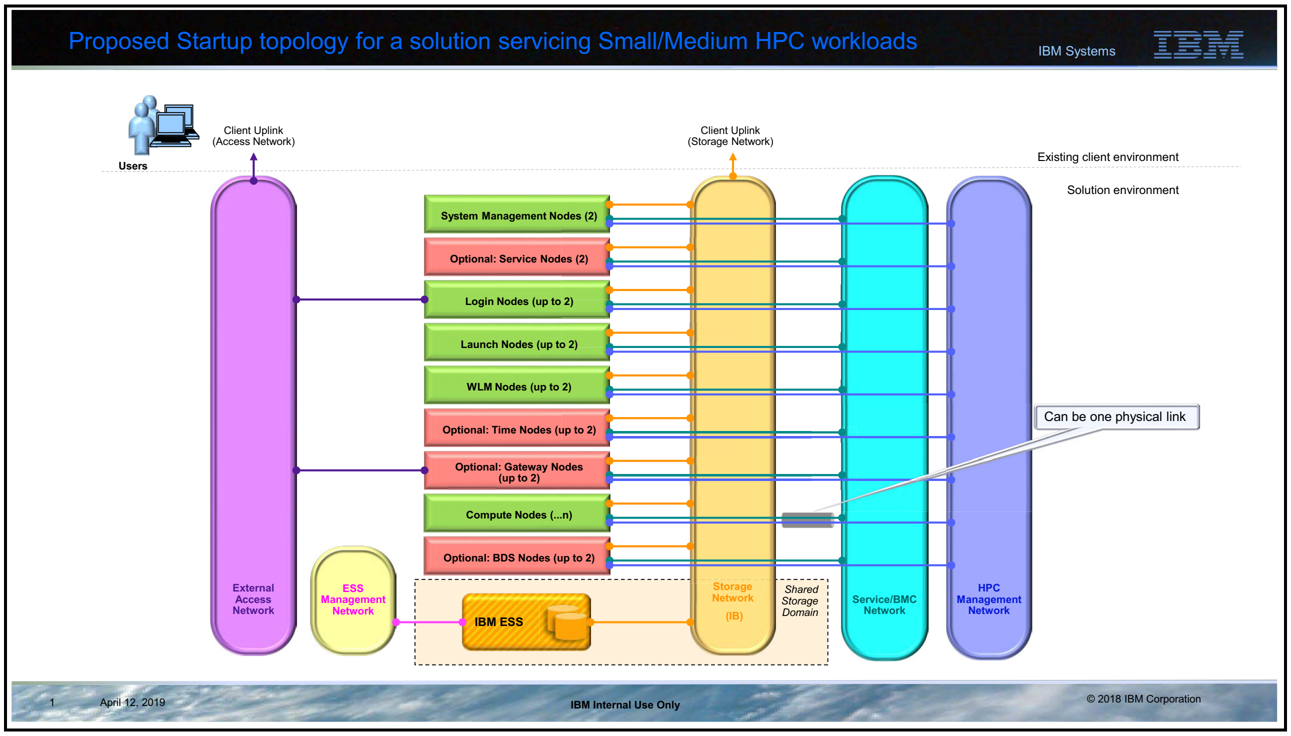


Figure 4-4 Medium HPC cluster components

4.3 Software stack mapping

Figure 4-5 shows different software components within an HPC cluster and how they map to different logical components.

Notice the integration of the different BDS components among other computing node types. For example, management nodes are also holding the filebeats stack while the login nodes hold the logstash for most of the BDS nodes for the Elasticsearch layer. This functional distribution can be consolidated by dedicating two nodes for Kibana, logstash, and filebeats while keeping the remainder of the BDS nodes in the Elasticsearch layer.

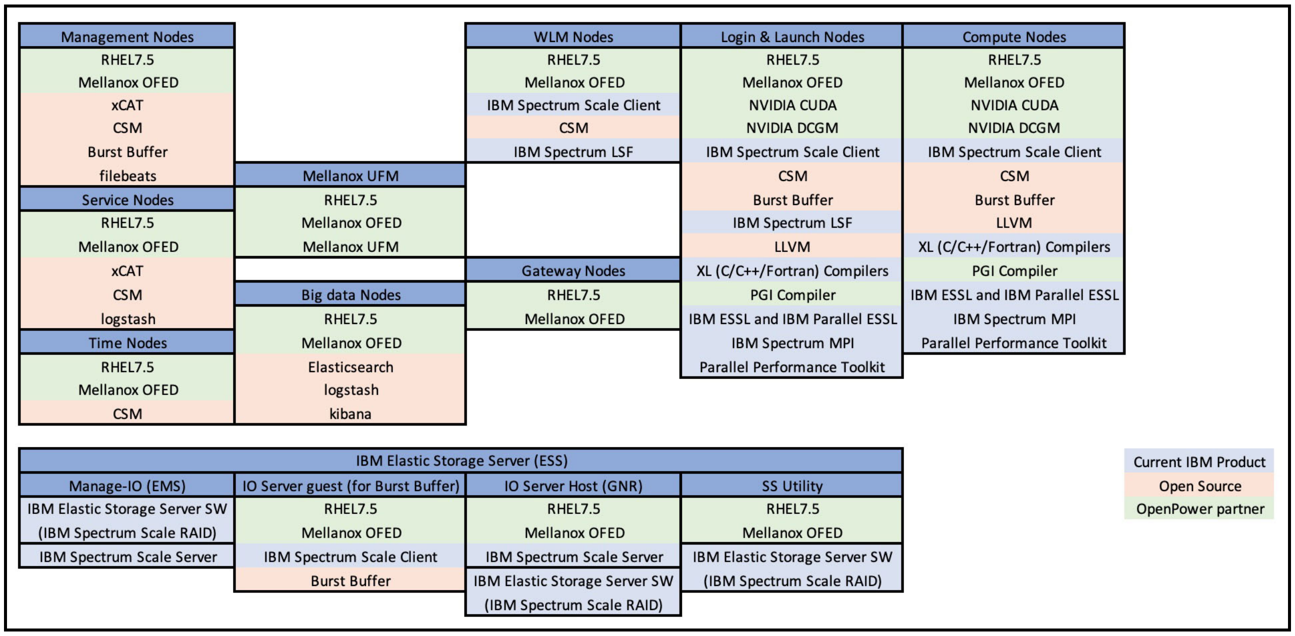
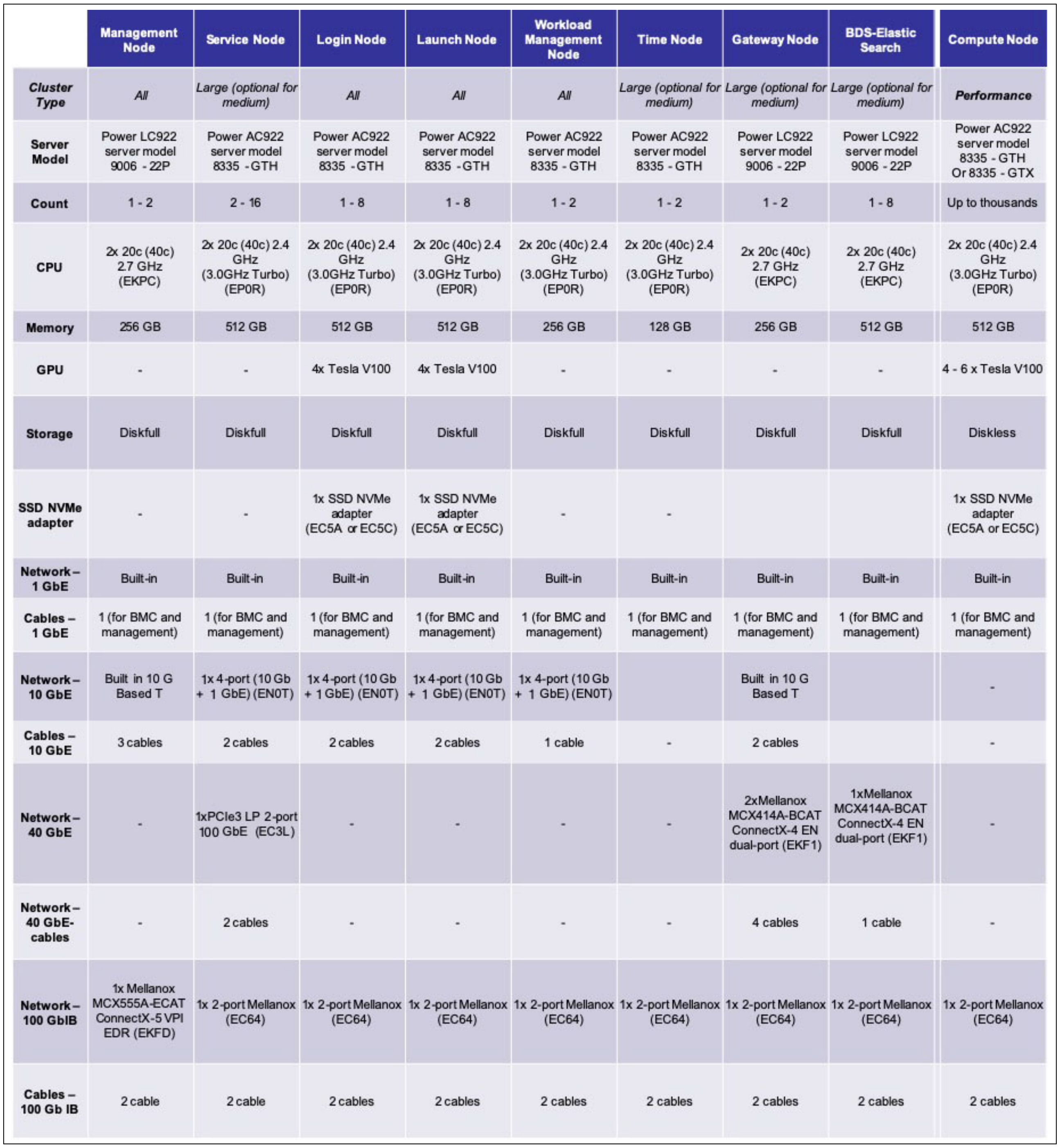


Figure 4-5 Software: node types mapping

4.4 Generic sizing of nodes

Figure 4-6 provides generic sizing of different node types that make up large clusters. The actual quantity and sizing of the nodes are client-scenario dependant. Workload type, job resource requirements, and the number of users are part of the factors that contribute to proper sizing.



	Management Node	Service Node	Login Node	Launch Node	Workload Management Node	Time Node	Gateway Node	BDS-Elastic Search	Compute Node
Cluster Type	*All*	*Large (optional for medium)*	*All*	*All*	*All*	*Large (optional for medium)*	*Large (optional for medium)*	*Large (optional for medium)*	***Performance***
Server Model	Power LC922 server model 9006 - 22P	Power AC922 server model 8335 -GTH	Power AC922 server model 8335 -GTH	Power AC922 server model 8335 -GTH	Power AC922 server model 8335 - GTH	Power AC922 server model 8335 - GTH	Power LC922 server model 9006 - 22P	Power LC922 server model 9006 - 22P	Power AC922 server model 8335 - GTH Or 8335 - GTX
Count	1 - 2	2 - 16	1 - 8	1 - 8	1 - 2	1 - 2	1 - 2	1 - 8	Up to thousands
CPU	2x 20c (40c) 2.7 GHz (EKPC)	2x 20c (40c) 2.4 GHz (3.0GHz Turbo) (EP0R)	2x 20c (40c) 2.4 GHz (3.0GHz Turbo) (EP0R)	2x 20c (40c) 2.4 GHz (3.0GHz Turbo) (EP0R)	2x 20c (40c) 2.4 GHz (3.0GHz Turbo) (EP0R)	2x 20c (40c) 2.4 GHz (3.0GHz Turbo) (EP0R)	2x 20c (40c) 2.7 GHz (EKPC)	2x 20c (40c) 2.7 GHz (EKPC)	2x 20c (40c) 2.4 GHz (3.0GHz Turbo) (EP0R)
Memory	256 GB	512 GB	512 GB	512 GB	256 GB	128 GB	256 GB	512 GB	512 GB
GPU	-	-	4x Tesla V100	4x Tesla V100	-	-	-	-	4 - 6 x Tesla V100
Storage	Diskfull	Diskfull	Diskfull	Diskfull	Diskfull	Diskfull	Diskfull	Diskfull	Diskless
SSD NVMe adapter	-	-	1x SSD NVMe adapter (EC5A or EC5C)	1x SSD NVMe adapter (EC5A or EC5C)	-	-			1x SSD NVMe adapter (EC5A or EC5C)
Network – 1 GbE	Built-in	Built-in	Built-in	Built-in	Built-in	Built-in	Built-in	Built-in	Built-in
Cables – 1 GbE	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)	1 (for BMC and management)
Network – 10 GbE	Built in 10 G Based T	1x 4-port (10 Gb + 1 GbE) (EN0T)	1x 4-port (10 Gb + 1 GbE) (EN0T)	1x 4-port (10 Gb + 1 GbE) (EN0T)	1x 4-port (10 Gb + 1 GbE) (EN0T)		Built in 10 G Based T		-
Cables – 10 GbE	3 cables	2 cables	2 cables	2 cables	1 cable	-	2 cables		-
Network – 40 GbE	-	1xPCIe3 LP 2-port 100 GbE (EC3L)	-	-	-	-	2xMellanox MCX414A-BCAT ConnectX-4 EN dual-port (EKF1)	1xMellanox MCX414A-BCAT ConnectX-4 EN dual-port (EKF1)	-
Network – 40 GbE-cables	-	2 cables	-	-	-	-	4 cables	1 cable	-
Network – 100 GbIB	1x Mellanox MCX555A-ECAT ConnectX-5 VPI EDR (EKFD)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)	1x 2-port Mellanox (EC64)
Cables – 100 Gb IB	2 cable	2 cable	2 cables	2 cables	2 cables	2 cables	2 cables	2 cables	2 cables

Figure 4-6 Generic sizing of different node types

4.5 Ethernet network layout

Figure 4-7 shows a typical Ethernet network layout of the different node types that participatE in a large HPC cluster.

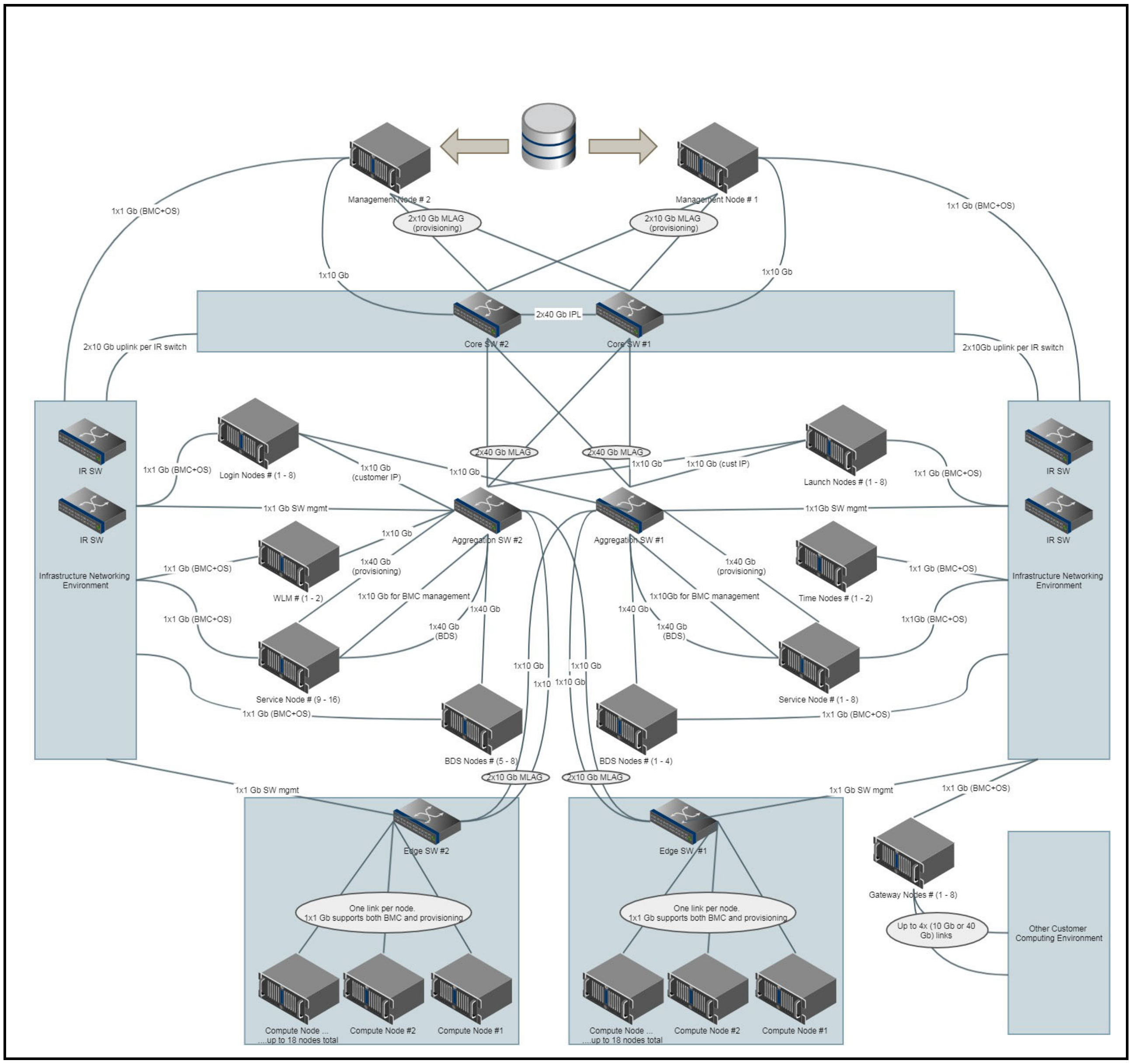


Figure 4-7 Typical large HPC cluster network connectivity

Figure 4-7 highlights the three layers of network switches:

1. Core layer
2. Aggregation layer
3. Edge layer, which includes the management level

Depending on the scale of the cluster, a switch that can be used for both core and aggregation levels is the IBM Switch (8831-S48), which is a Mellanox sourced Ethernet switch with the Mellanox part number MSX1410-BB2F2. This switch provides 48x 10 GbE + 12x 40 GbE ports.

For the edge layer, you can use the IBM Switch (8831-S52), which is a Mellanox sourced Ethernet switch. It provides 48x 1 GbE RJ45 + 4x 1/10 GbE SFP+ ports.

For more information about IBM and Mellanox joint solutions, see IBM and Mellanox Solutions.

4.6 InfiniBand network

InfiniBand is the predominant interconnect technology in the HPC market. InfiniBand has many characteristics that make it ideal for HPC:

- Low latency and high throughput
- Remote Direct Memory Access (RDMA)
- A flat Layer 2 that scales out to thousands of end points
- QoS
- Centralized management
- Multi-pathing
- Support for multiple topologies

The following section illustrates how to design an HPC cluster by using a Mellanox InfiniBand interconnect solution.

4.6.1 InfiniBand network topologies

There are several common topologies for an InfiniBand fabric. Here are some of those topologies:

- Fat tree

  A multi-root tree. This is the most popular topology.
- 2D/3D mesh

  Each node is connected to four or six other nodes: positive, negative, X axis, and Y axis.
- 2D/3D torus

  The X, Y, and Z ends of the 2D or 3D mashes are *wrapped around* and connected to the first node.

Other topologies also exist, such as dragonfly and hypercube. This section focuses on fat tree, which has optimized performance and scalability.

4.6.2 Fat tree topology

The most widely used topology in HPC clusters is the fat-tree topology. This topology typically enables the best performance at a large scale when configured as a *non-blocking* network. Where over-subscription of the network is tolerable, it is possible to configure the cluster in a blocking configuration.

A fat-tree cluster typically uses the same bandwidth for all links, and in most cases it uses the same number of ports in all of the switches.

The switches on the upper levels are called *spine switches*. The switches on the lowest level are called *leaf switches*, and the nodes are always connected to them. The smallest fat-tree topology has at least two levels.

Figure 4-8 illustrates a two-level, non-blocking fat-tree topology cluster with 36 port switches as a leaf (L1) layer.

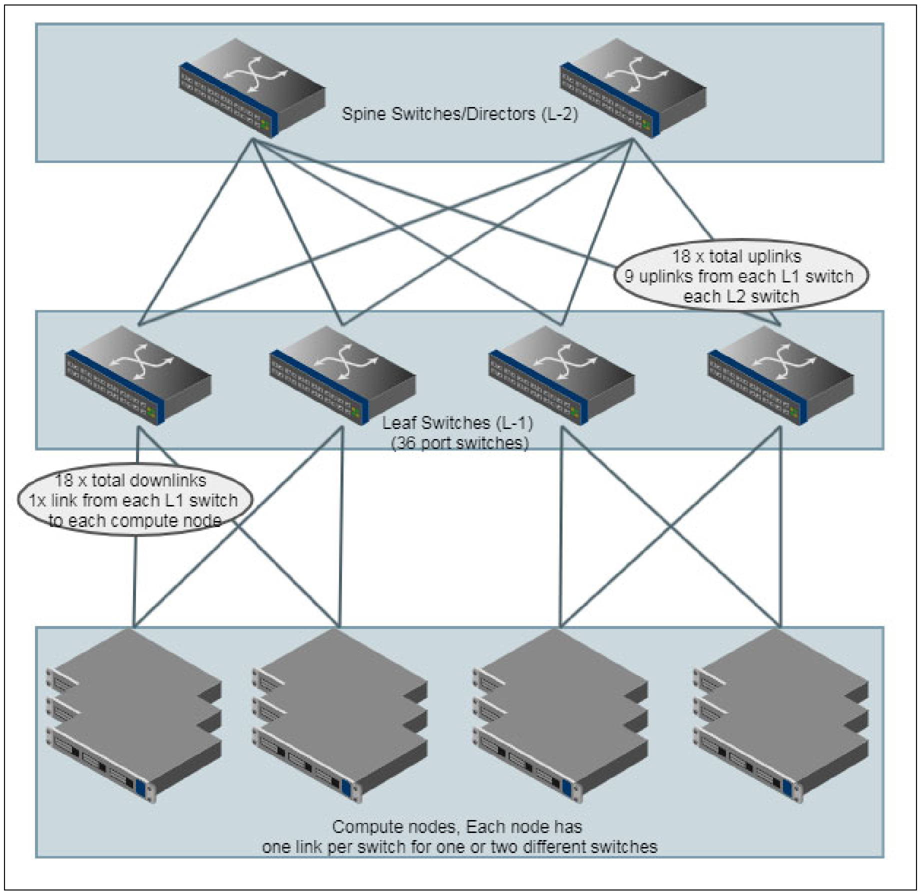

Spine Switches/Directors (L-2)
18 x total uplinks
9 uplinks from each L1 switch
each L2 switch
Leaf Switches (L-1)
(36 port switches)
18 x total downlinks
1x link from each L1 switch
to each compute node
Compute nodes, Each node has
one link per switch for one or two different switches

Figure 4-8 Two levels non-blocking fat-tree topology cluster

Fat tree cluster design rules

The following rules must be adhered to while building a fat-tree cluster[2]:

- *Non-blocking clusters* must be balanced. The same number of links must connect a Level 2 (L2) switch to every Level 1 (L1) switch. Whether over-subscription is possible depends on the HPC application and the network requirements.
- If the L2 switch is a director switch (that is, a switch with leaf and spine cards), all L1 switch links to an L2 switch must be evenly distributed among leaf cards. For example, if six links run between an L1 and L2 switch, they can be distributed to leaf cards as 1:1:1:1:1:1, 2:2:2, 3:3, or 6. They must never be mixed, for example, 4:2 or 5:1.
- Do not create routes that must traverse up, back down, and then up the tree again. This creates a situation that is called *credit loops* that manifest itself as traffic deadlocks in the cluster. In general, there is no way to avoid credit loops. Any fat-tree with multiple directors plus edge switches has physical loops that are avoided by using a routing algorithm such as *up-down*.
- Try to always use 36-port switches for L1 and director class switches for L2.

[2] Designing an HPC Cluster with Mellanox InfiniBand Solutions

Consider using the Mellanox InfiniBand Cluster Configurator because it helps configure clusters based on a fat tree topology with two levels of switch systems.

Note: For fat trees larger than 648 nodes, the HPC cluster must at least have three levels of hierarchy.

At the time of writing, the following documents provide the most recent Mellanox InfiniBand interconnect guides:

- Designing an HPC Cluster with Mellanox InfiniBand Solutions
- Understanding Up/Down InfiniBand Routing Algorithm
- How to Prevent InfiniBand Credit Loops
- Recommendations for Inter-Switch Cabling
- Cabling Considerations for Clos-5 Networks

4.7 Job launch overview

Note: For more information about IBM Spectrum LSF Job Step Manager (JSM) Version 10.2, see IBM Knowledge Center.

Figure 4-9 on page 47 provides an overview of the users' job launch and the sequence of events in relationship to different software and hardware stacks.

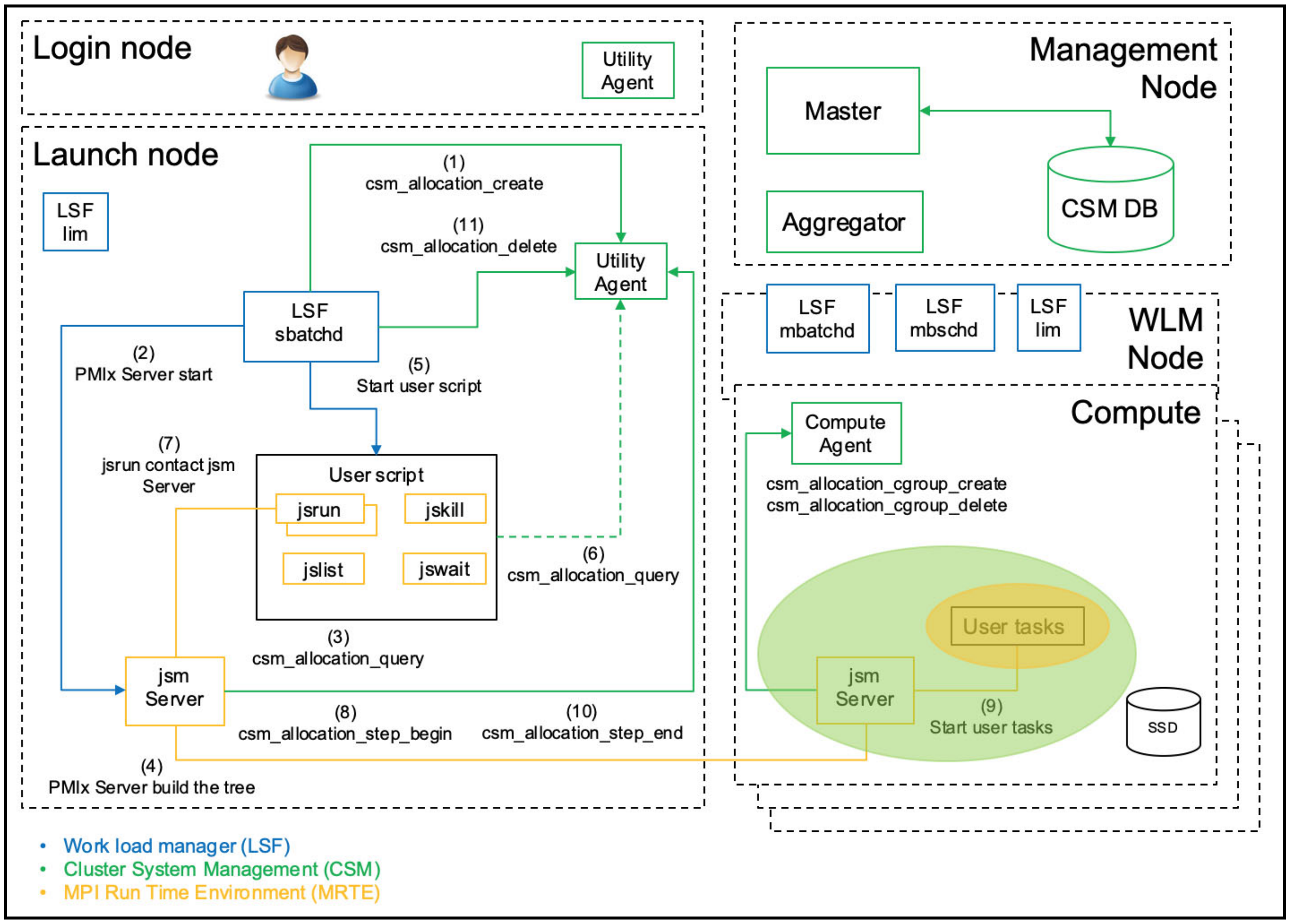


Figure 4-9 Users' job launch overview and events sequence

There are three components that are involved in the job launch:

1. The workload manager (WLM) (IBM Spectrum LSF)
2. The CSM
3. The Message Passing Interface (MPI) Runtime Environment (RTE)

The dashed lines in Figure 4-9 indicate optional operations.

After IBM Spectrum LSF decides to start a job by using the CSM infrastructure, it calls **`csm_allocation_create`** to reserve nodes for a job, run the job prolog, create the Job Step Manager (JSM) (PMIx) server, and launch a user script or executable file on the LN. As part of the CSM allocation, the CSM database is updated to include the job start information. Then, the compute nodes are put into job run mode and are accessible to the job owner.

The **`jsm`** daemon creates a job step launch tree on all the nodes that are determined by a **`csm_allocation_query`** call. This job step tree is created on behalf of the job that is allocated with **`csm_allocation create`**. Then, user job steps can be launched by the user with the **`jsrun`** command. This command communicates with the **`jsm`** daemon on the LN through a UNIX socket by passing job step requests. As part of the job step start and end, IBM JSM calls CSM to record step-specific information in the CSM database. After the last job step completes, the user script or executable file that is launched by IBM Spectrum LSF exits and IBM Spectrum LSF stops the **`jsm`** daemon. IBM Spectrum LSF then releases resources after calling the job epilog with a call to **`csm_allocation_delete`**. Then, the job resource usage is stored in the CSM database; the job is complete; and the node resources can be reused.

Important: This architecture provides the best performance and cluster utilization efficiency for jobs that require high parallelism and high CPU and GPU resources, such as jobs that require at least the full resources of the Power AC922 node.

Important: Users can include multiple independent or interrelated job steps within the same user submitted job script.

Part 2

Deployment

This part describes how to deploy the nodes and software in a cluster. It also describes the Cluster Administration and Storage Tools (CAST) to manage the solution.

The following chapters are included in this part:

- ▸ Nodes and software deployment
- ▸ Cluster Administration and Storage Tools

5

Nodes and software deployment

This chapter provides a comprehensive description about how to deploy the nodes that integrate into the high-performance computing (HPC) solution.

This chapter also introduces the Extreme Cluster and Cloud Administration Toolkit (xCAT) installation in the Power System AC922 server and the images that are created that contain the HPC software stack. The cluster runs Red Hat Enterprise Linux 7.5 ppc64 Little Endian diskful image on the management node and diskless image on the compute nodes. Both are bare-metal, non-virtualized installations.

The following topics are described in this chapter:

- Deployment overview
- System management
- xCAT deployment overview
- Initial xCAT management node installation on an IBM Power System LC922 server
- xCAT node discovery
- xCAT compute nodes (stateless)
- xCAT login nodes (stateful)

5.1 Deployment overview

The HPC clustered solution that is described in this book is based on the CORAL reference architecture and the fastest super computer as of the time of writing[1]. The base components of this cluster are the Power AC922 server, Red Hat Enterprise Linux 7.5 Little Endian, Mellanox high-speed InfiniBand networking, and the NVIDIA graphics processing unit (GPU) and NVLink technology that is described in Chapter 2, "IBM Power System AC922 server for HPC overview" on page 7 and in Chapter 3, "Software stack" on page 19.

On top of the base foundation, this chapter focuses on the installation of xCAT, a tool that is known for node deployment and administration. This chapter explains in detail how the images for the compute nodes and the other type of nodes are created and installed.

5.2 System management

The Power AC922 system is based on the OpenBMC platform, which supports out-of-band management through the Intelligent Platform Management Interface (IPMI) tool and the OpenBMC tool. This chapter describes how both of these utilities provide similar management capabilities and their respective examples.

5.2.1 The Intelligent Platform Management Interface tool

The IPMI tool is an open source command-line interface (CLI) that you use for managing and controlling OpenBMC based systems. It provides examples in Version 1.8.18 and later.

Here are the most common options and commands of the **`ipmitool`** command. The format of the command is:

```
ipmitool [options...] <command>
```

- Options:
  - **`-H <address>:`** The host name or IP address of the baseboard management controller (BMC) (only one is needed).
  - **`-P <password>:`** The remote BMC password. (The default in POWER9 processor-based servers is `OpenBMC`.)
  - **`-I <interface>:`** The interface type of the connection. (As a preferred practice, use lanplus as the interface for all the **`ipmitool`** commands against the BMC.)
- Commands:
  - **`chassis`**: Retrieves the chassis status and sets the power states.
  - **`sol`**: Facilitates Serial-over-LAN (SOL) connections.
  - **`mc`**: Interacts with the management controller.
  - lan: Configures the network interface.

[1] https://www.top500.org/

In Example 5-1, the **ipmitool** command is used to get the power status of the chassis. Then, the command is then followed by a command that powers on the system and checks the power status again.

Example 5-1 The ipmitool command: example commands

```
$ ipmitool -H <hostname/IP> -P OpenBmc -I lanplus chassis status
System Power         : onff
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
Power Control Fault  : false
Power Restore Policy : always-off
Last Power Event     :
Chassis Intrusion    : inactive
Front-Panel Lockout  : inactive
Drive Fault          : false
Cooling/Fan Fault    : false
Front Panel Control  : none

$ ipmitool -H <hostname/IP> -P OpenBmc -I lanplus chassis power on
Chassis Power Control: Up/On

$ ipmitool -H <hostname/IP> -P OpenBmc -I lanplus chassis power status
Chassis Power is on
```

With the **sol** command, you can establish a connection to the system serial port to view a node, as shown in Example 5-2. It is a useful option when you install or debug a node, or there is not a network configuration on the node to establish an SSH connection.

Example 5-2 The sol command options

```
$ ipmitool -H <hostname/IP> -P OpenBmc -I lanplus sol activate
Info: SOL payload already active on another session
$ ipmitool -H <hostname/IP> -U ADMIN -P admin -I lanplus sol deactivate
$ ipmitool -H <hostname/IP> -U ADMIN -P admin -I lanplus sol activate
[SOL Session operational. Use ~? for help]
p9r1n1 login:
```

Use the **mc** command to perform management console functions. Typically, you use this command only when the BMC is no longer responsive to commands (such as **chassis**).

Example 5-3 shows the most common use of this command to reset the BMC.

Example 5-3 The mc reset command

```
$ ipmitool -H <hostname/IP> -U ADMIN -P admin mc reset cold
Sent cold reset command to MC
```

You can use the **lan** command to configure the BMC network interface (although it is assumed that the current IP address is known and is passed as part of the **<arguments>**).

The following important options can be included:

- Display the BMC Ethernet Port network configuration:

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

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

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

- Set the BMC Ethernet Port for static IP address:

```
$ ipmitool <arguments> lan set 1 ipsrc static
$ ipmitool <arguments> lan set 1 ipaddr a.b.c.d
$ ipmitool <arguments> lan set 1 netmask e.f.g.h
$ ipmitool <arguments> lan set 1 defgw i.j.k.l
```

5.2.2 OpenBMC

OpenBMC is an alternative to the **`ipmitool`** command. OpenBMC offers a common structure and set of options to manage and administer the Power AC922 server. OpenBMC also offers a GUI version of the tool, which you can access by going to `<https://BMC IP address>`.

For more information about downloading and installing OpenBMC, see OpenBMC.

Note: This book does not cover installation of the OpenBMC tool; it focuses only on the CLI instance because the HPC cluster general administration is based on only the CLI.

Here are the OpenBMC top-level options:

`-H: <hostname/IP>`	Host name or IP address of the BMC.
`-U: <username>`	User name that is used to log in.
`-A:`	Provides a prompt to ask for the password.
`-P: <password>`	Password for the user name.
`-j:`	Changes the output format to JSON.
`-t:`	Location of the policy table to use.
`-T:`	Provides time statistics for logging in, running the command, and logging out.
`-V:`	Displays the current version of the OpenBMC tool.

Here are some examples of **`openbmc`** commands. For a thorough list of options, go to OpenBMC.

- To check the power status of the system, run the following command:

```
$ openbmctool -U <username> -P <password> -H <hostname/IP> chassis power status
```

- To power on the system, run the following command:

```
$ openbmctool -U <username> -P <password> -H <hostname/IP> chassis power on
```

- To power off the system normally, run the following command:

```
$ openbmctool -U <username> -P <password> -H <BMC IP address or BMC host name>
chassis power softoff
```

- To power off the system immediately, run the following command:

```
$ openbmctool -U <username> -P <password> -H <BMC IP address or BMC host name>
chassis power hardoff
```

- To do a warm reset of the BMC remotely and without an AC cycle, run the following command:

  ```
  $ openbmctool -U <username> -P <password> -H <BMC IP address or BMC host name>
  bmc reset warm
  ```

- To do a cold reset of the BMC remotely and without an AC cycle, run the following command:

  ```
  $ openbmctool -U <username> -P <password> -H <BMC IP address or BMC host name>
  bmc reset coldAdd text here (Body0).
  ```

5.2.3 Firmware update

The OpenBMC tool can perform firmware updates through both the GUI and CLI. To perform a CLI BMC firmware update on the Power AC922 server, complete the steps in Example 5-4.

> **Note:** The commands that are listed in Example 5-4 run in a Linux client node with connectivity to the cluster nodes, OpenBMC is installed in the Power AC922 server, and all **`openbmc`** commands are run from this client node. The **`cat`** commands at the beginning and ending of the example are run from the cluster node being upgraded by using **`ssh`** to its BMC IP address. The **`$`** sign denotes commands that run in the client node, and the **`#`** sign denotes commands that run inside the BMC operating system (OS).
>
> The BMC password is `OpenBMC` with a "zero" instead of the letter "O" in all the examples in this book.

Example 5-4 OpenBMC firmware update CLI sequence

```
$ ssh root@9.108.97.153 <BMC ip address>
# cat /var/lib/phosphor-software-manager/pnor/ro/VERSION
IBM-witherspoon-ibm-OP9-v2.0-2.14-prod
      op-build-v2.0-11-gb248194
      buildroot-2018.02.1-6-ga8d1126
      skiboot-v6.0.1
      hostboot-8ab6717d-p110cb65
      occ-77bb5e6
      linux-4.16.8-openpower2-p6a14c7f
      petitboot-v1.7.1-p50a5645
      machine-xml-7cd20a6
      hostboot-binaries-276bb70
      capp-ucode-p9-dd2-v4
      sbe-a596975
      hcode-b8173e8
# exit
$ openbmctool -H 9.108.97.153 -U root -P OpenBmc chassis power softoff
Attempting login...
Attempting to Power off gracefully...:
{
  "data": null,
  "message": "200 OK",
  "status": "ok"
}
User root has been logged out

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc chassis power status
```

```
Attempting login...
Chassis Power State: Off
Host Power State: Off
BMC Power State: Ready
User root has been logged out

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc firmware flash bmc -f
obmc-witherspoon-ibm-v2.1.ubi.mtd.tar
Attempting login...
This image has been found on the bmc. Activating image: 376af621
Firmware flash and activation completed. Please reboot the bmc and then boot the
host OS for the changes to take effect.
User root has been logged out

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc fru print | grep Activiv
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc firmware flash pnor -f
witherspoon-IBM-OP9-v2.0.8-2.7_prod.pnor.squashfs.tar
Attempting login...
This image has been found on the bmc. Activating image: 841024c1
Firmware flash and activation completed. Please reboot the bmc and then boot the
host OS for the changes to take effect.
User root has been logged out


$ openbmctool -H 9.108.97.153 -U root -P OpenBmc fru print | grep Activ
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",
      "Activation": "xyz.openbmc_project.Software.Activation.Activations.Active",
      "RequestedActivation":
"xyz.openbmc_project.Software.Activation.RequestedActivations.None",

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc mc reset cold
Attempting login...
Attempting to reboot the BMC...:
{
  "data": null,
  "message": "200 OK",
  "status": "ok"
}
User root has been logged out

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc chassis power status
Attempting login...
Chassis Power State: Off
Host Power State: Off
BMC Power State: Ready
User root has been logged out

$ openbmctool -H 9.108.97.153 -U root -P OpenBmc chassis power on
Attempting login...
Attempting to Power on...:
{
  "data": null,
  "message": "200 OK",
  "status": "ok"
}
User root has been logged out
$ ssh root@9.108.97.153
# cat /var/lib/phosphor-software-manager/pnor/ro/VERSION
IBM-witherspoon-ibm-OP9-v2.0.8-2.7-prod
      op-build-v2.0.8-2-gf54aed9
      buildroot-2018.02.1-6-ga8d1126
      skiboot-v6.0.8
      hostboot-d033213-pf8eafb0
      occ-084756c
      linux-4.16.13-openpower1-pcc4b089
      petitboot-v1.7.2-p26e7ade
      machine-xml-7cd20a6
      hostboot-binaries-hw080418a.op920
      capp-ucode-p9-dd2-v4
      sbe-55d6eb2
      hcode-hw082318a.op920
```

5.2.4 Boot order configuration

The Petitboot bootloader can automatically boot (or autoboot) from several types of devices in a certain order (that is, falling back to later devices if earlier devices cannot be used to perform a boot). This scenario requires a specific configuration of the device boot order (for Genesis-based node discovery; and diskful installation. These tasks are described in 5.3, “xCAT deployment overview” on page 59).

Petitboot must first attempt to boot from the network devices by using DHCP. If the network devices are not available, Petitboot attempts to boot from the disk devices. By following this order, the node can obtain its network and boot configuration from the xCAT management node (for example, the Genesis image or diskless installation) or fall back to boot an OS from disk if the network boot is not specified (for example, diskful installation).

To configure the boot order in the Petitboot bootloader, complete the following steps:

1. Power on the system by running the following command:

   ```
   $ ipmitool -I lanplus -H <BMC IP address> -P <password> chassis power on
   ```

2. Open the console by running the following command:

   ```
   $ ipmitool -I lanplus -H <BMC IP address> -P <password> sol activate
   ```

3. Wait for the Petitboot panel, as shown in Example 5-5.

   Example 5-5 Petitboot panel

   ```
   Petitboot  (v1.2.6-8fa93f2)                                 8335-GTH
   0000000000000000"
   "System  information System  configuration Language"
   "Rescan  devices"
   "Retrieve  config  from  URL"
   "*Exit  to  shell"
   "Enter=accept,  e=edit,  n=new,  x=exit,  l=language,  h=help
   "Welcome  to  Petitboot"
   ```

4. Configure the boot order for Petitboot. In the Petitboot panel, select `System configuration`.

5. In the Petitboot System Configuration panel (see Example 5-6), complete the following steps.

   Example 5-6 Petitboot System Configuration panel: DHCP on a specific interface setting

   ```
   Petitboot System Configuration
   Boot Order:  (None)
                [ Add Device ]
                [ Clear & Boot Any ]
                [ Clear ]

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

   Device:     (*) enP5p1s0f0 [70:e2:84:14:d6:81, link up]
               ( ) enP5p1s0f1 [70:e2:84:14:d6:82, link up]

   DNS Server(s): _______________________________ (eg. 192.168.0.2)
                 (if not provided by DHCP server)
   Disk R/W:   ( ) Prevent all writes to disk
               (*) Allow bootloader scripts to modify disks
               [ OK ]      [ Help ]     [ Cancel ]
   tab=next, shift+tab=previous, x=exit, h=help
   ```

   a. In the Boot order section, complete the following steps:

      i. Select `Clear`.

      ii. Select `Add Device`.

iii. In the Select a boot device to add panel (see Example 5-7), select `Any Network device` and click `OK`.

Example 5-7 Select a boot device to add panel

```
Select a boot device to add
         ( ) net: enP1p3s0f0 [mac: 98:be:94:59:fa:24]
         ( ) net: enP1p3s0f1 [mac: 98:be:94:59:fa:25]
         (*) Any Network device
         ( ) Any Disk device
         ( ) Any USB device
         ( ) Any CD/DVD device
         ( ) Any Device
         [ OK ]      [ Cancel ]
tab=next, shift+tab=previous, x=exit, h=help
```

iv. Select `Add Device` again.

v. In the Select a boot device to add panel, select `Any Disk device`, and click `OK`.

vi. Verify that the Boot order section is identical to Example 5-8.

Example 5-8 Petitboot system configuration panel: Boot order configuration

```
Petitboot System Configuration
Boot Order:  (0) Any Network device
             (1) Any Disk device
             [ Add Device ]
             [ Clear & Boot Any ]
             [ Clear ]
<...>
```

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

Note: When you select the `DHCP on all active interfaces` option, it might slow the boot process unnecessarily if multiple networks ports are cabled and active.

c. In the Device section, select the network interface for accessing the xCAT management node (if you selected DHCP on a specific interface in the network section).

d. Click `OK`.

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

5.3 xCAT deployment overview

This section provides an overview of the xCAT deployment and installation. It describes the management node and deploying the cluster nodes. xCAT (described in 3.5.1, “Extreme Cluster and Cloud Administration Toolkit” on page 28) is an open source tool to deploy and administer an HPC cluster. xCAT is continuously updated with new features and enhancements. For more information, see xCAT download and xCAT documentation.

5.3.1 xCAT database: Objects and tables

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

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

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

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

The xCAT database is stored in a SQLite database by default. Open source database engines, such as MySQL, MariaDB, and PostgreSQL, are also supported for large clusters.

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

```
$ man 5 xcatdb
```

5.3.2 xCAT node booting

The xCAT management node can control the boot method (or device) of the compute nodes by using the following methods:

- Change the temporary boot device configuration of the bootloader by using IPMI.
- Change the network boot configuration that is provided to the bootloader by using DHCP.
- Do not provide a network boot configuration to the bootloader so that it may boot from other devices than network adapters.

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

New or undiscovered nodes can be booted into node discovery, and known or discovered nodes into arbitrary stages (for example, OS provisioning, installed OS, basic shell environment, or node discovery again). For that purpose, the nodes' bootloader must be configured to boot automatically with the following device order:

- From the network interface on the xCAT management network (to obtain network and boot configuration by using DHCP/TFTP from the xCAT management node)

- From local disks

For more information, see 5.2.4, “Boot order configuration” on page 57.

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

5.3.3 xCAT node discovery

The xCAT node discovery (or node discovery process) consists of booting a new (or undiscovered) node so that the management node can identify (or *discover*) the node by using a specific method.

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

The following node discovery methods are available in xCAT, ranging from more manual to more automatic:

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

For more information about node discovery methods, see xCAT Hardware Discovery & Define Node.

The MTMS-based discovery can be summarized as the following sequence:

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

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

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

5.3.4 xCAT baseboard management controller discovery

The xCAT BMC discovery occurs automatically after node discovery. It can be summarized by the following sequence:

1. The xCAT management node attempts to match the node object definition to a temporary BMC object definition (with MTMS information) for new or undiscovered BMCs.
2. If a match is identified, the xCAT management node configures the BMC according to the BMC attributes in the node object definition (for example, network configuration) and removes the temporary BMC object.
3. The node then may respond to out-of-band management operations through IPMI or OpenBMC on its BMC IP address.
4. The BMC becomes a discovered BMC, and the corresponding node supports out-of-band operations (for example, power control and monitoring) by using its object definition in the xCAT management node.

5.3.5 xCAT installation types: Disks and state

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

- Diskful and stateful

  The OS is installed to disk and loaded from disk. Changes are written to disk (persistent).
- Diskless and stateless

  The OS is installed by using a different and contained method in the management node and loaded from the network. Changes are written to memory and discarded (not persistent).
- Diskless and statelite

  An intermediary type between stateless and stateful. The OS is installed by using a different and contained method in the management node and loaded from the network. Changes are written to memory and can be stored (persistent) or discarded (not persistent).

For more information about OS provisioning and state persistency, see xCAT IBM Power LE / OpenPOWER.

5.3.6 xCAT network interfaces: Primary and additional

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

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

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

For more information, see xCAT Configure Additional Network Interfaces - confignics.

Note: The Power AC922 server that is used for HPC shares the physical port for BMC and the primary network interface card (NIC) of the nodes. The BMC and primary NIC on each node have independent IPs on same network (management) but different subnets.

5.3.7 xCAT software kits

xCAT supports a software package format that is called *xCAT Software Kits* (*xCAT kits* or *kits*) that is tailored for installing software in xCAT clusters. Kits are used with some products of the IBM HPC software stack.

An xCAT software kit can include a software product's distribution packages, configuration information, scripts, and other files. It also includes xCAT-specific information for installing the appropriate product pieces, which are referred to as the *kit components*, to a particular node according to its environment and role in the xCAT cluster.

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

The incomplete or partial kits are indicated by file names with the `NEED_PRODUCT_PKGS` string, which can be converted to complete kits when combined with the product's distribution packages. Incomplete or partial kits are 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 OS installations.

For more information about xCAT software kits, see xCAT Software Kits.

5.3.8 xCAT synchronizing files

Synchronizing (sync) files to the nodes is a feature of xCAT that is used to distribute specific files from the management node to the new deploying or deployed nodes.

This function is supported for diskful or diskless nodes. Generally, the specific files are usually the system configuration files for the nodes in the `/etc` directory like `/etc/hosts` or `/etc/resolve.conf`. The files can also be the application programs configuration files for the nodes.

The advantages of this function are that is it can parallel-sync files to the nodes or nodegroup for the installed nodes, and it can automatically sync files to the newly installed node after the installation. Additionally, this feature also supports the flexible format to define the synced files in a configuration file that is called `synclist`.

The `synclist` file can be a common file for a group of nodes that use the same profile or OS image, or it can be the special one for a particular node. Considering that the location of the `synclist` file is used to find the `synclist` file, the common `synclist` is put in a specific location for Linux nodes or specified by the OS image.

For more information about xCAT file synchronization, see xCAT Synchronizing Files.

5.3.9 xCAT version

This section describes xCAT V2.14.3, which includes Red Hat Enterprise Linux-Alt Server 7.5 support for IBM PowerPC® 64-bit Little Endian (ppc64le) for the following functions:

- Provisioning types: Diskful and stateful, and diskless and stateless
- Support for Compute Unified Device Architecture (CUDA) Toolkit for NVIDIA GPUs
- Support for Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux for Mellanox InfiniBand adapters
- Support for kits with the IBM HPC Software
- Support for non-virtualized (or bare-metal) mode
- Power AC922 server:
  - Hardware discovery for BMCs
  - Hardware management with IPMI

For more information, see xCAT 2.14.3 release.

5.3.10 xCAT scenario for high-performance computing

This section describes the following xCAT networks and network addresses (it follows the network setup that is described in Figure 4-1 on page 34):

- Network address scheme: `10.network-number.0.0/16` (16-bit network mask)
- Management (OS) network (10 Gb Ethernet): `10.`*`1`*`.0.0/16`
- Service (BMC) network (1 Gb Ethernet): `10.`*`2`*`.0.0/16`
- High-performance interconnect (InfiniBand): `10.`*`10`*`.0.0/16`
- Site network (1 Gb Ethernet): `9.x.x.x/24`

Note: Depending on the adapters in the systems, the number and type of network interfaces can differ.

The network switch VLAN configuration can ensure that the management node can access all nodes' in-band and out-of-band (BMC) network interfaces, and that the non-management nodes can access only in-band network interfaces (but not out-of-band network interfaces).

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

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

The host names (or node names) are assigned to the POWER9 (thus, `p9`) processor-based compute nodes according to the following naming scheme:

- Node naming scheme: `p9r<rack-number>n<node-number-in-rack>`
- For example, for five racks and six systems per rack: `p9r1n1, p9r1n2, ..., p9r2n1, p9r2n2,... p9r5n6`

5.4 Initial xCAT management node installation on an IBM Power System LC922 server

This section describes the deployment of the xCAT management node with Red Hat Enterprise Linux Server 7.5 Alt for PowerPC 64-bit Little Endian (ppc64le) in non-virtualized (or bare-metal) mode on the Power LC922 server. The Power LC922 is a POWER9 processor-based server with two rear 3.5-inch disks and up to 12 front 3.5-inch disks. It is the best suited management server for a Power AC922 HPC cluster.

After you complete the steps that are described in this section, the management node is ready for the configuration and running of the xCAT node discovery process on other nodes in the cluster.

For more information, see Extreme Cluster and Cloud Administration Toolkit and see the following resources:

- *Installation Guide for Red Hat Enterprise Linux* (click **Install Guides** → **Installation Guide for Red Hat Enterprise Linux** → **Prepare the Management Node**.)
- *Configure xCAT* section (click **Admin Guide** → **Manage Clusters** → **IBM Power LE / OpenPOWER** → **Configure xCAT**.)

Note: The Power LC922 server is recommended for a management node. Although this specific type of node does not require the compute power that is found in the Power AC922 servers, it offers better storage capability.

5.4.1 Red Hat Enterprise Linux server

You can install Red Hat Enterprise Linux server by using one of the following methods:

- Virtual media (with the BMC GUI interface)
- USB device
- Network installation (for example, by using HTTP)

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

This section describes the installation process that uses two USB flash drives. This method is preferred because it is the easiest method and has the least dependencies. The OS is installed on the two rear disks by using this method.

For more information about other installation methods, see the following resources:

- Red Hat Enterprise Linux 7 Installation Guide
- Installing Linux on OPAL POWER9 systems

Prerequisites

The following components are required for the USB flash drive installation:

- A computer that is running Linux (preferred), macOS, or Windows
- Access to Red Hat Enterprise Linux Server 7.5 Alt ISO
- Two USB flash drives (one drive must be at least 4 GB)
- One of the following means to navigate in Petitboot:
  - Network connection to BMC and ipmitool (preferred)

– Serial connection
– VGA monitor and USB keyboard

Configuring the baseboard management controller IP

Before installing the system, configure the BMC so that the installation can be performed remotely from your desk:

1. Initialize the BMC configuration by using one of the following methods:
   a. Connect a USB keyboard and a VGA monitor to the server.
   b. Use a notebook with a serial connection to the server. For more information, see Connecting to your server with a serial console connection page.
2. Power on your server by pressing the power button on the front of your system. Your system powers on and shows the Petitboot panel. This process takes approximately 1 - 2 minutes to complete.

   Note: Do not walk away from your system. When Petitboot loads, your monitor becomes active and you must push any key to interrupt the boot process.

3. At the Petitboot bootloader main menu, select `Exit to shell`.
4. Set the BMC network settings for the first channel:
   a. Set the mode to static:
   ```
   $ ipmitool lan set 1 ipsrc static
   ```
   b. Set the BMC IP address:
   ```
   $ ipmitool admin lan set 1 ipaddr <ipaddr>
   ```
   c. Set the netmask:
   ```
   $ ipmitool admin lan set 1 netmask <netmask>
   ```
   d. Set the default gateway server:
   ```
   $ ipmitool lan set 1 defgw ipaddr <gateway_server>
   ```
5. Verify the new configuration by running the command that is shown in Example 5-9.

   Example 5-9 The ipmitool land print command
   ```
   $ ipmitool lan print 1
   <...>
   IP Address Source       : Static Address
   IP Address              : 9.x.x.x
   Subnet Mask             : 255.255.255.0
   MAC Address             : 70:e2:84:14:09:ad
   <...>
   Default Gateway IP      : 9.x.x.x
   Default Gateway MAC     : 98:be:94:00:47:84
   Backup Gateway IP       : 0.0.0.0
   Backup Gateway MAC      : 00:00:00:00:00:00
   802.1q VLAN ID          : Disabled
   802.1q VLAN Priority    : 0
   <...>
   ```
6. To activate the new configuration, you must reset the BMC by running the following command:
   ```
   $ ipmitool mc reset cold
   ```

7. Wait approximately 3 minutes. Then, attempt to ping the BMC IP to test the new configuration.

> **Note:** If your ping does not return successfully, complete the following steps:
>
> 1. Power off your system by running the **`ipmitool power off`** command.
> 2. Unplug the power cords from the back of the system. Wait 30 seconds and then apply power to boot the BMC.

Acquiring the network device name

To acquire the network device name for the Red Hat Enterprise Linux installation, complete the following steps:

1. Open the SOL console to the previously configured BMC (The default BMC user name and password is `root:0penBmc`. Note the zero in the first letter of the password.):

   ```
   $ ipmitool -I lanplus -H <hostname/IP> -P <password> sol activate
   ```

   > **Note:** Alternatively, you can use the serial connection at the node. For more information, see "Configuring the baseboard management controller IP" on page 66.

2. Check that the system is turned on:

   ```
   $ ipmitool -I lanplus -H <hostname/IP> -P <password> chassis power on
   ```

3. Wait for the Petitboot panel and choose `System information`, as shown in Example 5-10.

Example 5-10 Petitboot system information panel

```
Petitboot System Information
 ??????????????????????????????????????????????????????????????????????????????
 Network interfaces
 enp5p1s0f0:
  MAC:  98:be:94:68:ab:e8
  link: up

 enp5p1s0f1:
  MAC:  98:be:94:68:ab:e9
  link: down

 enp1s0f2:
  MAC:  98:be:94:68:ab:ea
  link: down

 enp1s0f3:
  MAC:  98:be:94:68:ab:eb
  link: down

 tunl0:
  MAC:  00:00:00:00:08:00
 ??????????????????????????????????????????????????????????????????????????????
 x=exit, h=help
```

Look for the `link: up` to locate which adapters have cables that are connected. In Example 5-10, `enP5p1s0f0` is the adapter that is connected to the site local area network (LAN) (all other cables are not connected yet). The network device name is needed as described in "Editing the kickstarter file for the second USB flash drive" on page 68.

Leave the serial console open because it is needed during the process to prepare the USB flash drives. To close the console, press the ~.

Preparing the USB flash drives

We created a kickstart file for you that helps you complete the basic installation and configuration ("Editing the kickstarter file for the second USB flash drive" on page 68) process. This kickstart file creates a software RAID1 (mdraid) on both rear disks, disables the firewall and selinux (required by xCAT), and configures the initial network interface.

Prepare the following USB flash drives:

- USB flash drive with the main Red Hat Enterprise Linux server 7.5 Alt installation ISO
- USB flash drive with the kickstarter configuration file

You must edit the kickstarter file and adapt it for your environment.

Note: You can add another partition to the first USB stick and use this partition for the kickstarter configuration. We want to keep the example configuration simple, so we use two separate USB flash drives.

Copying Red Hat Enterprise Linux ISO to the first USB flash drive

The following procedure assumes that you are using a Linux or a macOS system. Complete the following steps:

1. Download the Red Hat Enterprise Linux server 7.5-ALT installation ISO from Red Hat.
2. Run the **dd** command to copy the ISO file to the first USB flash drive, as shown in the following example:

   ```
   $ dd if=/home/user/Downloads/RHEL-ALT-7.5-20180315.0-Server-ppc64le-dvd1.iso
   of=/dev/<usb_device> bs=512k
   ```

Note: For more information about other OSs such as Windows, see Making Installation USB Media.

Editing the kickstarter file for the second USB flash drive

Complete the following steps:

1. To get the `smn_md.cfg` kickstarter file, see Appendix A, "Additional material" on page 327.
2. Open the `smn_md.cfg` file by using an editor of your choice and edit the first parameters, such as language, time zone, root password, and network for your environment, as shown in Example 5-11.

   Example 5-11 Editing the section of the smn_md.cfg kickstarter file

   ```
   ##########################
   #  Kickstart file for  #
   #  IBM LC922 server     #
   ##########################

   ######################
   # Edit the following #
   ######################

   #System language
   lang en_US
   ```

```
#System keyboard
keyboard us

#System timezone
timezone US/Eastern

#Root password. Default is cluster
rootpw --iscrypted $1$mzPFDiLI$spnL9ckOpRVD7LnxHbrVtO

#Network information
network --bootproto=static --ip=192.168.0.1 --netmask=255.255.255.0
--gateway=192.168.0.254 --nameserver=192.168.0.100,192.168.0.200
--device=enp5p1s0f0
network --hostname=xcat-mn.domain.com


###################################
# Do not touch anything below this #
###################################
<...>
```

You also might have to change the network device name. Match the device name with the ones in Example 5-10 on page 67 (they are in bold).

3. Generate a root password hash by running the following command:

   ```
   $ openssl passwd -1 <root_pw>
   $1$mzPFDiLI$spnL9ckOpRVD7LnxHbrVtO
   ```

   The resulting string in added to the `rootpw --iscrypted` line in the `.cfg` file. You also can use the preconfigured hash, which sets `cluster` as the root password.

4. Copy the edited kickstarter file to the second FAT32 formatted USB flash drive. Set the label of your USB drive to something informative, such as "KICKSTART".

5. Record the second USB flash drive UUID by running the following command:

   ```
   $ blkid
   /dev/sdc2: SEC_TYPE="msdos" UUID="BAE0-6F47" TYPE="vfat"
   ```

Starting the installation

To start the installation process, complete the following steps:

1. Put both USB keys only in the blue USB ports.
2. Open your serial console window or reconnect to the console.
3. Restart the server by running the following command:

   ```
   $ ipmitool -I lanplus -H <bmc-ip> -P <password> chassis power cycle
   ```

4. Wait for the Petitboot panel to load. Choose `Install Red Hat Enterprise Linux 7.5 (64-bit kernel)` and press E to edit the boot arguments, as shown in Example 5-12.

Example 5-12 Editing the boot arguments

```
<...>
[USB: sdn / 2018-10-19-18-33-07-00]
    Rescue a Red Hat Enterprise Linux system (64-bit kernel)
    Test this media & install Red Hat Enterprise Linux 7.5 (64-bit kernel)
 *  Install Red Hat Enterprise Linux 7.5 (64-bit kernel)
<...>
```

Note: Select `Rescan devices` if the USB device does not appear.

5. Change the boot arguments as shown in the following example:

```
ro inst.ks=hd:UUID=<UUID_KICKSTARTER>:/smn_md.cfg
inst.stage2=hd:UUID=<UUID_RHEL7_ISO>
```

Note: `<UUID_KICKSTARTER>` is your second USB flash drive with the kickstarter file that is described in "Editing the kickstarter file for the second USB flash drive" on page 68. `<UUID_RHEL7_ISO>` is your first USB flash drive with the Red Hat Enterprise Linux 7 ISO on it. It is the string in "[]" brackets after the device name in the edit menu that is shown in Example 5-13.

Example 5-13 shows the full string for our example.

Example 5-13 Boot argument string

```
Device:         (*) sdm [2018-10-19-18-33-07-00]
                ( ) Specify paths/URLs manually

 Kernel:         /ppc/ppc64/vmlinuz
 Initrd:         /ppc/ppc64/initrd.img
 Device tree:
 Boot arguments: ro inst.ks=hd:UUID=BAE0-6F47:/smn_md.cfg
inst.stage2=hd:UUID=2018-10-19-18-33-07-00
```

6. Click `OK`. Then, press Enter and Enter again to start the installation.

   The installation proceeds automatically. After the installation is finished, the system restarts.

   You must choose the correct boot target starting with `Disk` in Petitboot if you do not unplug the USB flash drive during the restart, as shown in Example 5-14.

Example 5-14 Choosing the correct boot target

```
[Disk: md126 / 62c0a482-733c-4e74-ac7c-22ee6483ab3d]
    Red Hat Enterprise Linux Server (0-rescue-eca7fb0809fa47b3b3f76eaa44869cd4)
 *  Red Hat Enterprise Linux Server (4.14.0-49.el7a.ppc64le) 7.5 (Maipo)
```

7. After the final disk boot, log in by using SSH with your root credentials that are configured in your kickstarter file.

Configuring the Red Hat Enterprise Linux Server 7.5 package repository

To install more packages and satisfy package dependencies for xCAT, configure a `yum` package repository for the Red Hat Enterprise Linux Server 7.5 packages.

You can configure the system for the Red Hat Enterprise Linux regular package update channels or at least the Red Hat Enterprise Linux Server 7.5 DVD 1 ISO. For simplicity, this section describes the Red Hat Enterprise Linux Server 7.5 DVD 1 ISO.

To configure the package repository for Red Hat Enterprise Linux Server 7.5 DVD 1, complete the following steps:

1. Copy the Red Hat Enterprise Linux Server 7.5 ISO content from the USB flash drive to the server, as shown in Example 5-15.

   Example 5-15 Copying the ISO content from the USB flash drive

   ```
   $ mkdir /mnt/usb
   $ mount -o ro /dev/disk/by-uuid/2018-10-19-18-33-07-00 /mnt/usb
   $ cp -r /mnt/usb /mnt/rhel7.5-dvd1
   $ chmod 500 /mnt/rhel7.5-dvd1
   $ umount /mnt/usb
   ```

2. Configure the repository:

   a. Install the RPM GPG key:

   ```
   $ rpm --import /mnt/rhel7.5-dvd1/RPM-GPG-KEY-redhat-release
   ```

   b. Create the yum package repository file, as shown in Example 5-16.

   Example 5-16 Creating the yum package repository file

   ```
   $ cat <<EOF >/etc/yum.repos.d/rhel7.5-dvd1.repo
   [rhel-7.5-dvd1]
   name=RHEL 7.5 Server DVD1
   baseurl=file:///mnt/rhel7.5-dvd1
   enabled=1
   gpgcheck=1
   EOF
   ```

You can verify that the package repository is configured correctly by running the command that is shown in Example 5-17.

Example 5-17 Verifying that the package repository is configured correctly

```
$ yum repolist
yum repolist
Loaded plugins: product-id, search-disabled-repos, subscription-manager
This system is not registered to Red Hat Subscription Management. You can use
subscription-manager to register.
repo id                              repo name
status
rhel-7.5-dvd1                        RHEL 7.5 Server DVD1
3.454
repolist: 3.454
```

Front disk formatting

Consider the following points as you configure the front drives for data:

- The `/install` directory for xCAT must be at least 500 GB because it consists of all your images and software packages.
- Track your logs and databases.
- Use LVM for flexible volume management.

The front drives can be formatted by using one of the following methods:

- Use the specific built-in hardware RAID controller. For more information, see the manufacturer's manual.
- Use LVM mirroring. For more information, see Red Hat Creating Mirrored Volumes.

5.4.2 xCAT packages and installation

xCAT is a collection of packages that are available for download at the xCAT project page.

The xCAT packages are organized into the following package repositories:

- xCAT Core Packages: Packages with the xCAT

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

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

xCAT is installed by using one of the following methods:

- Manually with online or offline RPM package repositories for the Red Hat Enterprise Linux, SUSE Linux Enterprise Server, and Debian packages (for Ubuntu).
- xCAT V2.12.1 added an installation tool called `go-xcat`. This tool simplifies the xCAT installation and update procedure and configures all the required repositories automatically.

This section describes the installation by using the new `go-xcat` tool. Complete the following steps:

1. Download the `go-xcat` tool, as shown in Example 5-18.

   Example 5-18 Downloading the go-xcat tool

   ```
   $ wget
   https://raw.githubusercontent.com/xcat2/xcat-core/master/xCAT-server/share/xcat
   /tools/go-xcat -O - >/tmp/go-xcat
   $ chmod +x /tmp/go-xcat
   ```

2. Run the tool without parameters to configure both xCAT repositories and install the latest version of xCAT, as shown in Example 5-19.

Note: You can specify a specific version of xCAT to install by using the `-x <version>` option.

Example 5-19 Running the go-xcat tool

```
$ /tmp/go-xcat install
Operating system:   linux
Architecture:       ppc64le
Linux Distribution: rhel
Version:            7.5

Reading repositories ...... done

xCAT is going to be installed.
Continue? [y/n] y
<... long yum output ...>
Complete!

xCAT has been installed!
========================

#If this is the first time xCAT has been installed, run the following
#commands to set environment variables into your PATH:

    for sh,
        `source /etc/profile.d/xcat.sh'
    or csh,
        `source /etc/profile.d/xcat.csh'
```

For an offline installation, you can specify the path to download the `xcat-core` and `xcat-dep` packages, as shown in the `go-xcat` help output in Example 5-20.

Example 5-20 The go-xcat help output

```
$ /tmp/go-xcat -h
go-xcat version 1.0.29

Usage: go-xcat [OPTION]... [ACTION]
Install xCAT automatically

Options:
Mandatory arguments to long options are mandatory for short options too.
  -h, --help                    display this help and exit
  --xcat-core=[URL]             use a different URL or path for the xcat-core
                                repository
  --xcat-dep=[URL]              use a different URL or path for the xcat-dep
                                repository
  -x, --xcat-version=[VERSION]  specify the version of xCAT; cannot use with
                                --xcat-core
  -y, --yes                     answer yes for all questions

Actions:
  install                       installs all the latest versions of xcat-core
```

```
                                and xcat-dep packages from the repository
  update                        updates installed xcat-core packages to the
                                latest version from the repository

Examples:
  go-xcat
  go-xcat install
  go-xcat update
  go-xcat --yes install
  go-xcat -x 2.12 -y install
  go-xcat --xcat-version=devel install
  go-xcat --xcat-core=/path/to/xcat-core.tar.bz2 \
          --xcat-dep=/path/to/xcat-dep.tar.bz2 install
  go-xcat --xcat-core=http://xcat.org/path/to/xcat-core.tar.bz2 \
          --xcat-dep=http://xcat.org/path/to/xcat-dep.tar.bz2 install

xCAT (Extreme Cluster and Cloud Administration Toolkit): <http://xcat.org/>
Full documentation at: <http://xcat-docs.readthedocs.io/en/stable/>
```

3. Verify that the package repositories are configured correctly, as shown in Example 5-21.

Example 5-21 Verifying that the repositories are configured correctly

```
$ yum repolist
<...>
repo id          repo name                 status
rhel-7.5-dvd1    RHEL 7.5 Server DVD1      3706+1
xcat-core        xCAT 2 Core packages      20
xcat-dep         xCAT 2 depedencies        32
repolist: 3.706
```

4. Verify that the xCAT service is running, as shown in Example 5-22.

Example 5-22 Verifying that the xCAT service is running

```
$ systemctl status xcatd
xcatd.service - xCAT management service
   Loaded: loaded (/usr/lib/systemd/system/xcatd.service; enabled; vendor
preset: disabled)
   Active: active (running) since Mon 2018-10-15 14:03:22 EDT; 2 days ago
<...>
```

5. Verify the version information and node type, as shown in Example 5-23.

Example 5-23 Verifying the version information and node type

```
$ source /etc/profile.d/xcat.sh
$ lsxcatd -a
Version 2.14.3 (git commit 7b7d9ab67589afec1ba58cd5138154290c529c0e, built Tue
Aug 21 07:16:00 EDT 2018)
This is a Management Node
dbengine=SQLite
```

5.4.3 Configuring more network interfaces

xCAT requires a static IP network configuration for the management node. The site network interface is configured during the kickstarter process. For all other networks, follow the instructions that are presented in this section.

Note: This requirement applies to any xCAT networks with communication between the management node and other nodes (for example, management and service networks).

You can configure an Ethernet network interface with a static IP address in one of many ways. This section describes the method that uses `sysconfig` or `ifcfg` files, and the **`nmcli`** command (Network Manager Command-Line Interface).

For more information, see Red Hat Enterprise Linux 7 Networking Guide at Red Hat documentation.

This section uses content from the following sections of the Red Hat Enterprise Linux 7 Networking Guide:

- Section 1.9, “Network configuration by using sysconfig files”
- Section 2.4.1, “Configuring a network interface with ifcfg files”

To configure a network interface with a static IP address, complete the following steps for the management (OS) network:

1. Create the `/etc/sysconfig/network-scripts/ifcfg-<`*`network-interface`*`>` file.

   For the scenario that is described in this chapter, the file resembles the following example:

   ```
   $ cat <<EOF >/etc/sysconfig/network-scripts/ifcfg-enP5p1s0f0
   DEVICE=enP5p1s0f0
   ONBOOT=yes
   BOOTPROTO=none
   IPADDR=10.1.0.1
   PREFIX=16
   IPV6INIT=yes
   EOF
   ```

2. Verify that the network configuration is not in effect immediately (no IPv4 or IPv6 address):

   ```
   $ ip addr show enP5p1s0f0
   4: enP5p1s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
   qlen 1000
       link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
   ```

3. Reload the network configuration for that network interface by running the **`nmcli`** command. The network configuration is loaded automatically on system boot.

   ```
   $ nmcli connection load /etc/sysconfig/network-scripts/ifcfg-enP51s0f0
   ```

4. Verify that the network configuration is in effect (including an IPv6 link-local address):

   ```
   $ ip addr show enP5p1s0f0
   4: enp1s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen
   1000
       link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
       inet 10.1.0.1/16 brd 10.1.255.255 scope global enP5p1s0f0
          valid_lft forever preferred_lft forever
       inet6 fe80::9abe:94ff:fe59:fa26/64 scope link
          valid_lft forever preferred_lft forever
   ```

5.4.4 Host name and alias

Although the kickstarter file configured the host name, you must verify that everything is set up correctly.

Verify that the short and long (fully qualified domain name) host names are detected:

```
$ hostname --short
xcat-mn

$ hostname --long
xcat-mn.xcat-cluster
```

Configure the host name to be resolved to the IP address in the management (OS) network by completing the following steps:

1. Add the host aliases in the `/etc/hosts` file:

```
$ echo "10.1.0.1 $(hostname -s) $(hostname -f)" >> /etc/hosts
$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.1.0.1    xcat-mn xcat-mn.xcat-cluster
```

2. Verify that the short host name resolves to the long host name, and that the **ping** test works:

```
$ ping -c1 xcat-mn
PING xcat-mn.xcat-cluster (10.1.0.1) 56(84) bytes of data.
64 bytes from xcat-mn.xcat-cluster (10.1.0.1): icmp_seq=1 ttl=64 time=0.025 ms

--- xcat-mn.xcat-cluster ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.025/0.025/0.025/0.000 ms
```

5.4.5 xCAT networks

The xCAT networks configuration is stored the `networks` table, and is available as objects of the `network` type.

Running the **makenetworks** command populates the `networks` table based on the current configuration of the network interfaces. It is run automatically during the installation of xCAT packages.

You can use the following commands to create, list, modify, and remove `network` objects, and list and edit the `networks` table:

- **mkdef -t network**
- **lsdef -t network**
- **chdef -t network**
- **rmdef -t network**
- **tabdump networks**
- **tabedit networks**

To configure the xCAT networks, populate the `networks` table by running the **makenetworks** command, remove any non-xCAT networks, rename the xCAT networks (optional), and create any other xCAT networks.

For this scenario, complete the following steps:

1. Populate the `networks` table by running the **`makenetworks`** command:

   ```
   $ makenetworks

   $ lsdef -t network
   10_1_0_0-255_255_0_0  (network)
   10_2_0_0-255_255_0_0  (network)
   9_x_x_x-255_255_255_0  (network)
   fd55:xxxx:xxxx:33e::/64  (network)
   ```

2. Remove any non-xCAT networks:

   ```
   $ rmdef -t network 9_x_x_x-255_255_255_0
   1 object definitions have been removed.

   $ rmdef -t network fd55:xxxx:xxxx:33e::/64
   1 object definitions have been removed.

   $ lsdef -t network
   10_1_0_0-255_255_0_0  (network)
   10_2_0_0-255_255_0_0  (network)
   ```

3. (Optional) Rename the xCAT networks:

   ```
   $ chdef -t network 10_1_0_0-255_255_0_0 -n net-mgmt
   Changed the object name from 10_1_0_0-255_255_0_0 to net-mgmt.

   $ chdef -t network 10_2_0_0-255_255_0_0 -n net-svc
   Changed the object name from 10_1_0_0-255_255_0_0 to net-mgmt.

   $ lsdef -t network
   net-mgmt  (network)
   net-svc (network)
   ```

4. Create any other xCAT networks:

   ```
   $ mkdef -t network net-app-ib net=10.10.0.0 mask=255.255.0.0
   1 object definitions have been created or modified.

   $ lsdef -t network
   net-app-ib (network)
   net-mgmt  (network)
   net-svc  (network)
   ```

   Note: The application networks lack the **`mgtifname`** (management interface name) attribute because the management node is not connected to them, that is, only some other nodes are, such as the compute nodes and login nodes.

   However, application networks must be defined in the management node for it to perform their network configuration on other nodes (by way of the management network).

5. Verify the xCAT networks configuration.

   You can run the **lsdef** command to list the configuration of a specific xCAT network or networks. The following example lists the management network and application network for InfiniBand port 1:

```
$ lsdef -t network net-mgmt,net-app-ib
Object name: net-mgmt
    gateway=<xcatmaster>
    mask=255.255.0.0
    mgtifname=enP5p1s0f0
    net=10.1.0.0
    tftpserver=10.1.0.1
Object name: net-app-ib
    mask=255.255.0.0
    net=10.10.0.0
```

   You can run the **tabdump** command to list the configuration of all xCAT networks:

```
$ tabdump networks
#netname,net,mask,mgtifname,<...>
"net-mgmt","10.1.0.0","255.255.0.0","enP5p1s0f0",<...>
"net-svc","10.2.0.0","255.255.0.0","enp1s0f3",<...>
"net-app-ib","10.10.0.0","255.255.0.0",,,,,,,,,,,,,,,
```

5.4.6 DNS server

The xCAT configures the DNS server based on the attributes of the `site` table, the `/etc/hosts` file, and node definitions. The **makedns** command applies the configuration.

The following attributes of the `site` table are used to configure the DNS server:

`dnsinterfaces`:	Host name (optional) and network interfaces for the DNS server to listen on.
`domain`:	DNS domain name for the cluster.
`forwarders`:	DNS servers for resolving non-cluster names, that is, the site's or external DNS servers.
`master`:	IP address of the xCAT management node on the management (OS) network, as known by the nodes.
`nameservers`:	DNS servers that are used by the compute nodes (usually, the IP address of the management node). The value <*xcatmaster*> indicates that the management node or service node that is managing a node (automatically defined to the correct IP address in the respective xCAT network) is more portable.

For more information, see the manual page of the `site` table by running the following command:

```
$ man 5 site
```

Running the **makedns** command generates the following configuration files for the DNS server, and reloads it:

- `/etc/named.conf`: Main configuration file (generated by **makedns -n**)
- `/var/named/*`: Zone files for network names and addresses (generated by **makedns -n**)

To configure the DNS server, complete the following steps:

1. Check the following basic configuration parameters in your site table:

   ```
   $ lsdef -t site -i dnsinterfaces,domain,forwarders,master,nameservers
   Object name: clustersite
       domain=xcal-cluster
       forwarders=9.x.x.x,9.x.x.x
       master=10.1.0.1
       nameservers=10.1.0.1
   ```

   Note: To get a detailed description for each parameter, run the **`tabdump -d site`** command.

2. The `dnsinterfaces` attribute of the `site` table is not configured yet. Set the attribute by running the following **`chdef`** command:

   ```
   $ chdef -t site dnsinterfaces='xcat-mn|enP5p1s0f0'
   1 object definitions have been created or modified.
   ```

3. Generate new configuration files for the DNS server by running the **`makedns -n`** command. The DNS server is automatically started or restarted as follows.

   ```
   $ makedns -n
   Handling xcat-mn.xcat-cluster in /etc/hosts.
   Handling localhost in /etc/hosts.
   Handling localhost in /etc/hosts.
   Getting reverse zones, this  take several minutes for a large cluster.
   Completed getting reverse zones.
   Updating zones.
   Completed updating zones.
   Restarting named
   Restarting named complete
   Updating DNS records, this  take several minutes for a large cluster.
   Completed updating DNS records.
   ```

   Note: It is important that no errors are reported in the output of the **`makedns`** command. The proper functioning of the DNS server is essential to several features in xCAT (for example, node discovery).

   If any errors are reported, check the messages, the contents of the `/etc/hosts` file, and any node definitions (by running the **`lsdef`** command) for errors or inconsistencies.

4. Verify that the DNS server is resolving internal and external names with the **`host`** command by completing the following steps:

   a. Install the `bind-utils` package (not installed with the minimal installation package set):

   ```
   $ yum install bind-utils
   ```

   b. Verify the name resolution of internal names. For example, the management node (that is, its short and long host names are associated and are resolved to its IP address in the management network):

   ```
   $ host xcat-mn 10.1.0.1
   Using domain server:
   Name: 10.1.0.1
   Address: 10.1.0.1#53
   Aliases:
   ```

```
xcat-mn.xcat-cluster has address 10.1.0.1
```

c. Verify the name resolution of external names:

```
$ host example.com 10.1.0.1
Using domain server:
Name: 10.1.0.1
Address: 10.1.0.1#53
Aliases:

example.com has address 93.184.216.34
example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

5.4.7 DHCP server

xCAT configures the DHCP server based on the attributes of the `site` and `networks` tables and the node definitions (for the reservation of IP address leases that are based on a MAC address). The **makedhcp** command applies the configuration.

The following attributes of the `site` and `networks` tables are used to configure the DHCP server:

`dhcpinterfaces` (`site` table):	Host name (optional) and network interfaces on which the DHCP server listens.
`dynamicrange` (`networks` table):	Range of IP addresses that are *temporarily* assigned during the node discovery process, which are required in the xCAT management and service networks.

Running the **makedhcp** command generates the following configuration files for the DHCP server and reloads it:

- `/etc/dhcp/dhcpd.conf`: Main configuration file (generated by **makedhcp -n**)
- `/var/lib/dhcpd/dhcpd.leases`: IP address leases (generated by **makedhcp -a**)

Note: The **:noboot** flag in the networks table prevents the distribution of the Genesis image in the network.

To configure the DHCP server, complete the following steps:

1. Set the `dhcpinterfaces` attribute of the `site` table by running the **chdef** command (note the **:noboot** flag), as shown in the following example for the scenario in this chapter:

   ```
   $ chdef -t site dhcpinterfaces='xcat-mn|enP5p1s0f0,xcat-mn|enp1s0fx:noboot'
   1 object definitions have been created or modified.
   ```

   You can verify the attributes by running the **lsdef** command:

   ```
   $ lsdef -t site -i dhcpinterfaces
   Object name: clustersite
       dhcpinterfaces=xcat-mn|enp1s0f0,xcat-mn|enp1s0fx:noboot
   ```

2. Remove the Genesis configuration file from the **:noboot** tagged network by running the **mknb** command.

   Note: The **mknb** (make network boot) command generates the network boot configuration file (specified in the /etc/dhcp/dhcpd.conf file) and Genesis image files. It is run automatically when the xCAT packages are installed.

   ```
   $ mknb ppc64
   Creating genesis.fs.ppc64.gz in /tftpboot/xcat
   ```

3. Generate the configuration file for the DHCP server by running the **makedhcp -n** command. The DHCP server is automatically started or restarted.

   ```
   $ makedhcp -n
   Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak

   The dhcp server must be restarted for OMAPI function to work
   Warning: No dynamic range specified for 10.1.0.0. If hardware discovery is
   being used, a dynamic range is required.
   Warning: No dynamic range specified for 10.2.0.0. If hardware discovery is
   being used, a dynamic range is required.
   ```

   Despite the message that is related to the need to restart the DHCP server, it is automatically restarted, as shown in the /var/log/messages file:

   ```
   $ tail /var/log/messages
   <...>
   <...> xcat[...]: xCAT: Allowing makedhcp -n for root from localhost
   <...> systemd: Starting DHCPv4 Server Daemon...
   <...> dhcpd: Internet Systems Consortium DHCP Server 4.2.5
   <...>
   ```

4. Generate the leases file for the DHCP server by running the **makedhcp -a** command. This step is required only if any node definitions exist (they do not yet exist in the scenario in this chapter).

   ```
   $ makedhcp -a
   ```

5.4.8 Intelligent Platform Management Interface credentials

xCAT configures the authentication credentials for IPMI commands (for example, power management, console sessions, BMC discovery, and network configuration) based on attributes from (in this order) node definitions, node groups definitions, and the passwd table.

To configure IPMI authentication credentials on individual nodes or on node groups, set the bmcpassword attribute on the node or node group object by running the **chdef** command:

```
$ chdef <node or group> bmcusername=”” bmcpassword=<IPMI password>
```

If the IPMI authentication credentials are common across some or all of the systems’ BMCs, you can set the common credentials in the passwd table. Any different credentials can be set in the respective node or node group objects.

For more information, see Configure passwords page of the xCAT documentation website.

To configure the IPMI authentication credentials, complete the following steps:

1. Set the `username` and `password` attributes of the `ipmi` key or row in the `passwd` table by running the **`chtab`** or **`tabedit`** commands:

   ```
   $ chtab key=ipmi passwd.password=OpenBmc
   ```

2. Verify the setting by running the **`tabdump`** command:

   ```
   $ tabdump -w key==ipmi passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "ipmi","","OpenBmc",,,,
   ```

   You can run the **`tabdump`** command without filter arguments to list the configuration of all entries in the `passwd` table:

   ```
   $ tabdump passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "omapi","xcat_key","<...>=",,,,
   "ipmi","","OpenBmc",,,,
   ```

   Note: If the IPMI authentication credentials are not set or invalid, some IPMI-based commands can show errors:

   ```
   $ rpower node status
   node: Error: Unauthorized name
   ```

5.5 xCAT node discovery

This section describes the xCAT node discovery (or hardware discovery) process. It covers the configuration steps that are required in the management node and instructions for performing the discovery of nodes in the cluster. xCAT provides the following methods for node discovery:

- Manual definition

  This method includes manual hardware information collection and node object definition. This example includes required node-specific and xCAT and platform-generic attributes:

  ```
  $ mkdef node1 \
     groups=all,ac922 \
     ip=10.1.1.1 mac=6c:ae:8b:6a:d4:e installnic=mac primarynic=mac \
     bmc=10.2.1.1 bmcpassword=OpenBmc \
     mgt=ipmi cons=ipmi netboot=petitboot
  ```

- MTMS-based discovery

  This process automatically collects MTM and S/N information from the node's BMC and OS (Genesis), and matches it with a minimal manually defined node object (with the `mtm` and `serial` attributes). This process automatically stores the hardware information in the matched node object.

- Sequential discovery

  This method automatically stores hardware information in a list of minimal node objects (with no attributes) in a sequential manner in the order that the nodes are booted.
- Switch-based discovery

  This method automatically identifies the network switch and port for the node (with the SNPMv3 protocol) and matches it with a minimal manually defined node object (with the `switch` and `switchport` attributes). This process automatically stores the hardware information in the matched node object.

This section describes the MTMS-based discovery method.

5.5.1 Verification of network boot configuration and Genesis image files

xCAT node discovery requires the node to boot the Genesis image from the network. It is important to verify that the files for network boot configuration and Genesis image are correctly in place.

To verify and generate the files for network boot configuration and the Genesis image, complete the following steps:

1. Verify the location of the platform-specific network boot configuration file in the `/etc/dhcp/dhcpd.conf` file. The scenario that is adopted in this chapter uses OPAL firmware.

   ```
   $ grep -A1 OPAL /etc/dhcp/dhcpd.conf
      } else if option client-architecture = 00:0e { #OPAL-v3
         option conf-file = "http://10.1.0.1/tftpboot/pxelinux.cfg/p/10.1.0.0_16";
   ```

   Note: The network boot configuration and Genesis image files are required only for the xCAT management network.

2. Verify that the file exists:

   ```
   $ ls /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   ```

   Note: The specified file might not exist in some cases, such as when the **mknb** command is not run after a change in the xCAT networks configuration. If this issue occurs, run **mknb ppc64** again.

3. Verify the content of the file. It must contain pointers to the platform-specific Genesis image files.

   ```
   $ cat /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   default "xCAT Genesis (10.1.0.1)"
      label "xCAT Genesis (10.1.0.1)"
      kernel http://10.1.0.1:80//tftpboot/xcat/genesis.kernel.ppc64
      initrd http://10.1.0.1:80//tftpboot/xcat/genesis.fs.ppc64.gz
      append "quiet xcatd=10.1.0.1:3001 "
   ```

4. Verify that the files of the Genesis image exist:

   ```
   $ ls -lh /tftpboot/xcat/genesis.{kernel,fs}.ppc64*
   -rw-r--r--. 1 root root 123M Oct 11 15:15 /tftpboot/xcat/genesis.fs.ppc64.gz
   -rwxr-xr-x. 1 root root  23M Apr  4  2018 /tftpboot/xcat/genesis.kernel.ppc64
   ```

Tip: To increase the verbosity of the node discovery in the nodes (which is useful for educational and debugging purposes), remove the **quiet** argument from the append line in the network boot configuration file by running the following command:

```
$ sed 's/quiet//' -i /tftpboot/pxelinux.cfg/p/10.1.0.0_16
```

5.5.2 Configuring the DHCP dynamic range

xCAT node discovery requires temporary IP addresses for nodes and BMCs until the association with the respective node objects and permanent network configuration occur. The IP address range that is reserved for that purpose is known as *dynamic range*, which is an attribute of network objects and is reflected in the configuration file of the DHCP server.

You must provide temporary IP addresses for the Management and Service Networks to handle the *in-band* network interface (used by the Petitboot bootloader and Genesis image) and *out-of-band* network interface (used by the BMC).

Depending on your network environment and configuration, the management node can use different or shared network interfaces for the Management and Service Networks. This consideration is important because the dynamic range is defined *per network interface* in the configuration file of the DHCP server. As described in 5.3.10, “xCAT scenario for high-performance computing” on page 64, the following networks are recommended:

- For different network interfaces, set the **dynamicrange** attribute on the Management and Service Networks.
- For a shared network interface, set the **dynamicrange** attribute in either one of the Management or Service Networks.

To set the dynamic range, complete the following steps:

1. Set the **dynamicrange** attribute for the management network by running the **chdef** command, as shown in the following example for the scenario in this chapter:

   ```
   $ chdef -t network net-mgmt dynamicrange=10.1.254.1-10.1.254.254
   1 object definitions have been created or modified.
   ```

   You can verify it by running the **lsdef** command:

   ```
   $ lsdef -t network net-mgmt -i dynamicrange
   Object name: net-mgmt
       dynamicrange=10.1.254.1-10.1.254.254
   ```

2. Generate the configuration file for the DHCP server by running the **makedhcp -n** command:

   ```
   $ makedhcp -n
   Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak
   ```

3. Verify the dynamic range in the configuration of the DHCP server:

   ```
   $ grep 'network\|subnet\|_end\|range' /etc/dhcp/dhcpd.conf
   ```

5.5.3 Configuring BMCs to DHCP mode

xCAT node discovery requires the BMCs' network configuration to occur in DHCP mode (until the association with the respective node objects and permanent network configuration occur).

Note: If the BMCs' network configuration cannot be changed (for example, because of network restrictions or maintenance requirements), skip the required steps for the BMC network configuration in the node discovery process. For more information, see 5.5.4, "Definition of temporary BMC objects" on page 87. Then, manually set the `bmc` attribute of one or more nodes to the respective BMC IP address.

To set the network configuration of the BMCs to DHCP mode, complete the following tasks:

- For BMCs that are in DHCP mode, no action is required.
- For BMCs with Static (IP) Address mode and a known IP address, complete the following steps:

  a. Set the `IP Address Source` attribute to `DHCP` by running the **`ipmitool`** command. Notice that the IP Address Source attribute is set to `Static Address`.

  ```
  $ ipmitool -I lanplus -H <ip> -P <password> lan print 1
  <...>
  IP Address Source       : Static Address
  IP Address              : 192.168.101.29
  Subnet Mask             : 255.255.255.0
  MAC Address             : 70:e2:84:14:02:54
  <...>
  ```

  Set it to DHCP:

  ```
  $ ipmitool -I lanplus -H <ip> -P <password> lan set 1 ipsrc dhcp
  ```

  Notice that the network configuration changes do not take effect immediately:

  ```
  $ ipmitool -I lanplus -H <ip> -P <password> lan print 1
  <...>
  IP Address Source       : DHCP Address
  IP Address              : 192.168.101.29
  Subnet Mask             : 255.255.255.0
  MAC Address             : 70:e2:84:14:02:54
  <...>
  ```

  b. Restart the BMC by running the **`ipmitool`** command, which is required for the network configuration changes to take effect:

  ```
  $ ipmitool -I lanplus -H <ip> -P <password> mc reset cold
  ```

  c. Wait for the BMC to perform initialization and network configuration.

  To determine when the BMC is back online and has acquired an IP address through DHCP, you can watch the `/var/log/messages` file for DHCP server log messages, as shown in the following example:

  ```
  $ tail -f /var/log/messages
  <...>
  <...> dhcpd: DHCPDISCOVER from 70:e2:84:14:02:54 via enP5p1s0f0
  <...> dhcpd: DHCPOFFER on 10.2.254.1 to 70:e2:84:14:02:54 via enP5p1s0f0
  <...> dhcpd: DHCPREQUEST for 10.2.254.1 (10.1.0.1) from 70:e2:84:14:02:54
  via enP5p1s0f0
  <...> dhcpd: DHCPACK on 10.2.254.1 to 70:e2:84:14:02:54 via enp1s0f3
  <...>
  ```

It is also possible to determine when the BMC is back online with an approach that is based on its IPv6 link-local address, which does not change across power cycles, network configuration steps, and so on (see Example 5-24). To identify the IPv6 link-local address of each BMC in a local network, see the “For BMCs with unknown IP address” bullet.

Example 5-24 Waiting for the BMC with the ping6 command and IPv6 link-local address

```
$ while ! ping6 -c 1 <IPv6-link-local-address>%<network-interface>; do echo
Waiting; done; echo Finished
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) ...
<...>
Waiting
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254)...
<...>
Waiting
<...>
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) ...
64 bytes from fe80::72e2:84ff:fe14:254: icmp_seq=1 ttl=64 time=0.669 ms
<...>
Finished
```

d. Verify that the network configuration changes are in effect by running the **ipmitool** command:

```
$ ipmitool -I lanplus -H <ip> -P <password> lan print 1
<...>
IP Address Source       : DHCP Address
IP Address              : 10.2.254.1
Subnet Mask             : 255.255.0.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

- For BMCs with unknown IP address (in DHCP or Static Address mode), complete the following steps:

a. Install the `nmap` package:

```
$ yum install nmap
```

b. Discover one or more IPv6 link-local addresses of one or more BMCs by running the **nmap** command (see Example 5-25).

Example 5-25 Discovering the IPv6 link-local address of BMCs with the nmap command

```
$ nmap -6 --script=targets-ipv6-multicast-echo -e enP5p1s0f0

Starting Nmap 6.40 ( http://nmap.org ) at <...>
Pre-scan script results:
| targets-ipv6-multicast-echo:
|   IP: fe80::9abe:94ff:fe59:f0f2  MAC: 98:be:94:59:f0:f2  IFACE: enP5p1s0f0
|   IP: fe80::280:e5ff:fe1b:fc99   MAC: 00:80:e5:1b:fc:99  IFACE: enP5p1s0f0
|   IP: fe80::280:e5ff:fe1c:9c3    MAC: 00:80:e5:1c:09:c3  IFACE: enP5p1s0f0
|   IP: fe80::72e2:84ff:fe14:259   MAC: 70:e2:84:14:02:59  IFACE: enP5p1s0f0
|   IP: fe80::72e2:84ff:fe14:254   MAC: 70:e2:84:14:02:54  IFACE: enP5p1s0f0
|_  Use --script-args=newtargets to add the results as targets
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 2.37 seconds
```

c. Perform the steps that are described in the “Static (IP) Address mode and known IP address” case. Replace the BMC’s IPv4 address in the **ipmitool** command for the IPv6 link-local address with the network interface as `zone index`, which is separated by the percent (%) symbol, as shown in the following example:

   <IPv6-link-local-address>%<network-interface>

5.5.4 Definition of temporary BMC objects

xCAT node discovery requires temporary node objects for BMCs until the association with the respective node objects and permanent network configuration occurs.

The temporary BMC objects are created by running the **bmcdiscover** command, which scans an IP address range for BMCs and collects information, such as machine type and model, serial number, and IP address. It can provide that information as objects in the xCAT database or the respective *stanzas* (a text-based description format with object name, type, and attributes). The objects are named after their MTMS information (which are obtained through IPMI).

The temporary BMC objects are automatically removed during the node discovery process after the respective node objects are matched and ready to refer to the BMCs. It is a simple task to have xCAT objects refer to the BMCs that are using temporary IP addresses (not yet associated with the respective node objects). This configuration allows for running xCAT commands (for example, power management and console sessions) before the network configuration of the BMC occurs.

The **bmcdiscover** command has the following requirements:

- The BMCs’ IP addresses must be within a known range (the dynamic range configuration, as described in 5.5.2, “Configuring the DHCP dynamic range” on page 84 and BMCs in DHCP mode, as described in 5.5.3, “Configuring BMCs to DHCP mode” on page 85).
- The IPMI authentication credentials must be defined in the `passwd` table (see “Intelligent Platform Management Interface credentials” on page 81) or by using command arguments.

The **bmcdiscover** command can be used with the following arguments:

-z:	Provides an object definition stanza.
-w:	Writes objects to the xCAT database.

To define temporary objects for the BMCs, complete the following steps:

1. Run the **bmcdiscover** command on the dynamic range of IP addresses:

```
$ bmcdiscover --range 10.2.254.1-254 -w
node-8335-gth-7882AAA:
        objtype=node
        groups=all
        bmc=10.2.254.1
        cons=ipmi
        mgt=ipmi
        mtm=8335-GTH
        serial=7882AAA
        nodetype=mp
        hwtype=bmc
```

2. Verify that the BMC objects are listed as node objects by running the **lsdef** command. Note the attributes or values `nodetype=mp` and `hwtype=bmc`.

```
$ lsdef
node-8335-gth-7882AAA (node)

$ lsdef node-8335-gth-7882AAA
Object name: node-8335-gth-7882AAA
    bmc=10.2.254.1
    cons=ipmi
    groups=all
    hwtype=bmc
    mgt=ipmi
    mtm=8335-GTH
    nodetype=mp
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=7882AAA
```

When BMC objects and IPMI authentication credentials are defined, you can run xCAT commands on the BMC objects, as shown in the following examples:

- The **rpower** command for power management
- The **rcons** command for console sessions (requires the **makeconservercf** command first)
- The **rsetboot** command for the boot-method selection

Note: It is always possible to run **ipmitool** commands to the BMC IP address as well.

5.5.5 Defining node objects

xCAT node discovery requires minimal node objects that can *match* the MTMS information, which can be collected either automatically by the Genesis image (by using the **mtm** and **serial** attributes) or manually. For more information, see 5.2.1, “The Intelligent Platform Management Interface tool” on page 52.

The node discovery also attempts to match the information with the temporary BMC objects to associate the node objects with their respective BMCs and perform the network configuration of the BMCs.

This section describes creating a node group to set the attributes that are common among nodes or based on regular expressions. For more information about xCAT regular expressions, see *Groups and Regular Expressions in Tables: Using Regular Expressions in the xCAT Tables*.

To define a node group, complete the following steps:

1. Create the `ac922` node group by running the **mkdef** command:

```
$ mkdef -t group ac922\
  ip='|p9r(\d+)n(\d+)|10.1.($1+0).($2+0)|' \
  bmc='|p9r(\d+)n(\d+)|10.2.($1+0).($2+0)|' \
  mgt=ipmi \
  cons=ipmi
Warning: Cannot determine a member list for group 'ac922'.
1 object definitions have been created or modified.
```

2. Verify the node group by running the **lsdef** command:

```
$ lsdef -t group ac922
Object name: ac922
    bmc=|p9r(\d+)n(\d+)|10.2.($1+0).($2+0)|
    cons=ipmi
    grouptype=static
    ip=|p9r(\d+)n(\d+)|10.1.($1+0).($2+0)|
    members=
    mgt=ipmi
```

To create a node that is part of the node group, complete the following steps:

1. Create a node by running the **mkdef** command and include the node group in the `groups` attribute. You can also modify a node and make it part of a group by running the **chdef** command.

```
$ mkdef p9r1n1 groups=all,ac922
1 object definitions have been created or modified.
```

   To create many nodes at the same time, you can use a node range:

```
$ mkdef p9r[1-5]n[1-6] groups=all,ac922
30 object definitions have been created or modified.
```

2. Verify that the node inherits the node group's attributes by running the **lsdef** command. Notice that the attributes that are based on regular expressions are evaluated according to the name of the node. Some other attributes are set by xCAT by default.

```
$ lsdef p9r1n1
Object name: p9r1n1
    bmc=10.2.1.1
    cons=ipmi
    groups=all,ac922
    ip=10.1.1.1
    mgt=ipmi
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfile,confignics
```

To set the `mtm` and `serial` attributes to match the BMC object, complete the following steps:

1. Set the attributes by running the **chdef** command. You can also set the attributes when creating the node object by running the **mkdef** command.

```
$ chdef p9r1n1 mtm=8335-GTH serial=7882AAA
1 object definitions have been created or modified.
```

> **Note:** The `mtm` and `serial` attributes are case-sensitive.

2. Verify the attributes by running the **lsdef** command:

```
$ lsdef p9r1n1
Object name: p9r1n1
    bmc=10.2.1.1
    cons=ipmi
    groups=all,s822lc
    ip=10.1.1.1
    mgt=ipmig
    mtm=8335-GTH
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfile,confignics
    serial=7882AAA
```

3. Enable the **bmcsetup** script for all nodes in the `ac922` group to set the BMC IP to static on first boot:

   ```
   $ chdef -t group ac922chain="runcmd=bmcsetup"
   ```

5.5.6 Configuring the host table, DNS, and DHCP servers

xCAT requires the node objects to be present and up-to-date in configuration files for the host table, DNS server, and DHCP server.

Note: This step is important for the node discovery process, which otherwise can show errors that are difficult to trace to specific misconfiguration steps.

The configuration files must be updated after the following changes are made, which are reflected in the configuration files:

- Adding or removing node objects
- Adding, modifying, or removing host names, IP addresses, or aliases for network interfaces

The order of the commands is relevant because some commands depend on changes that are performed by other commands. For more information, see the man pages of the **makehosts**, **makedns**, and **makedhcp** commands:

- **$ man makehosts**
- **$ man makedns**
- **$ man makedhcp**

To update the configuration files with the node objects, complete the following steps:

1. Update the host table by running the **makehosts** command:

   ```
   $ makehosts ac922
   ```

2. Verify that the node objects are present on the host table:

   ```
   $ cat /etc/hosts
   127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
   ::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
   10.1.0.1    xcat-mn.xcat-cluster xcat-mn
   10.1.1.1    p9r1n1 p9r1n1.xcat-cluster
   ```

3. Update the DNS server configuration by running the **makedns** command:

   ```
   $ makedns -n ac922
   Handling p9r1n1 in /etc/hosts.
   Getting reverse zones, this  take several minutes for a large cluster.
   Completed getting reverse zones.
   Updating zones.
   Completed updating zones.
   Restarting named
   Restarting named complete
   Updating DNS records, this  take several minutes for a large cluster.
   Completed updating DNS records.
   ```

4. Verify that the DNS server can resolve the name of the node by running the **host** command:

```
$ host p9r1n1 10.1.0.1
Using domain server:
Name: 10.1.0.1
Address: 10.1.0.1#53
Aliases:

p9r1n1.xcat-cluster has address 10.1.1.1
```

5. Update the DHCP server's configuration file by running the **makedhcp** command:

```
$ makedhcp -n
```

6. Update the DHCP server's leases file by running the **makedhcp** command:

```
$ makedhcp -a
```

5.5.7 Booting into node discovery

You can boot the nodes into node discovery during power on (or power cycle) if the boot order configuration is correct. For more information, see 5.2.4, "Boot order configuration" on page 57.

You can watch the progress of the node discovery process in the `/var/log/messages` file, which is described with comments in Example 5-26.

Example 5-26 Contents and comments for the /var/log/messages file during node discovery

```
$ tail -f /var/log/messages
...

Petitboot (DHCP client acquires an IP address, and releases it before booting):

... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPOFFER on 10.1.254.4 to 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPREQUEST for 10.1.254.4 (10.1.0.1) from 98:be:94:59:f0:f2 via
enP5p1s0f0
... dhcpd: DHCPACK on 10.1.254.4 to 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPRELEASE of 10.1.254.4 from 98:be:94:59:f0:f2 via enP5p1s0f0 (found)

Genesis (DHCP client acquires an IP address):

... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPOFFER on 10.1.254.5 to 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPREQUEST for 10.1.254.5 (10.1.0.1) from 98:be:94:59:f0:f2 via
enP5p1s0f0
... dhcpd: DHCPACK on 10.1.254.5 to 98:be:94:59:f0:f2 via enP5p1s0f0

Genesis (Communication with the xCAT Management Node; some error messages and
duplicated steps are apparently OK):

... xcat[30280]: xCAT: Allowing getcredentials x509cert
... xcat[17646]: xcatd: Processing discovery request from 10.1.254.5
... xcat[17646]: Discovery Error: Could not find any node.
... xcat[17646]: Discovery Error: Could not find any node.
... xcat[17646]: xcatd: Processing discovery request from 10.1.254.5
```

```
Genesis (The respective BMC object is identified, used for configuring the BMC
according to the node object, and then removed):

... xcat[17646]: Discovery info: configure password for
pbmc_node:node-8335-gth-0000000000000000.
... xcat[39159]: xCAT: Allowing rspconfig to node-8335-gth-0000000000000000
password= for root from localhost
... xcat[39168]: xCAT: Allowing chdef node-8335-gth-0000000000000000 bmcusername=
bmcpassword= for root from localhost
... xcat[17646]: Discover info: configure ip:10.2.1.1 for
pbmc_node:node-8335-gth-0000000000000000.
... xcat[39175]: xCAT: Allowing rspconfig to node-8335-gth-0000000000000000
ip=10.2.1.1 for root from localhost
... xcat[17646]: Discovery info: remove pbmc_node:node-8335-gth-0000000000000000.
... xcat[39184]: xCAT: Allowing rmdef node-8335-gtb-0000000000000000 for root from
localhost
... xcatd: Discovery worker: fsp instance: nodediscover instance: p9r1n1 has been
discovered
... xcat[17646]: Discovery info: configure password for
pbmc_node:node-8335-gth-0000000000000000.
... xcat[39217]: xCAT: Allowing chdef node-8335-gth-0000000000000000 bmcusername=
bmcpassword= for root from localhost
... xcat[17646]: Discover info: configure ip:10.2.1.1 for
pbmc_node:node-8335-gth-0000000000000000.
... xcat[17646]: Discovery info: remove pbmc_node:node-8335-gth-0000000000000000.

Genesis (DHCP client releases the temporary IP address, and acquires the permanent
IP address):

... dhcpd: DHCPRELEASE of 10.1.254.5 from 98:be:94:59:f0:f2 via enP5p1s0f0 (found)
... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPOFFER on 10.1.1.1 to 98:be:94:59:f0:f2 via enP5p1s0f0
... dhcpd: DHCPREQUEST for 10.1.1.1 (10.1.0.1) from 98:be:94:59:f0:f2 via
enP5p1s0f0
... dhcpd: DHCPACK on 10.1.1.1 to 98:be:94:59:f0:f2 via enP5p1s0f0

Genesis (Cleanup of the BMC discovery):

... xcat[39230]: xCAT: Allowing rmdef node-8335-gth-0000000000000000 for root from
localhost
... xcatd: Discovery worker: fsp instance: nodediscover instance: Failed to notify
10.1.254.5 that it's actually p9r1n1.

Genesis (Further communication with the xCAT Management Node):

... xcat[39233]: xCAT: Allowing getcredentials x509cert from p9r1n1
... xcat: credentials: sending x509cert
```

The Genesis image remains waiting for further instructions from the xCAT management node and is accessible through SSH.

Note: During the BMC network configuration steps, network connectivity can be lost (including the IPv6 link-local address). In this case, reset the connection by using in-band IPMI with the **ipmitool** command that is included in the Genesis image. This limitation might be addressed in a future xCAT or firmware version.

The steps to reset the BMC by using in-band IPMI on the Genesis image and wait for the BMC to come back online are shown in Example 5-27.

Example 5-27 Resetting the BMC by using in-band IPMI on the Genesis image

```
Reset the BMC via in-band IPMI on the Genesis image:

$ ssh p9r1n1 'ipmitool mc reset cold'

Wait for the BMC to come back online:
(This example is based on the link-local IPv6 address; you can also use the IPv4
address assigned on node discovery; e.g., ping 10.2.1.1)

$ while ! ping6 -c 1 -q fe80::72e2:84ff:fe14:254%enP5p1s0f0; do echo Waiting;
done; echo; echo Finished
PING fe80::72e2:84ff:fe14:254%enP5p1s0f0(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enP5p1s0f0 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

Waiting
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enP5p1s0f0 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

<...>
Waiting
PING fe80::72e2:84ff:fe14:254%enP5p1s0f0(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enP5p1s0f0 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.606/0.606/0.606/0.000 ms

Finished
```

You can watch the node discovery on the node console through IPMI. The BMC IP address can change as a result of the process. Therefore, use the BMC IPv6 link-local address (which does not change) for that purpose (see Example 5-28).

Example 5-28 Node discovery on the console by using IPMI with the rcons on ipmitool commands

```
With the rcons command:
(require initial configuration with the makeconservercf command):
$ makeconservercf
$ rcons node-8335-gth-0000000000000000

With the ipmitol command (via IPv6 link-local address):
$ ipmitool -I lanplus -H fe80::72e2:84ff:fe14:254%enp1s0f3 -P OpenBmc sol activate
```

You can verify that the node object now contains attributes that are obtained during the node discovery process (for example, hardware characteristics) and other xCAT attributes by running the **lsdef** command (see Example 5-29).

Example 5-29 Node object with attributes that are obtained during node discovery

```
$ lsdef p9r1n1
Object name: p9r1n1
    arch=ppc64
    bmc=10.2.1.1
    cons=ipmi
    cpucount=192
    cputype=POWER9 (raw), altivec supported
    disksize=sda:1000GB,sdb:1000GB
    groups=all,ac922
    ip=10.1.1.1
    mac=98:be:94:59:f0:f2
    memory=261482MB
    mgt=ipmi
    mtm=8335-GTH
    netboot=petitboot
    nodetype=mp
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=0000000000000000
    status=standingby
    statustime=11-25-2016 23:13:50
    supportedarchs=ppc64
```

5.6 xCAT compute nodes (stateless)

This section describes the deployment of an xCAT compute node with the IBM HPC software that is running on Red Hat Enterprise Linux Server 7.5 for PowerPC 64-bit Little Endian (ppc64le) in non-virtualized (or bare-metal) mode on the Power AC922 server.

The steps that are described in this section cover the stateless (diskless) installation. The image includes only runtime libraries to keep the image size as small as possible. For more information about a stateful image with full compiler support, see 5.7, “xCAT login nodes (stateful)” on page 133.

5.6.1 Network interfaces

The network interface that is associated with the management network is known as a *primary network interface* (or adapter). The network interfaces that are associated with other networks are known as *secondary* (or additional) *network interfaces* (or adapters).

For more information, see Configure Additional Network Interfaces - confignics.

Primary network interface

The primary network interface is the network interface that is connected to the management network. The following attributes are important for this network interface:

- `primarynic`

  This attribute identifies the primary network interface. Set it to `mac` to use the network interface with the MAC address that is specified by the `mac` attribute (collected during node discovery).

- `installnic`

  This attribute identifies the network interface that is used for OS installation, which is usually the primary network interface. Set it to `mac`.

To set the attributes for the primary network interface to `mac`, complete the following steps:

1. Set the `primarynic` and `installnic` by running the **`chdef`** command:

   ```
   $ chdef -t group ac922 \
       installnic=mac \
       primarynic=mac
   1 object definitions have been created or modified.
   ```

2. Verify the attributes by running the **`lsdef`** command:

   ```
   $ lsdef -t group ac922 -i installnic,primarynic
   Object name: ac922
       installnic=mac
       primarynic=mac
   ```

Secondary network interfaces

xCAT uses the information from the `nics` table to configure the network interfaces in the nodes to be part of the xCAT networks that are defined in the `networks` table. For example, the following attributes are used:

- The `nicips` attribute for IP addresses
- The `nicnetworks` attribute for xCAT networks (defined in the `networks` table)
- The `nictypes` attribute for the type of networks (for example, Ethernet or InfiniBand)
- (Optional) `nichostnamesuffixes` for appending per-network suffixes to host names

The attribute format for the `nics` table uses several types of field separators, which allows for each field to relate to multiple network interfaces with multiple values per network interface (for example, IP addresses and host names). The format is a comma-separated list of `interface!values` pairs (that is, one pair per network interface), where *`values`* is a pipe-separated list of values (that is, all values that are assigned to that network interface).

For values with regular expressions, include the xCAT regular expression delimiters or pattern around the value (that is, leading pipe, regular expression pattern, separator pipe, value, and trailing pipe).

Consider the following example:

- Two network interfaces: `eth0` and `eth1`
- Two IP addresses each (`eth0` with `192.168.0.1` and `192.168.0.2`; `eth1` with `192.168.0.3` and `192.168.0.4`):

  ```
  nicips='eth0!192.168.0.1|192.168.0.2,eth1!192.168.0.3|192.168.0.4'
  ```

- Two host name suffixes each (`eth0` with `-port0ip1` and `-port0ip2`; `eth1` with `-port1ip1` and `-port1ip2`):

  ```
  nichostnamesuffixes='eth0!-port0ip1|-port0ip2,eth1!-port1ip1|-port1ip2'
  ```

To configure the InfiniBand network interfaces of the compute nodes, complete the following steps:

1. Set the attributes of the `nics` table in the node group by running the **`chdef`** command:

   ```
   $ chdef -t group ac922\
       nictypes='ib0!InfiniBand' \
       nicnetworks='ib0!net-app-ib' \
       nichostnamesuffixes='ib0!-ib' \
       nicips='|p9r(\d+)n(\d+)|ib0!10.10.($1+0).($2+0)|'
   1 object definitions have been created or modified.
   ```

   Note: The following definition results in the same IP:

   ```
   nicips='|ib0!10.10.($2+0).($3+0)|'
   ```

2. Verify the attributes by running the **`lsdef`** command. They are represented with per-interface subattributes (that is, `<attribute>.<interface>=<values-for-the-interface>`).

   ```
   $ lsdef --nics -t group ac922
   Object name: ac922
       nichostnamesuffixes.ib0=-ib
       nicips.ib0=10.10.($1+0).($2+0)
       nicnetworks.ib0=net-app-ib
       nictypes.ib0=InfiniBand
   ```

   Note: To view the full `nicips` value, add **`--verbose`** to the **`lsdef`** command.

3. Verify that the attributes that are based on regular expressions have the correct values on a particular node by running the **`lsdef`** command:

   ```
   lsdef --nics p9r1n1
   Object name: p9r1n1
       nichostnamesuffixes.ib0=-ib
       nicips.ib0=10.10.1.1
       nicnetworks.ib0=net-app-ib
       nictypes.ib0=InfiniBand
   ```

4. Update the configuration files for the host table, DNS server, and DHCP server by running the **`makehosts`**, **`makedns`**, and **`makedhcp`** commands:

   ```
   $ makehosts ac922

   $ cat /etc/hosts
   127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
   ::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
   10.1.0.1    xcat-mn.xcat-cluster xcat-mn
   10.1.1.1    p9r1n1 p9r1n1.xcat-cluster
   10.10.1.1   p9r1n1-ib p9r1n1-ib.xcat-cluster
   ```

Note: If a network interface prefix or suffix is renamed or removed, you can update the host table by removing the node entry (or nodes or group entries) and adding them back by running the following commands:

```
$ makehosts -d nodes
$ makehosts nodes
```

```
$ makedns -n ac922
Handling p9r1n1 in /etc/hosts.
Handling p9r1n1-ib in /etc/hosts.
Getting reverse zones, this takes several minutes for a large cluster.
Completed getting reverse zones.
Updating zones.
Completed updating zones.
Restarting named
Restarting named complete
Updating DNS records, this takes several minutes for a large cluster.
Completed updating DNS records.

$ makedhcp -n
$ makedhcp -a
```

5. Add the **configNics** script to the list of postscripts for configuring the network interfaces. The InfiniBand network interfaces (2-port adapter) require the argument **--ibaports=2**.

```
$ chdef -t group ac922 --plus postscripts='confignics --ibaports=2'
1 object definitions have been created or modified.

$ lsdef -t group ac922 -i postscripts
Object name: ac922
    postscripts=confignics --ibaports=2
```

   For more information about the **confignics** script and the configuration of InfiniBand adapters, see the following xCAT documentation pages:

   – Configure Additional Network Interfaces - confignics
   – InfiniBand Network Configuration

Public or site network connectivity (optional)

The connectivity to the public or site networks for the compute nodes can be provided by using one of the following methods:

- By using the management node with network address translation (NAT)
- By using compute nodes with another xCAT network

To perform the required network configuration, follow the steps of either method.

Management node method

This method requires that the `gateway` attribute of the xCAT management network is set to the management node (the default setting is `<xcatmaster>`), and that the NAT rules are configured in the firewall. To connect, complete the following steps:

1. Verify that the `gateway` attribute of the management network is set to `<xcatmaster>` by running the **lsdef** command:

```
$ lsdef -t network net-mgmt -i gateway
Object name: net-mgmt
    gateway=<xcatmaster>
```

If not, set it by running the **chdef** command:

```
$ chdef -t network net-mgmt gateway='<xcatmaster>'
```

2. Verify the `routers` option of the DHCP server configuration file (based on the `gateway` attribute) in the management network that reflects the IP address of the management node:

```
$ grep 'subnet\|routers' /etc/dhcp/dhcpd.conf
  subnet 10.1.0.0 netmask 255.255.0.0 {
    option routers  10.1.0.1;
  } # 10.1.0.0/255.255.0.0 subnet_end
  subnet 10.2.0.0 netmask 255.255.0.0 {
    option routers  10.2.0.1;
  } # 10.2.0.0/255.255.0.0 subnet_end
```

If not, regenerate the DHCP server configuration files by running the **makedhcp** command:

```
$ makedhcp -n
$ makedhcp -a
```

3. Verify that the default route on the compute nodes is set to the IP address of the management node by running the **ip** command:

```
$ xdsh p9r1n1 'ip route show | grep default'
p9r1n1: default via 10.1.0.1 dev enP5p1s0f0
```

If not, restart the network service by running the **systemctl** command:

```
$ xdsh p9r1n1 'systemctl restart network'
```

4. Configure the `iptables` firewall rules for NAT in the **rc.local** script. Configure it to start automatically on boot and start the service manually this time.

Consider the following points regarding the network scenario in this chapter:

– The management network on network interface `enP5p1s0f0` is in the management node.

– The public or site network on network interface `enp1s0f2` is in the management node.

```
$ cat <<EOF >>/etc/rc.d/rc.local
iptables -t nat --append POSTROUTING --out-interface enp1s0f2 -j MASQUERADE
iptables --append FORWARD --in-interface enP5p1s0f0 -j ACCEPT
EOF

$ chmod +x /etc/rc.d/rc.local
$ systemctl start rc-local
```

Note: The default firewall in Red Hat Enterprise Linux Server 7.5 is `firewalld`, which is disabled by xCAT.

For more information about firewalld, see the Using Firewalls Red Hat Enterprise Linux documentation page.

5. You can verify the network connectivity on a running node (if any) by running the **ping** command:

```
$ xdsh p9r1n1 'ping -c1 example.com'
p9r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p9r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=4.07 ms
<...>
```

Compute nodes method

This method requires the `gateway` attribute of the xCAT management network *not* to be set, and another xCAT network (for the public or site network) to define the network interface, address, and gateway to be used. Complete the following steps:

1. Reset the `gateway` attribute of the management network by running the **`chdef`** command:

```
$ chdef -t network net-mgmt gateway=''

$ lsdef -t network net-mgmt -i gateway
Object name: net-mgmt
    gateway=
```

2. Define an xCAT network for the public or site network (for example, `net-site`):

```
$ mkdef -t network net-site net=9.x.x.x mask=255.255.255.0
1 object definitions have been created or modified.

# lsdef -t network
net-app-ib (network)
net-mgmt  (network)
net-site  (network)
net-svc  (network)

$ lsdef -t network net-site
Object name: net-site
    mask=255.255.255.0
    net=9.x.x.x
```

3. Modify the `nics` table to include the attributes of the new xCAT network by running the **`tabedit`** command or the **`chdef`** command. The network scenario that is described in this chapter uses a public or site network on network interface `enp1s0f1` in the compute nodes.

```
$ tabedit nics

OR:

$ chdef -t group ac922\
    nictypes='enp1s0f1!Ethernet,ib0!InfiniBand' \
    nicnetworks='enp1s0f1!net-site,ib0!net-app-ib' \
    nichostnamesuffixes='enp1s0f1!-site,ib0!-ib' \
    nicips='|p9r(\d+)n(\d+)|enp1s0f1!9.x.x.(181-$2),ib0!10.10.($1+0).($2+0)|' \
    nicextraparams='enp1s0f1!GATEWAY=9.x.x.254'
1 object definitions have been created or modified.
```

4. Verify the attributes by running the **`tabdump`** command or **`lsdef`** command:

```
$ tabdump nics
#node,nicips,nichostnamesuffixes,nichostnameprefixes,nictypes,niccustomscripts,
nicnetworks,nicaliases,nicextraparams,comments,disable
"ac922","|p9r(\d+)n(\d+)|enp1s0f1!9.x.x.(181-$2),ib0!10.10.($1+0).($2+0)|","enp
1s0f1!-site,ib0!-ib",,"enp1s0f1!Ethernet,ib0!InfiniBand",,"enp1s0f1!net-site,ib
0!net-app-ib",,"enp1s0f1!GATEWAY=9.x.x.254",,

$ lsdef --nics -t group ac922
Object name: ac922
<...>
    nicextraparams.enp1s0f1=GATEWAY=9.x.x.254
<...>
```

```
    nichostnamesuffixes.enp1s0f1=-site
<...>
    nicips.enp1s0f0=9.x.x.(181-$2)
<...>
    nicnetworks.enp1s0f1=net-site
<...>
    nictypes.enp1s0f1=Ethernet

$ lsdef --nics p9r1n1
Object name: p9r1n1
<...>
    nicextraparams.enp1s0f1=GATEWAY=9.x.x.254
<...>
    nichostnamesuffixes.enp1s0f1=-site
<...>
    nicips.enp1s0f1=9.x.x.180
<...>
    nicnetworks.enp1s0f1=net-site
<...>
    nictypes.enp1s0f1=Ethernet
```

5. Update the host table by running the **makehosts** command:

```
$ makehosts -d ac922
$ makehosts ac922

$ cat /etc/hosts
<...>
10.1.1.1 p9r1n1 p9r1n1.xcat-cluster
10.10.1.1 p9r1n1-ib p9r1n1-ib.xcat-cluster
9.x.x.180 p9r1n1-site p9r1n1-site.xcat-cluster
<...>
```

6. Update the DNS server configuration by running the **makedns** command:

```
$ makedns -n ac922
<...>
Handling p9r1n1-site in /etc/hosts.
<...>
Completed updating DNS records.
```

7. Update the DHCP server configuration by running the **makedhcp** command:

```
$ makedhcp -n
$ makedhcp -a
```

8. You can update the network configuration on a running node (if any) by running the **confignics** script of the **updatenode** command:

```
$ updatenode p9r1n1 -P confignics
```

9. You can verify the network connectivity on a running node (if any) by running the **ping** command:

```
$ xdsh p9r1n1 'ping -c1 example.com'
p9r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p9r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=3.94 ms
<...>
```

5.6.2 Red Hat Enterprise Linux operating system images

xCAT stores the configuration for installing OS in objects of type `osimage` (*operating system image*). To create `osimage` objects that are based on an OS installation disk image (for example, an ISO file), run the **`copycds`** command.

Setting the password for the root user

To set the root password, complete the following steps:

1. Set the `username` and `password` attributes of the `system` key or row in the `passwd` table by running the **`chtab`** command:

   ```
   $ chtab key=system passwd.username=root passwd.password=cluster
   ```

2. Verify the attributes by running the **`tabdump`** command:

   ```
   $ tabdump -w key==system passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "system","root","cluster",,,,
   ```

Note: If the attributes are not correctly set, the **`nodeset`** command shows the following error message:

```
# nodeset p9r1n1 osimage=rh75-compute-stateless
p9r1n1: Error: Unable to find requested filed <password> from table
<passwd>, with key <key=system,username=root>
Error: Some nodes failed to set up install resources on server
xcat-mn.xcat-cluster, aborting
```

Creating an osimage object

Initially, no `osimage` objects are available, as shown by the output of the following command:

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

To create `osimage` objects for the Red Hat Enterprise Linux Server 7.5 installation disk image, complete the following steps:

1. Run the **`copycds`** command on the Red Hat Enterprise Linux Server 7.5 ISO file:

   ```
   $ copycds /mnt/RHEL-ALT-7.5-20180315.0-Server-ppc64le-dvd1.iso
   Copying media to /install/rhels7.5/ppc64le
   Media copy operation successful
   ```

2. Verify that the `osimage` objects are present by running the **`lsdef`** command.

   ```
   $ lsdef -t osimage
   rhels7.5-alternate-ppc64le-install-compute  (osimage)
   rhels7.5-alternate-ppc64le-install-service  (osimage)
   rhels7.5-alternate-ppc64le-netboot-compute  (osimage)
   rhels7.5-alternate-ppc64le-statelite-compute  (osimage)
   ```

   For more information about the `osimage` objects, see 5.3.5, “xCAT installation types: Disks and state” on page 62.

3. Create a copy of the original `osimage` object that is named `rh75-compute-stateless`. It is optional but useful in case multiple `osimage` objects are maintained (for example, for multiple or different configurations of the same OS).

   You can accomplish this task by running the **`lsdef -z`** command, which provides the object stanza. Modify the stanza by running the **`sed`** command), and create an object that is based on it by running the **`mkdef -z`** command.

   ```
   $ osimage=rh75-compute-stateless

   $ lsdef -t osimage rhels7.5-alternate-ppc64le-netboot-compute -z \
   | sed "s/^[^# ].*:/$osimage:/" \
   | mkdef -z
   1 object definitions have been created or modified.
   ```

4. Verify the copied `osimage` object by running the **`lsdef`** command:

   ```
   $ lsdef -t osimage
   rh75-compute-stateless  (osimage)
   rhels7.5-alternate-ppc64le-install-compute  (osimage)
   rhels7.5-alternate-ppc64le-install-service  (osimage)
   rhels7.5-alternate-ppc64le-netboot-compute  (osimage)
   rhels7.5-alternate-ppc64le-statelite-compute(osimage)
   ```

5. To view more information about the new `osimage`, append the name of the object to the command in step 4, as shown in Example 5-30.

Example 5-30 Detailed osimage information

```
lsdef -t osimage $osimage
Object name: rh75-compute-stateless
    exlist=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.exlist
    imagetype=linux
    osarch=ppc64le
    osdistroname=rhels7.5-ppc64le
    osname=Linux
    osvers=rhels7.5-alternate
    otherpkgdir=/install/post/otherpkgs/rhels7.5/ppc64le
    pkgdir=/install/rhels7.5-alternate/ppc64le
    pkglist=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist

postinstall=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.postinstall
    profile=compute
    provmethod=netboot
    rootimgdir=/install/netboot/rhels7.5-alternate/ppc64le/compute
```

Note: The OS can be installed (without other components of the software stack) by running the **`nodeset`** command:

```
$ nodeset p9r1n1 osimage=rh75-compute-stateless
```

Changing the pkglist attribute

The use of multiple package lists is convenient for independently organizing the required packages for each component of the software stack. Although xCAT currently does not support multiple package lists in the `pkglist` attribute, it does support a package list to reference the contents of other package lists. This feature provides a way to achieve multiple package lists.

The following types of entries can be used in the `pkglist` file:

- RPM name without version numbers.
- Group or pattern name that is marked with a "`@`" (for stateful installation only).
- RPMs to be removed after the installation by marking them with a "`-`" (for stateful installation only).
- `#INCLUDE: <full file path>#` to include other pkglist files.
- `#NEW_INSTALL_LIST#` to signify that the listed RPMs will be installed with a new RPM **`install`** command (**`zypper`**, **`yum`**, or **`rpm`** as determined by the function that uses this file).

To change the `pkglist` attribute for a custom package list, complete the following steps:

1. Verify the current `pkglist` attribute and assign it to the **`old_list`** variable:

```
$ lsdef -t osimage rh75-compute-stateless -i pkglist
Object name: rh75-compute-stateless
    pkglist=/install/custom/netboot/rh/rh75-compute.pkglist

$ old_pkglist=/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist
```

2. In this example, we include the default Red Hat Enterprise Linux Server 7.5 `pkglist` and add the `numactl` package to it. Create the list that includes the old list. Then, add `numactl`.

```
$ new_pkglist=/install/custom/netboot/rh/rh75-compute.pkglist

$ mkdir -p $(dirname $new_pkglist)
$ echo "# RHEL Server 7.5 (original pkglist)" >> $new_pkglist
$ echo "#INCLUDE:${old_pkglist}#" >> $new_pkglist
$ echo "numactl" >> $new_pkglist
```

Note: All of your custom files must be in the `/install/custom` directory for easier distinction.

3. Verify the contents of the new list. The trailing "`#`" character at the end of the line is important for correct `pkglist` parsing.

```
$ cat $new_pkglist
# RHEL Server 7.5 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist#
numactl
```

4. Change the `pkglist` attribute to the new list by running the **`chdef`** command:

```
$ chdef -t osimage rh75-compute-stateless pkglist=$new_pkglist
1 object definitions have been created or modified.
```

5. Verify the `pkglist` attribute by running the **`lsdef`** command:

```
$ lsdef -t osimage rh75-compute-stateless -i pkglist
Object name: rh75-compute-stateless
    pkglist=/install/custom/netboot/rh/rh75-compute.pkglist
```

Changing the rootimgdir attribute

Each stateless image must have its own `rootimgdir`. The image is stored in the `rootimgdir=/install/netboot/rhels7.5/ppc64le/rh75-compute-stateless` directory. As a best practice, match the last level directory name with the name of the `osimage`.

Change the `rootimgdir` to `rh75-compute-stateless`:

```
$ lsdef -t osimage rh75-compute-stateless -i rootimgdir
```

```
Object name: rh75-compute-stateless
rootimgdir=/install/netboot/rhels7.5-alternate/ppc64le/compute

$ chdef -t osimage rh75-compute-stateless
rootimgdir=/install/netboot/rhels7.5-alternate/ppc64le/rh75-compute-stateless
1 object definitions have been created or modified.
```

Note: In the next sections, drivers and libraries are added to the stateless image. As a best practice, match the management and stateless image kernel level. If a new kernel level is installed in the management node, complete the procedure at Installing a new kernel in the Diskless image to update the stateless image kernel level to the same one before installing any driver.

5.6.3 NVIDIA CUDA Toolkit

The CUDA Toolkit can be installed with the RPM packages that are contained in the local repository package that is available for download in the NVIDIA website. The packages are marked for installation with the `otherpkgdir` and `otherpkglist` mechanisms to be installed during the Linux distribution installation.

Note: For a stateless installation, installing the CUDA packages *must* be done in the `otherpkglist` and *not* the `pkglist` as with a stateful installation.

For more information, see the xCAT Red Hat Enterprise Linux 7.5 LE documentation page.

To install the CUDA Toolkit, complete the following steps:

1. Install the `createrepo` package for creating package repositories:

   ```
   $ yum install createrepo
   ```

2. Download and extract the CUDA Toolkit RPM package.

   Note: The CUDA Toolkit is available for download at the NVIDIA CUDA downloads website.

   At the website, click **(Operating Systems) Linux** → **(Architecture) ppc64le** → **(Distribution) RHEL** → **(Version) 7** → **(Installer Type) rpm (local)** → **Download**.

   ```
   $ dir=/tmp/cuda
   $ mkdir -p $dir
   $ cd $dir

   $ curl -sOL https://.../cuda-repo-rhel7-9-2-local-9.2.148-1.ppc64le.rpm
   $ rpm2cpio cuda-repo-rhel7-9-2-local-9.2.148-1.ppc64le.rpm | cpio -id
   2430295 blocks
   ```

3. Copy the extracted package repository to a new `/install/software` directory.

   Note: All other software packages are in the central `/install/software` directory.

   ```
   $ dir_cuda=/install/software/cuda/9.2
   $ mkdir -p $dir_cuda

   $ ls -1d $dir_cuda/*
   ```

```
/install/software/cuda/9.2/cuda-9-2-9.2.148-1.ppc64le.rpm
<...>
/install/software/cuda/9.2/repodata
<...>
```

4. Download the CUDA 9.2 patch and copy the RPM files to `$dir_cuda`:

```
$ dir_patch=/tmp/cuda/patch
$ mkdir -p $dir_patch
$ cd $dir_patch
$ curl -sOL
https://.../patches/1/cuda-repo-rhel7-9-2-148-local-patch-1-1.0-1.ppc64le
$ rpm2cpio cuda-repo-rhel7-9-2-148-local-patch-1-1.0-1.ppc64le |cpio -id
$ cp $dir_patch/var/cuda-repo-9-2-148-local-patch-1/*.rpm $dir_cuda/
```

5. Download the NVIDIA driver **396.44** and copy all the RPM files to `$dir_cuda`:

```
$ dir_driver=/tmp/cuda/driver
$ mkdir -p /$dir_driver
$ cd $dir_driver
$ curl -sOL
http://us.download.nvidia.com/tesla/396.44/nvidia-driver-local-repo-rhel7-396.4
4-1.0-1.ppc64le.rpm
$ rpm2cpio nvidia-driver-local-repo-rhel7-396.44-1.0-1.ppc64le.rpm |cpio -id
$ cp $dir_driver/var/nvidia-driver-local-repo-396.44/*.rpm $dir_cuda/
$ cd
$ rm -r /tmp/cuda
```

6. The CUDA Toolkit contains RPMs that depend on the `dkms` package. This `dkms` package is provided by Extra Packages for Enterprise Linux (EPEL). For more information about EPEL, see Fedora's EPEL wiki page. Download the `dkms` RPM package from EPEL:

```
$ dir_deps=/install/software/cuda/deps
$ mkdir -p $dir_deps
$ cd $dir_deps

$ epel_url="https://dl.fedoraproject.org/pub/epel/7/ppc64le"
$ dkms_path="${epel_url}/Packages/d/$(curl -sL ${epel_url}/Packages/d | grep -o
'href="dkms-.*\.noarch\.rpm"' | cut -d '"' -f 2)"
$ curl -sOL "$dkms_path"

$ ls -1
dkms-2.6.0.1-1.el7.noarch.rpm
```

7. Create a package repository for the `dkms` package by running the **`createrepo`** command:

```
$ createrepo .
<...>

$ ls -1
dkms-2.6.0.1-1.el7.noarch.rpm
repodata
```

8. Create symlinks from the `otherpkglist` directory to the software directory:

```
$ lsdef -t osimage rh75-compute-stateless -i otherpkgdir
Object name: rh75-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le

$ otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le
$ cd $otherpkgdir
$ ln -s $dir_cuda cuda-9.2
$ ln -s $dir_deps cuda-deps

$ ls -l
lrwxrwxrwx 1 root root 26 22. Nov 15:55 cuda-9.2 -> /install/software/cuda/9.2
lrwxrwxrwx 1 root root 27 22. Nov 15:55 cuda-deps ->
/install/software/cuda/deps
```

9. Create the `otherpkglist` file with relative paths from `otherpkgdir` to `cuda-runtime` and the `dkms` package:

```
$ lsdef -t osimage rh75-compute-stateless -i pkglist,otherpkgdir,otherpkglist
Object name: rh75-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le
    otherpkglist=
    pkglist=/install/custom/netboot/rh/rh75-compute.pkglist

$ otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le
$ otherpkglist=/install/custom/netboot/rh/rh75-compute.otherpkglist

$ cd $otherpkgdir; find -L * -type f \( -name "cuda-runtime*" -o -name "dkms*" 
\)
cuda-9.2/cuda-runtime-9-2-9.2.148-1.ppc64le.rpm
cuda-deps/dkms-2.6.0.1-1.el7.noarch.rpm
$ cat <<EOF >>$otherpkglist
$ CUDA
cuda-deps/dkms
cuda-9.2/cuda-runtime-9-2
EOF
```

10. Set the `otherpkglist` attribute for the `osimage`:

```
$ chdef -t osimage rh75-compute-stateless --plus otherpkglist=$otherpkglist
1 object definitions have been created or modified.

$ lsdef -t osimage rh75-compute-stateless -i otherpkgdir,otherpkglist
Object name: rh75-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le
    otherpkglist=/install/custom/netboot/rh/rh75-compute.otherpkglist
```

11. Create a custom **`postinstall`** script that runs `cuda_power9_setup` and include it in the osimage definition:

```
$ lsdef -t osimage rh75-compute-stateless -i postinstall
Object name: rh75-compute-stateless
postinstall=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.postinstall

$ postinstall=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64.postinstall
$ new_postinstall=/install/custom/netboot/rh/rh75-compute.postinstall
$ cp $postinstall $new_postinstall
$ cat >>$new_postinstall <<-EOF
#install CUDA driver script on stateless
/install/postscripts/cuda_power9_setup
EOF
$ chdef -t osimage rh75-compute-stateless postinstall=$new_postinstall
$ lsdef -t osimage rh75-compute-stateless -i postinstall
Object name: rh75-compute-stateless
postinstall=/install/custom/netboot/rh/rh75-compute.postinstall
```

5.6.4 Mellanox OpenFabrics Enterprise Distribution

To install the Mellanox OFED for Linux, complete the following steps.

For more information, see the following xCAT documentation resources:

- Mellanox OFED Installation Script - Preparation page
- InfiniBand Network Configuration page

1. Download and copy the ISO file to the xCAT installation directory by running the following command.

   Note: The Mellanox OFED is available for download from the Mellanox downloads website.

   At the website, find the **IBM HPC and Technical Clusters** section and download the ISO file `MLNX_OFED_LINUX-4.3-4.0.5.1-rhel7.5alternate-ppc64le.iso` from the IBM MTM 8335-GT[W,C,G,H,X] row.

   Because you must accept a user agreement, the package must be downloaded by using a browser.

```
$ dir=/install/software/mofed
$ mkdir -p $dir

$ Download via browser, then scp to xcat management server
$ scp MLNX_OFED_LINUX-4.3-4.0.5.1-rhel7.5alternate-ppc64le.iso
xcat-mn:/install/software/mofed
```

2. Copy the **`mlnxofed_ib_install.v2`** script into the `postscripts` directory (with the required file name `mlnxofed_ib_install`). This script is a sample script that is intended to help the installation of the Mellanox OFED drivers:

```
$ script=/install/postscripts/mlnxofed_ib_install
$ cp /opt/xcat/share/xcat/ib/scripts/Mellanox/mlnxofed_ib_install.v2 $script
$ chmod +x $script
$ ls -l $script
-rwxr-xr-x 1 root root 16269 22. Nov 10:51
/install/postscripts/mlnxofed_ib_install
```

3. Include the **`--add-kernel-support`** argument to the list of default arguments (**`--without-32bit --without-fw-update --force`**) according to the xCAT documentation to rebuild kernel modules for the installed kernel version.

   Also, clear the `/tmp` directory after successful installation:

```
$ file=MLNX_OFED_LINUX-4.3-4.0.5.1-rhel7.5alternate-ppc64le.iso
$ args='--add-kernel-support --without-32bit --without-fw-update --force'

$ lsdef -t osimage rh75-compute-stateless -i postinstall
Object name: rh75-compute-stateless
    postinstall=/install/custom/netboot/rh/rh75-compute.postinstall

$ echo "# Install InfiniBand MOFED" >> $new_postinstall
$ echo "/install/postscripts/mlnxofed_ib_install -p $dir/$file -m $args -end-
-i \$installroot -n genimage" >> $new_postinstall
$ echo "rm -rf \$installroot/tmp/MLNX_OFED_LINUX*" >> $new_postinstall

$ cat $new_postinstall
```

```
<...>
# Install InfiniBand MOFED
/install/postscripts/mlnxofed_ib_install -p
/install/software/mofed/MLNX_OFED_LINUX-4.3-4.0.5.1-rhel7.5alternate-ppc64le
.iso -m --add-kernel-support --without-32bit --without-fw-update --force
-end- -i $installroot -n genimage
rm -rf $installroot/tmp/MLNX_OFED_LINUX*
```

Note: The **`-i $installroot`** argument is the installation root directory and is passed to the **`postinstallation`** script at run time as $1.

4. Certain dependency packages are required to install Mellanox OFED correctly. Add the InfiniBand package list that is provided by xCAT to the package list:

```
$ ib_list=/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist

$ lsdef -t osimage rh75-compute-stateless -i pkglist
Object name: rh75-compute-stateless
    pkglist=/install/custom/netboot/rh/rh75-compute.pkglist
$ pkglist=/install/custom/netboot/rh/rh75-compute.pkglist
$ cat $pkglist
   # RHEL Server 7.5 (original pkglist)
   #INCLUDE:/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist#
   numactl
$ echo "# InfiniBand MOFED Dependencies (sample pkglist)" >> $pkglist
$ echo "#INCLUDE:${ib_list}#" >> $pkglist

$ cat $pkglist
   # RHEL Server 7.5 (original pkglist)
   #INCLUDE:/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist#
   numactl
   # InfiniBand MOFED Dependencies (sample pkglist)
   #INCLUDE:/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist#
```

5.6.5 IBM XL C/C++ runtime libraries

To install the IBM Xpertise Library (XL) C/C++ Compiler xCAT software kit, complete the following steps. For more information, see the xCAT IBM Xpertise Library Compilers documentation page.

1. Download the partial kit (distributed by the xCAT project):

```
$ mkdir /tmp/xlc
$ cd /tmp/xlc

$ curl -sOL
https://xcat.org/files/kits/hpckits/2.14/rhels7.5/xlc-16.1.0-1-ppc64le.NEED_PRO
DUCT_PKGS.tar.bz2
```

2. Build the complete kit by combining the partial kit and the product packages by running the **`buildkit`** command:

```
$ dir=/install/software/compilers/xlc

$ ls -1 $dir
libxlc-16.1.0.0-180413g.ppc64le.rpm
libxlc-devel.16.1.0-16.1.0.0-180413g.ppc64le.rpm
```

```
libxlmass-devel.9.1.0-9.1.0.0-180412a.ppc64le.rpm
libxlsmp-5.1.0.0-180412a.ppc64le.rpm
libxlsmp-devel.5.1.0-5.1.0.0-180412a.ppc64le.rpm
xlc-license.16.1.0-16.1.0.0-180413g.ppc64le.rpm
xlc.16.1.0-16.1.0.0-180413g.ppc64le.rpm
$ buildkit addpkgs xlc-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir $dir
Extracting tar file /tmp/xlc/xlc-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2.
Please wait.
<...>
Kit tar file /tmp/xlc/xlc-16.1.0-1-ppc64le.tar.bz2 successfully built.

$ ls -1
xlc-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
xlc-16.1.0-1-ppc64le.tar.bz2
```

3. Add the kit to xCAT by running the **addkit** command:

```
$ lsdef -t kit
Could not find any object definitions to display.

$ addkit xlc-16.1.0-1-ppc64le.tar.bz2
Adding Kit xlc-16.1.0-1-ppc64le
Kit xlc-16.1.0-1-ppc64le was successfully added.

$ lsdef -t kit
xlc-16.1.0-1-ppc64le  (kit)

$ cd ..
$ rm -r /tmp/xlc
```

4. Verify the `kitcomponent` objects (and `description` fields) by running the **lsdef** command:

```
$ lsdef -t kitcomponent -w kitname==xlc-16.1.0-1-ppc64le -i description
Object name: xlc.compiler-compute-16.1.0-1-rhels-7-ppc64le
    description=XLC16 for compiler kitcomponent
Object name: xlc.license-compute-16.1.0-1-rhels-7-ppc64le
    description=XLC16 license kitcomponent
Object name: xlc.rte-compute-16.1.0-1-rhels-7-ppc64le
    description=XLC16 for runtime kitcomponent
```

5. Add the following `kitcomponent` (and dependencies) to the `osimage` object by running the **addkitcomp** command:

```
$ lsdef -t osimage rh75-compute-stateless -i kitcomponents,otherpkglist
Object name: rh75-compute-stateless
    kitcomponents=
    otherpkglist=/install/custom/netboot/rhel/rh75-compute.otherpkglist

$ addkitcomp --adddeps -i rh75-compute-stateless \
xlc.rte-compute-16.1.0-1-rhels-7-ppc64le
Assigning kit component xlc.rte-compute-16.1.0-1-rhels-7-ppc64le to osimage
rh75-compute-stateless
Kit components xlc.rte-compute-16.1.0-1-rhels-7-ppc64le were added to osimage
rh75-compute-stateless successfully

$ lsdef -t osimage rh75-compute-stateless -i kitcomponents,otherpkglist
Object name: rh75-compute-stateless
```

```
kitcomponents=xlc.license-compute-16.1.0-1-rhels-7-ppc64le,xlc.rte-compute-16.1
.0-1-rhels-7-ppc64le

otherpkglist=/install/osimages/rh75-compute-stateless/kits/KIT_DEPLOY_PARAMS.ot
herpkgs.pkglist,/install/custom/netboot/rhel/rh75-compute.otherpkglist,/install
/osimages/rh75-compute-stateless/kits/KIT_COMPONENTS.otherpkgs.pkglist
```

Note: For more information about the use of a subset of compute nodes for compiling applications, see 5.7, "xCAT login nodes (stateful)" on page 133.

5.6.6 IBM XL Fortran runtime libraries

To install the IBM XL Fortran Compiler xCAT kit, complete the following steps. For more information, see the xCAT IBM Xpertise Library Compilers documentation page.

1. Download the partial kit (distributed by the xCAT project):

```
$ mkdir /tmp/xlf
$ cd /tmp/xlf

$ curl -sOL
https://xcat.org/files/kits/hpckits/2.14/rhels7.5/xlf-16.1.0-1-ppc64le.NEED_PRO
DUCT_PKGS.tar.bz2
```

2. Build the complete kit by combining the partial kit and the product packages that are distributed in the installation media by running the **buildkit** command:

```
$ dir=/install/software/compilers/xlf

$ ls -1 $dir
libxlf-16.1.0.0-180413g.ppc64le.rpm
libxlf-devel.16.1.0-16.1.0.0-180413g.ppc64le.rpm
libxlmass-devel.9.1.0-9.1.0.0-180412a.ppc64le.rpm
libxlsmp-5.1.0.0-180412a.ppc64le.rpm
libxlsmp-devel.5.1.0-5.1.0.0-180412a.ppc64le.rpm
xlf-license.16.1.0-16.1.0.0-180413g.ppc64le.rpm
xlf.16.1.0-16.1.0.0-180413g.ppc64le.rpm
$ buildkit addpkgs xlf-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir $dir
Extracting tar file /tmp/xlf/xlf-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2.
Please wait. <...>
Kit tar file /tmp/xlf/xlf-16.1.0-1-ppc64le.tar.bz2 successfully built.

$ ls -1
xlf-16.1.0-1-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
xlf-16.1.0-1-ppc64le.tar.bz2
```

3. Add the kit to xCAT by running the **addkit** command:

```
$ addkit xlf-16.1.0-1-ppc64le.tar.bz2
Adding Kit xlf-16.1.0-1-ppc64le
Kit xlf-16.1.0-1-ppc64le was successfully added.

$ lsdef -t kit
xlc-16.1.0-1-ppc64le  (kit)
xlf-16.1.0-1-ppc64le  (kit)
$ cd ..
$ rm -r /tmp/xlf
```

4. Verify the `kitcomponent` objects (and `description` fields) by running the **`lsdef`** command:

```
$ lsdef -t kitcomponent -w kitname==xlf-16.1.0-1-ppc64le -i description
Object name: xlf.compiler-compute-16.1.0-1-rhels-7-ppc64le
    description=XLF16 for compiler kitcomponent
Object name: xlf.license-compute-16.1.0-1-rhels-7-ppc64le
    description=XLF16 license kitcomponent
Object name: xlf.rte-compute-16.1.0-1-rhels-7-ppc64le
    description=XLF16 for runtime kitcomponent
```

5. Add the following `kitcomponent` (and dependencies) to the `osimage` object by running the **`addkitcomp`** command:

```
$ addkitcomp --adddeps -i rh75-compute-stateless \
xlf.rte-compute-16.1.0-1-rhels-7-ppc64le
Assigning kit component xlf.rte-compute-16.1.0-1-rhels-7-ppc64le to osimage
rh75-compute-stateless
Kit components xlf.rte-compute-16.1.0-1-rhels-7-ppc64le were added to osimage
rh75-compute-stateless successfully

$ lsdef -t osimage rh75-compute-stateless -i kitcomponents
Object name: rh75-compute-stateless

kitcomponents=xlc.license-compute-16.1.0-1-rhels-7-ppc64le,xlc.rte-compute-16.1
.0-1-rhels-7-ppc64le,xlf.license-compute-16.1.0-1-rhels-7-ppc64le,xlf.rte-compu
te-16.1.0-1-rhels-7-ppc64le
```

5.6.7 Advance Toolchain runtime libraries

To install the Advance Toolchain, complete the following steps.

Note: For more information about the latest download links, see the Supported Linux Distributions section of the IBM Advance Toolchain for PowerLinux™ Documentation website.

1. Download the product packages:

```
$ dir=/install/software/compilers/at
$ mkdir -p $dir
$ cd $dir
$ wget
'ftp://ftp.unicamp.br/pub/linuxpatch/toolchain/at/redhat/RHEL7/at12.0/*-12.0-0.
ppc64le.rpm'
<...>

$ ls -1advance-toolchain-at12.0-devel-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-devel-debuginfo-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-mcore-libs-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-mcore-libs-debuginfo-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-perf-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-perf-debuginfo-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-runtime-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-runtime-at11.0-compat-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-runtime-debuginfo-12.0-0.ppc64le.rpm
advance-toolchain-at12.0-selinux-12.0-0.ppc64le.rpm
advance-toolchain-golang-at-12.0-0.ppc64le.rpm
```

2. Create a package repository by running the **createrepo** command:

```
$ createrepo .
<...>
```

3. Add packages to `otherkglist` of `osimage`:

```
$ lsdef -t osimage rh75-compute-stateless -i otherpkglist,otherpkgdir
Object name: rh75-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le

otherpkglist=/install/osimages/rh75-compute-stateless/kits/KIT_DEPLOY_PARAMS.ot
herpkgs.pkglist,/install/custom/netboot/rhel/rh75-compute.otherpkglist,/install
/osimages/rh75-compute-stateless/kits/KIT_COMPONENTS.otherpkgs.pkglist

$ otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le
$ otherpkglist=/install/custom/netboot/rh/rh75-compute.otherpkglist

$ cd $otherpkgdir
$ ln -s $dir at

$ ls -l
lrwxrwxrwx 1 root root 30 Oct  9 14:10 at -> /install/software/compilers/at
lrwxrwxrwx 1 root root 26 Oct  9 11:19 cuda-9.2 -> /install/software/cuda/9.2
lrwxrwxrwx 1 root root 27 Oct  9 11:19 cuda-deps -> /install/software/cuda/deps
lrwxrwxrwx 1 root root 69 Oct 10 12:04 xlc-16.1.0-1-rhels-7-ppc64le ->
/install/kits/xlc-16.1.0-1-ppc64le/repos/xlc-16.1.0-1-rhels-7-ppc64le
lrwxrwxrwx 1 root root 69 Oct 10 12:11 xlf-16.1.0-1-rhels-7-ppc64le ->
/install/kits/xlf-16.1.0-1-ppc64le/repos/xlf-16.1.0-1-rhels-7-ppc64le

$ echo "# Advance Toolchain" >> $otherpkglist
$ echo "at/advance-toolchain-at12.0-runtime" >> $otherpkglist
$ echo "at/advance-toolchain-at12.0-mcore-libs" >> $otherpkglist

$ cat $otherpkglist
   # CUDA
   cuda-deps/dkms
   cuda-9.2/cuda-runtime-9-2
   # Advance Toolchain
   at/advance-toolchain-at12.0-runtime
   at/advance-toolchain-at12.0-mcore-libs
```

Note: Advance Toolchain is approximately 2 GB after installation. For stateless nodes, it is recommended to install this package in a shared file system to save main memory on the compute node.

5.6.8 IBM Spectrum MPI

To install IBM Spectrum Message Passing Interface (IBM Spectrum MPI), complete the following steps:

1. Create a directory for the IBM Spectrum MPI and place all the RPMs:

```
$ dir=/install/software/smpi
$ mkdir -p $dir
$ cd $dir
//copy files or download from passport advantage to this directory.//
$ ls $dir
ibm_smpi-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
ibm_smpi_lic_s-10.02.09rtm5-rh7_20181010.ppc64le.rpm
ibm_smpi-libgpump-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
ibm_smpi-devel-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
ibm_spindle-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
ibm_smpi_gpusupport-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
ibm_smpi-pami_devel-10.02.00.09rtm5-rh7_20181010.ppc64le.rpm
```

2. Create a package repository by running the **createrepo** command:

```
$createrepo .
<...>
```

3. Add packages to `otherkglist` of `osimage`:

```
$ lsdef -t osimage rh75-compute-stateless -i otherpkglist,otherpkgdir
Object name: rh75-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le

otherpkglist=/install/osimages/rh75-compute-stateless/kits/KIT_DEPLOY_PARAMS.ot
herpkgs.pkglist,/install/custom/netboot/rhel/rh75-compute.otherpkglist,/install
/osimages/rh75-compute-stateless/kits/KIT_COMPONENTS.otherpkgs.pkglist

$ otherpkgdir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le
$ otherpkglist=/install/custom/netboot/rh/rh75-compute.otherpkglist

$ cd $otherpkgdir
$ ln -s $dir smpi

$ ls -l
lrwxrwxrwx 1 root root 30 Oct  9 14:10 at -> /install/software/compilers/at
lrwxrwxrwx 1 root root 26 Oct  9 11:19 cuda-9.2 -> /install/software/cuda/9.2
lrwxrwxrwx 1 root root 27 Oct  9 11:19 cuda-deps -> /install/software/cuda/deps
lrwxrwxrwx 1 root root 93 Oct 10 14:48 smpi -> /install/software/smpi
lrwxrwxrwx 1 root root 69 Oct 10 12:04 xlc-16.1.0-1-rhels-7-ppc64le ->
/install/kits/xlc-16.1.0-1-ppc64le/repos/xlc-16.1.0-1-rhels-7-ppc64le
lrwxrwxrwx 1 root root 69 Oct 10 12:11 xlf-16.1.0-1-rhels-7-ppc64le ->
/install/kits/xlf-16.1.0-1-ppc64le/repos/xlf-16.1.0-1-rhels-7-ppc64le

$ echo "# SMPI" >> $otherpkglist
echo "#ENV:IBM_SPECTRUM_MPI_LICENSE_ACCEPT=yes#" >> $otherpkglist
$ echo "smpi/ibm_smpi" >> $otherpkglist
$ echo "smpi/ibm_smpi_lic_s" >> $otherpkglist
$ echo "smpi/ibm_smpi-libgpump" >> $otherpkglist
$ echo "smpi/ibm_smpi-devel" >> $otherpkglist
$ echo "smpi/ibm_spindle" >> $otherpkglist
$ echo "smpi/ibm_smpi_gpusupport" >> $otherpkglist
$ echo "smpi/ibm_smpi-pami_devel" >> $otherpkglist

$ cat $otherpkglist
   # CUDA
   cuda-deps/dkms
   cuda-9.2/cuda-runtime-9-2
   # Advance Toolchain
   at/advance-toolchain-at12.0-runtime
   # SMPI
   #ENV:IBM_SPECTRUM_MPI_LICENSE_ACCEPT=yes#
   smpi/ibm_smpi
   smpi/ibm_smpi_lic_s
   smpi/ibm_smpi-libgpump
   smpi/ibm_smpi-devel
   smpi/ibm_spindle
   smpi/ibm_smpi_gpusupport
   smpi/ibm_smpi-pami_devel
```

5.6.9 IBM Parallel Performance Toolkit

To install IBM Parallel Performance Toolkit, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT by running the **addkit** command:

```
$ addkit /path/to/ppt-files/ppt-2.4.0-2.tar.bz2
Adding Kit ppedev-2.4.0-2
Kit ppt-2.4.0-2 was successfully added.

$ lsdef -t kit
ibm_smpi_kt-10.2.0.6-rh7-ppc64le  (kit)
ppt-2.4.0-2 (kit)
xlc-16.1.0-1-ppc64le  (kit)
xlf-16.1.0-1-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) by running the **lsdef** command:

```
$ lsdef -t kitcomponent -w kitname==ppt-2.4.0-2 -i description
Object name: ppt.compute-2.4.0-2-rhels-7.4-ppc64le
    description=IBM Parallel Performance Toolkit for compute nodes
Object name: ppt.license-2.4.0-2-rhels-7.4-ppc64le
    description=IBM Parallel Performance Toolkit license package
Object name: ppt.login-2.4.0-2-rhels-7.4-ppc64le
    description=IBM Parallel Performance Toolkit for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by running the **addkitcomp** command:

```
$ addkitcomp --adddeps -i rh75-compute-stateless \
ppt.compute-2.4.0-2-rhels-7.4-ppc64le
Assigning kit component ppt.compute-2.4.0-2-rhels-7.4-ppc64le to osimage
rh75-compute-stateless
Kit components ppt.compute-2.4.0-2-rhels-7.4-ppc64le were added to osimage
rh75-compute-stateless successfully

$ lsdef -t osimage rh75-compute-stateless -i kitcomponents
Object name: rh75-compute-stateless
```

```
kitcomponents=kitcomponents=xlc.license-compute-16.1.0-1-rhels-7-ppc64le,xlc.rt
e-compute-16.1.0-1-rhels-7-ppc64le,xlf.license-compute-16.1.0-1-rhels-7-ppc64le
,xlf.rte-compute-16.1.0-1-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.2.0.6-rh7
-rhels-7-ppc64le,ibm_spectrum_mpi-full-10.2.0.6-rh7-rhels-7-ppc64le,essl-licens
e-6.1.0-1-rhels-7.5-ppc64le,ppt.license-2.4.0-2-rhels-7.4-ppc64le,ppt.compute-2
.4.0-2-rhels-7.4-ppc64le
```

Note: To perform development and profiling tasks on a compute node, install the following `kitcomponent`:

```
ppt.login-2.4.0-2-rhels-7.4-ppc64le
```

5.6.10 IBM Engineering and Scientific Subroutine Library

To install IBM Engineering and Scientific Subroutine Library (IBM ESSL), complete the following steps.

Note: IBM ESSL V6.1.0-1 needs CUDA V9.2 or later.

1. Add the kit (distributed with the product media) to xCAT by running the **addkit** command:

```
$ addkit /path/to/essl-files/essl-6.1.0-1-ppc64le.tar.bz2
Adding Kit essl-6.1.0-1-ppc64le
Kit essl-6.1.0-1-ppc64le was successfully added.

$ lsdef -t kit
essl-6.1.0-1-ppc64le  (kit)
ibm_smpi_kt-10.2.0.6-rh7-ppc64le  (kit)
ppt-2.4.0-2  (kit)
xlc-16.1.0-1-ppc64le  (kit)
xlf-16.1.0-1-ppc64le  (kit)
```

2. Verify the `kitcomponent` objects (and `description` fields) by running the **lsdef** command:

```
$ lsdef -t kitcomponent -w kitname==essl-6.1.0-1-ppc64le -i description

Object name: essl-computenode-3264rte-6.1.0-1-rhels-7.5-ppc64le
    description=essl for compute nodes with 3264 rte only
Object name: essl-computenode-3264rtecuda-6.1.0-1-rhels-7.5-ppc64le
    description=essl for compute nodes with 3264 rte cuda only
Object name: essl-computenode-6.1.0-1-rhels-7.5-ppc64le
    description=essl for compute nodes
Object name: essl-computenode-6464rte-6.1.0-1-rhels-7.5-ppc64le
    description=essl for compute nodes with 6464 rte only
Object name: essl-computenode-nocuda-6.1.0-1-rhels-7.5-ppc64le
    description=essl for compute nodes
Object name: essl-license-6.1.0-1-rhels-7.5-ppc64le
    description=essl license for compute nodes
Object name: essl-loginnode-6.1.0-1-rhels-7.5-ppc64le
    description=essl for login nodes
Object name: essl-loginnode-nocuda-6.1.0-1-rhels-7.5-ppc64le
    description=essl for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by running the **`addkitcomp`** command:

```
$ addkitcomp --adddeps -i rh75-compute-stateless \
essl-computenode-6.1.0-1-rhels-7.5-ppc64le
Assigning kit component essl-computenode-6.1.0-1-rhels-7.5-ppc64le to osimage
rh75-compute-stateless
Kit components essl-computenode-6.1.0-1-rhels-7.5-ppc64le were added to osimage
rh75-compute-stateless successfully

$ lsdef -t osimage rh75-compute-stateless -i kitcomponents
Object name: rh75-compute-stateless

kitcomponents=xlc.license-compute-16.1.0-1-rhels-7-ppc64le,xlc.rte-compute-16.1
.0-1-rhels-7-ppc64le,xlf.license-compute-16.1.0-1-rhels-7-ppc64le,xlf.rte-compu
te-16.1.0-1-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.2.0.6-rh7-rhels-7-ppc64
le,ibm_spectrum_mpi-full-10.2.0.6-rh7-rhels-7-ppc64le,essl-license-6.1.0-1-rhel
s-7.5-ppc64le,essl-computenode-6.1.0-1-rhels-7.5-ppc64le,ppt.license-2.4.0-2-rh
els-7.4-ppc64le,ppt.compute-2.4.0-2-rhels-7.4-ppc64le
```

5.6.11 IBM Parallel Engineering and Scientific Subroutine Library

To install IBM Parallel ESSL, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT by running the **`addkit`** command:

```
$ addkit /path/to/pessl-files/pessl-5.4.0-0-ppc64le.tar.bz2
Adding Kit pessl-5.4.0-0-ppc64le.tar.bz2
Kit pessl-5.4.0-0-ppc64le was successfully added.

$ lsdef -t kit
essl-6.1.0-1-ppc64le  (kit)
ibm_smpi_kt-10.2.0.6-rh7-ppc64le  (kit)
pessl-5.4.0-0-ppc64le  (kit)
ppt-2.4.0-2  (kit)
xlc-16.1.0-1-ppc64le  (kit)
xlf-16.1.0-1-ppc64le  (kit
```

2. Verify the `kitcomponent` objects (and `description` fields) by running the **`lsdef`** command:

```
$ lsdef -t kitcomponent -w kitname==pessl-5.4.0-0-ppc64le -i description
Object name: pessl-computenode-3264rte-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for compute nodes with ESSL non-cuda runtime
Object name: pessl-computenode-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for compute nodes
Object name: pessl-computenode-nocuda-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for compute nodes
Object name: pessl-license-5.4.0-0-rhels-7.5-ppc64le
    description=pessl license for compute nodes
Object name: pessl-loginnode-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for login nodes
Object name: pessl-loginnode-nocuda-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by running the **`addkitcomp`** command:

```
$ addkitcomp --adddeps -i rh75-compute-stateless \
pessl-computenode-5.4.0-0-rhels-7.5-ppc64le
```

```
Assigning kit component pessl-computenode-5.4.0-0-rhels-7.5-ppc64le to osimage
rh75-compute-stateless
Kit components pessl-computenode-5.4.0-0-rhels-7.5-ppc64le were added to
osimage rh75-compute-stateless successfully

$ lsdef -t osimage rh75-compute-stateless -i kitcomponents
Object name: rh75-compute-stateless

kitcomponents=xlc.license-compute-16.1.0-1-rhels-7-ppc64le,xlc.rte-compute-16.1
.0-1-rhels-7-ppc64le,xlf.license-compute-16.1.0-1-rhels-7-ppc64le,xlf.rte-compu
te-16.1.0-1-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.2.0.6-rh7-rhels-7-ppc64
le,ibm_spectrum_mpi-full-10.2.0.6-rh7-rhels-7-ppc64le,essl-license-6.1.0-1-rhel
s-7.5-ppc64le,essl-computenode-6.1.0-1-rhels-7.5-ppc64le,pessl-license-5.4.0-0-
rhels-7.5-ppc64le,pessl-computenode-5.4.0-0-rhels-7.5-ppc64le,ppt.license-2.4.0
-2-rhels-7.4-ppc64le,ppt.compute-2.4.0-2-rhels-7.4-ppc64le
```

5.6.12 IBM Spectrum Scale (formerly IBM GPFS)

This section describes how to install the packages for IBM Spectrum Scale V5.0.1-2 and build and install the IBM GPFS Portability Layer packages.

The process to build the IBM GPFS Portability Layer requires a provisioned node (for example, a compute, login, or management node) with IBM Spectrum Scale packages that are installed (specifically, `gpfs.gpl`). Therefore, if the management node is not used to build the IBM GPFS Portability Layer, you cannot install IBM Spectrum Scale with the IBM GPFS Portability Layer in a single provisioning stage for one of the nodes, which is used to build the IBM GPFS Portability Layer. It is possible to do so afterward if the IBM GPFS Portability Layer package is made available for download in the management node (such as other IBM Spectrum Scale packages) for installation on other nodes.

For more information about the installation process, see the Installing GPFS on Linux nodes page. To install IBM Spectrum Scale V5.0.1-2, complete the following steps:

1. Install a script for the environment configuration.
2. Install the packages for IBM Spectrum Scale 5.0.1.-2.
3. Build the IBM GPFS Portability Layer.

Installing a script for the environment configuration

To install the script for the environment configuration, complete the following steps:

1. Create a script to set the **PATH** environment variable:

```
$ script=/install/postscripts/gpfs-setup

$ cat <<EOF >$script
#!/bin/bash

profile='/etc/profile.d/gpfs.sh'
echo 'export PATH=\$PATH:/usr/lpp/mmfs/bin' >\$profile
# REPLACE GPFSNODE with a GPFS management-node in the cluster, which
# compute nodes are allowed to ssh to as root:
#
/usr/lpp/mmfs/bin/mmsdrrestore -p GPFSNODE
/usr/lpp/mmfs/bin/mmstartup
EOF
```

```
$ chmod +x $script

$ ls -l $script
-rwxr-xr-x 1 root root 99 Nov 30 17:24 /install/postscripts/gpfs-setup
```

2. Include it in the `postscripts` list of the `osimage` object by running the **`chdef`** command:

```
$ lsdef -t osimage rh75-compute-stateless -i postbootscripts
Object name: rh75-compute-stateless
    postbootscripts=

$ chdef -t osimage rh75-compute-stateless --plus postbootscripts='gpfs-setup'
1 object definitions have been created or modified.

$ lsdef -t osimage rh75-compute-stateless -i postbooscripts
Object name: rh75-compute-stateless
    postscripts=gpfs-setup
```

Installing the packages for IBM Spectrum Scale V5.1.0-2

To install the packages, complete the following steps:

1. Extract the product packages of IBM Spectrum Scale V5.1.0-2:

```
$ dir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le/gpf
$ mkdir -p $dir

$ /path/to/Spectrum_Scale_Data_Management-5.0.1.2-ppc64LE-Linux-install
--text-only --silent --dir $dir
<...>
Extracting Product RPMs to
/install/post/otherpkgs/rhels7.5-alternate/ppc64le/gpfs
<...>
```

2. Create the respective package repository by running the **`createrepo`** command:

```
$ createrepo $dir
<...>

$ ls -1 $dir
gpfs.adv-5.0.1-2.ppc64le.rpm
gpfs.base-5.0.1-2.ppc64le.rpm
gpfs.callhome-ecc-client-5.0.1-2.noarch.rpm
gpfs.compression-5.0.1-2.ppc64le.rpm
gpfs.crypto-5.0.1-2.ppc64le.rpm
gpfs.docs-5.0.1-2.noarch.rpm
gpfs.ext-5.0.1-2.ppc64le.rpm
gpfs.gpl-5.0.1-2.noarch.rpm
gpfs.gskit-8.0.50-86.ppc64le.rpm
gpfs.gui-5.0.1-2.noarch.rpm
gpfs.hdfs-protocol-2.7.3-3.ppc64le.rpm
gpfs.java-5.0.1-2.ppc64le.rpm
gpfs.license.dm-5.0.1-2.ppc64le.rpm
gpfs.msg.en_US-5.0.1-2.noarch.rpm
gpfs.tct.client-1.1.5.2.ppc64le.rpm
gpfs.tct.server-1.1.5.2.ppc64le.rpm
repodata
```

3. Create the respective package list with a combination of commands:

```
$ list=/install/custom/netboot/rh/rh75-compute.otherpkglist
$ echo “####GPFS####” >> $list
$ rpm -qip *.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename $dir)/:" |
grep -v -E 'msg|java|callhome|crypto|gui|hdfs|client|server' >>$list


$ cat $list
gpfs/gpfs.adv
gpfs/gpfs.base
gpfs/gpfs.compression
gpfs/gpfs.docs
gpfs/gpfs.ext
gpfs/gpfs.gpl
gpfs/gpfs.gskit
gpfs/gpfs.license.dm
```

Building the IBM GPFS Portability Layer

The process to build the IBM GPFS Portability Layer requires a working node (for example, a management, login, or compute node) with IBM Spectrum Scale packages (specifically, `gpfs.gpl`) installed.

The node must be capable of building out-of-tree kernel modules (that is, with development packages such as `gcc`, `make`, and `kernel-devel` installed), and run the same kernel packages version and processor architecture as the target compute nodes for the IBM GPFS Portability Layer produced binary files.

For example, you can build the IBM GPFS Portability Layer in the management node (as in the example below) or an installed compute node (if any) if it is a system that is on a POWER9 processor-based server that is running Red Hat Enterprise Linux Server 7.5 alternate for ppc64le and matches the kernel packages version of the target compute nodes. This section describes the build process with an installed compute node.

Note: The IBM GPFS Portability Layer must be rebuilt and reinstalled if the kernel packages are updated. This process requires the `kernel-devel` package for the respective update version.

To perform the IBM GPFS Portability Layer build, complete the following steps:

1. Verify the IBM Spectrum Scale installation (**`PATH`** environment variable and RPM packages):

```
$ 'echo $PATH' | grep mmfs
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/lpp/mmfs/bin:/opt/ibutil
s/bin:/root/.local/bin:/root/bin

$ rpm -qa | grep ^gpfs
gpfs.gpl-5.0.1-2.noarch
gpfs.base-5.0.1-2.ppc64le
gpfs.adv-5.0.1-2.ppc64le
gpfs.license.dm-5.0.1-2.ppc64le
gpfs.gskit-8.0.50-86.ppc64le
gpfs.ext-5.0.1-2.ppc64le
gpfs.compression-5.0.1-2.ppc64le
```

2. Build the IBM GPFS Portability Layer.

   Verify that the build process writes the resulting IBM GPFS Portability Layer binary RPM package (`gpfs.gplbin-<kernel version>.<architecture>-<spectrum scale version>.rpm`) and finishes with an exit code of `zero` (success):

```
$/usr/lpp/mmfs/bin/mmbuildgpl --build-package
--------------------------------------------------------
mmbuildgpl: Building GPL module begins at Tue Oct 23 18:42:52 EDT 2018.
--------------------------------------------------------
<...>
Wrote:
/root/rpmbuild/RPMS/ppc64le/gpfs.gplbin-4.14.0-49.13.1.el7a.ppc64le-5.0.1-2.ppc
64le.rpm
--------------------------------------------------------
mmbuildgpl: Building GPL module completed successfully at Tue Oct 23 18:43:07
EDT 2018.
--------------------------------------------------------
```

3. Copy the package and create the respective package repository with the **createrepo** command:

```
$ dir=/install/post/otherpkgs/rhels7.5-alternate/ppc64le/gpfs-gpl
$ mkdir $dir

$ cp /root/rpmbuild/RPMS/ppc64le/gpfs.gplbin-*.rpm $dir
gpfs.gplbin-4.14.0-49.13.1.el7a.ppc64le-5.0.1-2.ppc64le.rpm <...>

$ createrepo $dir
<...>

$ ls -1 $dir
gpfs.gplbin-4.14.0-49.13.1.el7a.ppc64le-5.0.1-2.ppc64le.rpm
repodata
```

4. Create the respective package list by running the following commands:

```
$ echo $list
$/install/custom/netboot/rh/rh75-compute.otherpkglist

$ echo '#GPFS_GPL#' >>$list
$ rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" >>$list

$ cat $list
[root@mgmtnode1 gpfs]# cat $list
# CUDA
cuda-deps/dkms
cuda-9.2/cuda-runtime-9-2
# Advance Toolchain
at/advance-toolchain-at12.0-runtime
at/advance-toolchain-at12.0-mcore-libs
####GPFS####
gpfs/gpfs.license.dm
gpfs/gpfs.ext
gpfs/gpfs.gpl
gpfs/gpfs.gskit
gpfs/gpfs.compression
gpfs/gpfs.adv
```

```
gpfs/gpfs.base
#GPFS_GPL
gpfs-gpl/gpfs.gplbin-4.14.0-49.13.1.el7a.ppc64le
```

Note: You can install the IBM GPFS Portability Layer in an installed and running compute node by using the **otherpkgs** script of the **updatenode** command:

```
$ updatenode p9r1n1 -P otherpkgs
<...>
   p9r1n1: pkgsarray:  gpfs/gpfs.gplbin-4.14.0-49.13.1.el7a.ppc64le
, 2
<...>

$ xdsh p9r1n1 'rpm -qa | grep ^gpfs.gpl'


p9r1n1: gpfs.gpl-5.0.1-2.noarch
p9r1n1: gpfs.gplbin-4.14.0-49.13.el7.ppc64le
```

For more information about configuration and usage, see the Steps to establishing and starting your GPFS cluster page.

5.6.13 PGI runtime libraries

To install The Portland Group, Inc. (PGI) runtime libraries for OpenACC, complete the following steps. The scenario that is described in this section has an IBM Spectrum Scale file system available at `/gpfs/gpfs0/`; change `GPFSNODE` to `gpfs mgmt-node` to fit your scenario.

Note: PGI Compilers are available for download at the PGI Community Edition page.

At the website, click **Linux OpenPOWER** to begin the download process.

Because you must accept a user agreement, you must download this package with a browser.

No PGI Compiler runtime libraries-only packages are available. Therefore, libraries must be copied manually.

1. Create the `pgi` directory in the `GPFSNODE` GPFS:

```
$ dir=/gpfs/gpfs0/pgi
$ mkdir -p $dir

# Download via browser and scp to GPFSNODE pgi directory
$ scp pgilinux-2018-187-ppc64le.tar.gz gpfs-mn:/$dir
```

2. Extract the package:

```
$ cd $dir
$ tar -xf pgilinux-2018-187-ppc64le.tar.gz

$ ls -1
documentation.html
install
install_components
pgilinux-2018-187-ppc64le.tar.gz
```

3. Install the package by using the `./install` option and pointing to the `/gpfs/gpfs0/pgi` directory:

```
$ ./install
<...>
Do you accept these terms? (accept,decline) accept

<...>
Installation directory? [/opt/pgi] /gpfs/gpfs0/pgi
local directory on all hosts for shared objects: [/opt/pgi/18.7/share_objects]
/gpfs/gpfs0/pgi/18.7/share_objects


************************************************************************
CUDA Toolkit
************************************************************************
<...>
Do you accept these terms? (accept,decline)      accept
Installing PGI version 18.7 into /gpfs/gpfs0/pgi
#####################
<...>
Do you wish to update/create links in the 2018 directory? (y/n) y
<...>
Do you want to install OpenMPI onto your system? (y/n)n
<...>
Do you want the files in the install directory to be read-only? (y/n) y

Installation complete.
```

Note: Run **`add_network_host`** to create host-specific `localrc` files and run **`$ /gpfs/gpfs0/pgi/linuxpower*/bin/localrc.$host`** on all the other hosts on which you want to run PGI compilers.

For 64-bit PGI compilers on 64-bit Linux systems, run the following command:

```
$ /gpfs/gpfs0/pgitwo/linuxpower/18.7/bin/add_network_host
```

5.6.14 IBM Spectrum LSF integration with Cluster Systems Management

As mentioned in 3.4.1, "IBM Spectrum Load Sharing Facility" on page 28, a special version of IBM Spectrum Load Sharing Facility (IBM Spectrum LSF) is used for the HPC type of clusters in this book, it is different from typical the IBM Spectrum LSF product and IBM Spectrum LSF suite binary files. It is an RPM installation that mainly provides IBM Spectrum LSF functions that are integrated with IBM Cluster Systems Management (CSM) APIs.

This section describes how to install and configure this specific IBM Spectrum LSF version and do the integration with CSM.

Downloading and installing the CSM integration package

Complete the following steps:

1. Download the installation package for the IBM Spectrum LSF integration with CSM (see the following note). The installation package is `lsf-10.1-build_number.ppc64le_csm.bin`.

   Note: For more information about the location of the RPM files, contact your IBM Sales Representative.

2. Run the installation package to extract the RPM files:

   ```
   $ ./lsf-10.1-build_number.ppc64le_csm.bin
   ```

3. Accept the license agreement when prompted to continue with the file extraction.

   The extracted installation package contains the following RPM files:

   – `lsf-common-10.1-yyyyddmm.ppc64le.rpm`

   – `lsf-master-10.1-yyyyddmm.ppc64le.rpm`

   – `lsf-misc-10.1-yyyyddmm.ppc64le.rpm`

   – `lsf-server-10.1-yyyyddmm.ppc64le.rpm`

   In the RPM files, `yyyyddmm` is the build date.

4. Deploy the common, server, and master RPM packages on to the CSM hosts:

   – On the workload manager (WLM) node, run the **`rpm -ivh`** or **`yum install`** command to deploy the common, server, and master packages. By default, the package is installed in the `/opt/ibm/spectrumcomputing/lsf/` directory. You can use the **`--prefix`** option to change this installation directory.

   ```
   $rpm -ivh lsf-common-10.1-yyyyddmm.ppc64le.rpm
   lsf-server-10.1-build_number.ppc64le.rpm
   lsf-master-10.1-build_number.ppc64le.rpm
   ```

   – On the launch node (LN), run the **`rpm -ivh`** or **`yum install`** command to deploy the common and server packages. By default, the package is installed to the `/opt/ibm/spectrumcomputing/lsf/` directory. You can use the **`--prefix`** option to change this installation directory.

   ```
   rpm -ivh lsf-common-10.1-yyyyddmm.ppc64le.rpm
   lsf-server-10.1-build_number.ppc64le.rpm\
   ```

5. Verify that the installation is successful:

   – Verify that the IBM Spectrum LSF packages are installed:

   ```
   rpm -qa | grep lsf
   lsf-server-10.1-yyyyddmm.ppc64le
   lsf-common-10.1-yyyyddmm.ppc64le
   lsf-master-10.1-yyyyddmm.ppc64le
   ```

   – Run the **`rpm -qi`** command to find the build number in the package description.

6. Copy the entitlement file to the `conf/lsf.entitlement` subdirectory of the installation directory:

   – If you installed the packages with the default installation directory, copy the entitlement file to the `/opt/ibm/spectrumcomputing/lsf/conf/lsf.entitlement` directory.

   – If you specified a custom installation directory, copy the entitlement file to the `conf/lsf.entitlement` subdirectory of the directory that you specified with the **`--prefix`** installation option.

Configuring the Cluster Systems Management integration

To integrate CSM, complete the following steps:

1. Edit the `lsf.shared` file to add the name of the cluster with the CSM hosts and add the name and properties of the CSM resources:

   ```
   Begin Cluster
   ClusterName                    # Keyword
   ...
   cluster1                       # Example
   End Cluster
   ...
   Begin Resource
   RESOURCENAME         TYPE    INTERVAL INCREASING DESCRIPTION  # Keywords
   ...
      LN                Boolean  ()        ()        (CSM launch nodes)
      CN                Boolean  ()        ()        (CSM compute nodes)
      gpus              Numeric  ()        Y         (number of gpus on csm cn)
      vg_total_size     Numeric  ()        Y         (total ssd on csm cn)
      vg_available_size Numeric  15        N         (available ssd on csm cn)
      ssd0_wear         Numeric  15        Y         (ssd wear on csm cn)
   ...
   End Resource
   ```

2. Edit the *lsf.cluster.cluster1* file and add the relevant WLM and LN hosts and resources. For example, edit the `lsf.cluster.cluster1` file and add the following lines:

   ```
   Begin ClusterAdmins
   Administrators = lsfadmin
   End ClusterAdmins

   Begin Host
   HOSTNAME model type  server  RESOURCES #Keywords
   hostWLM    !    !       1     (mg)    #Example only. Use the real host name
   hostLN     !    !       1     (LN)    #Example only. Use the real host name
   End Host

   Begin ResourceMap
   RESOURCENAME      LOCATION
   gpus              (0@[wlm ln]) #none on WLM and LN
   vg_total_size     (0@[wm ln])  #none on WLM and LN
   vg_available_size ([default])
   ssd0_wear         ([default])
   End ResourceMap
   ```

3. Edit the `lsf.conf` file on the CSM hosts and define the CSM parameters:

   – On the WLM host, define the following parameters in the **`lsf.conf`** file:

     ```
     LSB_MIG2PEND=1
     EGO_DEFINE_NCPUS=cores
     LSF_CSM_LIB_PATH=/opt/ibm/csm/lib/libcsmi.so
     LSF_JSM_PATH=/opt/ibm/spectrum_mpi/jsm_pmix/bin/jsm
     LSB_BJOBS_DISPLAY_ENH=Y
     LSB_SHORT_HOSTLIST=1
     LSF_HPC_EXTENSIONS="CUMULATIVE_RUSAGE LSB_HCLOSE_BY_RES SHORT_EVENTFILE"
     LSB_BJOBS_PENDREASON_LEVEL=1
     LSF_ADD_HOST_PACK_FROM="30 6883 <master_candidate> <master> <launch_nodes>"
     #Parameters for easy mode job submission
     ```

```
LSB_CSM_JOBS=Y
LSB_EASY_PTILE=x #x is the total number of cores on each CN, but if
                 #core isolation is enforced cluster-wide, x is the
                 #total number of cores on each CN minus the
                 #core_isolation value
```

– On the LN host, define the following parameters in the `lsf.conf` file:

```
LSF_LIM_PORT=6809
LSB_MIG2PEND=1
LSB_RLA_PORT=6883
LSF_SERVER_HOSTS="hostWLM hostWLM_candidates"
# job run time parameters
LSF_CSM_LIB_PATH=/opt/ibm/csm/lib/libcsmi.so
LSF_JSM_PATH=/opt/ibm/spectrum_mpi/jsm_pmix/bin/jsm
# job run time environment variables
LSB_CSM_JOBS=Y
LSB_EASY_PTILE=x #x is the total number of cores on each CN, but if
                 #core isolation is enforced cluster-wide, x is the
                 #total number of cores on each CN minus the
                 #core_isolation value
# reporting CN to master lim
LSF_ADD_HOST_PACK_FROM="30 6883 <master_candidate> <master>" # replace
rla_port with the port number. Replace WLM_candidate LN WLMN with real host
names
```

4. On the WLM host, edit the `lsb.modules` file and configure the CSM modules:

```
Begin PluginModule
SCH_PLUGIN                      RB_PLUGIN                    SCH_DISABLE_PHASES
schmod_default                  ()                              ()
schmod_fcfs                     ()                              ()
schmod_fairshare                ()                              ()
schmod_limit                    ()                              ()
schmod_parallel                 ()                              ()
schmod_reserve                  ()                              ()
schmod_mc                       ()                              ()
schmod_preemption               ()                              ()
schmod_advrsv                   ()                              ()
schmod_ps                       ()                              ()
schmod_affinity                 ()                              ()
schmod_aps                      ()                              ()
schmod_datamgr                  ()                              ()
End PluginModule
```

5. On the WLM host, edit the IBM Spectrum LSF hosts file and add the names of all CSM hosts as needed. Add placeholder IP addresses because specific IP addresses are not needed. For example:

```
10.0.0.1 hostWLM
10.0.0.2 hostLN
```

6. On the WLM host, edit the `lsb.hosts` file and add the WLM and LN hosts. For example:

```
Begin Host
HOST_NAME  MXJ   r1m     pg    ls    tmp  DISPATCH_WINDOW  # Keywords
default      !   ()      ()    ()     ()     ()           # Use the same value
as the LSB_EASY_PTILE parameter
hostWLM      !   ()      ()    ()     ()     ()
hostLN       !   ()      ()    ()     ()     ()
End Host
```

7. On the IBM Spectrum LSF master host, define the parameter values for CSM job submission:

   – Define the CSM parameters for all user job queues in the `lsb.queues` file. For IBM Spectrum LSF job queues to work with CSM, add the following parameter values to all user job queues in the lsb.queues file:

   ```
   JOB_CONTROLS = SUSPEND[bmig $LSB_BATCH_JID]
   RERUNNABLE = Y
   ```

   – Optional: Define any default values for the CSM job submission options in the `lsf.conf` file. Define the **`LSB_JSM_DEFAULT`** parameter as the default value for the **`bsub -jsm`** option and the **`LSB_STEP_CGROUP_DEFAULT`** parameter as the default value for the **`bsub -step_cgroup option`**:

   ```
   LSB_JSM_DEFAULT=y | n

   LSB_STEP_CGROUP_DEFAULT=y | n
   ```

   For example:

   ```
   LSB_JSM_DEFAULT=y
   LSB_STEP_CGROUP_DEFAULT=y
   ```

   – Optional: Define any required values for the CSM job submission options at the queue level in the `lsb.queues` file. These settings override any job-level CSM options and append system level allocation flags to the job-level allocation flags. Define the **`CSM_REQ`** parameter with the CSM option settings to override the job-level CSM options:

   ```
   CSM_REQ= [jsm=y | n | d] [:step_cgroup=y | n] [: [-core_isolation 0 | 1 | 2
   | 3 | 4 | 5] [:cn_mem=mem_value] ] [:alloc_flags flag_str]
   ```

   Note: The options can appear in any order and none are required. The `flag_str` string value for the **`alloc_flags`** keyword cannot contain a colon (:).

   For example:

   ```
   CSM_REQ=jsm=n:step_cgroup=y:-core_isolation 3:cn_mem=1024
   ```

8. Restart the IBM Spectrum LSF daemons. On each WLM and LN host, run the following commands as root:

```
. /opt/ibm/spectrumcomputing/lsf/conf/profile.lsf
lsf_daemons stop
lsf_daemons start
```

Note: For more information about the IBM Spectrum LSF and CSM integration, see IBM Spectrum LSF with IBM Cluster Systems Management.

Chapter 6, “Cluster Administration and Storage Tools” on page 137details the deployment of Cluster Administration and Storage Tools (CAST) (CSM and IBM Burst Buffer (BB)).

5.6.15 Synchronizing the configuration files

Add all configuration files to your `osimage synclists` definition so that all the files are up-to-date and in sync across all nodes and images. When you generate or regenerate a stateless image or install a stateful image, the syncfiles are updated. You also can force the syncfile process by running the **`updatenode <node> -F`** command.

For more information, see Synchronizing Files.

The scenario in this section shows the synchronization of `/etc/hosts` to reduce host name and IP resolution time.

Note: If service nodes are used to manage you nodes (hierarchy), you must ensure that the service nodes were synchronized with the latest files from the management node before installing.

If you have a group of compute nodes (compute) that are going to be installed that are serviced by a service node, run the **`updatenode compute -f`** command before the installation to synchronize the current files to the service node. The node range is the compute node names, and **`updatenode`** determines which service nodes need updating.

Complete the following steps:

1. Create required folders:

```
$ syncdir=/install/custom/syncfiles/common
$ mkdir -p $syncdir

$ cd $syncdir
$ mkdir etc
$ cd etc
```

2. Create the hosts file (link to `/etc/hosts` in this case):

```
$ ln -s /etc/hosts hosts
$ pwd
/install/custom/syncfiles/common/etc
$ ls -l
lrwxrwxrwx 1 root root 10  1. Dez 15:24 hosts -> /etc/hosts
```

3. Create the `synclist` file.

Note: For more information about the synclist file syntax, see the The Format of synclist file page.

```
$ synclist=/install/custom/netboot/rh/rh75-compute.synclist

$ echo "/install/custom/syncfiles/common/etc/* -> /etc/" >> $synclist

$ cat $synclist
/install/custom/syncfiles/common/etc/* -> /etc/
```

4. Add the `synclist` to the `osimage` definition:

```
$ lsdef -t osimage rh75-compute-stateless -i synclists
Object name: rh75-compute-stateless
    synclists=
$ chdef -t osimage rh75-compute-stateless synclists=$synclist
1 object definitions have been created or modified.

$ lsdef -t osimage rh75-compute-stateless -i synclists
Object name: rh75-compute-stateless
    synclists=/install/custom/netboot/rh/rh75-compute.synclist
```

For more configuration files for a specific node type, create a directory in `/install/custom/syncfiles` and add the directory to the corresponding `synclist` file, as shown in the following example:

```
/install/custom/syncfiles/compute/opt
```

As of xCAT 2.14, nodes and node groups can be added to the `synclist` file. For more information, see the Support nodes in synclist file page.

5.6.16 Generating and packing the image

Because this image is a stateless image, you must generate and pack the image before providing it. The following steps are needed for a stateless image only:

1. Generate the stateless image by running the **`genimage`** command. It uses the `osimage` definition information to generate the image.

   ```
   $ genimage rh75-compute-stateless
   Generating image:
   <...>
   the initial ramdisk for statelite is generated successfully.
   ```

   This process can take some time. The installation process runs in a **`chroot`** environment on the management node. Monitor the building process carefully and watch for possible errors.

2. Check the content of the generated image:

   ```
   $ lsdef -t osimage rh75-compute-stateless -i rootimgdir
   Object name: rh75-compute-stateless
   rootimgdir=/install/netboot/rhels7.5-alternate/ppc64le/rh75-compute-stateless
   $ rootimgdir=/install/netboot/rhels7.5-alternate/ppc64le/rh75-compute-stateless

   $ ls -l $rootimgdir/rootimg
   lrwxrwxrwx  1 root root    7 Oct 10 16:07 bin -> usr/bin
   dr-xr-xr-x  4 root root 4096 Oct 18 19:00 boot
   drwxr-xr-x  2 root root 4096 Oct 22 16:23 dev
   drwxr-xr-x 75 root root 4096 Oct 22 16:27 etc
   drwxr-xr-x  2 root root 4096 Dec 14  2017 home
   lrwxrwxrwx  1 root root    7 Oct 10 16:07 lib -> usr/lib
   lrwxrwxrwx  1 root root    9 Oct 10 16:07 lib64 -> usr/lib64
   drwxr-xr-x  2 root root 4096 Dec 14  2017 media
   drwxr-xr-x  2 root root 4096 Dec 14  2017 mnt
   drwxr-xr-x 10 root root 4096 Oct 22 16:23 opt
   drwxr-xr-x  2 root root 4096 Oct 10 16:07 proc
   dr-xr-x---  2 root root 4096 Oct 11 19:06 root
   drwxr-xr-x 13 root root 4096 Oct 22 16:22 run
   lrwxrwxrwx  1 root root    8 Oct 10 16:07 sbin -> usr/sbin
   drwxr-xr-x  2 root root 4096 Dec 14  2017 srv
   drwxr-xr-x  2 root root 4096 Oct 10 16:07 sys
   drwxrwxrwt  8 root root 4096 Oct 22 16:24 tmp
   ```

```
drwxr-xr-x 15 root root 4096 Oct 11 16:57 usr
drwxr-xr-x 20 root root 4096 Oct 11 16:57 var
drwxr-xr-x  7 root root 4096 Oct 22 16:27 xcatpost
```

Note: After your image is generated, you can **`chroot`** to the image, install any other software, or modify the files manually. As a best practice, do the installation steps by using all of the mechanisms that are described in 5.3.5, "xCAT installation types: Disks and state" on page 62 because they ensure that a reproducible image is produced.

3. Pack the generated image with the **`packimage`** command:

```
$ packimage rh75-compute-stateless
Packing contents of
/install/netboot/rhels7.5-alternate/ppc64le/rh75-compute-stateless/rootimg
archive method:cpio
compress method:gzip
```

Note: The `packimage` process can be accelerated by installing the parallel compression tool `pigz` on the management node. For more information, see Accelerating the diskless initrd and rootimg generating.

5.6.17 Node provisioning

To start provisioning a compute node (or node group) and load the OS and software stack according to the `osimage` and node group objects, complete the following steps:

1. Define the `osimage` attribute of a node (or node group) by running the **`nodeset`** command:

```
$ nodeset p9r1n1 osimage=rh75-compute-stateless
p9r1n1: netboot rhels7.5-alternate-ppc64le-compute
```

2. You can verify the changes to the nodes attributes by running the **`lsdef`** command:

```
$ lsdef p9r1n1
Object name: p9r1n1
    <...>
    initrd=xcat/osimage/rh75-compute-stateless/initrd-stateless.gz
    <...>
kcmdline=imgurl=http://!myipfn!:80//install/netboot/rhels7.5-alternate/ppc64le/
rh75-compute-stateless/rootimg.cpio.gz XCAT=!myipfn!:3001 NODE=p9r1n1 FC=0
BOOTIF=70:e2:84:14:09:ae
    kernel=xcat/osimage/rh75-compute-stateless/kernel
    <...>
    os=rhels7.5-alternate
    <...>
    profile=compute
    provmethod=rh75-compute-stateless
    <...>
```

3. Set the boot method to network by running the **`rsetboot`** command (optional):

```
$ rsetboot p9r1n1 net
```

Note: This process is performed automatically by the **`nodeset`** command and is not required if the bootloader configuration for automatic boot is correct. For more information, see 5.2.4, "Boot order configuration" on page 57.

4. (Optional) Restart the node (or node group) by running the **rpower** command:

```
$ rpower p9r1n1 reset
```

Note: This process is performed automatically (within some time) if the Genesis image for node discovery is still running in the compute node (waiting for instructions from the management node).

5. You can watch the node's console by running the **rcons** command:

```
$ rcons p9r1n1
```

6. You can monitor the node boot progress in the node object and the `/var/log/messages` log file by running the following command:

```
$ lsdef  p9r1n1 -i status
Object name: p9r1n1
    status=powering-on

$ lsdef  p9r1n1 -i status
Object name: p9r1n1
    status=netbooting

lsdef  p9r1n1 -i status
Object name: p9r1n1
    status=booted

$tail -f /var/log/messages | grep p9r1n1
<...>
```

5.6.18 Postinstallation verification

This section describes the steps that are used to verify that the software stack is correctly provisioned.

Verifying the CUDA Toolkit

Verify all GPUs are listed by running the **nvidia-smi** command:

```
$ xdsh p9r1n1 'nvidia-smi --list-gpus'

p9r1n1: GPU 0: Tesla V100-SXM2-16GB (UUID:
GPU-3f48806d-2753-dce5-ee83-8ced3c835875)
p9r1n1: GPU 1: Tesla V100-SXM2-16GB (UUID:
GPU-261ba904-ad76-7d82-6d8f-d2a3d80744b4)
```

Verifying the Mellanox OpenFabrics Enterprise Distribution

To verify the installation of Mellanox OFED, complete the following steps:

1. Verify that the `openibd` service is correctly loaded and active by running the **systemctl** command:

```
xdsh p9r1n1 'systemctl status openibd'
p9r1n1: * openibd.service - openibd - configure Mellanox devices
p9r1n1:    Loaded: loaded (/usr/lib/systemd/system/openibd.service; enabled;
vendor preset: disabled)
p9r1n1:    Active: active (exited) since Tue 2018-10-23 19:25:46 GMT; 20h ago
p9r1n1:      Docs: file:/etc/infiniband/openib.conf
```

```
p9r1n1:  Main PID: 2470 (code=exited, status=0/SUCCESS)
p9r1n1:    CGroup: /system.slice/openibd.service
p9r1n1:
p9r1n1: Oct 23 19:25:46 p9r1n1.cluster systemd[1]: Starting openibd - configure
Mellanox devices...
p9r1n1: Oct 23 19:25:46 p9r1n1.cluster openibd[2470]: Loading HCA driver and
Access Layer:[  OK  ]
p9r1n1: Oct 23 19:25:46 p9r1n1.cluster systemd[1]: Started openibd - configure
Mellanox devices.
```

2. Verify the information and status of the InfiniBand adapter and ports by running the **ibstat** command:

```
xdsh p9r1n1 ibstat
p9r1n1: CA 'mlx5_0'
p9r1n1:         CA type: MT4121
p9r1n1:         Number of ports: 1
p9r1n1:         Firmware version: 16.21.0138
p9r1n1:         Hardware version: 0
p9r1n1:         Node GUID: 0xec0d9a0300cab918
p9r1n1:         System image GUID: 0xec0d9a0300cab918
p9r1n1:         Port 1:
p9r1n1:                 State: Active
p9r1n1:                 Physical state: LinkUp
p9r1n1:                 Rate: 100
p9r1n1:                 Base lid: 0
p9r1n1:                 LMC: 0
p9r1n1:                 SM lid: 0
p9r1n1:                 Capability mask: 0x04010000
p9r1n1:                 Port GUID: 0xee0d9afffecab918
p9r1n1:                 Link layer: InfiniBand
p9r1n1: CA 'mlx5_1'
p9r1n1:         CA type: MT4121
p9r1n1:         Number of ports: 1
p9r1n1:         Firmware version: 16.21.0138
p9r1n1:         Hardware version: 0
p9r1n1:         Node GUID: 0xec0d9a0300cab919
p9r1n1:         System image GUID: 0xec0d9a0300cab918
p9r1n1:         Port 1:
p9r1n1:                 State: Active
p9r1n1:                 Physical state: LinkUp
p9r1n1:                 Rate: 100
p9r1n1:                 Base lid: 0
p9r1n1:                 LMC: 0
p9r1n1:                 SM lid: 0
p9r1n1:                 Capability mask: 0x04010000
p9r1n1:                 Port GUID: 0xee0d9afffecab919
p9r1n1:                 Link layer: InfiniBand
```

Verifying the installed runtime libraries and Message Passing Interface

Verify the installed version of all runtime libraries and MPI by running the **rpm** command:

```
$ xdsh p9r1n1 'rpm -qa | \
   grep -E "xlc|xlf|advance-toolchain|ppt|smpi|essl|pessl"'
p9r1n1: essl.license-6.1.0-0.ppc64le
p9r1n1: essl.3264.rte-6.1.0-1.ppc64le
p9r1n1: pessl.common-5.4.0-0.ppc64le
```

```
p9r1n1: essl.rte-6.1.0-0.ppc64le
p9r1n1: xlf.license-compute-16.1.0-1.noarch
p9r1n1: libxlc-16.1.0.0-180413g.ppc64le
p9r1n1: essl-computenode-6.1.0-1.noarch
p9r1n1: ppt.compute-2.4.0-2.noarch
p9r1n1: advance-toolchain-at12.0-mcore-libs-12.0-0.ppc64le
p9r1n1: ibm_smpi_lic_s-10.02-p9_20180611.ppc64le
p9r1n1: libxlf-16.1.0.0-180413g.ppc64le
p9r1n1: essl.rte.common-6.1.0-0.ppc64le
p9r1n1: pessl.rte.common-5.4.0-0.ppc64le
p9r1n1: pessl.msg-5.4.0-0.ppc64le
p9r1n1: essl.6464.rte-6.1.0-1.ppc64le
p9r1n1: essl.3264.rtecuda-6.1.0-1.ppc64le
p9r1n1: essl.common-6.1.0-0.ppc64le
p9r1n1: xlf-license.16.1.0-16.1.0.0-180413g.ppc64le
p9r1n1: xlc.license-compute-16.1.0-1.noarch
p9r1n1: essl-license-6.1.0-1.noarch
p9r1n1: pessl-license-5.4.0-0.noarch
p9r1n1: xlc.rte-compute-16.1.0-1.noarch
p9r1n1: ppt_license-2.4.0-2.ppc64le
p9r1n1: ppt_mrnet-2.4.0-2.ppc64le
p9r1n1: advance-toolchain-at12.0-runtime-12.0-0.ppc64le
p9r1n1: ibm_smpi-10.02.00.06rtm1-rh7_20180811.ppc64le
p9r1n1: pessl.license-5.4.0-0.ppc64le
p9r1n1: pessl.3264.rte-5.4.0-0.ppc64le
p9r1n1: essl.msg-6.1.0-0.ppc64le
p9r1n1: xlc-license.16.1.0-16.1.0.0-180413g.ppc64le
p9r1n1: xlf.rte-compute-16.1.0-1.noarch
p9r1n1: pessl-computenode-5.4.0-0.noarch
p9r1n1: ppt_runtime-2.4.0-2.ppc64le
```

Verifying IBM Spectrum Scale

Complete the following steps:

1. Verify that the IBM Spectrum Scale packages are installed by running the **`rpm`** command:

   ```
   $ xdsh p9r1n1 'rpm -qa | grep ^gpfs'
   p9r1n1: gpfs.gpl-5.0.1-2.noarch
   p9r1n1: gpfs.gplbin-4.14.0-49.el7a.ppc64le-5.0.1-2.ppc64le
   p9r1n1: gpfs.base-5.0.1-2.ppc64le
   p9r1n1: gpfs.adv-5.0.1-2.ppc64le
   p9r1n1: gpfs.license.dm-5.0.1-2.ppc64le
   p9r1n1: gpfs.gskit-8.0.50-86.ppc64le
   p9r1n1: gpfs.gplbin-4.14.0-49.13.1.el7a.ppc64le-5.0.1-2.ppc64le
   p9r1n1: gpfs.ext-5.0.1-2.ppc64le
   p9r1n1: gpfs.compression-5.0.1-2.ppc64le
   ```

2. Verify that the node does not yet belong to any cluster by running the **`mmlscluster`** command:

   ```
   $ xdsh p9r1n1 'mmlscluster'
   p9r1n1: mmlscluster: This node does not belong to a GPFS cluster.
   p9r1n1: mmlscluster: Command failed. Examine previous error messages to
   determine cause.
   ```

(Optional) Checking public and site network connectivity

Verify the connectivity to external and non-cluster nodes by running the **ping** command:

```
$ xdsh p9r1n1 'ping -c1 example.com'
p9r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p9r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=3.94 ms
<...>
```

5.7 xCAT login nodes (stateful)

The process that is used to deploy an xCAT login node is similar to the xCAT compute node deployment. The main difference is the particular software stack components that are used on each node type. Also, other methods require some changes because the login nodes often use a `stateful` image.

Therefore, this section does not describe the deployment instructions for login nodes. Instead, it describes some examples of the differences in the software stack components and methods between login and compute nodes.

The following differences are typically considered for the login nodes:

- Use the `rhels7.5-alternate-ppc64le-install-compute osimage` (available images can be seen by running **`lsdef -t osimage`**) as an initial template.
- Consider the configuration of RAID and disk partitions. For more information, see the following websites:
  - Configure RAID before deploying the OS
  - Configure Disk Partition
- CUDA Toolkit often is not required because the extra compute-related hardware is not present on login nodes.
- The Mellanox OFED installation process is slightly different. Use the `postscripts node` attribute instead of the `postinstall osimage` attribute to start the **`mlnxofed_ib_install`** script. For more information, see the Diskful Installation page.
- The IBM XL C/C++ and Fortran compilers use the login nodes to compile applications. Therefore, the compiler package must be installed. Add the following appropriate xCAT software kits:

  ```
  Object name: xlc.compiler-compute-16.1.0-1-rhels-7-ppc64le
      description=XLC16 for compiler kitcomponent
  Object name: xlf.compiler-compute-16.1.0-1-rhels-7-ppc64le
      description=XLF16 for compiler kitcomponen
  ```

- Advance Toolchain needs at least the `advance-toolchain-at12.0-devel-12.0-0.ppc64le` package for compiler support. As a best practice, install `advance-toolchain-at12.0-perf-12.0-0.ppc64le.rpm` as well.
- The PGI compiler installation is different because the entire PGI compiler stack is needed. Complete the following steps to install it:

  a. Ensure that `pkglist` includes the following packages because they are a requirement for PGI compilers:

     - `gcc`
     - `gcc-c++`

b. Create a `postscript` to perform an unattended installation:

```
$ script=/install/postscripts/install_pgi_login
$ cat /install/postscripts/install_pgi_login
   #!/bin/bash

   INSTALL_TAR="/install/software/compilers/pgi/pgilinux-2016-1610-ppc64le.t
   ar.gz"
   INSTALL_PATH="/opt/pgi"
   PGI_PATH="/opt/pgi/linuxpower/16.10"
   PROFILE_FILENAME="/etc/profile.d/xcat-pgi.sh"

   # environment variables for unattended install
   export PGI_SILENT="true"
   export PGI_ACCEPT_EULA="accept"
   export PGI_INSTALL_DIR="$INSTALL_PATH"
   export PGI_INSTALL_TYPE="single"

   # get tar and extract it
   dir=/tmp/pgi_install
   mkdir $dir
   cd $dir
   wget -l inf -N --waitretry=10 --random-wait --retry-connrefused -t 10 -T
   60 -nH --no-parent "http://$MASTER/$INSTALL_TAR" 2> wget.log
   tar -xf "${INSTALL_TAR##*/}"

   # do installation
   ./install

   # set the paths required for pgi
   echo "export PGI=$INSTALL_PATH" > "${PROFILE_FILENAME}"
   echo "export PATH=$PGI_PATH/bin:\$PATH" >> "${PROFILE_FILENAME}"

   cd ..
   # comment for debugging
   rm -rf $dir

$ chmod +x $script
$ chdef -t <login_node> --plus postscripts=install_pgi_login
```

- The kits for PPT, IBM ESSL, and IBM Parallel ESSL provide kit components that are specifically for login nodes, for example:

```
Object name: ppt.login-2.4.0-2-rhels-7.4-ppc64le
    description=IBM Parallel Performance Toolkit for login nodes
Object name: essl-loginnode-6.1.0-1-rhels-7.5-ppc64le
    description=essl for login nodes
Object name: essl-loginnode-nocuda-6.1.0-1-rhels-7.5-ppc64le
    description=essl for login nodes

Object name: pessl-loginnode-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for login nodes
Object name: pessl-loginnode-nocuda-5.4.0-0-rhels-7.5-ppc64le
    description=pessl for login nodes
```

- With the parallel file system (IBM Spectrum Scale), the login node can access the data that is provided to or produced by the applications that are running in the compute nodes.
- The job scheduler (IBM Spectrum LSF) is typically required to submit jobs from the login nodes to the compute nodes.

6

Cluster Administration and Storage Tools

This chapter introduces Cluster Administration and Storage Tools (CAST), which is composed of two main components: IBM Cluster Systems Management (CSM) and IBM Burst Buffer (BB). The CSM and BB tools are developed for use on POWER processor-based systems only.

This chapter describes the installation and configuration of CSM and BB.

The following topics are described in this chapter:

- Cluster Systems Management
- Preparing CSM
- Burst Buffer

6.1 Cluster Systems Management

CSM is a C application programming interface (API) for managing a large cluster. CSM is developed for use with POWER processor-based systems only. This chapter is based on Version 1.3.0, which is available as of October 2018.

CSM offers a suite of tools for maintaining the cluster:

- Discovery and management of system resources
- Database integration (PostgreSQL)
- Job launch support (workload management APIs)
- Node diagnostics (diag APIs and scripts)
- Reliability, availability, and serviceability (RAS) events and actions
- Infrastructure health checks
- Python bindings for C APIs

6.2 Preparing CSM

All nodes participating in CSM must be running Red Hat Enterprise Linux for PPC64LE.

6.2.1 Software dependencies

CSM has software dependencies. Verify that the following packages are installed.

Hard dependencies

Table 6-1 lists the mandatory dependencies for CSM to configure.

Table 6-1 Mandatory software dependencies for CSM

Software	Version	Comments
Extreme Cluster and Cloud Administration Toolkit (xCAT)	2.13.3 or later	None.
PostgreSQL	9.2.18 or later	xCAT documentation for migration.
`openssl-libs`	1.0.1e-60 or later	None.
perl-YAML	0.84-5 or later	Required by the diagnostic tests.
perl-JSON	2.59 or later	Required by the diagnostic tests that obtain information from the Unified Fabric Manager (UFM).

Software	Version	Comments
`cast-boost`	1.60.0-4	None.
POWER9 firmware level	**Baseboard management controller (BMC):** ibm-v2.0-0-r46-0-gbed584c or later. **Host:** IBM-witherspoon-ibm-OP9_v1.19_1.185 or later	None.

Soft dependencies

Table 6-2 lists more dependencies that provide extra functions.

Table 6-2 Soft software dependencies

Software	Version	Comments
NVIDIA Data Center GPU Manager (DCGM)	datacenter-gpu-manager-1.4.2-1.ppc64le	► Needed by diagnostics, health check, and CSM graphics processing unit (GPU) inventory ► All nodes with GPUs
NVIDIA Compute Unified Device Architecture (CUDA) Toolkit	cuda-9.2.148-1.ppc64le.	All nodes with GPUs
NVIDIA Driver	cuda-drivers-396.47-1.ppc64le	► Needed by NVIDIA DCGM ► All nodes with GPUs
IBM Hardware Test Executive (HTX)	htxrhel72le-491-LE.ppc64le	► Needed by the diagnostics and health check ► All nodes ► HTX requires: – net-tools package (**`ifconfig`** command) – `mesa-libGLU-devel` and `mesa-libGLU` packages
IBM Spectrum Message Passing Interface (IBM Spectrum MPI)	10.02.00.05 or later	► Needed by the diagnostic tests dgemm, dgemm-gpu, jlink, and daxpy ► IBM Spectrum MPI requires – IBM Engineering and Scientific Subroutine Library (IBM ESSL) – IBM Xpertise Library (XL) Fortran
sudo	sudo-1.8.19p2-13.el7.ppc64le	Required by the diagnostic tests that must run as root
lm-sensors	3.4.0	Required by the diagnostic temperature tests

Setting up CSM is essential and must be done at the early stages of the HPC solution setup.

Table 6-3 lists other software dependencies.

Table 6-3 Other software dependencies

Software	Version	Comments
IBM Spectrum Load Sharing Facility (IBM Spectrum LSF)	10.1	None.
IBM Burst Buffer	1.3.0	None.
IBM POWER Hardware Monitor	ibm-crassd-0.8-15 or later	If installed on the service nodes, you can configure `ibmpowerhwmon` to create CSM RAS events from BMC sources by using **`csmrestd`**.

6.2.2 Installation

A set of RPMs is required to install CSM. To complete a clean CSM installation, follow the steps in this section.

6.2.3 CSM RPMs overview

The following RPMs are all of the required RPMs for CSM. These RPMs can be found on Box. Verify that you have all the required RPMs.

For CSM, all CSM RPMs have a version of `ibm-csm-component-1.3.0-<commit_sequence>.ppc64le.rpm`, for example, `ibm-csm-core-1.3.0-1401.ppc64le.rpm`. The `commit_sequence` portion is an increasing value that changes every time that a new commit is added to the repository. Because the `commit_sequence` changes so frequently, this document uses a wildcard for this portion of the RPM name.

Here are the CSM RPMs and their descriptions:

- `ibm-csm-core` holds the CSM infrastructure daemon and configuration file examples.
- `ibm-csm-api` holds all CSM APIs and the corresponding command-line interfaces (CLIs).
- `ibm-csm-hcdiag` holds the diagnostics and health check framework with all the available tests interfaces.
- `ibm-csm-db` holds CSM DB configuration files and scripts.
- `ibm-csm-bds` holds big data storage (BDS) configuration files, scripts, and documentation.
- `ibm-csm-bds-logstash` holds Logstash plug-ins, configuration files, and scripts.
- `ibm-csm-bds-kibana` holds Kibana plug-ins, configuration files, and scripts.
- `ibm-csm-restd` holds the CSM REST daemon, configuration file example, and a sample script.

Table 6-4 shows the installation for CSM RPMs to servers (nodes) based on their function.

Table 6-4 RPMs by node type

CSM RPM	Management node	Service node	Login, launch, or compute node	BDS node
`ibm-csm-core`	X	X	X	
`ibm-csm-api`	X	X	X	
`ibm-csm-hcdiag`	X	X	X	
`ibm-csm-db`	X			
`ibm-csm-bds`				X
`ibm-csm-bds-logstash`		X		X
`ibm-csm-bds-kibana`				X
`ibm-csm-restd`	X	X		

6.2.4 Installing CSM on to the management node

You start by installing `cast-boost` and then `flightlog` and CSM RPMs by running the **`rpm`** or **`yum`** command as follows:

```
$ rpm -ivh cast-boost-*.rpm
$ rpm -ivh ibm-flightlog-1.3.0-*.ppc64le.rpm \
ibm-csm-core-1.3.0-*.ppc64le.rpm \
ibm-csm-api-1.3.0-*.ppc64le.rpm \
ibm-csm-db-1.3.0-*.noarch.rpm \
ibm-csm-bds-1.3.0-*.noarch.rpm \
ibm-csm-hcdiag-1.3.0-*.noarch.rpm \
ibm-csm-restd-1.3.0-*.ppc64le.rpm
```

6.2.5 Installing CSM on to the service node

For service nodes, CSM supports diskless and diskful installation.

Diskless

To do a diskless installation, clone the existing images, add the following packages to the `otherpkgs` directory and list, and then run the **`genimage`** command. For more information, see 6.2.21, “Diskless images” on page 150.

```
cast-boost-*
ibm-flightlog-1.3.0-*.ppc64le
ibm-csm-core-1.3.0-*.ppc64le
ibm-csm-api-1.3.0-*.ppc64le
ibm-csm-bds-1.3.0-*.noarch
ibm-csm-bds-logstash-1.3.0-*.noarch
ibm-csm-hcdiag-1.3.0-*.noarch
ibm-csm-restd-1.3.0-*.ppc64le
```

Diskful

To do a diskful installation, first install the `cast-boot` RPMs by running the following command:

```
$ rpm -ivh cast-boost-*.rpm
```

Then, install the `flightlog` RPM and the following CSM RPMs by running the **`rpm`** command:

- `ibm-flightlog`
- `ibm-csm-core`
- `ibm-csm-api`
- `ibm-csm-bds`
- `ibm-csm-bds-logstash`
- `ibm-csm-hcdiag`
- `ibm-csm-restd`

```
$ rpm -ivh ibm-flightlog-1.3.0-*.ppc64le.rpm \
ibm-csm-core-1.3.0-*.ppc64le.rpm \
ibm-csm-api-1.3.0-*.ppc64le.rpm \
ibm-csm-bds-1.3.0-*.noarch.rpm \
ibm-csm-bds-logstash-1.3.0-*.noarch.rpm \
ibm-csm-hcdiag-1.3.0-*.noarch.rpm \
ibm-csm-restd-1.3.0-*.ppc64le.rpm
```

6.2.6 Installing CSM in the login, launch, and workload manager nodes

For the login, launch and workload manager (WLM) nodes, CSM supports diskless and diskful installation.

Diskless

To do a diskless installation, clone the existing images, add the following packages to the `otherpkgs` directory and list, and then run the **`rungenimage`** command. For more information, see 6.2.21, “Diskless images” on page 150.

```
cast-boost-*
ibm-flightlog-1.3.0-*.ppc64le
ibm-csm-core-1.3.0-*.ppc64le
ibm-csm-api-1.3.0-*.ppc64le
ibm-csm-hcdiag-1.3.0-*.noarch
```

Diskful

To do a diskful installation, install the `cast-boost` RPMs by running the following command:

```
$ rpm -ivh cast-boost-*.rpm
```

Install the `flightlog` RPM and the following CSM RPMs by running the **`rpm`** command:

- `ibm-flightlog`
- `ibm-csm-core`
- `ibm-csm-api`
- `ibm-csm-hcdiag`

```
$ rpm -ivh ibm-flightlog-1.3.0-*.ppc64le.rpm \
ibm-csm-core-1.3.0-*.ppc64le.rpm \
ibm-csm-api-1.3.0-*.ppc64le.rpm \
ibm-csm-hcdiag-1.3.0-*.noarch.rpm
```

6.2.7 Installing CSM in the compute nodes

For compute nodes, CSM supports diskless and diskful installation.

Diskless

To do a diskless installation, clone the existing images, add the following packages to the `otherpkgs` directory and list, and then run the **`genimage`** command. For more information, see 6.2.21, "Diskless images" on page 150.

```
cast-boost-*ibm-flightlog-1.3.0-*.ppc64le
ibm-csm-core-1.3.0-*.ppc64le
ibm-csm-api-1.3.0-*.ppc64le
ibm-csm-hcdiag-1.3.0-*.noarch
```

Diskful

To do a diskful installation, install the `cast-boost` RPMs by running the following command. Replace "`/path/to/rpms`" with the appropriate location for your system.

```
$ xdsh compute"cd /path/to/rpms; rpm -ivh cast-boost-*.rpm"
```

Install the `flightlog` RPM and the following CSM RPMs.

Note: Replace "`/path/to/rpms`" with the appropriate location for your system.

- `ibm-flightlog`
- `ibm-csm-core`
- `ibm-csm-api`
- `ibm-csm-hcdiag`

```
$ xdsh compute "cd /path/to/rpms; \
rpm -ivh ibm-flightlog-1.3.0-*.ppc64le.rpm \
ibm-csm-core-1.3.0-*.ppc64le.rpm \
ibm-csm-api-1.3.0-*.ppc64le.rpm \
ibm-csm-hcdiag-1.3.0-*.noarch"
```

6.2.8 Configuration

After the CSM components are installed, start the configuration by validating the open files limit by running the following command:

```
$ ulimit -n
50000
```

6.2.9 Configuring the CSM database

The CSM database (CSM DB) holds information about systems hardware configuration, hardware inventory, RAS, diagnostics, job steps, job allocations, and CSM configuration. This information is essential for the HPC system to run properly and for resources accounting.

On the management node, create the `csmdb` schema by running **`csm_db_script.sh`**, which is shown in Example 6-1. This script assumes that xCAT migrated to postgreSQL. For more information about this migration process, see Configure PostgreSQL - xCAT 2.14.5 documentation.

Example 6-1 The csm_db_script.sh script

```
$ /opt/ibm/csm/db/csm_db_script.sh
--------------------------------------------------------------------------------
[Start] Welcome to CSM database automation script
[Info] PostgreSQL is installed
[Info] csmdb database user: csmdb already exists
[Complete] csmdb database created
[Complete] csmdb database tables created
[Complete] csmdb database functions and triggers created
[Complete] csmdb table data loaded successfully into csm_db_schema_version
[Complete] csmdb table data loaded successfully into csm_ras_type
[Info ] csmdb DB schema version (16.1)
--------------------------------------------------------------------------------
```

This script creates the `csmdb` database and configures it with the default settings.

6.2.10 Default configuration files

To change the default configuration, complete the following steps:

1. On the management node, copy the default configuration and access control list (ACL) files from `/opt/ibm/csm/share/etc to /etc/ibm/csm` by running the following commands:

   ```
   $ cp /opt/ibm/csm/share/etc/*.cfg /etc/ibm/csm/
   $ cp /opt/ibm/csm/share/etc/csm_api.acl /etc/ibm/csm/
   ```

2. Review the configuration and ACL files. Make the following suggested updates (the host names can also be IP addresses especially if a particular network interface is wanted for CSM communication):

   – Substitute all **“**`__MASTER__`**”** occurrences in the configuration files with the management node host name.

   – On aggregator configurations, substitute **“**`__AGGREGATOR__`**”** with the corresponding service node host name.

   – On compute configurations, substitute **“**`__AGGREGATOR_A__`**”** with the assigned primary aggregator.

   – On compute configurations, substitute **“**`__AGGREGATOR_B__`**”** with the secondary aggregator or leave it untouched if you set up a system without failover.

   – If an aggregator is run on the management node too, make sure to provide a unique entry for `csm.net.local_client_listen.socket` to avoid name collision and strange behavior.

   – Create a Linux group for privileged access:

     - Add users to this group.
     - Make this group privileged in the ACL file.

> **Note:** For more information, see 6.3.1, “Configuring user control, security, and access level”, in the *CAST/CSM 1.0.0 Installation and Configuration Guide*.

– From the management node, copy the configuration files to the proper nodes by running the following commands:

```
$ xdcp compute /etc/ibm/csm/csm_compute.cfg /etc/ibm/csm/csm_compute.cfg
$ xdcp login,launch /etc/ibm/csm/csm_utility.cfg
/etc/ibm/csm/csm_utility.cfg
$ xdcp compute,login,launch /etc/ibm/csm/csm_api.acl
/etc/ibm/csm/csm_api.acl
$ xdcp compute,login,launch /etc/ibm/csm/csm_api.cfg
/etc/ibm/csm/csm_api.cfg
```

6.2.11 Configuring SSL

If you want an SSL setup, configure the `csm.net.ssl` section of the `config` file by editing the file as follows:

```
“ssl”: {
    “ca_file” : “<full path to CA file>”,
“cred_pem”: “<full path to credentials in pem format>”}
```

If the strings are non-empty, the daemon assumes that SSL is requested. If the SSL setup fails, it does not fall back to non-SSL communication.

6.2.12 Heartbeat interval

You can use CSM to tune the CSM daemons heartbeat interval by configuring the `csm.net` section of the `config` files as follows:

```
"net" :
{
    "heartbeat_interval" : 15,
    "local_client_listen" :
    {
        "socket"      : "/run/csmd.sock",
        "permissions" : 777,
        "group"       : ""
},
```

The heartbeat interval setting defines the time between two subsequent unidirectional heartbeat messages in case there is no other network traffic on a connection. The default is 15 seconds. If two daemons are configured with a different interval, they use the minimum of the two settings for the heartbeat on the connection, which means that you can configure a short interval between aggregator and master daemons and a longer interval between aggregator and compute daemons to reduce network interference on compute nodes.

It might take up to three intervals to detect a dead connection because of the following heartbeat process: After receiving a message, the daemon waits one interval to send its heartbeat one way. If it does not get a heartbeat after one more interval, it retries and waits for another interval before declaring the connection broken.

This setting must balance the requirements between fast detection of dead connections and network traffic impact. If a daemon fails or is shut down, the closed socket is detected immediately in many cases.

The heartbeat-based detection is mostly needed only for errors in the network hardware itself (for example, a broken or disconnected cable, switch, or port).

6.2.13 Environmental buckets

The daemons are enabled to run predefined environmental data collection. The execution is controlled by the `csm.data_collection` section in the configuration files. This function can list a number of buckets, and each one has a list of the predefined items. Currently, CSM supports two types of data collection items: *gpu* and *environmental*. The recommended configuration interval for these items is every 10 minutes. Data that is collected by using this function is sent from CSM compute daemons to CSM aggregator daemons to a Logstash instance for ingestion into the BDS.

6.2.14 Prolog and epilog scripts

To create allocations, both `privileged_prolog` and `privileged_epilog` scripts must be present in `/opt/ibm/csm/prologs/` on the compute nodes. Review the sample scripts and make any customizations that are required.

To use the packaged sample scripts (`/opt/ibm/csm/share/prologs/`), run the following command on the management node:

```
$ xdcp compute /opt/ibm/csm/share/prologs/* /opt/ibm/csm/prologs/
```

The command copies the following files to the compute nodes:

- `Privileged_prolog (access: 700)`
- `Privileged_prolog (access: 700)`
- `Privileged.ini (access: 600)`

The `privileged_prolog` and `privileged_epilog` files are Python scripts that have command-line arguments for type, user flags, and system flags. The type is either `allocation` or `step`, and the flags are space-delimited alphanumeric strings of flags. If a new version of one of these scripts is written, it must implement the following options:

- `--type [allocation|step]`
- `--user_flags "[string of flags, spaces allowed]"`
- `--sys_flags "[string of flags, spaces allowed]"`

The `privileged.ini` file configures the logging levels for the script. It is needed only if you are extending or using the packaged scripts. For more information, see the comments in the bundled scripts, the packaged `POST_README.md` file, or the configuring `allocation prolog` and `epilog` scripts in section 5.3.4, "Prolog/Epilog Scrips Compute", in the *CAST/CSM 1.0.0 Installation and Configuration Guide*.

6.2.15 CSM Pluggable Authentication Module

The `ibm-csm-core` RPM installs a Pluggable Authentication Module (PAM) module that can be enabled by a system administrator.

This module performs two operations:

- Prevents unauthorized users from obtaining access to a compute node.
- Places users who have active allocations in to the correct cgroup.

To enable the CSM PAM module for SSH sessions, complete the following steps:

1. Add always authorized users to `/etc/pam.d/csm/whitelist` (newline-delimited).

2. Uncomment the following lines from `/etc/pam.d/sshd`:

```
account    required     libcsmpam.so
session    required     libcsmpam.so
```

3. Restart the ssh daemon by running the following command:

```
$ systemctl restart  sshd.service
```

Non-root users who do not have an active allocation on the node and not whitelisted are now prevented from logging in to the node by using SSH. Users who have an active allocation are placed into the appropriate cgroup when logging in by using SSH.

For more information about the behavior and configuration of PAM, see `/etc/pam.d/csm/README.md` or the "Configuring the CSM PAM module" section in the *CAST/CSM 1.0.0 Installation and Configuration Guide*.

Note: The CSM PAM module must be enabled only on nodes that run the CSM compute daemon because SSH logins are restricted to root and users that are specified in the whitelist.

6.2.16 Starting the CSM daemons

First, start the CSM daemons dependency, and then start the CSM daemons.

Login, launch, and compute nodes

Start NVIDIA persistence and DCGM by running the following commands:

```
$ xdsh compute,service,utility "systemctl start nvidia-persistenced"
$ xdsh compute,service,utility "systemctl start dcgm"
```

Management node

Start the master daemon by running the following command:

```
$ systemctl start csmd-master
```

Start the aggregator daemon by running the following command:

```
$ systemctl start csmd-aggregator
```

Login and launch nodes

Start the utility daemon by running the following command:

```
$ xdsh login,launch "systemctl start csmd-utility"
```

Compute node

Start the compute daemon by running the following command:

```
$ xdsh compute "systemctl start csmd-compute"
```

6.2.17 Running the infrastructure health check

Run the CSM infrastructure health check on the login or launch node (LN) to verify the infrastructure status by running the following command:

```
# /opt/ibm/csm/bin/csm_infrastructure_health_check -v
Starting. Contacting local daemon...
```

```
Connected. Checking infrastructure... (this may take a moment. Please be
patient...)

###### RESPONSE FROM THE LOCAL DAEMON #######
MASTER: c650mnp06 (bounced=0; version=1.3.0)
        DB_free_conn_size: 10
        DB_locked_conn_pool_size: 0
        Timer_test: success
        DB_sql_query_test: success
        Multicast_test: success
        Network_vchannel_test: success
        User_permission_test: success
        UniqueID_test: success
Aggregators:1
      AGGREGATOR: c650f02p05 (bounced=0; version=1.3.0)
        Active_primary: 4
        Unresponsive_primary: 0
        Active_secondary: 0
        Unresponsive_secondary: 0

       Primary Nodes:
        Active: 4
           COMPUTE: c650f02p17 (bounced=5; version=1.3.0; link=PRIMARY)
           COMPUTE: c650f02p13 (bounced=5; version=1.3.0; link=PRIMARY)
           COMPUTE: c650f02p11 (bounced=5; version=1.3.0; link=PRIMARY)
           COMPUTE: c650f02p09 (bounced=5; version=1.3.0;link=PRIMARY)
        Unresponsive: 0

       Secondary Nodes:
        Active: 0
        Unresponsive: 0

Unresponsive Aggregators: 0
Utility Nodes:1
      UTILITY: c650f02p07 (bounced=0; version=1.3.0)

   Unresponsive Utility Nodes: 0

   Local_daemon: MASTER: c650mnp06 (bounced=0; version=1.3.0)
        Status:
   ############################################

Finished. Cleaning up...
Test complete: rc=0
#
```

Note: In some cases, the list and status of nodes might not be 100% accurate if there have been infrastructure changes immediately before or during the test. These changes usually result in timeout warnings and a rerun of the test returns an updated status.

If you have any unresponsive compute nodes, consider the following items:

- Unresponsive nodes do not show a daemon build version and do not list the connection type as primary or secondary.
- The unresponsive nodes cannot provide information about their configured primary or secondary aggregator. Instead, the aggregators report the last known connection status of those compute nodes. For example, if the compute node uses a connection as the primary link even if the compute configuration defines the connection as secondary, the aggregator shows this compute as an unresponsive primary node.

6.2.18 Setting up the environment for job launch

To update the compute nodes state to `IN_SERVICE`, run the CSM API **`csm_node_attributes_update`** command:

```
$ /opt/ibm/csm/bin/csm_node_attributes_update –s IN_SERVICE -n c650f02p09
```

6.2.19 Installing the configuring the CSM REST daemon

The CSM REST daemon must be installed and configured on the management node and service nodes. By using the CSM REST daemon, you can use `ibm-crassd` to create CSM RAS events for events that are detected from compute node BMCs. You also can use CSM Event Correlator to create CSM RAS events from console logs. For example, GPU XID errors are monitored by the CSM Event Correlator mechanism.

On the service nodes

Complete the following steps:

1. Install the `ibm-csm-restd` RPM if it is not already installed by running the following command:

   ```
   $ rpm -ivh ibm-csm-restd-1.3.0-*.ppc64le.rpm
   ```

2. Copy the default configuration file from `/opt/ibm/csm/share/etc` to `/etc/ibm/csm`:

   ```
   $ cp /opt/ibm/csm/share/etc/csmrestd.cfg /etc/ibm/csm/
   ```

3. Edit `/etc/ibm/csm/csmrestd.cfg` and replace "`__CSMRESTD_IP__`" with "`127.0.0.1`".

The CSM REST daemon requires that the local CSM daemon is running before it is started. In addition, `csmrestd` must be restarted whenever the local CSM daemon is restarted.

Start `csmrestd` by running the **`systemctl`** command:

```
$ systemctl start csmrestd
```

On the management node (optional)

If the CSM DB on your management node is not re-created as part of the CSM installation and you intend to enable IBM POWER HW Monitor collection of BMC RAS events from your service nodes, manually update the CSM RAS types in the CSM DB.

With all daemons stopped, run the following command:

```
$ /opt/ibm/csm/db/csm_db_ras_type_script.sh -l csmdb csm_ras_type_data.csv
```

This command imports any CSM RAS types in to the CSM DB that were added in later releases, and it is a no-op for any events that already exist in the CSM DB.

6.2.20 Uninstalling the CSM daemons

To uninstall the CSM daemons, complete the following steps:

1. Stop all CSM daemons by running the following commands:

```
$ xdsh compute "systemctl stop csmd-compute"
$ xdsh utility "systemctl stop csmd-utility"
$ systemctl stop csmd-aggregator
$ systemctl stop csmd-master
```

2. Delete the CSM DB by running the following command:

```
$ /opt/ibm/csm/db/csm_db_script.sh -d csmdb
```

3. Remove the RPMs by running the following commands:

```
$ xdsh compute,utility"rpm -e ibm-csm-core-1.3.0-*.ppc64le
ibm-csm-api-1.3.0-*.ppc64le ibm-flightlog-1.3.0-*.ppc64le
ibm-csm-hcdiag-1.3.0-*.noarch"

$ rpm -e ibm-csm-core-1.3.0-*.ppc64le ibm-csm-hcdiag-1.3.0-*.noarch
ibm-csm-db-1.3.0-*.noarch ibm-csm-api-1.3.0-*.ppc64le
ibm-csm-restd-1.3.0-*.ppc64le ibm-flightlog-1.3.0-*.ppc64le
```

4. Clean up the log and configuration logs by running the following commands:

```
$ xdsh compute,utility"rm -rf /etc/ibm /var/log/ibm"
$ rm -rf /etc/ibm/csm /var/log/ibm/csm
```

5. Stop the NVIDIA host engine (DCGM) by running the following commands:

```
$ xdsh compute,service,utility "systemctl start nvidia-persistenced"
$ xdsh compute,utility /usr/bin/nv-hostengine –t
```

6.2.21 Diskless images

Ensure that the following steps are completed on the xCAT management node:

1. xCAT is installed and the basic configuration is complete.
2. The steps in 6.2.4, “Installing CSM on to the management node” on page 141 are complete and the `cast-boost` RPMs are accessible in the `${HOME}/rpmbuild/RPMS/ppc64le` file.
3. The `ibm-flightlog-1.3.0-*.ppc64le` file is present on the xCAT management node.
4. You installed `createrepo` to build the other packages directory.

After verifying that these steps are complete, perform the following steps:

1. Generate the operating system images (`osimages`) for your version of Red Hat Enterprise Linux by running the following commands:

```
$ copycds RHEL-7.5-Server-ppc64le-dvd1.iso
$ lsdef –t osimage2.
```

2. Copy the `netboot` image of `osimages` and rename it by running the following commands:

```
$ image_input="rhels7.5-ppc64le-netboot-compute"
$ image_output="rhels7.5-ppc64le-diskless-compute"
$ lsdef -t osimage -z $image_input | sed -e "s/$image_input/$image_output/g" |
mkdef -z
```

3. Move the `cast-boost` RPMs to the `otherpkgdir` directory for the generation of the diskless image by running the following commands:

```
$ lsdef -t osimage rhels7.5-ppc64le-diskless-compute
$ cp cast-boost* /install/post/otherpkgs/rhels7.5/ppc64le/cast-boost
$ createrepo /install/post/otherpkgs/rhels7.5/ppc64le/cast-boost
```

4. Move the CSM RPMs to the `otherpkgdir` directory by running the following commands:

```
$ cp csm_rpms/* /install/post/otherpkgs/rhels7.5/ppc64le/csm
$ createrepo /install/post/otherpkgs/rhels7.5/ppc64le/csm
```

5. Run the **`creatrepo`** command:

```
$ createrepo /install/post/otherpkgs/rhels7.5/ppc64le
```

6. Add the following packages to a package list in `otherpkgdir` and then add the package list to the `osimage` by running the following command:

```
$ vi /install/post/otherpkgs/rhels7.5/ppc64le/csm.pkglist
cast-boost/cast-boost-*
csm/ibm-flightlog-1.3.0-*.ppc64le
csm/ibm-csm-core-1.3.0-*.ppc64le
csm/ibm-csm-api-1.3.0-*.ppc64le

$ chdef -t osimage rhels7.5-ppc64le-diskless-compute
otherpkglist=/install/post/otherpkgs/rhels7.5/ppc64le/csm.pkglist7.
```

7. Generate and package the diskless image by running the following commands:

```
$ genimage rhels7.5-ppc64le-diskless-compute
$ packimage rhels7.5-ppc64le-diskless-compute
```

6.3 Burst Buffer

BB is a cost-effective mechanism that can improve I/O performance for a large class of high-performance computing (HPC) applications without requiring intermediary hardware. BB provides the following functions:

- A fast storage tier between compute nodes and the traditional parallel file system
- Overlapping job stage-in and stage-out of data for checkpoint and restart
- Scratch volumes
- Extended memory I/O workloads

6.3.1 Installing Burst Buffer

This section describes the installation of BB.

Prerequisites

Red Hat Enterprise Linux 7.5 or later for POWER9 processor-based servers must be installed on the nodes. Use the following RPMs:

- `ibm-burstbuffer-1.3.0-x.ppc64le.rpm`
- `ibm-burstbuffer-lsf-1.3.0-x.ppc64le.rpm`
- `ibm-burstbuffer-mn-1.3.0-x.ppc64le.rpm`
- `ibm-csm-core-1.3.0-1347.ppc64le.rpm`

- `ibm-export_layout-1.3.0-x.ppc64le.rpm`
- `ibm-flightlog-1.3.0-x.ppc64le.rpm`

Security certificates

The connection between bbProxy and bbServer is secured by an X.509 certificate. To create a certificate, use the OpenSSL tool. For convenience, you can use a provided bash script:

```
$ /opt/ibm/bb/scripts/mkcertificate.sh
```

This command generates two files and only needs to be run on a single node:

```
-rw-r--r--1 root root  cert.pem
-rw-------1 root root key.pem
```

The `key.pem` file is the private key and must be kept secret. As a best practice, copy this file to `/etc/ibm/key.pem` on each of the bbServer nodes. The `cert.pem` file must be copied to all bbServer and compute nodes. The `cert.pem` can be deployed to the compute nodes in various ways:

- `cert.pem` can be placed in the shared IBM Spectrum Scale storage.
- `cert.pem` can be placed in the xCAT compute image.
- `cert.pem` can have **`rsync`** or **`scp`** run on it to send it to each compute node after boot.

IBM Elastic Storage Server I/O node virtual machine setup

On each IBM Elastic Storage Server I/O node virtual machine (VM) (or equivalent), install these RPMs:

- `ibm-burstbuffer-1.3.0-*.ppc64le.rpm`
- `ibm-flightlog-1.3.0-*.ppc64le.rpm`
- `ibm-csm-core-1.3.0-*.ppc64le.rpm`

The NVMe driver must be built with Remote Direct Memory Access (RDMA) enabled. The Mellanox OpenFabrics Enterprise Distribution (OFED) driver can this task if you run the following command:

```
% mlnxofedinstall --with-nvmf
```

Security files

The `cert.pem` and `key.pem` files must be placed in the `/etc/ibm` directory on each of the bbServer nodes. This task can be done after the VM is started or during image creation.

Starting BB services

The BB server can be started by running the following command on each IBM Elastic Storage Server I/O node VM (or equivalent):

```
$ /opt/ibm/bb/scripts/bbactivate --server
```

This command uses the default BB configuration file in the RPM (unless it is overridden by the **`--config`** flag) and starts the BB server. It also adds the NVMe over Fabrics block device pattern to `global_filter` in `/etc/lvm/lvm.conf` (unless the global filter line has been modified).

Setting up the compute node

This section describes the compute node setup.

Installing the RPMs

On each compute node, install these RPMs:

- `ibm-burstbuffer-1.3.0-*.ppc64le.rpm`
- `ibm-flightlog-1.3.0-*.ppc64le.rpm`
- `ibm-export_layout-1.3.0-*.ppc64le.rpm`
- `ibm-csm-core-1.3.0-*.ppc64le.rpm`

The NVMe driver must be built with RDMA enabled. The Mellanox OFED driver can accomplish this task by running the following command:

```
% mlnxofedinstall --with-nvmf
```

Security files

The `cert.pem` and `key.pem` files must be placed in the `/etc/ibm` directory on each of the compute nodes. This task can be done after the node is started or during image creation. The private key (`key.pem`) must not be placed on the compute node.

Generating the compute node and IBM Elastic Storage Server list

The BB has a static assignment of compute nodes to bbServers. This relationship is defined by two files that are specified by the **`bbactivate`** tool.

The first file (`nodelist`) is a list of the xCAT names for all compute nodes, with one compute node per line:

```
c650f07p23
c650f07p25
c650f07p27
```

This `nodelist` can be generated by running xCAT commands:

```
lsdef all | grep "Object name:" | cut -f 3 -d ' '
```

The second file (`esslist`) contains a list of IP addresses and ports for each bbServer. In the planned configuration, this is the IBM Elastic Storage Server I/O node VM IP address plus a well-known port (for example, 9001). For IBM Elastic Storage Server redundancy, place the two IBM Elastic Storage Server I/O nodes within the same IBM Elastic Storage Server.

```
20.7.5.1:9001 20.7.5.2:9001
20.7.5.3:9001 20.7.5.4:9001
```

Starting the Burst Buffer services

On each compute node, run the **`bbactivate`** script:

```
$ /opt/ibm/bb/scripts/bbactivate
```

Note: The **`bbactivate`** script attempts to assign the same number of compute nodes per IBM Elastic Storage Server I/O node. For *n* computes and *m* IBM Elastic Storage Server I/O nodes, the first *n* divided by *m* compute nodes are placed on the first IBM Elastic Storage Server I/O node in the list. The second group gets the second IBM Elastic Storage Server I/O node that is listed.

Running the bbServer on a different node than bbProxy requires a configured networked block device. If no block device is configured, the **`bbactivate`** script attempts to establish an NVMe over Fabrics connection between the two nodes when bbProxy is started.

Whenever a compute node is restarted or a solid-state drive (SSD) is replaced, rerun the **`bbactivate`** script.

Note: When bbServer establishes the NVMe over Fabrics connection, it uses the **`/opt/ibm/bb/scripts/nvmfConnect.sh`** script.

Setting up the launch and login node

This section describes how to set up the launch and login node.

Installing the RPMs

On each launch and login node, install these RPMs:

- `ibm-burstbuffer-1.3.0-*.ppc64le.rpm`
- `ibm-flightlog-1.3.0-*.ppc64le.rpm`
- `ibm-csm-core-1.3.0-*.ppc64le.rpm`
- `ibm-burstbuffer-lsf-1.3.0-*.ppc64le.rpm`

This **`burstbuffer-lsf`** RPM may be relocated by running the following command:

```
$ rpm--relocate /opt/ibm/bb/scripts=$LSF_SERVERDIR
```

Setting up IBM Spectrum LSF

You must do further IBM Spectrum LSF configuration to set up the data transfer queues. For more information, see IBM Knowledge Center.

As a best practice, add the following parameter to the `lsf.conf` file so that the BB `esub.bb` and `epsub.bb` scripts run when a job is submitted to set up key environment variables for `$BBPATH` and Burst Buffer Shared Checkpoint File System (BSCFS):

```
LSB_ESUB_METHOD=bb
```

Configuring Burst Buffer

The `ConfigurationA` directory is used to store job-specific BSCFS metadata between job execution and job stageout. Create a path in the parallel file system for BSCFS temporary files. The workpath must be accessible to all users.

A path is also needed to specify temporary storage for job-related metadata between the job submission through job stageout. It must be a location that can be written by the user and read by root, and accessible by nodes that are used for job submission and launch. It does not need to be accessible by the compute nodes. If the user home directories are readable by root, you can use **`--envdir=HOME`**.

For IBM Spectrum LSF configuration, several scripts must be copied into `$LSF_SERVERDIR`. The files that are copied from `/opt/ibm/bb/scripts` are `esub.bb`, `epsub.bb`, `esub.bscfs`, `epsub.bscfs`, **`bb_pre_exec.sh`**, and b**`b_post_exec.sh`**. The **`bbactivate`** script can automatically copy these files. Alternatively, the `burstbuffer-lsf` RPM is relocatable.

```
$/opt/ibm/bb/scripts/bbactivate --ln --bscfswork=$BSCFSWORK--envdir=HOME
--lsfdir=$LSF_SERVERDIR
```

Setting up the management node (optional)

This section describes the management node setup. This section is optional.

Installing the RPMs

On the CSM management node, install the following RPM:

```
ibm-burstbuffer-mn-1.3.0-*.ppc64le.rpm
```

Adding Burst Buffer RAS in to the CSM database

RAS definitions for the BB can be added to CSM postgreSQL tables by running the following script:

```
$ /opt/ibm/csm/db/csm_db_ras_type_script.sh -l csmdb /opt/ibm/bb/scripts/bbras.csv
```

The command must be run on the CSM management node. The `ibm-burstbuffer-mn` RPM must also be installed on the management node.

If the RAS definitions are not added, the bbProxy log shows errors when posting any RAS messages, but these errors are benign.

Stopping the services

To stop the BB processes, run the **`bbactivate`** script:

```
$ /opt/ibm/bb/scripts/bbactivate --shutdown
```

To disconnect all NVMe over Fabrics connections, from each I/O node run the following command:

```
$ nvme disconnect –n burstbuffer
```

Using BB administrator failover

There might be times in which the node that is running bbServer must be taken down for scheduled maintenance. The BB provides a mechanism to migrate transfers to a backup bbServer. The backup bbServer is defined in the configuration file under `backupcfg`.

To switch to the backup server on three compute nodes cn1, cn2, and cn3, run the following command:

```
$ sudo /opt/ibm/bb/scripts/setServer –server=backup–hosts=cn1,cn2,cn3
```

To switch back to the primary server on three compute nodes cn1, cn2, and cn3, run the following command:

```
$ sudo /opt/ibm/bb/scripts/setServer –server=primary–hosts=cn1,cn2,cn3
```

If submitting the switchover by using an IBM Spectrum LSF job that runs as root, the **`–hosts`** parameter may be removed because `setServer` uses the compute nodes that are assigned by IBM Spectrum LSF.

Optional configurations

This section contains optional configurations.

Single node loopback

The bbProxy and bbServer can run on the same node, although this is a development or bringup configuration (for example, a bbAPI-using application development). The IBM Elastic Storage Server I/O node list contains a line specifying a loopback address (`127.0.0.1:9001`) for each compute node. Both lists must have the same number of lines.

Configuring bbProxy without Cluster Systems Manager

The bbProxy can update CSM on the state of the logical volumes and ensure RAS by using CSM interfaces. This task is automatically configured by the **`bbactivate`** script:

```
$ /opt/ibm/bb/scripts/bbactivate --csm
$ /opt/ibm/bb/scripts/bbactivate --nocsm
```

The default is to enable CSM.

Configuring without Health Monitor

The BB has an external process that can monitor the bbProxy to bbServer connection. If the connection goes offline, the health monitor either attempts to reestablish the connection or (if defined) establishes a connection with the backup server.

By default, `bbactivate` starts the BB health monitor. This behavior can be changed by running the **`--nohealth`** command.

6.3.2 Using the Burst Buffer

This section describes how to use the BB.

Using bbcmd

The **`bbcmd`** command runs on the LNs and interacts with the bbProxy on the targeted compute nodes. For more information, run the following command:

```
$ bbcmd --help
```

When **`bbcmd`** is run on an LN, the following argument must be specified:

```
--target=0-3
```

The **`--target`** argument can be loosely interpreted as the indexes to the list of hosts for the job. The default host list can be overridden by using the **`--hostlist`** option. If that option is not specified, **`bbcmd`** generates it based on either the CSM allocation or IBM Spectrum LSF environment variables.

bbCmd documentation

The bbCmd documentation is available in two forms: Man page and help text.

To access the man page, run the following command:

```
$ man bbcmdb
```

To access the help text, run the following command:

```
$ bbcmd --help
```

Using the bbAPI library

If you add the following options to the **`compile`** command, you must include the bbAPI headers and the library, and set the **`rpath`** to point to the bbAPI library:

```
-I/opt/ibm -Wl,--rpath /opt/ibm/bb/lib -L/opt/ibm/bb/lib -lbbAPI
```

An example test case can be built by running the following command:

```
$ mpicc /opt/ibm/bb/tests/test_basic_xfer.c -o test_basic_xfer -I/opt/ibm
-L/opt/ibm/bb/lib -Wl,--rpath=/opt/ibm/bb/lib -lbbAPI
```

Using BSCFS

To set up the BSCFS environment, the stage-in and stage-out scripts need a special indicator from the user to avoid incurring a setup impact on each job.

When the user specifies an IBM Spectrum LSF application (**`-a`**) of BSCFS, the stage-in and stage-out scripts manage the configuration of the BSCFS mount point:

```
$ bsub -q bbq -a bscfs -stage storage=100 jsrun /gpfs/joeuser/myexecutable
```

After the command starts running, the BSCFS FUSE mount point is created.

Environment variables

Table 6-5 shows the BB environment variables.

Table 6-5 BB environment variables

Variable name	Description
BBPATH	Path to the XFS file system root directory. This is set by the IBM Spectrum LSF **esub.bb** script.
LSF_STAGE_JOBID or LSF_JOBID	**LSFjobID.LSF_STAGE_JOBID** refers to the job ID that is submitted by the user during stage-in and stage-out scripts (in this context, **LSB_JOBID** refers to the data transfer job).
PMIX|_NAMESPACE	Job Step ID. This is set by **jsrun**.
BSCFS_MNT_PATH	Mount point of BSCFS (usually `/bscfs`). This is set by the IBM Spectrum LSF **esub.bb** script.
BSCFS_PFS_PATH	This is the path into the parallel file system that BSCFS mirrors. This variable must be set by the user for BCSFS to function.
BCSFS_WORK_DIR	This is the directory that BCSFS uses to place control files. This variable must be set by the user for BSCFS to function.
BSCFS_STGIN_LISTFILE	This is a file that specifies the BSCFS files that the user wants pre-staged. This variable must be set by the user to trigger BSCFS pre-staging.

6.3.3 Troubleshooting

This section provides problem determination information.

Capturing debug information

If there is a failure, as a best practice you should preserve the state of the following files:

- `/var/log/bbserver/*`
- `/var/log/bbproxy/*`
- Any core files

The `/var/log` path contains the proxy and server console logs and flightlogs.

The JSON detailed error status is useful too. This status can either be obtained from the output of **bbcmd** or by calling the **BB_ GetLastErrorDetails()** function.

To inspect the flightlogs, run either of the following commands:

- **`$ decoder /var/log/bbserver/FL*`**
- **`$ decoder /var/log/bbproxy/FL*`**

Recovering from abnormal termination

If a logical volume that was previously allocated by the BB services is not mounted when bbProxy is restarted, bbProxy removes such logical volumes as part of the restart. However, if such logical volumes are still mounted, they are not removed during the restart of bbProxy. If the logical volumes are not removed, it might be necessary to remove them after the services are restarted. Before you remove the logical volume first, it must be unmounted.

If job IDs are reused or recycled and metadata on one or more of the bbServers already has existing information about that job ID, a best practice is to remove that metadata by running the following command:

```
bbcmd removejobinfo –jobid
```

If bbProxy does not start because of an existing UNIX socket file, remove the `bb.unixpath` file from the configuration file. If you are using the provided configuration file or if the `bb.unixpath` key is not found in the configuration file, this location is `/run/bb_socket`.

If bbServer's metadata becomes stale or corrupted, you can recover it by stopping all instances of bbProxy and bbServer and then removing the directory that is provided by the `bb.bbserverMetadataPath` key in the configuration file. All bbServers on the system must use the same copy of the metadata as given by this key in the configuration file. If you use the provided configuration file or if the `bb.bbserverMetadataPath` key is not found in the configuration file, this location is `/gpfs/gpfs0/bbmetadata`.

If bbServer or bbProxy fail when attempting to bind to an address, double-check that a bbServer or bbProxy process is not already running on the system. If no processes are found, it is possible that Linux needs a couple of minutes before reclaiming the port.

Part 3

Application development

This part describes how to compile and run your application. The idea behind the chapters in this part is to show you how to use the clustered high-performance computing (HPC) environment to run your workload and take advantage of the computing power of the solution. This part also includes a chapter describing how to measure and tune the applications.

The following chapters are included in this part:

7

Compilation, execution, and application development

This chapter provides information about software, compilers, and tools that can be used for application development and tuning on the IBM Power System AC922 server. A few development models also are described.

The following topics are described in this chapter:

- Compiler options
- Porting applications to IBM Power Systems servers
- IBM Engineering and Scientific Subroutine Library
- IBM Parallel Engineering and Scientific Subroutine Library
- Using POWER9 vectorization
- Development models

7.1 Compiler options

Compiler options are one of the main tools that are used to debug and optimize the performance of your code during development. Several compilers, including IBM Xpertise Library (XL); GNU Compiler Collection (GCC); The Portland Group, Inc. (PGI); and NVIDIA compilers, support the latest IBM POWER9 processor-based features and enhancements.

7.1.1 IBM XL compiler options

IBM XL C/C++ for Linux V16.1.0 and IBM XL Fortran for Linux V16.1.0 support POWER9 processors with new features and enhancements, such as Little Endian distributions support, compiler optimizations, and built-in functions for POWER9 processors. In addition, many Open Multi-Processing (OpenMP) 4.5 features were introduced[1], such as the constructs that enable the loading and offloading of applications and data to NVIDIA graphics processing unit (GPU) technology.

By default, these compilers generate code that runs on various IBM Power Systems servers. Options can be added to exclude older processors that are not supported by the target application. The following major IBM XL compiler options control this support:

`-mcpu`	Specifies the family of the processor architecture for which the instruction code is generated.
`-mtune`	Indicates the processor 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.
`-qcache`	Defines a specific cache or memory geometry.

The **`-mcpu=pwr9`** suboption produces object code that contains instructions that run on the POWER9 processor-based hardware platforms. With the **`-mtune=pwr9`** suboption, optimizations are tuned for the POWER9 processor-based hardware platforms. This configuration can enable better code generation because the compiler uses 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 IBM 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, use a more advanced compiler optimization. For numerical or compute-intensive codes, the IBM 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 IBM XL C/C++ support for POWER9 processor-based servers, see IBM Knowledge Center.

For more information about IBM XL Fortran, see the IBM Knowledge Center.

[1] OpenMP is an application programming interface (API) that facilitates the development of parallel applications for shared memory systems.

Optimization parameters

The strength of the IBM XL compilers is in their optimization and ability to improve code generation. Optimized code runs with greater speed, uses less machine resources, and increases your productivity.

For IBM XL C/C++ V16.1.0, the available compiler options to maximize application development performance are listed in Table 7-1.

Table 7-1 Optimizations levels and options

Base optimization level	Other options that are implied by level	Other suggested options	Other options with possible benefits
`-O0`	None	`-mcpu`	None

`-O2`	`-qmaxmem=8192`	`-mcpu`
		`-mtune`
			`-qmaxmem=-1`
			`-qhot=level=0`

`-O3`	`-qnostrict`
	`-qmaxmem=-1`
	`-qhot=level=0`
		`-mcpu`
		`-mtune`
			`-qpdf`

`-O4`	`-qnostrict`
	`-qmaxmem=-1`
	`-qhot`
	`-qipa`
	`-qarch=auto`
	`-qtune=auto`
	`-qcache=auto`
		`-mcpu`
		`-mtune`
		`-qcache`
			`-qpdf`
			`-qsmp=auto`

`-O5`	All of -O4
	`-qipa=level=2`
		`-mcpu`
		`-mtune`
		`-qcache`
			`-qpdf`
			`-qsmp=auto`

Several options are used to control the optimization and tuning process so that users can improve the performance of their application at run time.

When you compile programs with any of the following sets of options, the compiler automatically attempts to vectorize calls to system math functions. It does so by calling the equivalent vector functions in the Mathematical Acceleration Subsystem (MASS) libraries, with the exceptions of **vdnint**, **vdint**, **vcosisin**, **vscosisin**, **vqdrt**, **vsqdrt**, **vrqdrt**, **vsrqdrt**, **vpopcnt4**, and **vpopcnt8**:

- **-qhot -qignerrno -qnostrict**
- **-O3 -qhot**
- **-O4**
- **-O5**

If the compiler cannot vectorize, it automatically tries to call the equivalent MASS scalar functions. For automatic vectorization or scaling, the compiler uses versions of the MASS functions that are contained in the system library `libxlopt.a`.

In addition to any of the sets of options, if the compiler cannot vectorize when the **-qipa** option is in effect, it tries to inline the MASS scalar functions before it decides to call them.

Not all options benefit all applications. Tradeoffs sometimes occur between an increase in compile time, a reduction in debugging capability, and the improvements that optimization can provide. For more information about the optimization and tuning process and writing optimization-friendly source code, see the options that are listed in Table 7-2 and *Optimization and Programming Guide - XL C/C++ for Linux, V16.1.0, for Little Endian distributions,* SC27-8046-00.

Table 7-2 Optimization and tuning options

Option name	Description
-qhot	Performs aggressive and high-order loop analysis and transformations during optimization (implies -O2 optimization). Look for the suboptions that are available for **-qhot** to enable even more loop transformation, cache reuse, and loop parallelization when it is used with **-qsmp**.
-qipa	Enables or customizes a class of optimizations that is known as interprocedural analysis (IPA). These optimizations enable a whole-program analysis at one time rather than a file-by-file basis. The IBM XL compiler receives the power to restructure your application and perform optimizations, such as inlining for all functions and disambiguation of pointer references and calls. Look at the **-qipa** suboptions for more features.
-qmaxmem	Limits the amount of memory that the compiler allocates during the time 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, where results from a sample program execution that was compiled with **-qpdf1** are used to improve optimization near conditional branches and in frequently run code sections afterward with **-qpdf2**, which creates a profiled optimized executable file.
-p, -pg, -qprofile	The compiler prepares the object code for profiling by attaching monitoring code to the executable file. This code counts the number of times each routine is called and creates a `gmon.out` file if the executable file can run successfully. Afterward, a tool, such as **gprof**, can be used to generate a runtime profile of the program.
-qinline	Attempts to inline functions instead of generating calls to those functions for improved performance.
-qstrict	Ensures that optimizations that are performed by default at the **-O3** and higher optimization levels, and, optionally at **-O2**, and do not alter the semantics of a program.

Option name	Description
`-qsimd`	Controls whether the compiler can automatically use vector instructions for processors that support them.
`-qsmp`	Enables parallelization of program code automatically by the compiler. Options such as schedulers and chunk sizes can be enabled to the code by this option.
`-qoffload`	Enables support for offloading OpenMP target regions to a NVIDIA GPU. Use the OpenMP `#pragma omp target` directive to define a target region. Moreover, for `-qoffload` to take effect, you must specify the `-qsmp` option during compilation.
`-qunroll`	Controls loop unrolling for improved performance.

For more information about the IBM XL C/C++ compiler options and for IBM XL Fortran, see the following IBM Knowledge Center pages:

- IBM Knowledge Center
- IBM Knowledge Center

7.1.2 GNU Compiler Collection compiler options

For GCC, a minimum level of compiler optimization is `-O2`, and the suggested level of optimization is `-O3`. The GCC default is a strict mode, but the `-ffast-math` option disables strict mode. The `-O fast` option combines `-O3` with `-ffast-math` in a single option. Other important options include `-fpeel-loops`, `-funroll-loops`, `-ftree-vectorize`, `-fvect-cost-model`, and `-mcmodel=medium`.

Support for the POWER9 processor is now available on GCC-8.2.1 (which is available from Advanced Toolchain for Linux on Power V12) by using the `-mcpu=power9` and `-mtune=power9` options. The `-mcpu=power9` options automatically enable the options that are shown in Table 7-3.

Table 7-3 POWER9 processor support with GCC-8.2.1

Option name	Description
`-mfloat128-hardware`	Enables IEEE 754 128-bit floating point precision, which is implemented in hardware on POWER9 processor-based servers.
`-misel`	Enables the generation of integer select (ISEL) instruction, which uses the condition register set and allows conditional execution while eliminating branch instruction.
`-mmodulo`	Generates code by using the Power ISA V3.0 integer instructions (modulus, count trailing zeros, array index support, and integer multiply/add).

Option name	Description
-mpower9-fusion	Fuses certain operations together for better performance on POWER9 processor-based servers.
-mpower9-minmax	Enables Power ISA V3.0 MIN/MAX instructions that do not require **-ffast-math** or **-fhonor-nans**.
-mpower9-misc	Uses certain scalar instructions that were added in Power ISA V3.0.
-mpower9-vector	Generates code that uses the vector and scalar instructions that were added in Power ISA V3.0. Also enables using built-in functions that allow more direct access to the vector instructions.

The application binary interface (ABI) type to use for intrinsic vectorizing can be set by specifying the **-mveclibabi=mass** option and linking to the MASS libraries, which enables more loops with **-ftree-vectorize**. The MASS libraries support only static archives for linking. Therefore, the following explicit naming and library search order is required for each platform or mode:

- POWER9 32-bit: **-L<MASS-dir>/lib -lmassvp9 -lmass_simdp9 -lmass -lm**
- POWER9 64-bit: **-L<MASS-dir>/lib64 -lmassvp9_64 -lmass_simdp9_64 -lmass_64 -lm**

For more information about GCC support on POWER9 processor-based servers, see the GNU Manual website.

Optimization parameters

The most commonly used optimization options are listed in Table 7-4.

Table 7-4 Optimization options for GNU Compiler Collection

Option name	Description
-O, -O1	With the **-O** option, the compiler tries to reduce code size and execution time without performing any optimizations that add 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 turns on the **-finline-functions**, **-frename-registers**, **-funswitch-loops**, **-fpredictive-commoning**, **-fgcse-after-reload**, **-ftree-vectorize**, **-fvect-cost-model**, **-ftree-partial-pre**, and **-fipa-cp-clone** options.
-O0	Reduces compilation time and grants debuggers full access to the code. Do not optimize.
-Os	Optimizes 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.

Option name	Description
-ffast-math	Sets **-fno-math-errno**, **-funsafe-math-optimizations**, **-fno-trapping-math**, **-ffinite-math-only**, **-fno-rounding-mat**h, **-fno-signaling-nans**, and **-fcx-limited-range**. This option causes the preprocessor macro **__FAST_MATH__** to be defined. This option must never be turned on by any **-O** option besides the **-Ofast** option because it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules, or 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**, **-frerun-cse-after-loop**, **-fweb**, and **-frename-registers** options.
-fno-inline	Do not expand any functions inline apart from those functions that are marked with the **always_inline** attribute. This setting is the default setting when not optimizing. Single functions can be exempted from inlining by marking them with the **noinline** attribute.
-fno-math-errno	Do not set ERRNO after math functions are called that are run with a single instruction, for example, **sqrt**. A program that relies on IEEE exceptions for math error handling can use this flag for speed while maintaining IEEE arithmetic compatibility.
-finline-functions	Consider all functions for inlining even if they are not declared inline. The compiler heuristically decides which functions are worth integrating in this way. If all calls to a specific function are integrated and the function is declared static, the function is normally not output as assembly language code in its own right. This option is enabled at level **-O3**.

7.2 Porting applications to IBM Power Systems servers

The first challenge that often is encountered when working on a Power Systems server is the fact that software that is written in C, C++, or Fortran includes some significant differences from applications that were developed for x86 and x86_64 systems. Therefore, considering that many applications exist on these architectures, the work of writing and porting applications are assumed to involve a large amount of time, effort, and investment. If done manually, this task can be burdensome.

For example, consider the snippet that is shown in Example 7-1.

Example 7-1 x86 code that does not work properly on Power Systems servers

```
#include <stdio.h>

int main()
{
   char c;

   for( c=0 ; c>=0 ; c++)
```

```
      printf("%d: %c\n", (int) c, c);

   return 0;
}
```

The code that is shown in Example 7-1 on page 167 runs 256 times and displays the expected characters on a x86_64 system, but turns into an infinite loop in a Power Systems server. Considering that characters on a Power Systems server are mapped by default to an unsigned character and x86 characters are mapped to a signed character, the port that is shown in Example 7-2 is necessary to fix the infinite loop challenge.

Example 7-2 Power Systems port

```
--- char_x86.c  2018-10-11 11:10:04.820296533 -0400
+++ char_ppc.c  2018-10-11 11:12:13.785381982 -0400
@@ -1,7 +1,7 @@
 #include <stdio.h>
 int main()
 {
-       char c;
+       signed char c;
        for( c=0 ; c>=0 ; c++)
        printf("%d: %c\n", (int) c, c);
        return 0;
```

If you are coming from a 32-bit environment, several problems can occur because of long pointer data types sizes and the alignment of your variables, as shown in Table 7-5.

Table 7-5 Data types

Data type	32-bit mode		64-bit mode	
	Size	**Alignment**	**Size**	**Alignment**
Long, signed long, and unsigned long	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
Pointer	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
size_t (defined in the header file <cstddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
ptrdiff_t (defined in the header file <cstddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries

Moreover, an architecture that has built-in functions can also be a problem. Consider an example of single-instruction multiple-data (SIMD) vectors for the following code that is designed for an x86 system, as shown in Example 7-3.

Example 7-3 Code with x86 built -in functions that need porting

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

typedef float v4sf __attribute__ ((mode(V4SF)));
```

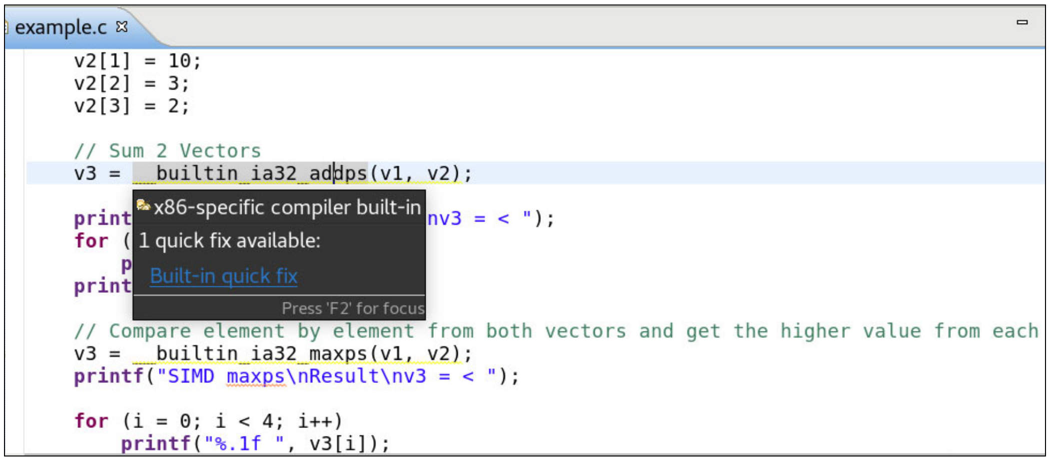
```
int main()
{
   int i;
   v4sf v1, v2, v3;

   //Vector 1
   v1[0] = 20;
   v1[1] = 2;
   v1[2] = 9;
   v1[3] = 100;

   //Vector 2
   v2[0] = 2;
   v2[1] = 10;
   v2[2] = 3;
   v2[3] = 2;

   // Sum 2 Vectors
   v3 = __builtin_ia32_addps(v1, v2);

   printf("SIMD addition\nResult\nv3 = < ");
   for (i = 0; i < 4; i++)
      printf("%.1f ", v3[i]);
    printf(">\n");

   // Compare element by element from both vectors and get the higher value from 
each position
    v3 = __builtin_ia32_maxps(v1, v2);

   printf("SIMD maxps\nResult\nv3 = < ");
   for (i = 0; i < 4; i++)
      printf("%.1f ", v3[i]);
    printf(">\n");

   return 0;
}
```

This code uses x86 built-in functions such as **`__builtin_ia32_addps`** and **`__builtin_ia32_maxps`**. Although you can review documentation to find the corresponding functions (if they exist between these architectures), but this strategy can be burdensome.

To mitigate all of these challenges, IBM created an all-in-one solution for developing and porting applications on Power Systems servers. This solution is called IBM Software Development Kit for Linux on Power, which is described at the IBM Software Development Kit for Linux on Power (SDK) website.

Among other features, this software development kit (SDK) aims to aid developers to code, fix, simulate, port, and run software locally in a x86_64 machine, virtually in a x86_64 simulation, or remotely in a Power Systems server. Currently, this SDK integrates the Eclipse integrated development environment (IDE) with IBM XL C/C++, Advance Toolchain, and open source tools, such as OProfile, Valgrind, and Autotools.

Figure 7-1 shows the detection of such of function in the code that is shown in Example 7-3 on page 168.

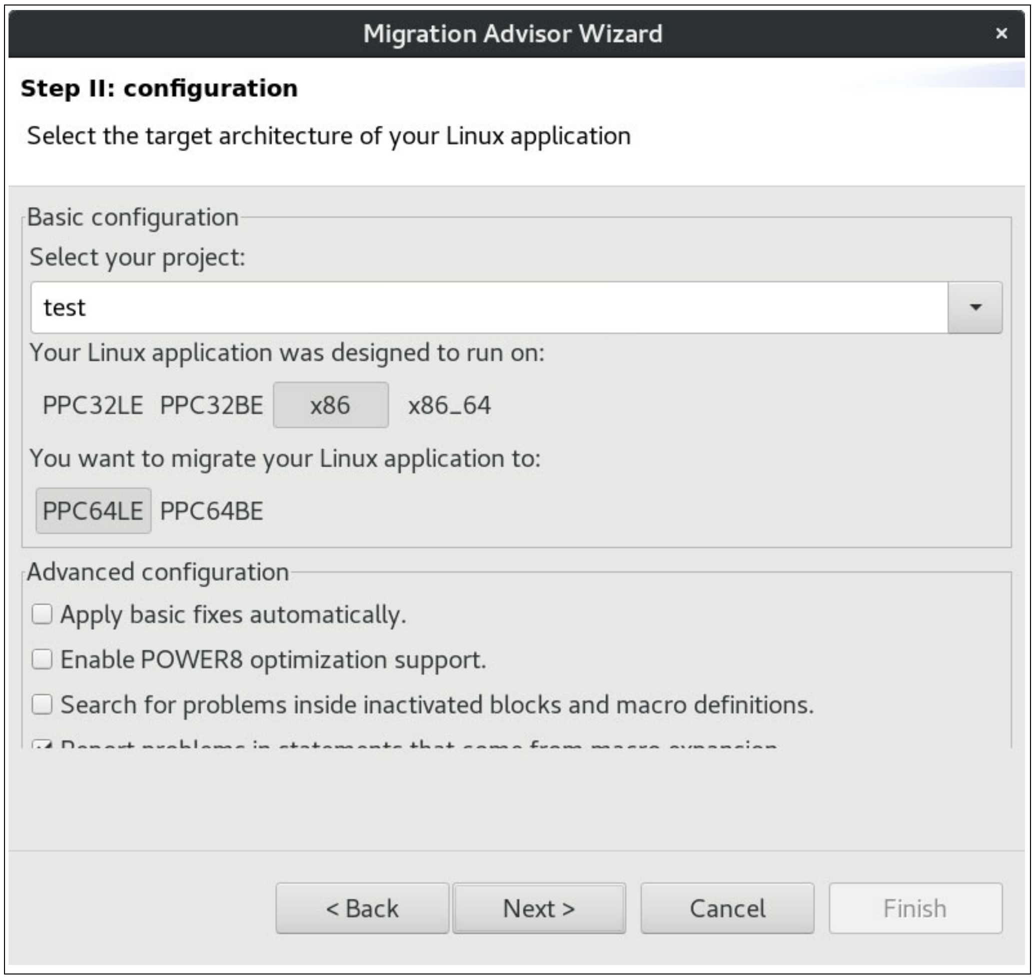


```
    v2[1] = 10;
    v2[2] = 3;
    v2[3] = 2;

    // Sum 2 Vectors
    v3 = __builtin_ia32_addps(v1, v2);

    print                         nv3 = < ");
    for (
        p
    print

    // Compare element by element from both vectors and get the higher value from each
    v3 = __builtin_ia32_maxps(v1, v2);
    printf("SIMD maxps\nResult\nv3 = < ");

    for (i = 0; i < 4; i++)
        printf("%.1f ", v3[i]);
```

Figure 7-1 Porting problems in Example 7-3 on page 168 that are detected by ibm-sdk-lop

In addition, this tool integrates the IBM Feedback Directed Program Restructuring (IBM FDPR®) and **pthread** monitoring tools, which are designed to analyze Power Systems servers These tools include powerful porting and analytic tools, such as Migration Advisor (MA), Source Code Advisor, and CPI Breakdown.

By starting the Migration Advisor Wizard for the code that is shown in Example 7-3 on page 168, two options become available, as shown in Figure 7-2.


Migration Advisor Wizard

Step II: configuration

Select the target architecture of your Linux application

Basic configuration

Select your project: test

Your Linux application was designed to run on: PPC32LE PPC32BE [✓ x86] x86_64

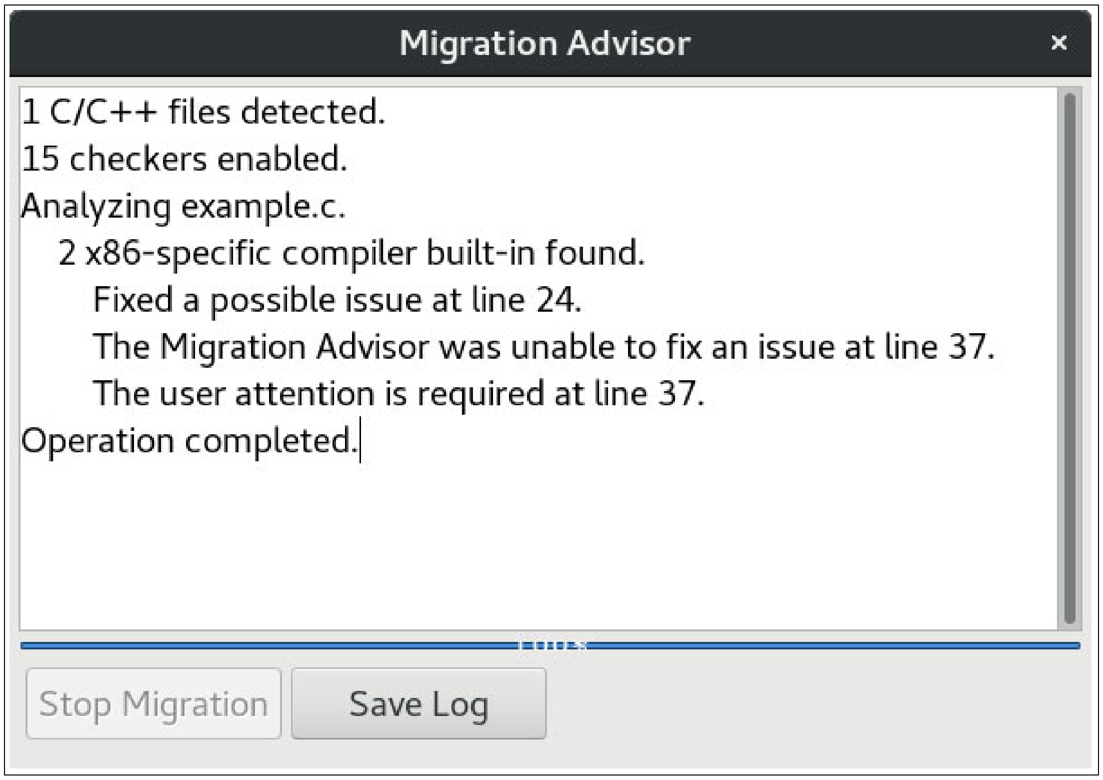
You want to migrate your Linux application to: [✓ PPC64LE] PPC64BE

Advanced configuration

- ☐ Apply basic fixes automatically.
- ☐ Enable POWER8 optimization support.
- ☐ Search for problems inside inactivated blocks and macro definitions.

Figure 7-2 Migration Wizard enables porting code from different architectures and endianness

The code that is shown in Example 7-3 on page 168 leads to the port report, as shown in Figure 7-3.



```
1 C/C++ files detected.
15 checkers enabled.
Analyzing example.c.
   2 x86-specific compiler built-in found.
      Fixed a possible issue at line 24.
      The Migration Advisor was unable to fix an issue at line 37.
      The user attention is required at line 37.
Operation completed.
```

Figure 7-3 Porting log showing changes that were performed on Example 7-3 on page 168

Consider the changes that the MA performed, which are shown in Example 7-4.

Example 7-4 Porting changes that performed by the MA on Example 7-3 on page 168

```
--- vector_before_migration     2018-10-15 12:39:24.435114426 -0400
+++ vector_after_migration      2018-10-15 12:39:56.342548064 -0400
@@ -3,6 +3,13 @@
 typedef float v4sf __attribute__ ((mode(V4SF)));
+
+#ifdef __PPC__
+__vector float vec__builtin_ia32_maxps(__vector float a, __vector float b) {
+       //TODO: You need to provide a PPC implementation.
+}
+#endif
+
 int main()
 {
        int i;
@@ -21,7 +28,12 @@
        v2[3] = 2;
        // Sum 2 Vectors
+
+#ifdef __PPC__
+    v3 = vec_add(v1, v2);
+#else
        v3 = __builtin_ia32_addps(v1, v2);
```

```
+#endif
        printf("SIMD addition\nResult\nv3 = < ");
        for (i = 0; i < 4; i++)
@@ -29,8 +41,12 @@
        printf(">\n");
        // Compare element by element from both vectors and get the higher value
from each position
-       v3 = __builtin_ia32_maxps(v1, v2);
+#ifdef __PPC__
+    v3 = vec__builtin_ia32_maxps(v1, v2);
+#else
+    v3 = __builtin_ia32_maxps(v1, v2);
+#endif
        printf("SIMD maxps\nResult\nv3 = < ");
        for (i = 0; i < 4; i++)
```

The wizard found a corresponding function for the function **`__builtin_ia32_addps()`**, and informed us that we must implement our own version of function **`__builtin_ia32_maxps()`** on **`vec__builtin_ia32_maxps()`**, which is now before **`main()`**. Moreover, all PPC wrappers were created.

By using this tool, you can automatically solve many porting errors from x86 to Power Systems architectures by clicking a button on the `ibm-sdk-lop` interface. For more information about the SDK features, see *Developing software using the IBM Software Development Kit for Linux on Power*.

7.3 IBM Engineering and Scientific Subroutine Library

The IBM Engineering and Scientific Subroutine Library (IBM ESSL) includes the following runtime libraries:

- IBM ESSL Serial Libraries and IBM ESSL Symmetric Multiprocessor (SMP) Libraries
- IBM ESSL SMP Compute Unified Device Architecture (CUDA) Library

The following mathematical subroutines, in nine computational areas, are tuned for performance:

- Linear algebra subprograms
- Matrix operations
- Linear algebraic equations
- Eigensystem analysis
- Fourier transforms, convolutions and correlations, and related computations versus sorting and searching
- Interpolation
- Numerical quadrature
- Random number generation

7.3.1 IBM ESSL Compilation in Fortran, IBM XL C/C++, and GCC/G++

The IBM ESSL subroutines are callable from programs that are written in Fortran, C, and C++. Table 7-6, Table 7-8 on page 175, and Table 7-10 on page 176 list how the compilation on Linux depends on the type of IBM ESSL library (serial, SMP, or SMP CUDA); environment (32-bit integer and 64-bit pointer or 64-bit integer, or 64-bit pointer); and compiler (XLF, IBM XL C/C++, or gcc/g++) that is used.

Table 7-6 Fortran compilation commands for IBM ESSL

Type of IBM ESSL library	Environment	Compilation command
Serial	32-bit integer and 64-bit pointer	`xlf_r -O -qnosave program.f -lessl`
	64-bit integer and 64-bit pointer	`xlf_r -O -qnosave program.f -lessl6464`

SMP	32-bit integer and 64-bit pointer	`xlf_r -O -qnosave -qsmp program.f -lesslsmp`
		`xlf_r -O -qnosave program.f -lesslsmp -lxlsmp`

	64-bit integer and 64-bit pointer	`xlf_r -O -qnosave -qsmp program.f -lesslsmp6464`
		`xlf_r -O -qnosave program.f -lesslsmp6464 -lxlsmp`

SMP CUDA	32-bit integer and 64-bit pointer	`xlf_r -O -qnosave -qsmp program.f -lesslsmpcuda -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64`
		`xlf_r -O -qnosave program.f -lesslsmpcuda -lxlsmp -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64`

To use the Fastest Fourier Transform in the West (FFTW) wrapper libraries, include the header file `fftw3.f`, which contains the constant definitions. To use these definitions, complete one of the following tasks:

- Add the following line to your Fortran application:

```
include "fftw3.f”
```

- Add the `fftw3.f` header file in your application.

You also can compile and link with the FFTW wrapper libraries by running the commands that are listed in Table 7-7.

Table 7-7 Fortran compilation guidelines (assume that the FFTW wrappers are in /usr/local/include)

Type of IBM ESSL library	Environment	Compilation command
Serial	32-bit integer and 64-bit pointer	`xlf_r -O -qnosave program.f -lessl -lfftw3_essl -I/usr/local/include -L/usr/local/lib64`
SMP	32-bit integer and 64-bit pointer	`xlf_r -O -qnosave program.f -lesslsmp -lfftw3_essl -I/usr/local/include -L/usr/local/lib64`

Table 7-8 lists the compilation guidelines for IBM ESSL by using IBM XL C/C++.

Table 7-8 IBM XL C/C++ compilation commands (cc_r for XLC, xlC_r for XLC++) for IBM ESSL

Type of IBM ESSL library	Environment	Compilation command
Serial	32-bit integer and 64-bit pointer	`cc_r (xlC_r) -O program.c -lessl -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
	64-bit integer and 64-bit pointer	`cc_r (xlC_r) -O -D_ESV6464 program.c -lessl6464 -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
SMP	32-bit integer and 64-bit pointer	`cc_r (xlC_r) -O program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
	64-bit integer and 64-bit pointer	`cc_r (xlC_r) -O -D_ESV6464 program.c -lesslsmp6464 -lxlf90_r -lxlsmp -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
SMP CUDA	32-bit integer and 64-bit pointer	`cc_r (xlC_r) -O program.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`

To compile and link the FFTW wrapper libraries, you must use the commands that are listed in Table 7-9. Also, use the header file `fftw3_essl.h` instead of `fftw3.h`.

Table 7-9 IBM XL C compilation guidelines to use the FFTW wrapper libraries

Type of IBM ESSL library	Environment	Compilation command
Serial	32-bit integer and 64-bit pointer	`cc_r (xlC_r) -O program.c -lessl -lxlf90_r -lxlfmath -lfftw3_essl -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib -I/usr/local/include -L/usr/local/lib64`
SMP	32-bit integer and 64-bit pointer	`cc_r (xlC_r) -O program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -lfftw3_essl -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib -I/usr/local/include -L/usr/local/lib64`

Table 7-10 lists the compilation commands for gcc and g++.

Table 7-10 C/C++ compilation commands (gcc for C and g++ for C++) for IBM ESSL[2]

Type of IBM ESSL library	Environment	Compilation command
Serial	32-bit integer and 64-bit pointer	`gcc (g++) program.c -lessl -lxlf90_r -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
	64-bit integer and 64-bit pointer	`gcc (g++) -D_ESV6464 program.c -lessl6464 -lxlf90_r -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
SMP	32-bit integer and 64-bit pointer	`gcc (g++) program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
	64-bit integer and 64-bit pointer	`gcc (g++) -D_ESV6464 program.c -lesslsmp6464 -lxlf90_r -lxlsmp -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`

[2] The IBM ESSL SMP libraries require the IBM XL OpenMP run time. The gcc OpenMP run time is not compatible with the IBM XL OpenMP run time.

Type of IBM ESSL library	Environment	Compilation command
SMP CUDA	32-bit integer and 64-bit pointer	**gcc (g++) program.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lm -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib**

To compile and link the FFTW wrapper libraries, you must use the commands that are listed in Table 7-9 on page 176. Also, use the header file `fftw3_essl.h` instead of `fftw3.h`.

Table 7-11 lists the IBM XL C compilation guidelines to use for the FFTW wrapper libraries, assuming that the FFTW wrappers files are installed in `/usr/local/include`.

Table 7-11 IBM XL C compilation guidelines to use the FFTW wrapper libraries

Type of IBM ESSL library	Environment	Compilation command
Serial	32-bit integer and 64-bit pointer	**gcc (g++) program.c -lessl -lxlf90_r -lxlfmath -lfftw3_essl -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib -I/usr/local/include -L/usr/local/lib64**
SMP	32-bit integer and 64-bit pointer	**gcc (g++) program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -lfftw3_essl -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib -I/usr/local/include -L/usr/local/lib64**

7.3.2 IBM ESSL example

To demonstrate the power of the IBM ESSL API, Example 7-5 shows a C sample source code that uses IBM ESSL to perform the well-known `dgemm` example with matrixes of dimension [20000, 20000] while measuring the execution time and performance of this calculation.

Example 7-5 IBM ESSL C dgemm_sample.c source code for a [20,000x20,000] calculation of dgemm

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <essl.h> //ESSL header for C/C++

//Function to calculate time in milliseconds
long timevaldiff(struct timeval *starttime, struct timeval *finishtime)
{
  long msec;
  msec=(finishtime->tv_sec-starttime->tv_sec)*1000;
  msec+=(finishtime->tv_usec-starttime->tv_usec)/1000;
  return msec;
}

int main()
{
  struct timeval start, end;
  double diff;
  long n, m, k;
  double *a, *b, *c;
  double rmax, rmin;
  double seed1, seed2, seed3;
  double flop;

  //Seeds for matrix generation
  seed1 = 5.0;
  seed2 = 7.0;
  seed3 = 9.0;

  //Maximum and minimum value elements of matrixes
  rmax = 0.5;
  rmin = -0.5;

  //Size of matrixes
  n = 20000; m = n; k =n;

  //Number of additions and multiplications
  flop = (double)(m*n*(2*(k-1)));

  //Memory allocation
  a = (double*)malloc(n*k*sizeof(double));
  b = (double*)malloc(k*m*sizeof(double));
  c = (double*)malloc(n*m*sizeof(double));

  //Matrix generation
  durand(&seed1,n*k,a); //DURAND are included to ESSL, not to CBLAS
  cblas_dscal(n*k,rmax-rmin,a,1);
  cblas_daxpy(n*k,1.0,&rmin,0,a,1);

  durand(&seed2,k*m,b);
  cblas_dscal(k*m,rmax-rmin,b,1);
  cblas_daxpy(k*m,1.0,&rmin,0,b,1);

  durand(&seed3,n*m,c);
  cblas_dscal(n*m,rmax-rmin,c,1);
  cblas_daxpy(n*m,1.0,&rmin,0,c,1);

  //Matrix multiplication (DGEMM)
  gettimeofday(&start,NULL);
  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                             m,n,k,1.0,a,n,b,k,1.0,c,n);
  gettimeofday(&end,NULL);

  //Print results
  printf("%.3lf seconds, ",(double)timevaldiff(&start,&end)/1000);
  printf("%.3lf MFlops\n",flop/(double)timevaldiff(&start,&end)/1000.0);

  //Memory deallocation
  free(a);
  free(b);
  free(c);

  return 0;
}
```

The code uses IBM ESSL routines that are called by C Basic Linear Algebra Subprograms (CBLAS) interfaces, which enable more options to specify matrix order (for example, `column-major` or `row-major`). The calculation that is performed by `dgemm` is done by using the following formula:

$C = \alpha[A] \times [B] + \beta[C]$

Alpha and *beta* are real scalar values, and *A*, *B*, and *C* are matrixes of conforming shape.

The algorithm is shown in Example 7-5 on page 177.

Example 7-6 shows how to compile and run the algorithm that is shown in Example 7-5 on page 177 by using the serial version of IBM ESSL and the XLC compiler instructions that are listed in Table 7-8 on page 175.

Example 7-6 Compilation and execution of dgemm_sample.c for serial IBM ESSL

```
$ xlc_r -O3 dgemm_sample.c -lessl -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/5.1.0/lib
-L/opt/ibm/xlf/16.1.0/lib -R/opt/ibm/lib -o dgemm_serial

$ ./dgemm_serial
670.141 seconds, 23874.379 MFlops
```

For more information about how to use and tune this example to gain performance, see 8.2.1, “IBM ESSL execution in multiple CPUs and GPUs” on page 233.

For more information about the new IBM ESSL features, see IBM Knowledge Center.

For more information about the IBM ESSL SMP CUDA library, see IBM Knowledge Center.

7.4 IBM Parallel Engineering and Scientific Subroutine Library

IBM Parallel ESSL V5.4.0 is a highly optimized mathematical subroutine library for clusters of POWER8 processor-based nodes. IBM Parallel ESSL supports the Single Program Multiple Data (SPMD) programming model that uses the Message Passing Interface (MPI) library. It also assumes that your program is using the SPMD programming model. This configuration means that all parallel tasks are identical and work on different sets of data.

IBM Parallel ESSL supports only 32-bit integer and 64-bit pointer environment libraries, which must be used with the IBM Parallel Environment Runtime Edition Message Passage Interface Chameleon (MPICH) library.

IBM Parallel ESSL subroutines cover the following computational areas:

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- Level 3 PBLAS
- Linear algebraic equations
- Eigensystem analysis and singular value analysis
- Fourier transforms
- Random number generation

IBM Parallel ESSL uses calls of the IBM ESSL subroutines for computational purposes.

For communication, Basic Linear Algebra Communications Subprograms (BLACS) is included, which is based on MPI.

7.4.1 Program development

During the development process of your program, the BLACS subroutines must be used. To include IBM Parallel ESSL calls into the code of the program, complete the following steps:

1. Start the process grid by using the BLACS subroutines (**`BLACS_GET`** call and **`BLACS_GRIDINIT`** or **`BLACS_GRIDMAP`** after it).
2. Distribute data across the process grid. Try to use different block sizes to improve the performance of the program. For example, some IBM Parallel ESSL subroutines use a GPU for large sizes of blocks (about 2000 by 2000).
3. Call the IBM Parallel ESSL subroutine on all processes of the BLACS process grid.
4. Aggregate the results of IBM Parallel ESSL runs from all processes.
5. Call the BLACS subroutines to clean the process grid and exit, such as **`BLACS_GRIDEXIT`** or **`BLACS_EXIT`**.

Example 7-7 shows a sample Fortran dgemm implementation that uses the IBM Parallel ESSL version of the PDGEMM subroutine for a 20000 by 20000 matrix size. PDGEMM works by using the following formula:

C = α[A] x [B] + β[C}

A*lpha* and *beta* are real scalar values*, and A*, *B*, and *C* are matrixes of conforming shape.

The algorithm is shown in Example 7-7.

Example 7-7 IBM Parallel ESSL Fortran example source code pdgemm_sample

```
      program pdgemm_sample
      implicit none
      real*8, allocatable, dimension(:) :: a,b,c
      integer, dimension(9) :: desca,descb,descc
      integer m,n,k
      integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
      integer acol, bcol, ccol
      real*8, parameter :: alpha = 2.d0
      real*8, parameter :: beta  = 3.d0
      integer, parameter :: ia = 1, ib = 1, ic = 1
      integer, parameter :: ja = 1, jb = 1, jc = 1

      integer, parameter :: nb = 200
! Bigger size for CUDA runs
!     integer, parameter :: nb = 3000
! Initialization of process grid
      call blacs_pinfo(iam,np)
      if (np.ne.20) then
        print *, 'Test expects 20 nodes'
        stop 1
      else
        nr = 5
        nc = 4
      endif
      call blacs_get(0,0,icontxt)
      call blacs_gridinit(icontxt,'r',nr,nc)
      call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Size of matrixes
      m = 20000
      n = 20000
      k = 20000

! Fill parameters for PDGEMM call
      desca(1) = 1
      desca(2) = icontxt
      desca(3) = m
      desca(4) = k
      desca(5:6) = nb
      desca(7:8) = 0
      desca(9) = numroc(m,nb,mr,0,nr)
      acol = numroc(k,nb,mc,0,nc)

      descb(1) = 1
      descb(2) = icontxt
      descb(3) = k
      descb(4) = n
      descb(5:6) = nb
      descb(7:8) = 0
      descb(9) = numroc(k,nb,mr,0,nr)
      bcol = numroc(n,nb,mc,0,nc)

      descc(1) = 1
      descc(2) = icontxt
      descc(3) = m
      descc(4) = n
      descc(5:6) = nb
      descc(7:8) = 0
      descc(9) = numroc(m,nb,mr,0,nr)
      ccol = numroc(n,nb,mc,0,nc)

      allocate(a(desca(9)*acol))
      allocate(b(descb(9)*bcol))
      allocate(c(descc(9)*ccol))

! PDGEMM call
      a = 1.d0
      b = 2.d0
      c = 3.d0
      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
     &            beta,c,ic,jc,descc)

! Deallocation of arrays and exit from BLACS
      deallocate(a)
      deallocate(b)
      deallocate(c)
      call blacs_gridexit(icontxt)
      call blacs_exit(0)
      end
```

For more information about using BLACS with the IBM Parallel ESSL library, see IBM Knowledge Center.

For more information about the concept of development by using the IBM Parallel ESSL library, see this IBM Knowledge Center.

7.4.2 Using GPUs with the IBM Parallel ESSL

GPUs can be used by the local MPI tasks in the following ways within the IBM Parallel ESSL programs:

- GPUs are not shared.

  This setting means that each MPI task on a node uses unique GPUs. For this case, the local rank of MPI tasks can be used.

  Example 7-8 shows how to work with local rank by using the **MP_COMM_WORLD_LOCAL_RANK** variable. It is created from Example 7-7 on page 180 with another section that gets the local rank of the MPI task and assigns this task to a respective GPU by rank. Change the size of the process grid to 4 by 2 to fit your cluster, which has two nodes with 4 GPUs on each node, and uses a block size of 3000 by 3000.

  Example 7-8 IBM Parallel ESSL Fortran example source code for non-shared GPUs

```
program pdgemm_sample_local_rank
      implicit none
      real*8, allocatable, dimension(:) :: a,b,c
      integer, dimension(9) :: desca,descb,descc
      integer m,n,k
      integer ids(1)
      integer ngpus
      integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
      integer acol, bcol, ccol
      real*8, parameter :: alpha = 2.d0
      real*8, parameter :: beta  = 3.d0
      integer, parameter :: ia = 1, ib = 1, ic = 1
      integer, parameter :: ja = 1, jb = 1, jc = 1

      integer, parameter :: nb = 3000

      character*8 rank
      integer lrank,istat

! Initialization of process grid
      call blacs_pinfo(iam,np)
      if (np.ne.8) then
        print *, 'Test expects 8 nodes'
        stop 1
      else
        nr = 4
        nc = 2
      endif
      call blacs_get(0,0,icontxt)
      call blacs_gridinit(icontxt,'r',nr,nc)
      call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Get local rank and assign respective GPU
      call getenv('MP_COMM_WORLD_LOCAL_RANK',value=rank)
      read(rank,*,iostat=istat) lrank
      ngpus = 1
      ids(1) = lrank
      call setgpus(1,ids)

! Size of matrixes
      m = 20000
      n = 20000
      k = 20000

! Fill parameters for PDGEMM call
      desca(1) = 1
      desca(2) = icontxt
      desca(3) = m
      desca(4) = k
      desca(5:6) = nb
      desca(7:8) = 0
      desca(9) = numroc(m,nb,mr,0,nr)
      acol = numroc(k,nb,mc,0,nc)

      descb(1) = 1
      descb(2) = icontxt
      descb(3) = k
      descb(4) = n
      descb(5:6) = nb
      descb(7:8) = 0
      descb(9) = numroc(k,nb,mr,0,nr)
      bcol = numroc(n,nb,mc,0,nc)

      descc(1) = 1
      descc(2) = icontxt
      descc(3) = m
      descc(4) = n
      descc(5:6) = nb
      descc(7:8) = 0
      descc(9) = numroc(m,nb,mr,0,nr)
      ccol = numroc(n,nb,mc,0,nc)

      allocate(a(desca(9)*acol))
      allocate(b(descb(9)*bcol))
      allocate(c(descc(9)*ccol))
```

```
! PDGEMM call
      a = 1.d0
      b = 2.d0
      c = 3.d0
      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
    &           beta,c,ic,jc,descc)

! Deallocation of arrays and exit from BLACS
      deallocate(a)
      deallocate(b)
      deallocate(c)
      call blacs_gridexit(icontxt)
      call blacs_exit(0)
      end
```

- GPUs are shared.

  This setting is used when the number of MPI tasks per node oversubscribe the GPUs. IBM Parallel ESSL recommends using NVIDIA Multi-Process Service (MPS), as described in 9.5.2, "CUDA Multi-Process Service" on page 292. The process allows you to use multiple MPI tasks concurrently by using GPUs.

  Note: It is possible that IBM Parallel ESSL cannot allocate memory on the GPU. In this case, you can reduce the number of MPI tasks per node.

  To inform IBM Parallel ESSL which GPUs to use for MPI tasks, use the **SETGPUS** subroutine. Example 7-9 shows the updated version of Example 7-15 on page 200, where IBM Parallel ESSL uses only one GPU for each MPI task, and the GPU is assigned in round-robin order. The algorithm is shown in Example 7-9.

Example 7-9 Calling the SETGPUS subroutine for one GPU

```
program pdgemm_sample
      implicit none
      real*8, allocatable, dimension(:) :: a,b,c
      integer, dimension(9) :: desca,descb,descc
      integer m,n,k
      integer ids(1)
      integer ngpus
      integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
      integer acol, bcol, ccol
      real*8, parameter :: alpha = 2.d0
      real*8, parameter :: beta  = 3.d0
      integer, parameter :: ia = 1, ib = 1, ic = 1
      integer, parameter :: ja = 1, jb = 1, jc = 1

      integer, parameter :: nb = 3000

! Initialization of process grid
      call blacs_pinfo(iam,np)
      if (np.ne.20) then
        print *, 'Test expects 20 nodes'
        stop 1
      else
        nr = 5
```

```
          nc = 4
        endif
        ngpus = 1
        ids(1) = mod(iam,4)
        call setgpus(ngpus,ids)
        call blacs_get(0,0,icontxt)
        call blacs_gridinit(icontxt,'r',nr,nc)
        call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Size of matrixes
        m = 20000
        n = 20000
        k = 20000

! Fill parameters for PDGEMM call
        desca(1) = 1
        desca(2) = icontxt
        desca(3) = m
        desca(4) = k
        desca(5:6) = nb
        desca(7:8) = 0
        desca(9) = numroc(m,nb,mr,0,nr)
        acol = numroc(k,nb,mc,0,nc)

        descb(1) = 1
        descb(2) = icontxt
        descb(3) = k
        descb(4) = n
        descb(5:6) = nb
        descb(7:8) = 0
        descb(9) = numroc(k,nb,mr,0,nr)
        bcol = numroc(n,nb,mc,0,nc)

        descc(1) = 1
        descc(2) = icontxt
        descc(3) = m
        descc(4) = n
        descc(5:6) = nb
        descc(7:8) = 0
        descc(9) = numroc(m,nb,mr,0,nr)
        ccol = numroc(n,nb,mc,0,nc)

        allocate(a(desca(9)*acol))
        allocate(b(descb(9)*bcol))
        allocate(c(descc(9)*ccol))

! PDGEMM call
        a = 1.d0
        b = 2.d0
        c = 3.d0
        call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
       &            beta,c,ic,jc,descc)

! Deallocation of arrays and exit from BLACS
        deallocate(a)
```

```
      deallocate(b)
      deallocate(c)
      call blacs_gridexit(icontxt)
      call blacs_exit(0)
      end
```

Example 7-8 on page 182 explicitly states that you should use two GPUs per MPI task. To change this configuration, change the following lines:

```
ngpus = 1
ids(1) = mod(iam,4)
```

to the following lines:

```
ngpus = 2
ids(1) = mod(iam,2)*2
ids(2) = mod(iam,2)*2 + 1
```

7.4.3 Compilation

The IBM Parallel ESSL subroutines can be called from 64-bit-environment application programs, which are written in Fortran, C, and C++. Compilation commands of source code for different types of IBM Parallel ESSL libraries (SMP and SMP CUDA) that use MPICH libraries are described in Table 7-12, Table 7-13 on page 187, Table 7-14 on page 187, Table 7-15 on page 188, and Table 7-16 on page 188.

You do not need to modify your Fortran compilation procedures when IBM Parallel ESSL for Linux is used. When linking and running your program, you must modify your job procedures to set up the necessary libraries. If you are accessing IBM Parallel ESSL for Linux from a Fortran program, you can compile and link by running the commands that are listed in Table 7-12.

Table 7-12 Fortran compilation commands for IBM Parallel ESSL by using IBM Spectrum MPI libraries

Type of IBM Parallel ESSL library	Compilation command
64-bit SMP	`mpifort -O program.f -lesslsmp -lpesslsmp -lblacssmp -lxlsmp`
64-bit SMP CUDA	`mpifort -O xyz.f -lesslsmpcuda -lpesslsmp -lblacssmp -lxlsmp -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64`

The IBM Parallel ESSL for Linux header files `pessl.h` and `Cblacs.h`, which are used for C and C++ programs, are installed in the `/usr/include` directory. When linking and running your program, you must modify your job procedures to set up the necessary libraries. If you are accessing IBM Parallel ESSL for Linux from a C program, you can compile and link by running the commands that are listed in Table 7-13.

Table 7-13 C program compile and link commands for use with IBM Spectrum MPI

Type of IBM Parallel ESSL library	**Compilation command**

64-bit SMP	`mpicc -O program.c`
	`-lesslsmp -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp -lxlfmath`
	`-lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`mpicc -O program.c`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp`
	`-lxlfmath -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

To use gcc as your C compiler, you can compile and link by running the commands that are listed in Table 7-14.

Table 7-14 C program gcc compile and link commands for use with IBM Spectrum MPI

Type of IBM Parallel ESSL library	**Compilation command**

64-bit SMP	`export OMPI_CC=gcc`
	
	`mpicc -O xyz.c -lesslsmp -lpesslsmp -lblacssmp -lxlf90_r`
	`-lxl -lxlsmp -lxlfmath -lm -lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`export OMPI_CC=gcc`
	
	`mpicc -O program.c`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxl -lxlsmp`
	`-lxlfmath -lm -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64`
	`-R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

To develop C++ programs with Parallel EESL, use the commands that are listed in Table 7-15.

Table 7-15 C++ program compile and link commands for use with IBM Spectrum MPI

Type of IBM Parallel ESSL library	Compilation command

64-bit SMP	`mpiCC -O program.C`
	`-lesslsmp -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp -lxlfmath`
	`-lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`mpiCC -O program.C`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp`
	`-lxlfmath -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64`
	`-R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

To use g++ as your C++ compiler, you can compile and link by running the commands that are listed in Table 7-16.

Table 7-16 C++ program g++ compile and link commands for use with IBM Spectrum MPI

Type of IBM Parallel ESSL library	Compilation command

64-bit SMP	`export OMPI_CXX=g++`
	
	`mpiCC -O program.C`
	`-lesslsmp -lpesslsmp -lblacssmp -lxlf90_r -lxl -lxlsmp`
	`-lxlfmath -lm -lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`export OMPI_CXX=g++`
	
	`mpiCC -O program.C`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxl -lxlsmp`
	`-lxlfmath -lm -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64`
	`-R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

For more information about all of the features of the IBM Parallel ESSL library, see IBM Knowledge Center.

7.5 Using POWER9 vectorization

The SIMD instructions are building blocks that are used to perform parallelism at CPU level. The Power Architecture implements SIMD through the VMX and VSX technologies, which are specified in Power Instruction Set Architecture (Power ISA) V3.0 for POWER9 processors.

Techniques to use data parallelism through SIMD instructions are called *vectorization*. They can be used by the compiler in the form of auto-vectorization transformations or made available to applications as APIs.

GCC and IBM XL compilers provide vector APIs that are based on the AltiVec specification for C, C++, and Fortran. Their API is composed of built-in functions (also known as intrinsics), defined vector types, and extensions to the programming language semantics. Each compiler vector implementation is explained in the following sections.

For more information about the auto-vectorization features of the GCC and IBM XL compilers, see 7.1, “Compiler options” on page 162.

7.5.1 AltiVec operations with GCC

GCC provides a modified API that uses AltiVec operations for the POWER family of processors. This interface is made available by including `<altivec.h>` in the source code and by using the **`-maltivec`** and **`-mabi=altivec`** compiling functions. This feature is also made available if the code is compiled with **`-mvsx`** because this option enables **`-maltivec`** with more features that use VSX instructions.

For more information about the AltiVec API specification, see the “PowerPC AltiVec Built-in Functions” section of *Using the GNU Compiler Collection (GCC)*.

The following features are implemented:

- Add the keywords **`__vector`**, **`vector`**, **`__pixel`**, **`pixel`**, **`__bool`**, and **`bool`**. The **`vector`**, **`pixel`**, and **`bool`** keywords are implemented as context-sensitive and 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 GCC implementation does not allow a `typedef` name as a type identifier. You must use the actual **`__vector`** keyword, for example, **`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.
- A 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 often 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).
- An assignment operation (`a = b`) is allowed only if both sides have the same vector types.

- The address operation `&p` is valid if `a` is the `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 >>), and comparison operators (==, !=, <, <=, >, and >=) can be used on these types. The compiler generates the correct SIMD instructions for the hardware.

Table 7-17 shows vector data type extensions as implemented by GCC. Vector types are signed by default unless an otherwise `unsigned` keyword is specified. The only exception is `vector char`, which is unsigned by default. The hardware does not have instructions for supporting `vector long long` and `vector bool long long` types, but they can be used for float-point/integer conversions.

Table 7-17 Vector types as implemented by GCC

Vector types	**Description**
`vector char`	Vector of a sixteen 8-bit characters.
`vector bool`	Vector of a sixteen 8-bit unsigned characters.
`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 integers.
`vector bool int`	Vector of four 32-bit integers.
`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 integers. It is implemented in 64-bit mode only. Requires compile with **`-mvsx`**.
`vector long long`	Vector of two 64-bit signed integers.
`vector bool long`	Vector of two 64-bit signed integers.

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*.

7.5.2 AltiVec operations with IBM XL

The IBM XL compiler family provides an implementation of AltiVec APIs through feature extensions for C and C++. Fortran extensions for vector support are also available.

IBM XL C/C++

To use vector extensions, the application must be compiled with **`-mcpu=pwr9`**, and **`-qaltivec`** must be in effect.

The IBM XL C/C++ implementation defines the **`vector`** (or alternatively, **`__vector`**) keywords that are used in the declaration of vector types.

Similar to GCC implementation, IBM XL C/C++ allows unary, binary, and relational operators to be applied to vector types. It implements all data types that are listed in Table 7-17 on page 190.

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; therefore, 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 IBM XL C/C++ 16.1.0 vector support, vector initialization, and the **`vec_step`** operator, see IBM Knowledge Center.

For more information about built-in functions for vector operations, see IBM Knowledge Center.

IBM XL Fortran

To use vector extensions, the application must comply with **`-qarch=pwr9`**.

The IBM 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 also are extensions to the language. They must be used within vectors only.

Vectors are automatically aligned to 16 bytes, but exceptions apply. For more information about vector types on IBM XL Fortran 16.1.0, see IBM Knowledge Center.

For the list of vector intrinsic procedures that are available with IBM XL Fortran V16.1.0, see IBM Knowledge Center.

Example 7-10 uses the IBM XL Fortran vector library to perform the following calculation:

$C = \alpha[A] + [B]$

A*lpha* and *beta* are real scalar values, and *A*, *B*, and *C* are matrixes of conforming shapes.

Example 7-10 Fortran program that demonstrates use of IBM XL compiler vectors

```
SUBROUTINE VSX_TEST
implicit none

real*8, allocatable :: A(:), B(:), C(:), CT(:)
real*8 alpha
```

```
  integer*8 max_size, ierr
  integer*8 i, j, it
  integer*8 ia, ialign
  integer   n, nalign

  vector(real*8) va1, va2, va3
  vector(real*8) vb1, vb2
  vector(real*8) vc1, vc2
  vector(real*8) valpha

  max_size = 2000
  alpha = 2.0d0

  ierr = 0
  allocate(A(max_size),stat=ierr)
  allocate(B(max_size),stat=ierr)
  allocate(C(max_size),stat=ierr)
  allocate(CT(max_size),stat=ierr)
  if (ierr .ne. 0) then
    write(*,*) 'Allocation failed'
    stop 1
  endif

  do i = 1, max_size
    a(i) = 1.0d0*i
    b(i) = -1.0d0*i
    ct(i) = alpha*a(i) + b(i)
  enddo

  ia     = LOC(A)
  ialign = IAND(ia, 15_8)
  nalign = MOD(RSHIFT(16_8-ialign,3_8),7_8) + 2

!  Compute Head
  j = 1
  do i = 1, nalign
    C(j) = B(j) + alpha * A(j)
    j = j + 1
  enddo

  n = max_size - nalign - 4
  it = rshift(n, 2)

  va1 = vec_xld2( -8, A(j))
  va2 = vec_xld2(  8, A(j))
  va3 = vec_xld2( 24, A(j))

  va1 = vec_permi( va1, va2, 2)
  va2 = vec_permi( va2, va3, 2)

  vb1 = vec_xld2(  0, B(j))
  vb2 = vec_xld2( 16, B(j))

  do i = 1, it-1
    vc1 = vec_madd( valpha, va1, vb1)
```

```
61        vc2 = vec_madd( valpha, va2, vb2)
62
63        va1 = va3
64        va2 = vec_xld2( 40, A(j))
65        va3 = vec_xld2( 56, A(j))
66
67        va1 = vec_permi( va1, va2, 2)
68        va2 = vec_permi( va2, va3, 2)
69
70        call vec_xstd2( va1,  0, C(j))
71        call vec_xstd2( va2, 16, C(j))
72
73        vb1 = vec_xld2( 32, B(j))
74        vb1 = vec_xld2( 48, B(j))
75
76        j = j + 4
77      enddo
78
79      vc1 = vec_madd( valpha, va1, vb1)
80      vc2 = vec_madd( valpha, va2, vb2)
81
82      call vec_xstd2( va1,  0, C(j))
83      call vec_xstd2( va2, 16, C(j))
84
85 !     Compute Tail
86      do i = j, max_size
87        C(i) = B(i) + alpha * A(i)
88      enddo
89
90      do i = 1, 10
91        write(*,*) C(i), CT(i)
92      enddo
93
94      END SUBROUTINE VSX_TEST
```

Lines 11 - 14 in Table 7-11 on page 177 show a declaration of vectors with 16 elements of 8 bytes of real types. Called methods **vec_xld2** (load), **vec_permi** (permuting), and **vec_madd** (fused) are multiply add SIMD operations that are applied to the vector types.

7.6 Development models

The high-performance computing (HPC) solution that is proposed in this book contains a software stack that allows for the development of C, C++, and Fortran applications by using different parallel programming models. In this context, applications can be implemented by using *pure models*, such as MPI, OpenMP, CUDA, OpenACC, Parallel Active Messaging Interface (PAMI), or OpenSHMEM, or by using some combinations of these models (also known as *hybrid models*).

This section describes aspects of IBM Parallel Environment, compilers (GNU and IBM XL families), libraries, and toolkits that developers can use to implement applications on pure or hybrid parallel programming models. It does not describe how those applications can be run with IBM Parallel Environment (for more information, see 8.3, "Using IBM Parallel Environment V2.3" on page 241.

7.6.1 OpenMP programs with IBM Parallel Environment

OpenMP applies a shared memory parallel programming model of development. It is a multi-platform and directive-based API that is available to many languages, including C, C++, and Fortran.

The development and execution of OpenMP applications are fully supported by IBM Parallel Environment Runtime Edition. OpenMP parallelism uses directives that use what is known as *shared memory parallelism* because it defines various types of parallel regions. The parallel regions can include iterative and non-iterative segments of code.

The uses of the **`#pragma omp`** directives can be put in to the following general categories:

- Defines parallel regions in which work is done by threads in parallel (**`#pragma omp parallel`**). Most of the OpenMP directives statically or dynamically bind to an enclosing parallel region.
- Defines how work is distributed or shared across the threads in a parallel region (**`#pragma omp sections`**, **`#pragma omp for`**, **`#pragma omp single`**, and **`#pragma omp task`**).
- Controls synchronization among threads (**`#pragma omp atomic`**, **`#pragma omp master`**, **`#pragma omp barrier`**, **`#pragma omp critical`**, **`#pragma omp flush`**, and **`#pragma omp ordered`**).
- Defines the scope of data visibility across parallel regions within the same thread (**`#pragma omp threadprivate`**).
- Controls synchronization (**`#pragma omp taskwait`** and **`#pragma omp barrier`**).
- Controls data or computation that is on another computing device.

You can specify the visibility context for selected data variables by using some OpenMP clauses. Scope attribute clauses are listed in Table 7-18.

Table 7-18 OpenMP variable scope on IBM Parallel Environment

Data scope attribute clause	Description
`private`	The **`private`** clause declares the variables in the list to be private to each thread in a team.
`firstprivate`	The **`firstprivate`** clause declares the variables in the list to be private to each thread in a team with the value that variable had before the parallel section.
`lastprivate`	Similar to **`firstprivate`**, **`lastprivate`** updates the variable in the outer scope of the parallel region with the last value it had in the parallel region.
`shared`	The **`shared`** clause declares the variables in the list to be shared among all the threads in a team. All threads within a team access the same storage area for shared variables.
`reduction`	The **`reduction`** clause performs a reduction on the scalar variables that appear in the list by using a specified operation.
`default`	The **`default`** clause allows the user to affect the data-sharing attribute of the variables that appeared in the parallel construct.

To demonstrate some of the clauses that are listed in Table 7-18 on page 194, Example 7-11 shows the following simple parallel algorithm that can be used to calculate the dot product of two vectors:

$$\left(A \cdot B = \sum_{i=1}^{N} a_i b_i\right), A = (a_1, a_2, \ldots, a_N); B = (b_1, b_2, \ldots, b_N)$$

A and *B* are vectors of conforming shape. The algorithm is shown in Example 7-11.

Example 7-11 IBM ESSL C dgemm_sample.c source code for a [20,000x20,000] calculation of dgemm

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <sys/time.h>
#include <omp.h>

void print_v(double*,int);

int main(int argc, char* argv[])
{
        if(argc!=4) { printf("No verbose || size of vectors || or number of 
threads provided\nAborting...\n"); return -1; }

        int i, verbose, N, n_threads;
        double *a, *b, sum;

        sum = 0.0;
        verbose = atoi(argv[1]);
        N = atoi(argv[2]);
        n_threads = atoi(argv[3]);
        a = (double*) malloc(sizeof(double)*N);
        b = (double*) malloc(sizeof(double)*N);

        for(i=0;i<N;i++)
                a[i]=b[i]=(double)(i+1);

        #pragma omp parallel for default(none) firstprivate(N) private(i) 
shared(a,b) reduction(+:sum) num_threads(n_threads)
        for(i=0; i<N; i++)
                sum += a[i] * b[i];

if(verbose) { printf("[A] = "); print_v(a,N); printf("[B] = "); print_v(b,N); }

        printf("[A]*[B] = %.2f\n", sum);

        return 0;
}

void print_v(double *v, int N)
{
        int i; printf("[");
        for(i=0;i<N;i++)
        {
                if( i!=(N-1) )
                        printf("%.2f ",v[i]);
                else
                        printf("%.2f]\n",v[i]);
        }
}
```

Line 27 from the source code that is shown in Example 7-11 on page 195 is of interest in this section:

```
   #pragma omp parallel for default(none) firstprivate(N) private(i)
shared(a,b) reduction(+:sum) num_threads(n_threads)
```

- The **`default(none)`** clause is used to make the compiler remind you that it must know the scope of each variable in the parallel section.
- The **`firstprivate(N)`** clause is used because you want each thread to have its own `N` value and the previous value of `N`. However, **`private(i)`** is there because you want the loop variable to be thread-independent and you do not need its previous value. Also, the shared variables are the data that is accessed by each thread independently because of the private **`i`** index.
- The **`reduction(sum)`** clause is there because you want to add all independent products in the sum variable after the loop.
- The **`num_thread(n_threads)`** clause is there to define the number of threads to run the **`#pragma omp`** parallel section.

Example 7-12 shows how to compile the source code.

Example 7-12 Compiling an OpenMP program with IBM XL C

```
$ xlc_r -O3 -Wall -o dot_xlc dot_xlc.c  -qsmp=omp -mcpu=pwr9
-mtune=pwr9
```

From a compiler perspective, IBM XL C/C++ 16.1.0 and Fortran 16.1.0 support OpenMP API V4.5. Similarly, the GCC compiler that is provided by the IBM Advance Toolchain for Linux on Power is based on OpenMP API V4.5.

For more information about IBM XL 16.1.0 and OpenMP, see *Optimization and Programming Guide for Little Endian Distributions*.

For more information about OpenMP examples and the directives and their applications, see *OpenMP Application Programming Interface: Examples*.

For more information about offloading code to GPU, see the Offloading Support in GCC page.

7.6.2 CUDA C programs with the NVIDIA CUDA Toolkit

The development of parallel programs that use the General Purpose GPU (GPGPU) model is provided in the Power AC922 server with support for the NIVIDIA CUDA Toolkit 9.2 for Linux on POWER9 Little Endian.

This section describes relevant aspects of the CUDA development in the proposed solution. This section also describes characteristics of the NVIDIA V100 GPU, integration between compilers, and the availability of libraries. For more information about the NVIDIA CUDA development model and resources, see the NVIDIA CUDA Zone website.

Understanding the NVIDIA V100 GPU CUDA capabilities

The Power AC922 server supports two, four, or six NVIDIA Tesla V100 GPUs, each implementing the Tesla V100/SXM2 CUDA compute architecture. The Tesla V100 architecture delivers CUDA compute capability V7.0.

Table 7-19 lists the available resources (per GPU) and the limitations for the CUDA C++ applications that are found with the **`deviceQuery`** script that is available with the sample codes from NVIDIA.

Table 7-19 CUDA available resources per GPU

GPU resources	Value
Total of global memory	15360 MB (16,106,127,360 bytes)
Total amount of constant memory	64 KB
Shared memory per block	48 KB
Stream Multiprocessors (SMs)	80
Maximum warps per SM	64
Threads per Warps	32
Maximum threads per SM	2048
Maximum thread blocks per SM	32
Maximum threads per block	1024
Maximum grid size	(x, y, z) = (2147483647, 65535, 65535)
Max dimension size of a thread block (x,y,z)	(x,y,z) = (1024, 1024, 64)
Maximum Texture Dimension Size (x,y,z)	1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
Maximum layered texture size and number of layers	1D=(32768), 2048 layers, 2D=(32768, 32768), 2048 layers

The GV100 GPU that is included in the Tesla V100 has the following innovative features:

- Extreme performance: Powering HPC, deep learning (DL), and many more GPU computing areas.
- Second-generation NVLink: The NVIDIA new high-speed, high-bandwidth interconnect for maximum application scalability includes cache coherence capabilities with IBM POWER9 processor-based servers.
- High-bandwidth memory (HBM2): The fastest, high-capacity, efficient, and stacked GPU memory architecture.
- Unified memory and compute preemption: Significantly improved programming model.
- 12 nm FFN: Enables more features, higher performance, and improved power efficiency.

NVIDIA nvcc compiler

The NVIDIA **nvcc** compiler driver is responsible for generating the final executable file, which is a combination of host (CPU) and device (GPU) codes.

IBM XL and GCC compilers can be used to generate the host code and their flags. As described in 7.1, "Compiler options" on page 162, this process produces an optimized code to run in the POWER9 processor. By default, `nvcc` uses the GCC, unless the **-ccbin** flag is passed to set another back-end compiler.

Note: The `nvcc` compiler uses the GCC C++ compiler by default to generate the host code. If wanted, the IBM XL C++ compiler can instead use the **-ccbin xlC** option.

The NVIDIA Tesla V100 GPUs are built on Tesla GV100 architecture, which provides CUDA compute capability V7.0. Use the **-gencode** compiler option to set the virtual architecture and binary compatibility. For example, **-gencode arch=compute_70,code=sm_70** generates code that is compatible with the Tesla GV100 architecture (virtual architecture V7.0).

Example 7-13 shows a CUDA program that prints to standard output the index values for 2D thread blocks. The executable file is built with **nvcc**, which uses xlC as the host compiler (by using the **-ccbin xlC** option). Parameters to xlC are passed by the **-Xcompiler** option.

Example 7-13 CUDA C program and nvcc compilation

```
$ cat -n hello.cu
#include<cuda_runtime.h>
#include<stdio.h>

  __global__ void helloKernel() {
      int tidx = threadIdx.x;
      int tidy = threadIdx.y;
      int blockidx = blockIdx.x;
      int blockidy = blockIdx.y;

      printf("I am CUDA thread (%d, %d) in block (%d, %d)\n",
        tidx, tidy, blockidx, blockidy);
  };

int main(int argc, char* argv[]) {
  dim3 block(16,16);
  dim3 grid(2,2);
  helloKernel<<<grid, block>>>();
  cudaDeviceSynchronize();
  return 0;
}

$ nvcc -ccbin xlC -Xcompiler="-qarch=pwr9 -qtune=pwr9 -qhot -O3" -gencode
arch=compute_70,code=sm_70 hello.cu -o helloCUDA
$ ./helloCUDA
I am CUDA thread (0, 10) in block (1, 0)
I am CUDA thread (1, 10) in block (1, 0)
I am CUDA thread (2, 10) in block (1, 0)
I am CUDA thread (3, 10) in block (1, 0)
I am CUDA thread (4, 10) in block (1, 0)
I am CUDA thread (5, 10) in block (1, 0)
<... Output omitted ...>
```

The nvcc also supports cross-compilation of CUDA C and C++ code to PowerPC 64-bit Little Endian (ppc64le).

Note: The support for POWER9 Little Endian was introduced in CUDA Toolkit 9.1.

For more information about the nvcc compilation process and options, see the NVIDIA CUDA Compiler NVCC page.

CUDA libraries

Accompanying the CUDA Toolkit for Linux are the following mathematical utility and scientific libraries that use the GPU to improve performance and scalability:

- cuBLAS: Basic linear algebra subroutines
- cuFFT: Fast fourier transforms
- cuRAND: Random number generation
- cuSOLVER: Dense and sparse direct linear solvers and Eigen solvers
- cuSPARSE: Sparse matrix routines
- Thrust: Parallel algorithm and data structures

Example 7-14 shows the use of the following cuBLAS API to calculate the dot product of two vectors.

$$\left(A \cdot B = \sum_{i=1}^{N} a_i b_i\right), A = (a_1, a_2, \ldots, a_N); B = (b_1, b_2, \ldots, b_N)$$

A and *B* are vectors of conforming shape.

This algorithm uses IBM XLC++ (xlC) to generate the host code and compile it by using the **-lcublas** flag that links dynamically with the cuBLAS library.

Example 7-14 Sample code for CUDA C by using cuBLAS library

```
#include<cublas_v2.h>
#include<cuda_runtime.h>
#include<cuda_runtime_api.h>
#include<stdlib.h>
#include<stdio.h>

#define N 10000

int main(int argc, char* argv[]) {
        int const VEC_SIZE = N*sizeof(double);
        double* h_vec_A = (double*) malloc(VEC_SIZE);
        double* h_vec_B = (double*) malloc(VEC_SIZE);

        double *d_vec_A, *d_vec_B, result;
        cublasStatus_t status;
        cublasHandle_t handler;
        cudaError_t error;
        // Initialize with random numbers between 0-1
        int i;
        for(i=0; i<N; i++) {
                h_vec_A[i] = (double) (rand() % 100000)/100000.0;
                h_vec_B[i] = (double) (rand() % 100000)/100000.0;
        }
cudaMalloc((void **)&d_vec_A, VEC_SIZE);
        cudaMalloc((void **)&d_vec_B, VEC_SIZE);
        cudaMemcpy(d_vec_A, h_vec_A, VEC_SIZE ,cudaMemcpyHostToDevice);
        cudaMemcpy(d_vec_B, h_vec_B, VEC_SIZE ,cudaMemcpyHostToDevice);

        // Initialize cuBLAS
        status = cublasCreate(&handler);
        // Calculate DOT product
        status = cublasDdot(handler, N , d_vec_A, 1, d_vec_B, 1, &result);
        if(status != CUBLAS_STATUS_SUCCESS) {
                printf("Program failed to calculate DOT product\n");
                return EXIT_FAILURE;
        }
        printf("The DOT product is: %G\n", result);

        // Tear down cuBLAS
        status = cublasDestroy(handler);
        return EXIT_SUCCESS;
}
```

The source code that is shown in Example 7-14 on page 199 is built and run as shown in Example 7-15.

Example 7-15 Building and running source code and cuBLAS sample code

```
$ make
/usr/local/cuda-9.2/bin/nvcc -ccbin xlC  -Xcompiler -qtune=pwr9  -Xcompiler -qhot
-Xcompiler  -O3 -Xcompiler -qarch=pwr9   -gencode arch=compute_70,code=compute_70
-o cublas-example.o -c
/usr/local/cuda-9.2/bin/nvcc -ccbin xlC -Xcompiler -qtune=pwr9 -Xcompiler
-qarch=pwr9 -gencode arch=compute_30,code=compute_30 -o cublas-example
cublas-example.o  -lcublas

$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-9.2/lib64/
$ ./cublas-example
The DOT product is:
2500.22
```

7.6.3 OpenACC

As an option that is provided by the OpenPower foundation, IBM works with NVIDIA to enable the use of PGI compilers in POWER8 and POWER9 processor-based systems. As with the IBM XL, this compiler also can use most of the features that NVLINK technology provides to offload code to the Tesla GPUs.

The offload through the PGI compiler is performed by the OpenACC model, which was developed similar to OpenMP to ease the programming of heterogeneous CPU and GPU hybrid software for programmers. By using simple clauses such as **`#pragma acc kernels{}`**, the programmer can indicate loops where parallelism might be found during the time the compiler creates CUDA kernels under the hood for the programmer.

In this section, consider the integral of the following curve that leads to the value of pi if performed in the closed interval [0,1] that is shown in Table 7-8 on page 175:

$$\int_0^1 \frac{4}{1+x^2}dx = \pi \approx 3,1416$$

The simple addition of the **#pragma acc kernels{}** (the bolded line in Example 7-16) results in a significant performance increase.

Example 7-16 Parallel code for integration by using openacc

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include <sys/time.h>
#include <omp.h>

int32_t timer(struct timeval *, struct timeval *, struct timeval *);

int32_t main(int32_t argc, char* argv[])
{
        if(argc!=4)
        {
                printf("No number of steps OR number of blocks OR number of 
threads were provided\nAborting...\n");
                return -1;
        }

        uint32_t i, N, n_blocks, n_threads;
        double x, pi, step, sum;
        struct  timeval start_princ, finish_princ, diff_princ;

        sum = 0.0;
        N = atol(argv[1]);
        n_blocks = atol(argv[2]);
        n_threads = atol(argv[3]);
        step = 1.0 / ((double) N);

        gettimeofday(&start_princ,NULL);
                #pragma acc kernels
                for(i=0; i<N; i++)
                {
                        x = (i + 0.5) * step;
                        sum += 4.0 / ( 1.0 + x * x);
                }
                pi = step * sum * n_blocks * n_threads;
        gettimeofday(&finish_princ,NULL);

        timer(&diff_princ, &finish_princ, &start_princ);
        printf("PI: %.15f\nExecution Time   = %ld.%06ld s\n", pi, 
diff_princ.tv_sec, diff_princ.tv_usec);
        fprintf(stderr,"%ld.%06ld\n", diff_princ.tv_sec, diff_princ.tv_usec);
        return 0;
}

int32_t timer(struct timeval *result, struct timeval *t2, struct timeval *t1)
{
        int32_t diff = (t2->tv_usec + 1000000 * t2->tv_sec) - (t1->tv_usec +
1000000 * t1->tv_sec);
        result->tv_sec = diff / 1000000;
        result->tv_usec = diff % 1000000;
        return (diff<0);
}
```

Example 7-16 on page 201 is an excellent starting ground to parallelize software through OpenACC. However, to achieve better performance the programmer must fine-tune the **`#pragma acc`** calls. Therefore, consider the effect of sharing the integral loop among multiple blocks and threads, as shown in Example 7-17.

Example 7-17 OpenACC parallelization in n_blocks with n_threads each

```
--- acc_pi_simple.c     2018-10-15 18:12:44.158132792 -0400
+++ acc_pi.c    2018-10-15 18:13:08.187224187 -0400
@@ -26,7 +26,7 @@
         step = 1.0 / ((double) N);
         gettimeofday(&start_princ,NULL);
-                #pragma acc kernels
+                #pragma acc parallel loop reduction(+:sum) device_type(nvidia)
vector_length(n_threads) gang worker num_workers(n_blocks)
                 for(i=0; i<N; i++)
                 {
                         x = (i + 0.5) * step;
```

In Example 7-17, you divide the for loop between blocks of several threads. Also, you must add a reduction clause to add all individual computation between the blocks at the end of the execution.

The compilation command for our configuration that uses a Tesla V100 is shown in Example 7-18. The **`-acc`** option tells the compiler to process the source recognizing **`#pragma acc`** directives. At the same time, **`-Minfo`** tells the compiler to share information about the optimization during the compilation process. Also, **`all`** stands for `accel,inline,ipa,loop,lre,mp,opt,par,unified,vect` and the intensity asks for the computing loop information.

Example 7-18 OpenACC compilation example

```
$ pgcc -acc -Minfo=all,intensity -ta=tesla:cc70 -o acc_pi acc_pi.c -lm
main:
     29, Accelerator kernel generated
         Generating Tesla code
         30, #pragma acc loop gang, worker(n_blocks), vector(n_threads) blockIdx.x
threadIdx.y threadIdx.x */
             Generating reduction(+:sum)
     29, Generating implicit copy(sum)
```

In addition, the **`-ta`** flag chooses the target accelerator, which can be set to the host CPU by using the option **`multicore`**, and therefore run in the 160 cores. However, in our example, it is set to our Tesla V100 GPU that has compute capacity 70. You can find more information by running the **`pgaccelinfo`** command. To find the correct **`cc gpu`** flag, run the command that is shown in Example 7-19.

Example 7-19 Finding more information about the NVIDIA board

```
$ pgaccelinfo
CUDA Driver Version:           9020
NVRM version:                  NVIDIA UNIX ppc64le Kernel Module  396.37  Tue Jun 
12 17:21:36 PDT 2018
Device Number:                 0
Device Name:                   Tesla V100-SXM2-16GB
Device Revision Number:        7.0
Global Memory Size:            16106127360
Number of Multiprocessors:     80
Concurrent Copy and Execution: Yes
Total Constant Memory:         65536
Total Shared Memory per Block: 49152
Registers per Block:           65536
Warp Size:                     32
Maximum Threads per Block:     1024
Maximum Block Dimensions:      1024, 1024, 64
Maximum Grid Dimensions:       2147483647 x 65535 x 65535
Maximum Memory Pitch:          2147483647B
Texture Alignment:             512B
Clock Rate:                    1530 MHz
...
...
... Output Omited
...
PGI Default Target:            -ta=tesla:cc70
```

You can also use the command that is shown in Example 7-20.

Example 7-20 Finding the compute capacity of your board

```
$ pgaccelinfo |grep -m 1 "PGI Default Target"
PGI Default Target:            -ta=tesla:cc70
```

For more information about running OpenACC parallelization and the performance that is achieved in Example 7-17 on page 202, see 8.2.2, “OpenACC execution and scalability” on page 240.

For more information about how to use and install `openacc`, see the following resources:

- Resources page of the OpenACC website
- OpenACC Courses page

7.6.4 IBM XL C/C++ and Fortran offloading

Another useful feature of the Power AC922 server is using OpenMP directives to perform GPU offloading through IBM XL C/C++ programs. The combination of the POWER processors with the NVIDIA GPUs provides a robust platform for heterogeneous HPC that can run several scalable technical computing workloads efficiently. The computational capability is built on top of massively parallel and multithreaded cores within the NVIDIA GPUs and the IBM POWER processors.

For more information about the power of this feature, see *Optimization and Programming Guide for Little Endian Distributions*.

IBM XL Fortran and IBM XL C/C++ V16.1.0 support the OpenMP API V4.5 specification. Compute-intensive parts of an application and associated data can be offloaded to the NVIDIA GPUs by using the supported OpenMP preprocessor directives device constructs to control parallel processing.

Note: The **`pragma`** calls take effect only when parallelization is enabled with the **`-qsmp`** compiler option. Also, the compilation also must include **`-qoffload`** to offload.

For example, you can use the **`omp`** target directive to define a target region, which is a block of computation that operates within a distinct data environment and is intended to be offloaded into a parallel computation device during execution.

Table 7-20 lists some of the supported OpenMP 4.5 **`pragma`** clauses.

Table 7-20 Supported OpenMP 4.5 pragma clauses

IBM XL Fortran	IBM XL C/C++	Description
`TARGET DATA`	`omp target data`	The **`omp target data`** directive maps variables to a device data environment and defines the lexical scope of the data environment that is created. The **`omp target data`** directive can reduce data copies to and from the offloading device when multiple target regions are using the same data.
`TARGET ENTER DATA`	`omp target enter data`	The **`omp target enter data`** directive maps variables to a device data environment. The **`omp target enter data`** directive can reduce data copies to and from the offloading device when multiple target regions are using the same data and when the lexical scope requirement of the **`omp target dataconstruct`** is not appropriate for the application.

IBM XL Fortran	IBM XL C/C++	Description (Fort.)
`TARGET EXIT DATA`	`omp target exit data`	The **`omp target exit data`** directive unmaps variables from a device data environment. The **`omp target exit data`** directive can limit the amount of device memory when you use the **`omp target enter data`** construct to map items to the device data environment.
`TARGET`	`omp target`	The **`omp target`** directive instructs the compiler to generate a target task, that is, to map variables to a device data environment and to run the enclosed block of code on that device. Use the **`omp target`** directive to define a target region, which is a block of computation that operates within a distinct data environment and is intended to be offloaded onto a parallel computation device during execution.
`TARGET UPDATE`	`omp target update`	The **`omp target update`** directive makes the list items in the device data environment consistent with the original list items by copying data between the host and the device. The direction of data copying is specified by motion-type.
`DECLARE TARGET`	`omp declare target`	The **`omp declare target`** directive specifies that variables and functions are mapped to a device so that these variables and functions can be accessed or run on the device.
`TEAMS`	`omp teams`	The **`omp teams`** directive creates a collection of thread teams. The master thread of each team runs the teams.
`DISTRIBUTE`	`omp distribute`	The **`omp distribute`** parallel for directive runs a loop by using multiple teams where each team typically consists of several threads. The loop iterations are distributed across the teams in chunks in round-robin fashion.
`DISTRIBUTE PARALLEL DO`	`omp distribute parallel for`	The **`omp distribute parallel for`** directive runs a loop by using multiple teams where each team typically consists of several threads. The loop iterations are distributed across the teams in chunks in round-robin fashion.

For more information about all supported IBM XL OpenMP clauses, see *Compiler Reference for Little Endian Distributors*.

To test the scalability of this feature, use the following dgemm example:

$$C = \alpha[A] \cdot [B] + \beta[C]$$

A*lpha* and *beta* are real scalar values, and *A*, *B*, and *C* are matrixes of conforming shape.

Example 7-21 presents a column major order implementation of the dgemm algorithm. This code was created to present the compilation and execution of a real example of an algorithm by using IBM XL C offloading on NVIDIA GPUs. In addition, it aims to present a comparison between a hand implementation against the advantages of using the vast API of IBM Parallel ESSL, such as the implementations that are described in 7.4.2, "Using GPUs with the IBM Parallel ESSL" on page 182. Finally, this implementation was designed to ease the execution of a scalability curve, which is created for a broader perspective of the performance gain that is achieved by offloading the code to the GPU.

Example 7-21 Column major order implementation of dgemm to test scalability of IBM XL C offloading

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <omp.h>

#define interval 0.5
#define verbose 0

double  float_rand();
void    timer(struct timeval *, struct timeval *, struct timeval *);
void    print_mat(double *mat, long N);

int main(int argc, char* argv[])
{
        if(argc!=4)
        {
                printf("No matrix dimension OR number of blocks OR number of
threads were provided\nAborting...\n");
                return -1;
        }

        struct timeval diff, start, end;
        long i, j, k, N, n_blocks, n_threads;
        double *a, *b, *c, *tmp, alpha, beta, sum;
        double flop, exec_time;

        srand((unsigned int)time(NULL));

        N = atol(argv[1]);
        n_blocks = atol(argv[2]);
        n_threads = atol(argv[3]);

        alpha = ((double)1.3);
        beta  = ((double)2.4);
        flop  = (double)(N*N*(2*(N-1)));

        a = (double*) malloc(N * N * sizeof(double));
        b = (double*) malloc(N * N * sizeof(double));
        c = (double*) malloc(N * N * sizeof(double));
        tmp = (double*) malloc(N * N * sizeof(double));

   for(i=0;i<N;i++)
        {
                for(j=0;j<N;j++)
                {
                        a[ (i) * N + (j) ] = float_rand();
                        b[ (i) * N + (j) ] = float_rand();
                        c[ (i) * N + (j) ] = float_rand();
                }
        }

        if(verbose)
        {
                printf("A\n");   print_mat(a,N);
                printf("\nB\n"); print_mat(b,N);
                printf("\nC\n"); print_mat(c,N);
        }

        gettimeofday(&start,NULL);
                #pragma omp target map(to: a[0:N*N], b[0:N*N], c[0:N*N]) map(from:
tmp[0:N*N])
                #pragma omp teams num_teams(n_blocks) thread_limit(n_threads)
                #pragma omp distribute parallel for private(sum)
                for(i=0;i<N;i++)
                {
                        for(j=0;j<N;j++)
                        {
                                sum = 0.0;
                                for(k=0;k<N;k++)
                                {
                                        sum += a[ (i)*N + (k) ] * b[ (k)*N + (j)
];
                                }
                               tmp[ (i)*N + (j) ] = alpha * sum + beta * c[ (i)*N
+ (j) ];
                        }
                }
        gettimeofday(&end,NULL);

        memcpy(c, tmp, N * N * sizeof(double));

        if(verbose)
        {
                printf("\nC = alpha * A * B + beta * C\n");
                print_mat(c,N);
        }

        timer(&diff, &end, &start);
        exec_time = diff.tv_sec + 0.000001 * diff.tv_usec;
        printf("\nExecution time = %lf s, %lf MFlops\n\n", exec_time, (flop /
exec_time));
        fprintf(stderr, "%.6lf,%.6lf\n", exec_time, (flop / exec_time));

        free(a); free(b); free(c);

        return 0;
}

double float_rand()
{
        double a = ((double)-1.0) * interval;
        double b = interval;
        return b + ((double)rand() / ( RAND_MAX / (a-b) ) ) ;
}

void timer(struct timeval *result, struct timeval *t2, struct timeval *t1)
{
        long diff = (t2->tv_usec + 1000000 * t2->tv_sec) - (t1->tv_usec + 1000000
* t1->tv_sec);
        result->tv_sec = diff / 1000000;
        result->tv_usec = diff % 1000000;
}

void print_mat(double *mat, long N)
{
        long i, j;

        for(i=0;i<N;i++)
        {
                for(j=0;j<N;j++)
                {
                        ( mat[ (i)*N + (j) ] >= 0.0) ? printf(" %f ",mat[ (i)*N +
(j) ]) : printf("%f ",mat[ (i)*N + (j) ]);
                }
                printf("\n");
        }
}
```

Example 7-22 shows how to compile the source code.

Example 7-22 How to compile a program that uses IBM XL offloading

```
$ xlc_r -O3 -Wall -o dgemm_xlc dgemm_xlc.c -qsmp=omp -qoffload -mcpu=pwr9
-mtune=pwr9
```

Depending on the computation logic, some algorithms can perform better when column major order is used than line major order, and vice versa. This issue usually occurs because of the increase of the cache hit ratio, but one of these configurations can mask a false sharing situation.

Although Example 7-21 on page 206 dynamically allocates the matrixes and calculates the indexes of the matrix throughout all the code, the IBM ESSLs in Example 7-5 on page 177 performed this function in line 60 by selecting only one parameter at the dgemm function.

Therefore, it is important to study and use the highly optimized IBM ESSL API as much as possible.

Example 7-21 on page 206 also shows the syntax of copying dynamically allocated vectors to the GPU (normal vector uses the name of the variable). The data copy is performed on **`#pragma omp target`** call on the bolded line of Example 7-21 on page 206, where we use the **`map()`** function to copy A, B, and C into the GPU, and the `tmp` out of the GPU at the end of the calculation. This process avoids multiple copies at the end of each iteration of the outer for loop.

We then call the **`#pragma omp teams`** that are nested with the **`thread_limit`** to run our code in warps, as a CUDA code can. Finally, we use **`#pragma omp distribute`** to share the code according to our blocks of a specific number of threads.

For more information about all the features of IBM XL offloading, see Product documentation for XL C/C++ for Linux, V16.1.0, for Little Endian distributions.

For more information about OpenMP examples, see the OpenMP 4.5 documentation.

7.6.5 MPI programs with IBM Parallel Environment V2.3

Note: IBM Parallel Environment is being deprecated in favor IBM Spectrum Message Passing Interface (IBM Spectrum MPI), which is a lighter and high-performance implementation of the MPI standard.

For more information, see 7.6.9, "MPI programs that use IBM Spectrum MPI" on page 220.

The following sections remain in this book for completeness and migration purposes. These sections will be removed in future releases of this publication.

- 7.6.5, "MPI programs with IBM Parallel Environment V2.3" on page 209
- 7.6.6, "Hybrid MPI and CUDA programs with IBM Parallel Environment" on page 215
- 7.6.7, "OpenSHMEM programs in IBM Parallel Environment" on page 218
- 7.6.8, "Parallel Active Messaging Interface programs" on page 219

The MPI development and runtime environment that is provided by IBM Parallel Environment V2.3 includes the following general characteristics:

- Provides the implementation of the MPI V3.0 standard based on the open source MPICH project.
- The MPI library uses the PAMI protocol as a common transport layer.
- Supports the MPI application in C, C++, and Fortran.
- Supports 64-bit applications only.
- Supports GNU and IBM XL compilers.
- MPI operations can be carried out on main or user-space threads.
- The I/O component (also known as Message Passing Interface I/O (MPI-IO)) is an implementation of ROMIO[3] that is provided by MPICH V3.1.2.
- Provides a CUDA-aware MPI implementation.

[3] For more information about ROMIO, see ROMIO: A High-Performance Portable MPI-IO Implementation.

- Uses a shared memory mechanism for message transport between tasks on the same compute node. In contrast, the User Space communication subsystem, which provides direct access to a high-performance communication network by way of an InfiniBand adapter, is used for internode tasks.
- Allows message stripping, failover, and recovery on multiple or single (with some limitations) network configurations.
- Allows for dynamic process management.

This section introduces some MPI implementation aspects of IBM Parallel Environment Runtime Edition and general guidance about how to build parallel applications. For more information, see *IBM Parallel Environment Runtime Edition for Linux: MPI Programming Guide*.

Message Passing Interface API

The MPI implementation of IBM Parallel Environment is based on MPICH. For information about the MPI API, see the MPICH website.

Provided compilers

The compilers provide a set of compilation scripts that is used to build parallel applications that support GNU and IBM XL family compilers for C, C++, and Fortran.

C, C++, and Fortran applications are built by using **mpcc** (C), **mpCC** (C++), and **mpfort** (Fortran) compilation scripts that are linked with the threaded version of MPI and **poe** libraries by default. They also apply some instrumentation on the binary file so that **poe** is indirectly started to manage the parallel execution.

The **mpicc** (C), **mpicxx** (C++), **mpif77** (Fortran77), and **mpif90** (Fortran 90) compilation scripts are designed to build MPICH-based parallel applications. A program that is compiled with those scripts can be run through **poe**.

Example 7-23 shows the **mpicc** command compiling an MPI C application by using the GCC compiler.

Example 7-23 IBM Parallel Environment Runtime Edition mpicc command to compile an MPI C program

```
$ export PATH=/opt/ibmhpc/pecurrent/base/bin:$PATH
$ mpicc -compiler gnu -O3 -mcpu=power9 -mtune=power9 -o myApp main.c
```

To display the command that is run to compile the application, use the **-show** option. In Example 7-23, the **mpicc -show** command produces the following output:

```
$ mpicc -show -compiler gnu -O3 -mcpu=power8 -mtune=power8 -o myApp main.c
/usr/bin/gcc -Xlinker --no-as-needed   -O3 -mcpu=power9 -mtune=power9 -o  myApp
main.c  -m64 -D__64BIT__ -Xlinker --allow-shlib-undefined -Xlinker
--enable-new-dtags -Xlinker -rpath -Xlinker /opt/ibmhpc/pecurrent/mpich/gnu/lib64
-I/opt/ibmhpc/pecurrent/mpich/gnu/include64 -I/opt/ibmhpc/pecurrent/base/include
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64 -lmpi
```

All of these compilation scripts use the IBM XL compilers unless the **MP_COMPILER** variable or **-compiler** option is set, which instructs the compilers to use another compiler. You can use the **gnu** or **xl** option values to use GNU or IBM XL compilers. For third-party compilers, use the fully qualified path (for example, `MP_COMPILER=/opt/at12.0/bin/gcc`).

Note: The compilation scripts that are provided by the latest version of the IBM Parallel Environment Runtime Edition are installed in the `/opt/ibmhpc/pecurrent/base/bin` directory.

Details of the MPI-IO implementation

The ROMIO implementation of IBM Parallel Environment Runtime Edition is configured to use the IBM Spectrum Scale file system, which delivers high-performance I/O operations. Some environment variables are also introduced so that users can control the behavior of some operations, such as collective aggregations.

Note: Although the configuration also supports NFS and POSIX compliance file systems, some limitations can apply.

The file system detection mechanism uses system calls unless the parallel file system is set by using the **`ROMIO_FSTYPE_FORCE`** environment variable. By passing hints to ROMIO, you may make many changes to the default configuration of MPI-IO, which are set in the **`ROMIO_HINTS`** environment variable.

Local rank property

The MPI implementation that IBM Parallel Environment provides has a mechanism to determine the rank of a task among others tasks that are running in the same machine. This ranking is also known as the *task local rank*.

You can use the read-only **`MP_COMM_WORLD_LOCAL_RANK`** variable to obtain the local rank. Each task reads its local attributed rank and is available through the runtime environment.

Example 7-24 shows an MPI C program that reads the **`MP_COMM_WORLD_LOCAL_RANK`** variable and prints its value to standard output.

Example 7-24 Simple MPI C program that prints the task local rank

```
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>

int main(int argc, char* argv[])
{
   int world_rank, world_size, local_rank;
   char hostname[255];

   MPI_Init(&argc, &argv);

   gethostname(hostname, 255);
   MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);
   local_rank = atoi(getenv("MP_COMM_WORLD_LOCAL_RANK"));
   printf("Task %d: running on node %s and local rank %d\n", world_rank, hostname, 
local_rank);

   MPI_Finalize();
   return EXIT_SUCCESS;
}
```

The output of the program in Example 7-24 on page 211 is shown in Example 7-25.

Example 7-25 Output of a simple MPI C program that prints the local rank

```
$ mpcc main.c
$ MP_RESD=poe MP_PROCS=5 ./a.out
Task 0: running on node xcat-mn.xcat-cluster and local rank 0
Task 1: running on node xcat-mn.xcat-cluster and local rank 1
Task 4: running on node xcat-mn.xcat-cluster and local rank 4
Task 2: running on node xcat-mn.xcat-cluster and local rank 2
Task 3: running on node xcat-mn.xcat-cluster and local rank 3
```

Switching MPI configurations by using environment modules

The process of compiling and running a parallel application with IBM Parallel Environment requires setting several environment variables. To ease this task, there are some available predefined profiles that use environment modules[4] to change the system's variables. Since IBM Parallel Environment V2.3, the following development profiles for MPI applications are provided:

- `perf`: Compile the MPI application with IBM XL and set it to run in development mode with minimal error checking.
- `debug`: Compile the MPI application with IBM XL and set it to run in development mode with the debug versions of the libraries.
- `trace`: Compile the MPI application with IBM XL and set it to run in development mode with the trace libraries.

The environment module command (**`module`**) can be installed in the system. For Red Hat Enterprise Linux 7.3, it can be installed by running the following command:

```
# yum install environment-modules
```

After the module command is installed in the system, you can list all the available modules, add the modules that are provided by IBM Parallel Environment, and load modules, as shown in Example 7-26.

Example 7-26 Using modules to set an environment to build and run MPI applications

```
$ module avail

-----------------------------------------------------------------------
/usr/share/Modules/modulefiles
-----------------------------------------------------------------------
dot         module-git  module-info modules     null        use.own
$ module use -a /opt/ibmhpc/pecurrent/base/module
$ echo $MODULEPATH
/usr/share/Modules/modulefiles:/etc/modulefiles:/opt/ibmhpc/pecurrent/base/module
$ module avail

-----------------------------------------------------------------------
/usr/share/Modules/modulefiles
-----------------------------------------------------------------------
dot         module-git  module-info modules     null        use.own
```

[4] For more information about Linux environment modules, see Modules - Software Environment Management.

```
----------------------------------------------------------------------
/opt/ibmhpc/pecurrent/base/module
-----------------------------------------------------------------------
pe2300.xl.debug pe2300.xl.perf  pe2300.xl.trace
$ module whatis pe2300.xl.debug
pe2300.xl.debug      : Adds PE environment variables for xl compiler and debug
develop mode to user environment.

$ module whatis pe2300.xl.perf
pe2300.xl.perf       : Adds PE environment variables for xl compiler and
performance develop mode to user environment.

$ module whatis pe2300.xl.trace
pe2300.xl.trace      : Adds PE environment variables for xl compiler and trace
develop mode to user environment.

$ env | grep MP
$ module load pe2300.xl.perf
  Adds these PE settings into your environment:

  MP_COMPILER=xl
  MP_EUIDEVELOP=min
  MP_MPILIB=mpich
  MP_MSG_API=mpi
  MP_CONFIG=2300
$ env | grep MP
MP_EUIDEVELOP=min
MP_CONFIG=2300
MP_MSG_API=mpi
MP_MPILIB=mpich
MP_COMPILER=xl
$ module load pe2300.xl.debug
  Adds these PE settings into your environment:

  MP_COMPILER=xl
  MP_EUIDEVELOP=debug
  MP_MPILIB=mpich
  MP_MSG_API=mpi
  MP_CONFIG=2300
$ env | grep MP
MP_EUIDEVELOP=debug
MP_CONFIG=2300
MP_MSG_API=mpi
MP_MPILIB=mpich
MP_COMPILER=xl
```

Simulating different SMT modes

The example cluster that is shown in Figure 7-4 contains two nodes (Power AC922 systems), each of which has two sockets. One nonuniform memory access (NUMA) node corresponds to one socket and has 20 physical POWER9 cores inside it. The systems are configured with simultaneous multithreading (SMT)-4, which means that a physical core can split a job between four logical CPUs (threads).

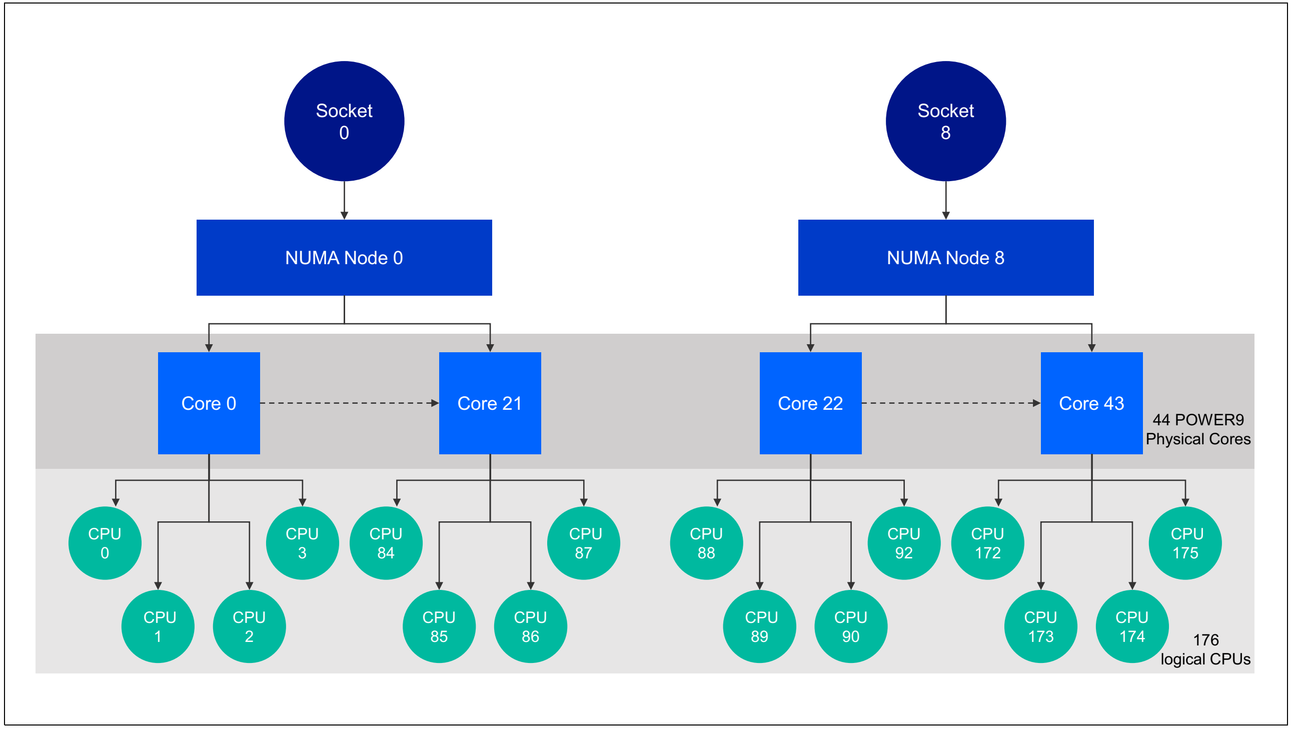


Figure 7-4 Structure of a node in a cluster

IBM Parallel Environment MPI provides run options or environment variables that help to simulate runs with different SMT modes. The following cases describe two ways where a job uses only one logical CPU from each POWER9 core (SMT-1 mode) and fully uses all 176 logical CPUs by using the code in Example 7-15 on page 200 with IBM Parallel ESSL calls:

- To configure the job to use only one logical CPU per POWER9 core, run the following command:

  ```
  MP_TASK_AFFINITY=core:1 MP_RESD=poe MP_PROCS=44 ./test_pdgemm
  ```

  `MP_TASK_AFFINITY=core:1` directs the IBM Parallel Environment MPI that each MPI task can use only one POWER9 core. Overall, the full cluster has 88 POWER9 cores (each node contains 44 cores), so a maximum of 88 MPI (`MP_PROCS=88`) tasks can be used for such run.

  Another possible solution to simulate SMT-1 mode is to take 20 MPI tasks with two POWER8 cores for each of them by running the following command:

  ```
  MP_TASK_AFFINITY=core:2 MP_RESD=poe MP_PROCS=44 ./test_pdgemm
  ```

- The following command shows how to use the cluster and all 176 logical CPUs per node:

  ```
  MP_TASK_AFFINITY=cpu:16 MP_RESD=poe MP_PROCS=22 ./test_pdgemm
  ```

  `MP_TASK_AFFINITY=cpu:16` means when you use the IBM Parallel Environment MPI that each MPI task can use only 16 logical CPUs. Each node in the cluster contains 176 logical CPUs, so 11 MPI tasks per node and 22 for the overall cluster can be used in this run.

If you want to change the CPU affinity variable and still use all logical CPUs, the number of MPI tasks must be changed. The following commands describe the alternative calls:

```
MP_TASK_AFFINITY=cpu:8 MP_RESD=poe MP_PROCS=44 ./test_pdgemm
MP_TASK_AFFINITY=cpu:4 MP_RESD=poe MP_PROCS=88 ./test_pdgemm
MP_TASK_AFFINITY=cpu:2 MP_RESD=poe MP_PROCS=176 ./test_pdgemm
```

7.6.6 Hybrid MPI and CUDA programs with IBM Parallel Environment

The IBM Parallel Environment compilers and runtime environment support building and running hybrid of MPI and CUDA programs.

Building the program

In this use case, you organize sources on separate files for MPI and CUDA codes, use different compilers to build the objects, and then link them by using the MPI compiler. Example 7-27 shows this procedure.

Example 7-27 Building hybrid MPI and CUDA programs

```
$ mpCC -o helloMPICuda_mpi.o -c helloMPICuda.cpp
$ nvcc -ccbin g++ -m64 -gencode arch=compute_70,code=sm_70 -o helloMPICuda.o -c
helloMPICuda.cu
$ mpCC -o helloMPICuda helloMPICuda_mpi.o helloMPICuda.o
-L/usr/local/cuda-9.2/lib64 -lcudart
```

You build the MPI source by using the **`mpCC`** script, and you use the **`nvcc`** compiler to build the CUDA code. Then, the object files are linked to the executable file and the CUDA runtime library. Implicitly, **`mpCC`** links the executable file to **`libmpi`** (MPI), **`libpami`** (PAMI), and **`libpoe`** (Parallel Operating Environment (POE)).

Source files that contain both MPI and CUDA code (the *spaghetti programming style*) can be compiled, but this style is not a good programming practice. In this case, you can compile the code by using **`nvcc`** (CUDA C compiler) and setting the MPI library and the headers as shown in the following example:

```
$ nvcc -I/opt/ibmhpc/pecurrent/mpich/gnu/include64/
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64/ -L/opt/ibmhpc/pecurrent/base/gnu/lib64/
-lmpi -lpami main.cu
```

CUDA-aware MPI support

The CUDA-aware MPI feature of IBM Parallel Environment grants tasks direct access to the GPU memory's buffers for message passing operations, which can improve significantly the application performance.

Note: The CUDA-aware MPI API was introduced in IBM Parallel Environment V2.3.

The CUDA development model has the following general steps:

1. Allocate data on the host (CPU) memory.
2. Allocate data on the device (GPU) memory.
3. Move the data from host to device memory.
4. Perform some computation (kernel) on that data on the device.
5. Move processed data from the device back to the host memory.

In the context of send-receive communication without the CUDA-aware MPI capability, the tasks cannot access the GPU memory. Therefore, step 5 on page 215 is required because the data must be on the host memory before it is sent. However, with CUDA-aware MPI the task accesses the portion of memory that is allocated in step 2 on page 215, which means that data cannot be staged into host memory (step 5 on page 215 is then optional).

Note: By default, CUDA-aware MPI is disabled on IBM Parallel Environment Runtime Edition. You can enable it by exporting the **`MP_CUDA_AWARE`** environment variable by setting `yes` (enable) or `no` (disable).

The code that is shown in Example 7-28 shows how the CUDA-aware feature can be used within a hybrid of CUDA and MPI programs. It is a simple MPI program that is meant to run two jobs where task 0 initializes an array, increments its values by one using the GPU computation, and then sends the result to task 1. In line 56, the call to the **`MPI_Send`** function uses the device buffer (allocated in line 40) directly.

Example 7-28 Simple CUDA-aware MPI program

```
1 #include<cuda_runtime.h>
2 #include<mpi.h>
3 #include<stdio.h>
4 #include<stdlib.h>
5 #include<assert.h>
6
7  __global__ void vecIncKernel(int* vec, int size) {
8     int tid = blockDim.x * blockIdx.x + threadIdx.x;
9     if(tid < size) {
10        vec[tid] += 1;
11     }
12  }
13
14 #define ARRAY_ELEM 1024
15 #define ARRAY_ELEM_INIT 555
16
17 int main(int argc, char* argv[]) {
18   int taskid, numtasks, tag=0;
19   MPI_Status status;
20   int array_size = sizeof(int) * ARRAY_ELEM;
21   int threadsPerBlock = 32;
22   int blockSize = ceil(ARRAY_ELEM/threadsPerBlock);
23
24   MPI_Init(&argc, &argv);
25   MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
26   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
27
28   if(numtasks != 2) {
29     printf("This program must run only 2 tasks\n");
30   }
31   /*
32    * Task 0: initialize an array, increment its values by 1,
33    *  and send it to Task 1.
34    */
35   if(taskid == 0) {
36       int *vSend = (int*) malloc(array_size);
37       for(int i=0; i < ARRAY_ELEM; i++)
```

```
        vSend[i]=ARRAY_ELEM_INIT;
      int *vDev;
      if(cudaMalloc((void **)&vDev, array_size) == cudaSuccess) {
        cudaMemcpy(vDev,vSend, array_size,cudaMemcpyHostToDevice);
        vecIncKernel<<< blockSize, threadsPerBlock>>>(vDev, ARRAY_ELEM);
        cudaDeviceSynchronize();
      } else {
        printf("Failed to allocate memory on GPU device\n");
        MPI_Abort(MPI_COMM_WORLD, MPI_ERR_OTHER);
        exit(0);
      }
      if(strcmp(getenv("MP_CUDA_AWARE"),"yes") != 0) {
          printf("Cuda-aware MPI is disabled, MPI_Send will fail.\n");
      }
      /*
       * CUDA-AWARE MPI Send: using the buffer allocated in GPU device.
       *  Do not need to transfer data back to host memory.
       */
      MPI_Send(vDev, ARRAY_ELEM, MPI_INT, 1, tag, MPI_COMM_WORLD);
  } else {
      /*
       * Task 1: receive array from Task 0 and verify its values
       *  are incremented by 1.
       */
      int *vRecv = (int*) malloc(array_size);
      MPI_Recv(vRecv, ARRAY_ELEM, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
      int expected = ARRAY_ELEM_INIT+1;
      for(int i=0; i < ARRAY_ELEM_INIT; i++) {
        assert(vRecv[i]==expected);
      }
  }

  MPI_Finalize();
  return 0;
}
```

The program that is shown in Example 7-28 on page 216 was compiled and run twice, as shown in Example 7-29. The first run enables the CUDA-aware MPI (see the line starting with `MP_CUDA_AWARE=yes`). Then, it runs with the feature disabled (the line starting with `MP_CUDA_AWARE=no`), which forces it to exit with a segmentation fault. In this situation, you can implement remediation by copying the buffer back to the host memory and by using it in the MPI message pass call.

Example 7-29 Compiling and running the simple CUDA-aware MPI program

```
$
LD_LIBRARY_PATH=/opt/ibmhpc/pecurrent/mpich/gnu/lib64/:/opt/ibmhpc/pecurrent/base/
gnu/lib64/:$LD_LIBRARY_PATH
$ nvcc -I/opt/ibmhpc/pecurrent/mpich/gnu/include64/
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64/ -L/opt/ibmhpc/pecurrent/base/gnu/lib64/
-lmpi -lpami main.cu
$ MP_CUDA_AWARE=yes MP_RESD=poe MP_PROCS=2 poe ./a.out
$ MP_CUDA_AWARE=no MP_RESD=poe MP_PROCS=2 poe ./a.out
  5 Cuda-aware MPI is disabled, MPI_Send will fail.
  6 ERROR: 0031-250  task 0: Segmentation fault
```

As of IBM Parallel Environment V2.3, the CUDA-aware MPI is implemented for the following features:

- All two-sided (point-to-point) communication (blocking and non-blocking and intra- and intercommunication)
- All blocking intracommunicator collectives

However, the following features are not supported in the context of CUDA-aware MPI:

- One-sided communications
- One-sided synchronization calls
- Fabric Collective Accelerator (FCA)
- Non-blocking collective API calls
- Intercommunicator collective API
- The collective selection with the **`pami_tune`** command

Using MPI local rank to balance the use of GPU devices

You can implement a policy to grant shared access and load balance to GPU devices among local tasks (running on the same compute node) by using the local rank feature that is available in the IBM Parallel Environment MPI. For example, one algorithm can limit the number of tasks that use GPUs to avoid oversubscription.

For more information about the IBM Parallel Environment local rank, see 7.6.5, “MPI programs with IBM Parallel Environment V2.3” on page 209.

Starting concurrent CUDA kernels from multiple tasks

There are some considerations about shared use of GPUs with multiple MPI tasks. For more information, see 9.5.2, “CUDA Multi-Process Service” on page 292.

7.6.7 OpenSHMEM programs in IBM Parallel Environment

OpenSHMEM[5] provides a specification API and reference implementation for the Partitioned Global Address Space (PGAS) parallel programming model, which abstracts the concept of global shared memory on processes that are distributed across different address spaces. The OpenSHMEM implementation uses a communication library that often uses Remote Direct Memory Access (RDMA) techniques.

IBM Parallel Environment provides an OpenSHMEM runtime library (`libshmem.so`), compiler scripts, and tools that enable developing and running OpenSHMEM programs. The PAMI library is used for communication among processing elements of the PGAS program, and it can use the RDMA capabilities of the InfiniBand interconnect to improve performance.

As of IBM Parallel Environment V2.3, support is available only for parallel programs that are written in C and C++. The IBM Parallel Environment implementation is based on the OpenSHMEM API specification V1.2[6] with some minor deviations. For more information about supported and unsupported routines, see the OpenSHMEM API support page.

The OpenSHMEM tools are installed in the `/opt/ibmhpc/pecurrent/base/bin` directory. The compile scripts are **`oshcc`** (C) and **`oshCC`** (C++). The **`oshrun`** script runs programs, although you can use **`poe`** as an alternative.

[5] For more information about OpenSHMEM, see OpenSHEM.
[6] For the OpenSHMEM V1.2 specification document, see OpenSHMEM Application Programming Interface.

Example 7-30 shows the OpenSHMEM program. The API can be used only after a shared memory section is initialized (line 8). A symmetric memory area (the **basket** variable that is declared in line 6) is written by all processing elements (the call to `shmem_int_put` in line 14) on IBM Parallel Environment zero. Then, all of the elements read the variable from IBM Parallel Environment zero (the call to `shmem_int_get` in line 15) to print its value.

Example 7-30 Simple OpenSHMEM program

```
 1 #include<shmem.h>
 2 #include<stdio.h>
 3 #include<stdlib.h>
 4
 5   int main() {
 6     static int basket; // Global shared (symetric) variable
 7     int cents_p, cents_g; // Processing element's local variable
 8     shmem_init(); // Initialize SHMEM
 9     int my_pe = shmem_my_pe(); // Processing element's number
10     //printf("Hello, I am PE %d\n", my_pe);
11     cents_p = rand()%10;
12     shmem_barrier_all();
13     if(my_pe != 0)
14       shmem_int_put(&basket, &cents_p, 1, 0);
15     shmem_int_get(&cents_g, &basket, 1, 0);
16     printf("Hello, I am PE %d. I put %d cents but I get %d\n", my_pe, cents_p, 
cents_g);
17     shmem_finalize(); // Finalize SHMEM
18
19     return 0;
20   }
```

The source code that is shown in Example 7-30 can be compiled by using the **oshcc** script, as shown in Example 7-31. It is a common practice that the name of an OpenSHMEM executable file includes the suffix *.x*.

Example 7-31 Compiling an OpenSHMEM program

```
$ oshcc -O3 hellosh.c -o hellosh.x
$ ls
hellosh.c  hellosh.x
```

By default, **oshcc** or **oshCC** compiles the application with **xlc** or **xlC** unless it is not installed or the **MP_COMPILER** environment variable is set to use **gcc** or **g++**.

Start the program by using the **oshrun** script. Alternatively, you can use **poe** directly. For more information about running OpenSHMEM programs, see 8.3.3, “Running OpenSHMEM programs” on page 246.

7.6.8 Parallel Active Messaging Interface programs

PAMI is a low-level protocol that is the foundation of communications on MPI and OpenSHMEM implementations of the IBM Parallel Environment Runtime Edition. It provides an API for programs to access PAMI capabilities, such as collective communications and RDMA that use InfiniBand Host Channel Adapters (HCAs).

Many pieces of sample code that can be used to demonstrate PAMI subroutines are included with IBM Parallel Environment Runtime Edition and are available in the `/opt/ibmhpc/pecurrent/ppe.samples/pami` folder. Example 7-32 shows how to build the PAMI samples and run a program (`alltoall.exe`) that uses the all-to-all communication functions.

Example 7-32 Building and running PAMI sample code

```
$ cp -r /opt/ibmhpc/pecurrent/ppe.samples/pami .
$ cd pami/
$ make
$ cd pami_samples
$ vi host.list
$ MP_RESD=poe MP_PROCS=4 ./coll/alltoall.exe
# Context: 0
# Alltoall Bandwidth Test(size:4) 0x1000e400, protocol: I1:Alltoall:P2P:P2P
# Size(bytes)  iteration    bytes/sec      usec
# -----------      -----------    -----------    ---------
           1             100       413223.1         2.42
           2             100       840336.1         2.38
           4             100      1659751.0         2.41
           8             100      3347280.3         2.39
          16             100      6722689.1         2.38
          32             100     13502109.7         2.37
          64             100     26778242.7         2.39
         128             100     53112033.2         2.41
         256             100     81012658.2         3.16
         512             100    171812080.5         2.98
        1024             100    320000000.0         3.20
        2048             100    552021563.3         3.71
        4096             100    869639065.8         4.71
```

For more information, see *IBM Parallel Environment Runtime Edition for Linux: PAMI Programming Guide*.

7.6.9 MPI programs that use IBM Spectrum MPI

IBM Spectrum MPI V0-2.0.6 is a high-performance implementation of the MPI standard. It is widely used in the HPC industry for developing scalable and parallel applications in IBM Power Systems servers. IBM Spectrum MPI supports a broad range of industry-standard platforms, interconnects, and operating systems (OSs), which helps ensure that parallel applications can run almost anywhere.

IBM Spectrum MPI includes the following features:

- Portability: A parallel software developer can create one program in a small cluster and scale it through the interconnects in a large cluster. This feature reduces development time and effort.
- Network optimization: IBM Spectrum MPI supports various networks and interconnects.
- Collective optimization: Through the `libcollectives` library, IBM Spectrum MPI can enable GPU buffers and enhance performance and scalability. It provides advanced logic to determine the fastest algorithm for any given collective operation.
- GPU support: Provides built-in support for NVIDIA GPUs.

IBM Spectrum MPI is not ABI compatible. For example, it does not run with OpenMPI, Platform MPI, or IBM Parallel Environment Runtime Edition. Multithread I/O also is not supported on IBM Spectrum MPI.

The IBM Spectrum MPI collectives component (`libcollectives`) does not support intercommunicators. For intercommunicator collective support, IBM Spectrum MPI relies on OpenMPI collective components.

For more information about the usage, installation, and limitations of IBM Spectrum MPI V10-1.0.2, see the following resources:

- *IBM Spectrum MPI Version 10 Release 2.0.6 User's Guide*
- *IBM Spectrum MPI Version 10 Release 2.0.6 Installation*
- *IBM Spectrum MPI Version 10 Release 2.0.6 Admin Guide*

7.6.10 Migrating from an IBM Parallel Environment Runtime Edition environment to IBM Spectrum MPI

Table 7-21 lists the instructions for porting code from IBM Parallel Environment Runtime Edition to IBM Spectrum MPI.

Table 7-21 IBM Parallel Environment Runtime Edition tasks and IBM Spectrum MPI equivalents

Task	IBM Parallel Environment Runtime Edition method	IBM Spectrum MPI method
Running programs	`poe program [args] [options]`	`mpirun [options] program [args]`
Compiling programs	The following compiler commands: ▶ **`mpfort`**, **`mpic77`**, or **`mpif90`** ▶ **`mpcc`** or **`mpicc`** ▶ **`mpCC`**, **`mpic++`**, or **`mpicxx`** Alternatively, you can use the following environment variable setting: `MP_COMPILER = xl | gcc | nvcc`	The following compiler commands: ▶ **`mpfort`** ▶ **`mpicc`** ▶ **`mpiCC`**, **`mpic++`**, or **`mpicxx`** Alternatively, you can use the following environment variable settings: ▶ `OMPI_CC = xl | gcc` ▶ `OMPI_FC = xlf | gfortran` ▶ `OMPI_CXX = xlC | g++`
Determining a rank before *MPI_Init*.	The **`MP_CHILD`** environment variable	The **`OMPI_COMM_WORLD_RANK`** environment variable
Specifying the local rank	The **`MPI_COMM_WORLD_LOCAL_RANK`** environment variable	The **`OMPI_COMM_WORLD_LOCAL_RANK`** environment variable
Setting affinity	The environment variables: ▶ **`MP_TASK_AFFINITY = cpu`** ▶ **`MP_TASK_AFFINITY = core`** ▶ **`MP_TASK_AFFINITY = mcm`** ▶ **`MP_TASK_AFFINITY = cpu:n`** ▶ **`MP_TASK_AFFINITY = core:n`** ▶ **`MP_TASK_AFFINITY = 1`**	The **`mpirun`** options: ▶ **`-aff width:hwthread`** ▶ **`-aff width:core`** ▶ **`-aff width:numa`** ▶ **`--map-by ppr:$MP_TASKS_PER_NODE:node:pe=N --bind-to hwthread`** ▶ **`--map-by ppr:$MP_TASKS_PER_NODE:node:pe=N --bind-to core`** ▶ **`-aff none`**

Task	IBM Parallel Environment Runtime Edition method	IBM Spectrum MPI method
Setting CUDA-aware	The `MP_CUDA_AWARE` environment variable	The `mpirun -gpu` option
Setting FCA	The `MP_COLLECTIVE_OFFLOAD` environment variable	The `mpirun -FCA` and `-fca` options
Setting RDMA	The following environment variables: ▶ `MP_USE_BULK_XFER` ▶ `MP_BULK_MIN_MSG_SIZE`	Use the RDMA default when `MSG_SIZE` is greater than 64.
Controlling the level of the debug messages	The `MP_INFOLEVEL` environment variable	The `mpirun -d` option
Setting STDIO	The following environment variables: ▶ `MP_STDINMODE` ▶ `MP_STOUTMODE` ▶ `MP_LABELIO`	The `mpirun -stdio *` options
Specifying the number of tasks	The `MP_PROCS` environment variable	The `mpirun -np *` option
Specifying a host list file	The `MP_HOSTFILE` environment variable	The `mpirun -hostfile *` option

For more information about migration, see *IBM Spectrum MPI Version 10 Release 1.0.2 User's Guide*.

7.6.11 Using IBM Spectrum MPI

IBM Spectrum MPI is an implementation of OpenMPI, which makes the source code structure similar to what you find in OpenMPI. For example, this section describes the six most-used procedures of IBM Spectrum MPI, as listed in Table 7-22.

Table 7-22 IBM Spectrum most used procedures

IBM Spectrum MPI procedure	Function
`MPI_Init()`	Starts the MPI function in a program.
`MPI_Comm_rank()`	The ID of the MPI process that is running.
`MPI_Comm_size()`	The total number of MPI processes that were created.
`MPI_Send()`	Sends data to another MPI process.
`MPI_Recv()`	Receives data from another MPI process.
`MPI_Finalize()`	Ends MPI functions in a program.

To demonstrate these procedures, Example 7-33 shows the implementation of a simple parallel algorithm that calculates the well-known integration of a curve following the trapezoidal rule, which is mathematically described as the discretization of an integral, as shown in the following examples:

$$\int_a^b f(x)\,dx \approx g(x)$$

$$h = \frac{b-a}{N}$$

$$g(x) \approx \frac{h}{2} \sum_{k=0}^{N-1} (f(x_k) + f(x_{k+1}))$$

$$g(x) \approx \frac{h}{2}[f(x_0) + 2f(x_1) + \ldots + 2f(x_{N-1}) + f(x_N)] \approx \frac{h}{2}(f(x_0) + f(x_N)) + h \sum_{k=1}^{N-1} f(x_k)$$

$$x_k = a + hk, k = 0, 1, \ldots, N$$

The trapezoidal rule creates infinitesimal trapezoids under a curve, and through their area summation, the integral value of the curve in a close interval can be estimated. The bigger the number of trapezoids, the more accurate the result of the estimation is.

Why the interest in this section? Because of the mathematical properties of addition and association, you can split this integral even in p processes (for simplicity, an even share between the number of processes and trapezoids). Consider the IBM Spectrum MPI code that is shown in Example 7-33.

Example 7-33 IBM Spectrum MPI code for a trapezoidal integration

```
#include <stdio.h>
#include <mpi.h>

float g(float x)
{
        return x*x;
}

float integral(float a, float b, int n, float h)
{
        int i;

        float x;
        float local_sum;

        x = a;
        local_sum = ( (g(a) + g(b)) / 2.0 );

        for (i = 1; i <= n-1; i++)
        {
                x += h;
                local_sum += g(x);
        }

        return local_sum * h;
}

int main(int argc, char *argv[])
{
        int     src;            // Process id of local integral calculations
        int     dst = 0;        // Process id of father that sums local calcs
        int     tag = 0;        // MPI tag value for messages. Not used here.
        int     np;             // Number of processes
        int     p_rank;         // Process rank

        float   a = -3.0;       // Left interval limit
        float   b =  3.0;       // Right interval limit
        int     n = 10000;      // Number of trapezoids

        float   local_a;        // Local Left interval limit
        float   local_b;        // Local Right interval limit
        int     local_n;        // Local Number of trapezoids

        float   h;              // Trapezoid length of base
        float   local_sum;      // Local Integral per process
        float   total_sum;      // Total local_sum of whole interval

        MPI_Status status;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &p_rank);
        MPI_Comm_size(MPI_COMM_WORLD, &np);

        h = (b-a) / n;          // Base length of trapezoids in all interval
        local_n = n / np;       // Chunk size of trapezoids for each process

        // Division of full interval using the rank of each process
        local_a = a + p_rank * local_n * h;
        local_b = local_a + local_n * h;
        local_sum = integral(local_a, local_b, local_n, h);

        if (p_rank == 0)
        {
                total_sum = local_sum;
                for(src = 1; src < np; src++)
                {
             MPI_Recv(&local_sum, 1, MPI_FLOAT, src, tag, MPI_COMM_WORLD, &status);
                        printf("MPI_Recv {%d <- %d} = %f\n", p_rank, src, local_sum);
                        total_sum = total_sum + local_sum;
                }
        }
```

```
71.        else
72.        {
73.                printf("MPI_Send {%d -> %d} = %f\n", p_rank, dst, local_sum);
74.                MPI_Send(&local_sum, 1, MPI_FLOAT, dst, tag, MPI_COMM_WORLD);
75.        }
76.
77.        if(p_rank == 0)
78.                printf("\nEstimate of local_sum of x^2 = %f\nInterval
   [%.3f,%.3f]\nUsing %d trapezoids\n\n", total_sum, a, b, n);
79.
80.        MPI_Finalize();
81.        return 0;
82.}
```

In line 4 in Example 7-33 on page 223, notice that the function of interest is g(x) = x^2. In lines 36 - 37, notice the interval that is being calculated in the closed interval of [-3;+3]. Also, in the main function, observe that the **MPI_Init()** function is used to start the MPI program. Immediately afterward, **MPI_Comm_rank()** and **MPI_Comm_size()** are called to fetch the current ID of each process that is created by **MPC_Init** and the total number of processes, respectively.

At run time, IBM Spectrum MPI fetches the number of processes from the environmental variable (or the CLI) and creates processes for multiple instances of this program. Each program features its unique rank ID, which is the target of interest in this code.

The **integral()** function sequentially calculates the trapezoid integral in a closed local interval section by following the formulas that were described in 7.6.11, “Using IBM Spectrum MPI” on page 222. However, in lines 57 - 59 in Example 7-33 on page 223, notice that the local variables that were created by combining the limits from the interval with the number of trapezoids and the rank of each MPI process created separate local intervals of the main interval. Therefore, you can calculate the trapezoid integral for each section independently by each MPI process.

What is still necessary is to join the results of each local integral? In line 61 in Example 7-33 on page 223, check whether you are running the main process (which have `p_rank` equal to `0`). If you are running the main process, iterate among all the MPI processes instances by using the **MPI_Recv** function to fetch their partial calculation and add it to `total_sum`. However, if you are not running the process, you are in a child process and use **MPI_Send** to send the partial calculations to the main process.

Describe what happens by using **MPI_Send()** and **MPI_Recv()**. Consider the following points:

- **MPI_Send** (**buf**, **count**, **type**, **dest**, **tag**, and **comm**) is a blocking MPI operation that uses the following arguments:
  - **buf** is the address of the buffer that is being sent.
  - **count** is the number of elements in the send buffer.
  - **type** is the **MPI_Datatype** of the elements in the buffer.
  - **dest** is the node rank ID of the destination process.
  - **tag** is the tag that the message can receive.
  - **comm** is the communicator of this message.
- **MPI_Recv** (**buf**, **count**, **type**, **src**, **tag**, **comm**, and **status**) is a blocking MPI operation that uses the following arguments:
  - **buf** is the address of the buffer where the process receives data.
  - **count** is the number of elements in the receiving buffer.
  - **type** is the **MPI_Datatype** of the elements in the buffer.

– **src** is the node rank ID of the process from where data is coming.
– **tag** is the tag that the message can receive.
– **comm** is the communicator of this message.
– **status** is the object status.

To use the *MPI_Datatype*, see Table 7-23.

Table 7-23 Using MPI_Datatype

MPI_Datatype	C type equivalent
MPI_C_BOOL	_Bool
MPI_CHAR	char (text)
MPI_UNSIGNED_CHAR	unsigned char (integer)
MPI_SIGNED_CHAR	signed char (integer)
MPI_INT	signed int
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_C_DOUBLE_COMPLEX	double _Complex

Finally, analyze the compilation and running of Example 7-33 on page 223. You use **mpicc** to compile and **mpirun** to run the parallel program, as shown in Example 7-34. On the **mpirun** call, you provide the number of processes that you want to run the code. Use the **-pami_noib** option because you use only one compute node to create the processes; no InfiniBand is connected between other nodes.

Note: Although we iterate in what looks like a sequential order, our output has no order.

Example 7-34 Compiling with IBM Spectrum MPI

```
$ /opt/ibm/spectrum_mpi/bin/mpicc -O2 trap.c -o trap.mpi

$ /opt/ibm/spectrum_mpi/bin/mpirun -np 5 -pami_noib ./trap.mpi

MPI_Send {3 -> 0} = 1.871934
MPI_Send {1 -> 0} = 1.872049
MPI_Recv {0 <- 1} = 1.872049
MPI_Recv {0 <- 2} = 0.144001
MPI_Recv {0 <- 3} = 1.871934
MPI_Recv {0 <- 4} = 7.056407

Estimate of local_sum of x^2 = 17.999882
Interval [-3.000,3.000]
Using 10000 trapezoids

MPI_Send {2 -> 0} = 0.144001
MPI_Send {4 -> 0} = 7.056407
```

If you perform an analytic integration of g(x) = x^2 in [-3;3], you find the value of 18.

8

Running parallel software, performance enhancement, and scalability testing

This chapter describes some techniques that are used to gain performance out of CPUs and graphics processing units (GPUs) in POWER9 processor-based systems.

This chapter provides the results of running some of the source code that was presented in Chapter 7, “Compilation, execution, and application development” on page 161. This code is run to demonstrate a few tips to you about how to enhance, test, and scale parallel programs in POWER9 processor-based systems.

The following topics are described in this chapter:

- Controlling the running of multithreaded applications
- Performance enhancements and scalability tests
- Using IBM Parallel Environment V2.3
- Using IBM Spectrum LSF
- Running tasks with IBM Spectrum MPI

8.1 Controlling the running of multithreaded applications

To gain performance for computing applications that run on a computing node, you typically use multiple threads, cores, or GPUs. The runtime environment has several options to support runtime fine-tuning of multithreaded programs.

This chapter describes how to control the Open Multi-Processing (OpenMP) workload by setting certain environment variables in IBM Xpertise Library (XL) C. Then, the chapter shows how to control OpenMP by integrating IBM Engineering and Scientific Subroutine Library (IBM ESSL).

This chapter also shows the tools that you can use to retrieve or set the affinity of a process at run time. This chapter also shows how to control the nonuniform memory access (NUMA) policy for processes and shared memory. GPUs are used as examples to explain the tools that can be used to retrieve or set the affinity of a process at run time. This chapter also shows how to control the NUMA policy for processes and shared memory[1].

8.1.1 Running OpenMP applications

A user can control the running of OpenMP applications by setting environment variables. This section describes only a subset of the environment variables that are especially important for technical computing workloads.

Note: The environment variables that are prefixed with **`OMP_`** are defined by the OpenMP standard. Other environment variables that are described in this section are specific to a particular compiler.

Distribution of a workload

The **`OMP_SCHEDULE`** environment variable controls the distribution of a workload among threads. If the workload items have uniform computational complexity, the static distribution fits well in most cases. If an application does not specify the scheduling policy internally, a user can set it to `static` at run time by exporting the environment variable, as shown in the following example:

```
export OMP_SCHEDULE="static"
```

However, the application can control the scheduling policy from the source code, as shown in the following example:

```
#pragma omp parallel for schedule(kind [,chunk size])
```

Table 8-1 shows the options for the OpenMP scheduler of a parallel section.

Table 8-1 Scheduling options for an OpenMP parallel section

Type of scheduling	Description
Static	Divides the loop as evenly as possible into chunks of work to the threads. By default, the chunk size is calculated by the **`loop_count`** divided by **`number_of_threads`**.
Dynamic	By using the internal work queue, this scheduling provides blocks of different chunk sizes to each thread at cost of extra processing. By default, the chunk size is equal to 1.

[1] This section is based on the content that originally appeared in Chapter 7, “Tuning and debugging applications”, of *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263.

Type of scheduling	Description
Guided	Works similar to dynamic scheduling, but it orders the chunks of work from the largest to the smallest, which can handle imbalances more properly. By default, the chunk size is calculated by **`loop_count`** divided by **`number_of_threads`**.
Auto	The compiler decides between static, dynamic, and guided.
Runtime	Use the **`OMP_SHEDULE`** environment variable. The advantage of this scheduler is that rewriting or recompiling the source code is unnecessary.

Specifying the number of OpenMP threads

OpenMP applications are often written so that they can create an arbitrary number of threads. In this case, the user sets the **`OMP_NUM_THREADS`** environment variable to specify the number of threads to create. For example, to run the application with 20 threads, use this command:

```
export OMP_NUM_THREADS=20
```

Again, the application can control the scheduling policy from the source code, as shown in the following example:

```
#pragma omp parallel ... num_threads(4)
```

Showing OpenMP data

The OpenMP 4.5 standard introduced the **`OMP_DISPLAY_ENV`** environment variable to display the OpenMP version and list the internal control variables (ICVs). The OpenMP run time prints data to the `stderr` output stream. If the user sets the value to `TRUE`, the OpenMP version number and initial values of ICVs are printed.

The `VERBOSE` value instructs the OpenMP run time to augment the output with the values of vendor-specific variables. If the value of this environment variable is set to `FALSE` or undefined, no information is printed. This variable is useful when you must be certain that the runtime environment is configured as expected at the moment that the program loads.

Placement of OpenMP threads

The POWER9 processor can handle multiple hardware threads simultaneously. With the increasing number of logical processors, the operating system (OS) kernel scheduler has more possibilities for automatic load balancing. For technical computing workloads, the fixed position of threads within the server is typically preferred.

The IDs of the logical processors are zero-based. Each logical processor has the same index, regardless of the simultaneous multithreading (SMT) mode. Logical processors are counted starting from the first core of the first socket. The first logical processor of the second core features the index 4. Number the logical processors of the second socket only after you finish the numbering of the logical processors of the first socket.

IBM XL compilers

For the OpenMP application that is compiled with IBM XL compilers, you must use the *XLSMPOPTS* environment variable to control thread placement. This environment variable includes many suboptions, and only a few of these options control thread binding. You can use the combination of **`startproc`** and **`stride`** suboptions, or the **`procs`** suboption. Consider the following points:

- The **`startproc`** suboption is used to specify the starting logical processor number for binding the first thread of an application. The **`stride`** suboption specifies the increment for the subsequent threads. For example, the following value of **`XLSMPOPTS`** instructs the OpenMP runtime environment to bind OpenMP threads to logical processors 80, 84, 88, and so on, up to the last available processor:

  ```
  export XLSMPOPTS=startproc=80:stride=4
  ```

- A user can also explicitly specify a list of logical processors to use for thread binding with the **`procs`** suboption. For example, to use only even-numbered logical processors of a processor's second core, specify the following value of **`XLSMPOPTS`**:

  ```
  export XLSMPOPTS=procs=8,10,12,14
  ```

Note: The **`startproc`**, **`stride`**, and **`procs`** suboptions were deprecated in favor of the **`OMP_PLACES`** environment variable. IBM intends to remove these suboptions in upcoming releases of the IBM XL compiler runtime environment.

For more information about the **`XLSMPOPTS`** environment variable, see the XLSMPOPTS section of the online manuals at the following websites:

- XL C/C++ for Linux
- XL Fortran for Linux

GNU Compiler Collection compilers

For the OpenMP application that is compiled with GNU Compiler Collection (GCC) compilers, use the **`GOMP_CPU_AFFINITY`** environment variable. Assign a list of the logical processors that you want to use to the **`GOMP_CPU_AFFINITY`** environment variable. The syntax and semantics are the same as with the **`procs`** suboption of the IBM XL compilers **`XLSMPOPTS`** environment variable. For more information about the **`GOMP_CPU_AFFINITY`** environment variable, see the corresponding section of the GCC manual.

Support for thread binding in recent versions of the OpenMP standard

The OpenMP 3.1 revision introduced the **`OMP_PROC_BIND`** environment variable. The Open MP 4.0 revision introduced the **`OMP_PLACES`** environment variable. These variables control thread bindings and affinities in a similar manner to **`XLSMPOPTS`** and **`GOMP_CPU_AFFINITY`**, although their syntax slightly differs.

Performance affect

For more information about the thread binding affect on the performance, see 9.1, "Effects of basic performance tuning techniques" on page 260. An easy-to-use code also is available to generate your own binding map. For more information, see 9.3, "Sample code for the construction of thread affinity strings" on page 283.

8.1.2 Setting and retrieving process affinity at run time

By running 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 start a new application with a certain affinity by specifying a mask or a list of logical processors. The Linux scheduler restricts the application threads to a certain set of logical processors only.

You can also use the **taskset** command when an application creates many threads and you want to set the affinity for highly loaded threads only. In this circumstance, identify the process identifiers (PIDs) of highly loaded running threads and perform binding only on those threads. You can discover these threads by examining the output of the **top -H** command.

Knowing the PID, you can also use the **taskset** command to retrieve the affinity of the corresponding entity (a thread or a process).

8.1.3 Controlling the NUMA policy for processes and shared memory

By running the **numactl** command, you can specify a set of nodes and logical processors on which you want your application. In the current context, you can assume that this tool defines a *node* as a group of logical processors that are associated with a particular memory controller. For POWER9 processors, such a node is a whole processor or a GPU. Processors have CPUs and memory, although GPUs have only memory.

To discover the indexes of nodes and estimate the memory access penalty, run the **numactl** command with the **-H** argument. Example 8-1 shows the corresponding output for a 44-core IBM Power System AC922 (Model 8335-GTW) server with six GPUs.

Example 8-1 The numactl -H command output in a 44-core Power AC922 server

```
$ numactl -H
available: 8 nodes (0,8,250-255)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87
node 0 size: 261692 MB
node 0 free: 231086 MB
node 8 cpus: 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167
168 169 170 171 172 173 174 175
node 8 size: 261748 MB
node 8 free: 232644 MB
node 250 cpus:
node 250 size: 16128 MB
node 250 free: 16126 MB
node 251 cpus:
node 251 size: 16128 MB
node 251 free: 16126 MB
node 252 cpus:
node 252 size: 16128 MB
node 252 free: 16126 MB
node 253 cpus:
node 253 size: 16128 MB
node 253 free: 16125 MB
node 254 cpus:
node 254 size: 16128 MB
node 254 free: 16126 MB
node 255 cpus:
node 255 size: 16128 MB
node 255 free: 16126 MB
node distances:
node   0   8  250  251  252  253  254  255
  0:  10  40  80  80  80  80  80  80
  8:  40  10  80  80  80  80  80  80
 250:  80  80  10  80  80  80  80  80
 251:  80  80  80  10  80  80  80  80
 252:  80  80  80  80  10  80  80  80
 253:  80  80  80  80  80  10  80  80
 254:  80  80  80  80  80  80  10  80
 255:  80  80  80  80  80  80  80  10
```

You can pass the indexes of these nodes to **`numactl`** as an argument for the **`-N`** option to bind the process to specific *nodes.* To bind the process to specific *logical processors*, use the **`-C`** option. In the latter case, the indexes follow the same conventions as the OpenMP environment variables and a **`taskset`** command.

The memory placement policy significantly affects the performance of technical computing applications. You can enforce a certain policy by running the **`numactl`** command. The **`-l`** option instructs the OS to always allocate memory pages on the current node. Use the **`-m`** option to specify a list of nodes that the OS can use for memory allocation. You need to use the **`-i`** option to ask the OS for a round-robin allocation policy on specified nodes.

To exemplify the use of **`numactl`** and the effects of running on remote memory, create a simple application that allocates 10 GB memory, as shown in Example 8-2.

Example 8-2 A Python program that allocates 10 GB memory of local versus GPU memory

```
$ cat > 10GB.py << 'EOF'
GB = 1024*1024*1024
a = "a" * (10 * GB)
EOF

$ time numactl --cpunodebind 0 --membind=0 python 10GB.py
real    0m0.776s
user    0m0.258s
sys     0m0.516s


$ time numactl --cpunodebind 0 --membind=255 python 10GB.py
real    0m2.950s
user    0m0.240s
sys     0m2.708s
```

Running on CPU local memory (distance 10), this is less than 1 second. Then, if you run it on GPU memory (distance 80), it takes almost 3 seconds.

Note: Both onboard DDR4 memory and high-bandwidth memory (HBM2) GPU memory is available for user processes. CPU processes prefer to use the much closer distance onboard memory, but if you use more than the amount of memory that you allocated, you automatically start using the slower GPU memory, which can have negative effects on your application.

To avoid accidentally binding interactive processes to GPU memory, bind **sshd** to use memory only from the CPU nodes, as shown in Example 8-3.

Example 8-3 Systemd configuration for binding sshd to CPU memory

```
# mkdir /etc/systemd/system/sshd.service.d/
# cat > /etc/systemd/system/sshd.service.d/numactl.conf << 'EOF'
[Service]
ExecStart=
ExecStart=/usr/bin/numactl -m 0,8 /usr/sbin/sshd -D $OPTIONS
EOF
# systemctl daemon-reload
# systemctl restart sshd
# systemctl status sshd
```

8.2 Performance enhancements and scalability tests

This section provides details about performance enhancements and scalability tests.

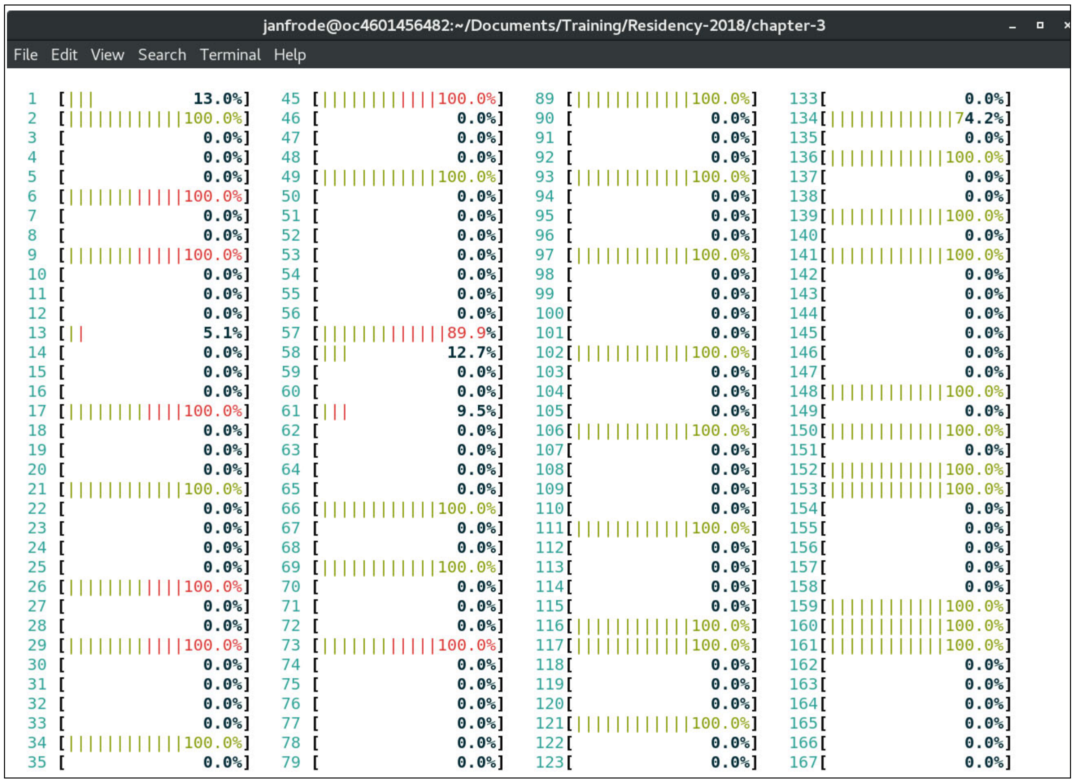
8.2.1 IBM ESSL execution in multiple CPUs and GPUs

One of the many features of IBM ESSL is its ability to compile the same source code to run automatically in different CPUs without any parallel programming that must be performed by the developer. This feature is demonstrated in Example 8-4, which describes how to compile and run Example 7-6 on page 179 by using the symmetric multiprocessor (SMP) version of IBM ESSL and the XLC compiler, as described in 8.1.1, “Running OpenMP applications” on page 228.

Example 8-4 Compilation and running of dgemm_sample.c for SMP IBM ESSL

```
$ export XLSMPOPTS=parthds=44

$ xlc_r -O3 dgemm_sample.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath
-L/opt/ibm/xlsmp/5.1.0/lib -L/opt/ibm/xlf/16.1.0/lib -R/opt/ibm/lib -o dgemm_smp

$ ./dgemm_smp
29.638 seconds, 539820.501 MFlops
```

To set the number of SMP threads, use **XLSMPOPTS**. Therefore, 44 CPUs are running at maximum throughput when this feature is used, as shown in Figure 8-1.


janfrode@oc4601456482:~/Documents/Training/Residency-2018/chapter-3
File Edit View Search Terminal Help

```
1  [|||          13.0%]   45 [||||||||||||100.0%]   89 [||||||||||||100.0%]   133[              0.0%]
2  [||||||||||||100.0%]   46 [              0.0%]   90 [              0.0%]   134[|||||||||||||74.2%]
3  [              0.0%]   47 [              0.0%]   91 [              0.0%]   135[              0.0%]
4  [              0.0%]   48 [              0.0%]   92 [              0.0%]   136[||||||||||||100.0%]
5  [              0.0%]   49 [||||||||||||100.0%]   93 [||||||||||||100.0%]   137[              0.0%]
6  [||||||||||||100.0%]   50 [              0.0%]   94 [              0.0%]   138[              0.0%]
7  [              0.0%]   51 [              0.0%]   95 [              0.0%]   139[||||||||||||100.0%]
8  [              0.0%]   52 [              0.0%]   96 [              0.0%]   140[              0.0%]
9  [||||||||||||100.0%]   53 [              0.0%]   97 [||||||||||||100.0%]   141[||||||||||||100.0%]
10 [              0.0%]   54 [              0.0%]   98 [              0.0%]   142[              0.0%]
11 [              0.0%]   55 [              0.0%]   99 [              0.0%]   143[              0.0%]
12 [              0.0%]   56 [              0.0%]   100[              0.0%]   144[              0.0%]
13 [||            5.1%]   57 [|||||||||||||89.9%]   101[              0.0%]   145[              0.0%]
14 [              0.0%]   58 [|||          12.7%]   102[||||||||||||100.0%]   146[              0.0%]
15 [              0.0%]   59 [              0.0%]   103[              0.0%]   147[              0.0%]
16 [              0.0%]   60 [              0.0%]   104[              0.0%]   148[||||||||||||100.0%]
17 [||||||||||||100.0%]   61 [|||           9.5%]   105[              0.0%]   149[              0.0%]
18 [              0.0%]   62 [              0.0%]   106[||||||||||||100.0%]   150[||||||||||||100.0%]
19 [              0.0%]   63 [              0.0%]   107[              0.0%]   151[              0.0%]
20 [              0.0%]   64 [              0.0%]   108[              0.0%]   152[||||||||||||100.0%]
21 [||||||||||||100.0%]   65 [              0.0%]   109[              0.0%]   153[||||||||||||100.0%]
22 [              0.0%]   66 [||||||||||||100.0%]   110[              0.0%]   154[              0.0%]
23 [              0.0%]   67 [              0.0%]   111[||||||||||||100.0%]   155[              0.0%]
24 [              0.0%]   68 [              0.0%]   112[              0.0%]   156[              0.0%]
25 [              0.0%]   69 [||||||||||||100.0%]   113[              0.0%]   157[              0.0%]
26 [||||||||||||100.0%]   70 [              0.0%]   114[              0.0%]   158[              0.0%]
27 [              0.0%]   71 [              0.0%]   115[              0.0%]   159[||||||||||||100.0%]
28 [              0.0%]   72 [              0.0%]   116[||||||||||||100.0%]   160[||||||||||||100.0%]
29 [||||||||||||100.0%]   73 [||||||||||||100.0%]   117[||||||||||||100.0%]   161[||||||||||||100.0%]
30 [              0.0%]   74 [              0.0%]   118[              0.0%]   162[              0.0%]
31 [              0.0%]   75 [              0.0%]   119[              0.0%]   163[              0.0%]
32 [              0.0%]   76 [              0.0%]   120[              0.0%]   164[              0.0%]
33 [              0.0%]   77 [              0.0%]   121[||||||||||||100.0%]   165[              0.0%]
34 [||||||||||||100.0%]   78 [              0.0%]   122[              0.0%]   166[              0.0%]
35 [              0.0%]   79 [              0.0%]   123[              0.0%]   167[              0.0%]
```

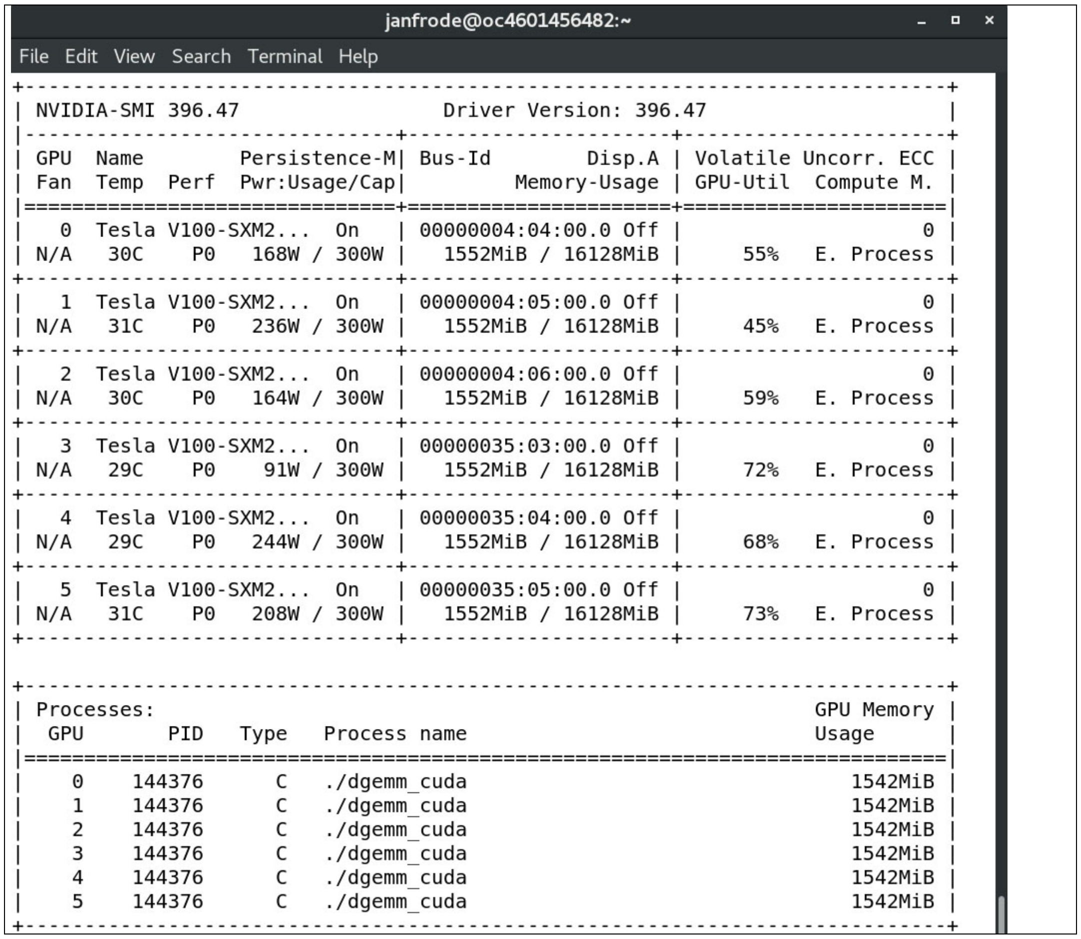
Figure 8-1 Screen capture of htop during the run of a dgemm_smp calculation of order [20,000x20,000]

This result presents a gain of approximately 18x for SMP execution when compared to its serial runs.

Performance can be enhanced further by using the SMP Compute Unified Device Architecture (CUDA) version of IBM ESSL, which enables the use of four GPUs to run the code, as shown on Example 8-5 and in Figure 8-2 on page 235.

Example 8-5 Compilation and execution of dgemm_sample.c for SMP GPU IBM ESSL

```
$ export XLSMPOPTS=parthds=20

$ xlc_r -O3 dgemm_sample.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas
-lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64
-L/opt/ibm/xlsmp/5.1.0/lib -L/opt/ibm/xlf/16.1.0/lib -R/opt/ibm/lib -o dgemm_cuda

$ ./dgemm_cuda
7.201 seconds, 2221802.527 MFlops
```



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.47                 Driver Version: 396.47                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  On   | 00000004:04:00.0 Off |                    0 |
| N/A   30C    P0   168W / 300W |   1552MiB / 16128MiB |     55%   E. Process |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  On   | 00000004:05:00.0 Off |                    0 |
| N/A   31C    P0   236W / 300W |   1552MiB / 16128MiB |     45%   E. Process |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  On   | 00000004:06:00.0 Off |                    0 |
| N/A   30C    P0   164W / 300W |   1552MiB / 16128MiB |     59%   E. Process |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  On   | 00000035:03:00.0 Off |                    0 |
| N/A   29C    P0    91W / 300W |   1552MiB / 16128MiB |     72%   E. Process |
+-------------------------------+----------------------+----------------------+
|   4  Tesla V100-SXM2...  On   | 00000035:04:00.0 Off |                    0 |
| N/A   29C    P0   244W / 300W |   1552MiB / 16128MiB |     68%   E. Process |
+-------------------------------+----------------------+----------------------+
|   5  Tesla V100-SXM2...  On   | 00000035:05:00.0 Off |                    0 |
| N/A   31C    P0   208W / 300W |   1552MiB / 16128MiB |     73%   E. Process |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0    144376      C   ./dgemm_cuda                                 1542MiB |
|    1    144376      C   ./dgemm_cuda                                 1542MiB |
|    2    144376      C   ./dgemm_cuda                                 1542MiB |
|    3    144376      C   ./dgemm_cuda                                 1542MiB |
|    4    144376      C   ./dgemm_cuda                                 1542MiB |
|    5    144376      C   ./dgemm_cuda                                 1542MiB |
+-----------------------------------------------------------------------------+
```

Figure 8-2 Screen capture of nvidia-smi during the run of dgemm_cuda

A 95x speedup gain can be observed when GPUs are used for this calculation. In addition, the CUDA run reaches over 3.6 teraflops (TFLOPS), as shown in Figure 8-3.

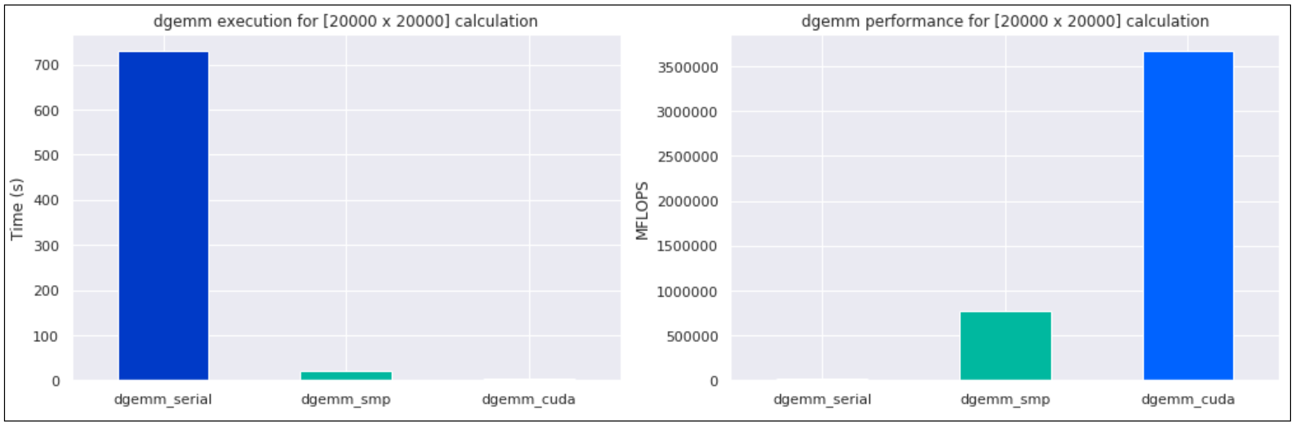


Figure 8-3 Running and performance of dgemm

Another strategy that was used on older GPUs was adjusting the Power Cap Limit, which can improve the performance of parallel software on GPU cards. However, Volta GPUs have the Power Cap at maximum capacity (see Example 8-6) when compared against Figure 8-3 on page 235. Therefore, this strategy is obsolete for performance improvement.

Example 8-6   Compilation and run of dgemm_sample.c for SMP IBM ESSL

```
$ export XLSMPOPTS=parthds=20
$ xlc_r -O3 dgemm_sample.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath
-L/opt/ibm/xlsmp/5.1.0/lib -L/opt/ibm/xlf/16.1.0/lib -R/opt/ibm/lib -o dgemm_smp

$ ./dgemm_smp
38.130 seconds, 419596.119 MFlops
```

For more information about the new IBM ESSL features, see the Engineering and Scientific Subroutine Library page.

Running on different SMT modes

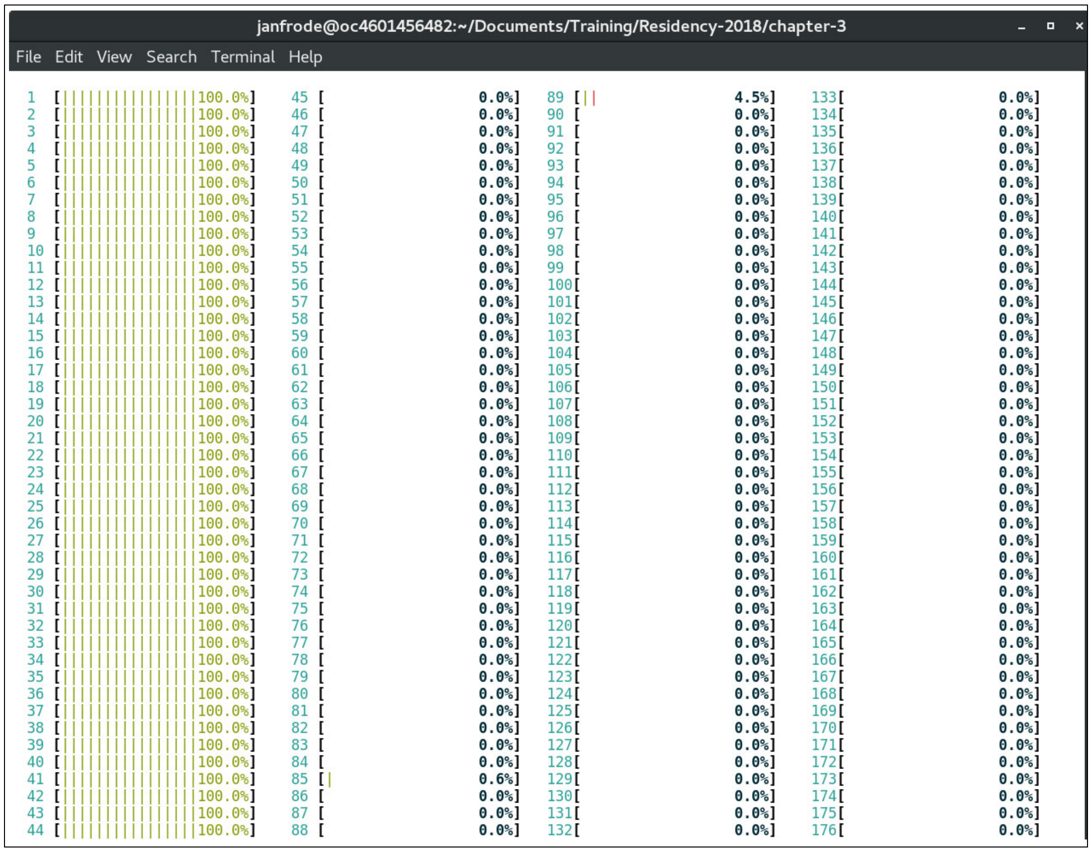
Example 8-4 on page 233 shows how to request several CPUs of the OS to run a job. You can also use the environment variable **XLSMPOPTS** to control the distribution.

The sample system has 44 physical POWER9 cores and 176 logical CPUs. Every 4 CPUs depend on one physical core. By default, the system is set to SMT-4 mode, which means that all 4 CPUs per core can be used. In the first run that is shown in Example 8-7, the example uses CPUs 0 - 43. In the second run, only even numbers of CPUs are used, which simulates SMT-2 mode. The third run simulates SMT-1 mode.

Example 8-7   Different CPU binding for running with 44 symmetric multiprocessor threads

```
$ export
XLSMPOPTS=parthds=44:PROCS=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21
,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43
$ ./dgemm_smp
63.517 seconds, 251888.471 MFlops

$ export
XLSMPOPTS=parthds=22:PROCS=0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,
40,42
$ ./dgemm_smp
70.810 seconds, 225945.488 MFlops

$ export
XLSMPOPTS=parthds=44:PROCS=0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,7
6,80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156,16
0,164,168,172
$ ./dgemm_smp
20.729 seconds, 771826.909 MFlops
```

Figure 8-4 shows the result of `dgemm_smp` with the first export configuration.


janfrode@oc4601456482:~/Documents/Training/Residency-2018/chapter-3
File Edit View Search Terminal Help

```
1  [||||||||||||||||100.0%]    45 [                  0.0%]    89 [||                4.5%]    133[                  0.0%]
2  [||||||||||||||||100.0%]    46 [                  0.0%]    90 [                  0.0%]    134[                  0.0%]
3  [||||||||||||||||100.0%]    47 [                  0.0%]    91 [                  0.0%]    135[                  0.0%]
4  [||||||||||||||||100.0%]    48 [                  0.0%]    92 [                  0.0%]    136[                  0.0%]
5  [||||||||||||||||100.0%]    49 [                  0.0%]    93 [                  0.0%]    137[                  0.0%]
6  [||||||||||||||||100.0%]    50 [                  0.0%]    94 [                  0.0%]    138[                  0.0%]
7  [||||||||||||||||100.0%]    51 [                  0.0%]    95 [                  0.0%]    139[                  0.0%]
8  [||||||||||||||||100.0%]    52 [                  0.0%]    96 [                  0.0%]    140[                  0.0%]
9  [||||||||||||||||100.0%]    53 [                  0.0%]    97 [                  0.0%]    141[                  0.0%]
10 [||||||||||||||||100.0%]    54 [                  0.0%]    98 [                  0.0%]    142[                  0.0%]
11 [||||||||||||||||100.0%]    55 [                  0.0%]    99 [                  0.0%]    143[                  0.0%]
12 [||||||||||||||||100.0%]    56 [                  0.0%]    100[                  0.0%]    144[                  0.0%]
13 [||||||||||||||||100.0%]    57 [                  0.0%]    101[                  0.0%]    145[                  0.0%]
14 [||||||||||||||||100.0%]    58 [                  0.0%]    102[                  0.0%]    146[                  0.0%]
15 [||||||||||||||||100.0%]    59 [                  0.0%]    103[                  0.0%]    147[                  0.0%]
16 [||||||||||||||||100.0%]    60 [                  0.0%]    104[                  0.0%]    148[                  0.0%]
17 [||||||||||||||||100.0%]    61 [                  0.0%]    105[                  0.0%]    149[                  0.0%]
18 [||||||||||||||||100.0%]    62 [                  0.0%]    106[                  0.0%]    150[                  0.0%]
19 [||||||||||||||||100.0%]    63 [                  0.0%]    107[                  0.0%]    151[                  0.0%]
20 [||||||||||||||||100.0%]    64 [                  0.0%]    108[                  0.0%]    152[                  0.0%]
21 [||||||||||||||||100.0%]    65 [                  0.0%]    109[                  0.0%]    153[                  0.0%]
22 [||||||||||||||||100.0%]    66 [                  0.0%]    110[                  0.0%]    154[                  0.0%]
23 [||||||||||||||||100.0%]    67 [                  0.0%]    111[                  0.0%]    155[                  0.0%]
24 [||||||||||||||||100.0%]    68 [                  0.0%]    112[                  0.0%]    156[                  0.0%]
25 [||||||||||||||||100.0%]    69 [                  0.0%]    113[                  0.0%]    157[                  0.0%]
26 [||||||||||||||||100.0%]    70 [                  0.0%]    114[                  0.0%]    158[                  0.0%]
27 [||||||||||||||||100.0%]    71 [                  0.0%]    115[                  0.0%]    159[                  0.0%]
28 [||||||||||||||||100.0%]    72 [                  0.0%]    116[                  0.0%]    160[                  0.0%]
29 [||||||||||||||||100.0%]    73 [                  0.0%]    117[                  0.0%]    161[                  0.0%]
30 [||||||||||||||||100.0%]    74 [                  0.0%]    118[                  0.0%]    162[                  0.0%]
31 [||||||||||||||||100.0%]    75 [                  0.0%]    119[                  0.0%]    163[                  0.0%]
32 [||||||||||||||||100.0%]    76 [                  0.0%]    120[                  0.0%]    164[                  0.0%]
33 [||||||||||||||||100.0%]    77 [                  0.0%]    121[                  0.0%]    165[                  0.0%]
34 [||||||||||||||||100.0%]    78 [                  0.0%]    122[                  0.0%]    166[                  0.0%]
35 [||||||||||||||||100.0%]    79 [                  0.0%]    123[                  0.0%]    167[                  0.0%]
36 [||||||||||||||||100.0%]    80 [                  0.0%]    124[                  0.0%]    168[                  0.0%]
37 [||||||||||||||||100.0%]    81 [                  0.0%]    125[                  0.0%]    169[                  0.0%]
38 [||||||||||||||||100.0%]    82 [                  0.0%]    126[                  0.0%]    170[                  0.0%]
39 [||||||||||||||||100.0%]    83 [                  0.0%]    127[                  0.0%]    171[                  0.0%]
40 [||||||||||||||||100.0%]    84 [                  0.0%]    128[                  0.0%]    172[                  0.0%]
41 [||||||||||||||||100.0%]    85 [|                 0.6%]    129[                  0.0%]    173[                  0.0%]
42 [||||||||||||||||100.0%]    86 [                  0.0%]    130[                  0.0%]    174[                  0.0%]
43 [||||||||||||||||100.0%]    87 [                  0.0%]    131[                  0.0%]    175[                  0.0%]
44 [||||||||||||||||100.0%]    88 [                  0.0%]    132[                  0.0%]    176[                  0.0%]
```

Figure 8-4 Result of dgemm_smp with the first export configuration

For the third run with SMT-1 mode, running one thread on each POWER9 core gives 50% performance improvement compared to runs without CPU binding.

IBM ESSL SMP CUDA library options

The IBM ESSL SMP CUDA library has the following options that are controlled by the user:

- Control which GPUs IBM ESSL uses.

  By default, IBM ESSL uses all available devices, but you can change it by using the environment variable **`CUDA_VISIBLE_DEVICES`** or the **`SETGPUS`** subroutine. GPUs are numerated in the OS starting with 0. For example, if you want to use only the second and third GPUs in your run, set the environment variable as shown in the following example:

  ```
  $ export CUDA_VISIBLE_DEVICES=1,2
  ```

  You also can place the call into the code, as shown in Example 8-8.

  Example 8-8 Binding IBM ESSL code to GPUs

  ```
  int ids[2] = {1, 2}; //GPUs IDs
  int ngpus = 2; //Number of GPUs
  ...
  setgpus(ngpus, ids);
  /*your ESSL SMP CUDA call*/
  ```

  You can also specify a different order of devices, which can be useful in cases when you want to reduce latency between specific CPUs and GPUs.

- Disable or enable hybrid mode.

  By default, IBM ESSL runs in hybrid mode. Therefore, IBM ESSL routines use POWER9 CPUs and NVIDIA GPUs. To disable this mode and start using only GPUs, you must specify the following environment variable:

  ```
  export ESSL_CUDA_HYBRID=no
  ```

  To re-enable it, unset this variable or set it to `yes`.

- Pin host memory buffers.

  The following options are provided by IBM ESSL:

  – Not pin host memory buffers (default).

  – Allow IBM ESSL to pin buffers alone. To use this option, set the following environment variable:

    ```
    export ESSL_CUDA_PIN=yes
    ```

  – Provide information to IBM ESSL that you will pin the buffers before the IBM ESSL routines calls:

    ```
    export ESSL_CUDA_PIN=pinned
    ```

Example 8-9 shows runs of source code from Example 7-5 on page 177 that are scaled up to a 10000x10000 matrix with different IBM ESSL SMP CUDA library options.

Example 8-9 The dgemm_sample.c runs with different symmetric multiprocessor CUDA options

```
#! /bin/sh -
export XLSMPOPTS=startproc=0:stride=4:parthds=44
xlc_r -O3 dgemm_sample.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas
-lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64
-L/opt/ibm/xlsmp/5.1.0/lib -L/opt/ibm/xlf/16.1.0/lib -R/opt/ibm/lib -o dgemm_cuda

export CUDA_VISIBLE_DEVICES=0,1,2,3,4,5
export ESSL_CUDA_HYBRID=yes
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 6 GPUs hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 6 GPUs non-hybrid mode: $VAR"

export CUDA_VISIBLE_DEVICES=0,1,2,3,4
export ESSL_CUDA_HYBRID=yes
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 5 GPUs (1st, 2nd, 3rd, 4th, 5th) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 5 GPUs (1st, 2nd, 3rd, 4th, 5th) non-hybrid mode: $VAR"

export CUDA_VISIBLE_DEVICES=0,1,3,4
export ESSL_CUDA_HYBRID=yes
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 4 GPUs (1st, 2nd, 4th, 5th) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
```

```
echo "SMP CUDA run with 4 GPUs (1st, 2nd, 4th, 5th) non-hybrid mode: $VAR"

export CUDA_VISIBLE_DEVICES=0,1,3
export ESSL_CUDA_HYBRID=yes
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 3 GPUs (1st, 2nd, 4th) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 3 GPUs (1st, 2nd, 4th) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,3
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 2 GPUs (1st, 4th) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 2 GPUs (1st, 4th) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (1st) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (1st) non-hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=3
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (4th) hybrid mode: $VAR"

export ESSL_CUDA_HYBRID=no
VAR=`./dgemm_cuda`
echo "SMP CUDA run with 1 GPU (4th) non-hybrid mode: $VAR"
```

The results of the runs are shown in Example 8-10.

Example 8-10 Result of different IBM ESSL symmetric multiprocessor CUDA runs

```
SMP CUDA run with 6 GPUs hybrid mode: 89.127 seconds, 22439664.748 MFlops
SMP CUDA run with 6 GPUs non-hybrid mode: 89.527 seconds, 22339405.989 MFlops
SMP CUDA run with 5 GPUs (1st, 2nd, 3rd, 4th, 5th) hybrid mode: 101.032 seconds,
19795510.333 MFlops
SMP CUDA run with 5 GPUs (1st, 2nd, 3rd, 4th, 5th) non-hybrid mode: 101.505
seconds, 19703265.849 MFlops
SMP CUDA run with 4 GPUs (1st, 2nd, 4th, 5th) hybrid mode: 121.712 seconds,
16432069.147 MFlops
SMP CUDA run with 4 GPUs (1st, 2nd, 4th, 5th) non-hybrid mode: 124.152 seconds,
16109124.299 MFlops
SMP CUDA run with 3 GPUs (1st, 2nd, 4th) hybrid mode: 155.352 seconds,
12873860.652 MFlops
SMP CUDA run with 3 GPUs (1st, 2nd, 4th) non-hybrid mode: 157.673 seconds,
12684353.060 MFlops
```

```
SMP CUDA run with 2 GPUs (1st, 4th) hybrid mode: 229.629 seconds, 8709614.204
MFlops
SMP CUDA run with 2 GPUs (1st, 4th) non-hybrid mode: 229.331 seconds, 8720931.754
MFlops
SMP CUDA run with 1 GPU (1st) hybrid mode: 418.384 seconds, 4780249.723 MFlops
SMP CUDA run with 1 GPU (1st) non-hybrid mode: 423.952 seconds, 4717468.015 MFlops
SMP CUDA run with 1 GPU (4th) hybrid mode: 422.105 seconds, 4738110.186 MFlops
SMP CUDA run with 1 GPU (4th) non-hybrid mode: 426.181 seconds, 4692794.845 MFlops
```

In this set of examples, we see that not much extra performance is gained by using the hybrid mode (0.5 - 2%). It is probably better to use only GPUs in this example and save the CPUs for other tasks.

For more information about the IBM ESSL SMP CUDA library, see the Using the ESSL SMP CUDA Library page.

8.2.2 OpenACC execution and scalability

This section describes testing the performance of the OpenACC model in our system. Example 7-17 on page 202 is run for an average of three times to create a speedup[2] per block analysis that is shown in Figure 8-5.

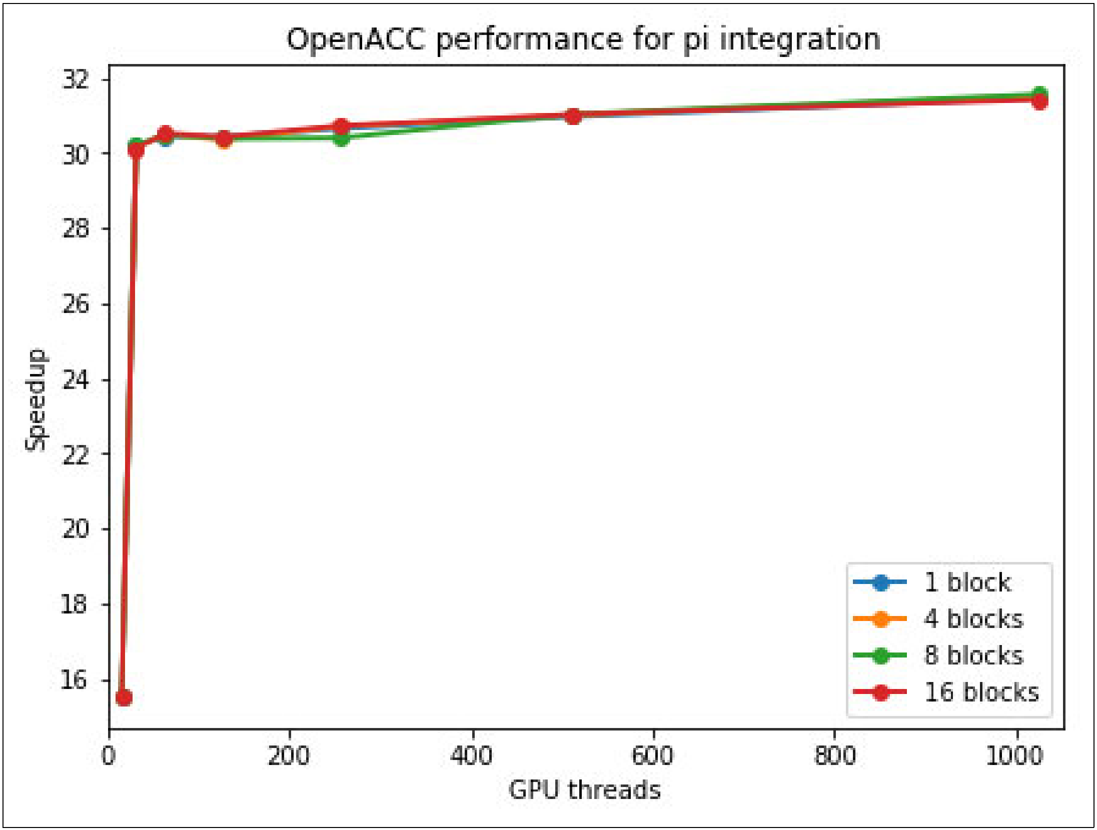


Figure 8-5 Scalability curve from the pi integration in OpenACC in Example 7-17 on page 202

[2] The improvement speed of the run of a fixed task is achieved by increasing computing elements to process that task. It is the ratio between the sequential time of running a task divided by the time that is taken to run that task in parallel with a number of processing elements.

As shown in Figure 8-5 on page 240, an instant gain with all block configurations is observed, which results in an approximately 30x speedup with the pi parallelization example by using 10,000,000,000,000 steps. This result likely indicates that the parallel part of this piece of code was reduced to its minimum, which created the speedup threshold that is stated in Ahmdahl's law[3].

Considering that parallelization occurs only in a FOR loop where the measurement is performed, it is a fair conclusion that a threshold of maximum performance is achieved.

8.3 Using IBM Parallel Environment V2.3

This section describes the running of a parallel application through the IBM Parallel Environment Runtime.

Note: IBM Parallel Environment is being deprecated in favor IBM Spectrum Message Passing Interface (IBM Spectrum MPI), which is a lighter and high-performance implementation of the Message Passing Interface (MPI) standard.

For more information about some differences, porting, and other related topics, see 7.6.9, "MPI programs that use IBM Spectrum MPI" on page 220.

8.3.1 Running applications

The IBM Parallel Environment provides an environment to manage the execution of parallel applications. To start the Parallel Operating Environment (POE), run the **poe** command.

Running an application with POE spreads processes across the cluster nodes. Consider the following points:

- Parallel tasks are created on compute nodes.
- There is one instance of the partition manager daemon (PMD) per compute node, which features the tasks of the running application.
- The POE process is on the submitting node (home node) machine where it was started.

The PMD controls communication between the home **poe** process and the created tasks in a specific compute node. The PMD is used to pass the standard input, output, and error streams of the home POE to the tasks.

However, if the PMD process exits abnormally, such as with **kill** signals or by the **bkill** command of the IBM Spectrum Load Sharing Facility (IBM Spectrum LSF), the shared memory that is used for intranode message passing cannot get cleaned up properly. In this case, use the **ipcrm** command to reclaim that memory.

The environment works in two modes: interactive and batch. It also allows Single Program Multiple Data (SPMD) and Multiple Program, Multiple Data (MPMD) programs.

Many environment variables are in place to control the behavior of POE. Some of the variables are described next.

[3] Ahmdahl's Law is a known calculation in parallel computing of the theoretical speedup that can be achieved when computing a fixed task by using multiple resources. It states that the speedup is limited by the serial part of code. Therefore, the programmer seeks to grow the parallel part of the code as much as possible.

The number of tasks in the parallel application is specified by **MP_PROCS** variable, which is equivalent to the **-procs** option of the **poe** command. The application is set with 10 tasks in the following example:

```
$ MP_PROCS=10 poe ./myApp
```

The **poe** command manages the running of applications that are implemented with different models and eventually mixed. To set the messaging application programming interface (API), set the **MP_MSG_API** variable (equivalent to the **-msg_api** option). The accepted values are `MPI`, `PAMI`, and `shmem`. It does not need to be set for MPI programs because it is the default. In following command, **poe** is called to run a 10 task OpenSHMEM program:

```
$ MP_PROCS=10 MP_MSG_API=shmem poe ./myApp
```

Compute nodes allocation

The partition manager can connect to an external resources manager that determines the allocation of the compute nodes. For example, the configuration that is described in 8.4, "Using IBM Spectrum LSF" on page 247 can be configured to integrate seamlessly with IBM Parallel Environment so that hosts are selected by the IBM Spectrum LSF **bsub** command for batch jobs submission.

By default, the allocation uses any resources manager that is configured. However, it can be disabled by setting **MP_RESD** (or **-resd option**). Also, the native partition manager nodes allocation mechanism is enabled by `MP_RESD=poe` and require a hosts list file.

Example 8-11 shows what a host list file looks like when a resource manager is not used with POE. `MP_RESD=poe` is exported to enable the internal host allocation mechanism.

Example 8-11 Running a parallel application without a resource manager through IBM Parallel Environment

```
$ cat host.list
! Host list - One host per line
! Tasks 0 and 2 run on p9r1n1 host
! Tasks 1 and 3 run on p9r2n2 host
p9r1n1
p9r2n2
p9r1n1
p9r2n2
$ MP_PROCS=4 MP_RESD=poe poe ./myApp
```

The format and content of the hosts list file changes whether a resource manager is used. For more information, see the IBM Parallel Environment documentation.

If POE needs to point out the host file location, use the **MP_HOSTFILE** variable (same as the **-hostfile** option of **poe**). If it is not set, **poe** looks for a file that is named `host.list` in the local directory.

When a resource manager is in place, the system administrators often configure pools of hosts. Therefore, consider using the *MP_RMPOOL* variable (or **-rmpool** option) to determine which of the pools of machines were configured (if any) by the administrators to use.

Other variables are available to configure the resource manager behavior. The **MP_NODES** (**-nodes** option) environmental variable sets the number of physical nodes, and the **MP_TASKS_PER_NODE** (**-tasks_per_node** option) variable sets the number of tasks per nodes.

Considerations about network configuration

The IBM Parallel Environment provides variables to control and configure the network adapters that are used by the parallel applications. Some variables might be implicitly set depending on the combination of other settings or by the resource manager.

To specify the network adapters to use for message passing, set the **`MP_EUIDEVICE`** variable (or **`-euidevice`** option). The variable accepts the values `sn_all` (one or more windows are on each network) or `sn_single` (all windows are on a single network). The `sn_all` value is frequently used to enable protocol stripping, failover, and recovery.

Network adapters can be shared or dedicated. That behavior is defined by using the **`MP_ADAPTER_USE`** variable (or **`-adapter_use`** option). It accepts the `shared` and `dedicated` values.

Considerations about Remote Direct Memory Access

IBM Parallel Environment implements message passing by using Remote Direct Memory Access (RDMA) through the InfiniBand interconnect. In such a mechanism, memory pages are automatically pinned and buffer transferences are handled directly by the InfiniBand adapter without the host CPU involvement.

RDMA on messaging passing is disabled by default. Set `MP_USE_BULK_XFER=yes` to enable the bulk data transfer mechanism. Also, set the **`MP_BULK_MIN_MSG_SIZE`** variable to set the minimum message length for bulk transfer.

Considerations about affinity

Several levels of affinity are available for parallel applications through the **`poe`** command. These levels are also controlled by environment variables or **`poe`** options. Because resource managers usually employ their own affinity mechanisms, those variables can be overwritten or ignored.

The primary variable to control placement of MPI tasks is **`MP_TASK_AFFINITY`** (or the **`-task_affinity`** option) when a resource manager is not used. You can bind tasks at the physical processor (`core` value), logical CPU (`cpu` value), and multi-chip module (`mcm` value) levels.

For example, the following command allocates one core per task:

```
$ poe -task_affinity core -procs 2 ./myApp
```

For more examples of **`MP_TASK_AFFINITY`** being used to control task affinity, see 7.6.5, "MPI programs with IBM Parallel Environment V2.3" on page 209.

The **`MP_CPU_BIND_LIST`** (or **`-cpu_bind_list`** option) and **`MP_BIND_MODE`** (or **`-bind_mode spread`** option) environment variables can be used with **`MP_TASK_AFFINITY`** to further control the placement of tasks. **`MP_CPU_BIND_LIST`** specifies a list of processor units for establishing task affinity. In the following command, affinity is restricted to only the second core of each socket:

```
$ poe -task_affinity core -cpu_bind_list 0/16,8/1040 -procs 2
```

When a resource manager is used, the **`MP_PE_AFFINITY`** variable can be set to `yes` so that **`poe`** assumes control over affinity. However, if IBM Spectrum LSF is used and it sets the affinity, **`poe`** acknowledges the allocated CPUs. If `MP_PE_AFFINITY=yes` is enabled in IBM Spectrum LSF batch jobs, it enables the InfiniBand adapter affinity.

To help define affinity, IBM Parallel Environment Runtime Edition provides the **`cpuset_query`** command, which shows information about the current assignments of a running program. It also shows the topology of any specific compute node.

As shown in Example 8-12, **`cpuset_query -t`** shows the topology of an IBM Power System S822LC server that is running in SMT-8 mode with two sockets with 10 cores each with eight hardware threads.

Example 8-12 Running the cpuset_query command to show the node topology

```
$ cpuset_query -t
MCM(Socket): 251
MCM(Socket): 0
        CORE: 12
                CPU: 13
                CPU: 14
                CPU: 12
                CPU: 15
        CORE: 40
                CPU: 41
                CPU: 42
                CPU: 40
                CPU: 43
        CORE: 28
                CPU: 31
                CPU: 28
                CPU: 30
                CPU: 29

                CPU: 15
<... Output Omitted ...>
MCM(Socket): 8
        CORE: 2116
                CPU: 149
                CPU: 151
                CPU: 148
                CPU: 150
        CORE: 2076
                CPU: 110
                CPU: 109
                CPU: 111
                CPU: 108
        CORE: 2104
                CPU: 139
                CPU: 137
                CPU: 138
                CPU: 136
<... Output Omitted ...>
```

You can check the affinity by running the **`cpuset_query`** command through **`poe`**, as shown in Example 8-13. The command shows a two-tasks program that is started with affinity at a core level. Each task is allocated (see CPUs with value `1`) with a full core that has four hardware threads on an SMT-4 mode node.

Example 8-13 Running the cpuset_query command to show task affinity

```
MCM/QUAD(251) contains:
[Total cpus for MCM/QUAD(251)=0]
MCM/QUAD(0) contains:
cpu13, cpu41, cpu31, cpu9, cpu21,
cpu78, cpu11, cpu68, cpu7, cpu58,
cpu86, cpu48, cpu76, cpu38, cpu66,
cpu28, cpu5, cpu56, cpu84, cpu18,
cpu46, cpu74, cpu36, cpu64, cpu26,
cpu3, cpu54, cpu82, cpu16, cpu44,
cpu72, cpu34, cpu62, cpu24, cpu1,
cpu52, cpu80, cpu14, cpu42, cpu70,
cpu32, cpu60, cpu22, cpu50, cpu79,
cpu12, cpu40, cpu69, cpu30, cpu8,
cpu59, cpu87, cpu20, cpu49, cpu77,
cpu10, cpu39, cpu67, cpu29, cpu6,
cpu57, cpu85, cpu19, cpu47, cpu75,
cpu37, cpu65, cpu27, cpu4, cpu55,
cpu83, cpu17, cpu45, cpu73, cpu35,
cpu63, cpu25, cpu2, cpu53, cpu81,
cpu15, cpu43, cpu71, cpu33, cpu61,
cpu23, cpu0, cpu51, [Total cpus for MCM/QUAD(0)=88]
MCM/QUAD(254) contains:
[Total cpus for MCM/QUAD(254)=0]
MCM/QUAD(252) contains:
[Total cpus for MCM/QUAD(252)=0]
MCM/QUAD(250) contains:
[Total cpus for MCM/QUAD(250)=0]
MCM/QUAD(255) contains:
[Total cpus for MCM/QUAD(255)=0]
MCM/QUAD(253) contains:
[Total cpus for MCM/QUAD(253)=0]
MCM/QUAD(8) contains:
cpu149, cpu110, cpu139, cpu98, cpu167,
cpu100, cpu129, cpu88, cpu157, cpu119,
cpu147, cpu175, cpu109, cpu137, cpu96,
cpu165, cpu127, cpu155, cpu117, cpu145,
cpu173, cpu107, cpu135, cpu94, cpu163,
cpu125, cpu153, cpu115, cpu143, cpu171,
cpu105, cpu133, cpu92, cpu161, cpu123,
cpu151, cpu113, cpu141, cpu103, cpu131,
cpu90, cpu121, cpu111, cpu99, cpu168,
cpu101, cpu89, cpu158, cpu148, cpu138,
cpu97, cpu166, cpu128, cpu156, cpu118,
cpu146, cpu174, cpu108, cpu136, cpu95,
cpu164, cpu126, cpu154, cpu116, cpu144,
cpu172, cpu106, cpu134, cpu93, cpu162,
cpu124, cpu152, cpu114, cpu142, cpu170,
cpu104, cpu132, cpu91, cpu160, cpu122,
cpu150, cpu112, cpu140, cpu169, cpu102,
cpu130, cpu159, cpu120, [Total cpus for MCM/QUAD(8)=88]
Total number of MCMs/QUADs found  = 8
Total number of COREs found       = 44
Total number of CPUs found        = 176
cpuset for process 130902 (1 = in the set, 0 = not included)
cpu0  = 1
cpu1  = 1
cpu2  = 1
```

```
cpu3  = 1
<... Output Omitted ...>
```

Considerations about the CUDA-aware Message Passing Interface

IBM Parallel Environment Runtime Edition implements a CUDA-aware MPI mechanism, but it is disabled by default. To enable it, set the **MP_CUDA_AWARE** variable to `yes`. For more information, see 7.6.6, "Hybrid MPI and CUDA programs with IBM Parallel Environment" on page 215.

8.3.2 Managing applications

IBM Parallel Environment Runtime Edition provides a set of commands to manage stand-alone **poe** jobs. This section introduces the most common commands.

Canceling an application

Because the **poe** command handles the signals of all tasks in the partition, sending an interrupt (SIGINT) or terminate (SIGTERM) signal triggers all remote processes. If **poe** runs with 100413 process ID, you can end the program by running the following command:

```
$ kill -s SIGINT 100413
```

However, if some remote processes are orphans, run **poekill** ***program_name*** to end all remaining tasks. In fact, **poekill** can send any signal to all the remote processes.

Suspending and resuming a job

To cancel a **poe** process, suspend and resume applications by using signals. Run a **poekill** or **kill** command to send a SIGTSTP to suspend the **poe** process (which triggers the signal to all tasks).

The application can be resumed by sending a SIGCONT to continue **poe**. Run a **poekill**, **kill**, **fg**, or **bg** command to deliver the signal.

8.3.3 Running OpenSHMEM programs

You can set the following environment variables to run an OpenSHMEM program with IBM Parallel Environment Runtime Edition by running a **poe** command:

- `MP_MSG_API=shmem`

  Instructs **poe** to use the OpenSHMEM message passing API.
- `MP_USE_BULK_XFER=yes`

  Enables the use of RDMA in Parallel Active Messaging Interface (PAMI).

You can use `MP_PROCS=<num>` set the number of processing elements.

NAS Parallel Benchmarks with OpenSHMEM

The NAS Parallel Benchmarks[4] (NPBs) are a well-known suite of parallel applications that are often used to evaluate high-performance computers. The benchmarks provide programs, kernels, and problem solvers that simulate aspects such as computation, data movement, and I/O of real scientific applications. You can select the size of the workload that each benchmark processes among a list of classes (A, B, C, and so on).

[4] For more information about NPBs, see http://www.nas.nasa.gov/publications/npb.html.

A version of NPB that was rewritten by using OpenSHMEM in C and Fortran was released by the OpenSHMEM group. NPB3.2-SHMEM is the implementation of NPB 3.2. It provides some benchmarks in Fortran and only one in C, as shown in Example 8-14.

Example 8-14 The openshmen-npbs implementation

```
$ git clone https://github.com/openshmem-org/openshmem-npbs
$ cd openshmem-npbs/C
$ cd config
$ cp suite.def.template suite.def
$ cp make.def.template make.def
# Set CC in make.def
$ cd ../
$ make is NPROCS=2 CLASS=A
make suite
$ cd bin
$ ls
host.list  is.A.2
[developer@redbook01 bin]$ MP_RESD=poe oshrun -np 2 ./is.A.2
```

This section uses the Integer Sort kernel implementation of NPB3.2-SHMEM to demonstrate the use of OpenSHMEM with IBM Parallel Environment, and the effect of some configurations on the performance of the application.

The Integer Sort benchmark was compiled by using the **oshcc** compiler script of the IBM Parallel Environment and the IBM XL C compiler.

The workload class C of the Integer Sort benchmark was run as shown in Example 8-14.

8.4 Using IBM Spectrum LSF

IBM Spectrum LSF is a load and resources manager that enables shared access to cluster hosts while maximizing occupancy and the efficient use of resources.

IBM Spectrum LSF provides a queue-based and policy-driven scheduling system for a user's batch jobs that employs mechanisms to optimize resource selection and allocation based on the requirements of the application.

All development models that are described in Chapter 7, "Compilation, execution, and application development" on page 161 are fully supported by IBM Spectrum LSF. The preferred way to run applications in a production cluster is by using the IBM Spectrum LSF job submission mechanism. Also, you can manage any job. This section describes how to submit and manage jobs by using IBM Spectrum LSF commands.

8.4.1 Submit jobs

This section describes the job submission process. To submit a job to IBM Spectrum LSF, run the **bsub** command. By using IBM Spectrum LSF, you can submit the job by using command-line options, interactive command-line mode, or a control file. The tool provides the following options for fine-grained job management:

- Control input and output parameters.
- Define limits.
- Specify submission properties.

- Notify users.
- Control scheduling and dispatch.
- Specify resources and requirements.

The simplest way to submit a job is to use the command-line options, as shown in Example 8-15.

Example 8-15 IBM Spectrum LSF bsub command to submit a job by using command-line options

```
$ bsub -o %J.out -e %J.err -J 'omp_affinity' -q short './command'
Job <212> is submitted to queue <short>.
```

Example 8-15 shows some basic options of the **bsub** command. The flags specify standard output (**-o**), input (**-i**), and error (**-e**) files. As shown in Example 8-15, **-J** sets the job name, but it can also be used to submit multiple jobs (also know as an *array of jobs*), and **-q** sets the queue of which it can be a part. If the **-q** flag is not specified, the default queue is used (usually the `normal` queue). The last option that is shown in Example 8-15 is the application to be run.

Using the shell script is convenient when you must submit jobs regularly or that require many parameters. The file content that is shown in Example 8-16 is a regular shell script that embodies special comments (lines starts with `#BSUB`) to control the behavior of the **bsub** command, and runs the **noop** application by using the `/bin/sh` interpreter.

Example 8-16 Shell script to run IBM Spectrum LSF batch job

```
#!/bin/sh

#BSUB -o %J.out -e %J.err
#BSUB -J serial

./noop
```

You can run the **bsub myscript** or **bsub < myscript** commands to submit a script to IBM Spectrum LSF. In the first case, `myscript` is not spooled, which means that changes on the script take effect if the job is still running. Conversely, **bsub < myscrypt** (see Example 8-17) spools the script.

Example 8-17 IBM Spectrum LSF bsub command to submit a script job

```
$ bsub < noop_lsf.sh
Job <220> is submitted to default queue <normal>.
```

Considerations for OpenMP programs

You can use environment variables to control the running of OpenMP applications as described in 8.1.1, “Running OpenMP applications” on page 228. By default, the **bsub** command propagates all variables on the submitting host to the environment on the target machine.

Example 8-18 shows some OpenMP control variables (**OMP_DISPLAY_ENV**, **OMP_NUM_THREADS**, and **OMP_SCHEDULE**), which are exported to the environment before a job is scheduled to run on the p8r2n2 host. The content of the 230.err file, where errors are logged, shows that those variables are propagated on the remote host.

Example 8-18 Exporting OpenMP variables to IBM Spectrum LSF

```
$ export OMP_DISPLAY_ENV=true
$ export OMP_NUM_THREADS=20
$ export OMP_SCHEDULE="static"
$ bsub -m "p9r2n2" -o %J.out -e %J.err ./affinity
Job <230> is submitted to default queue <normal>.
$ cat 230.err

OPENMP DISPLAY ENVIRONMENT BEGIN
  _OPENMP = '201307'
  OMP_DYNAMIC = 'FALSE'
  OMP_NESTED = 'FALSE'
  OMP_NUM_THREADS = '20'
  OMP_SCHEDULE = 'STATIC'
  OMP_PROC_BIND = 'FALSE'
  OMP_PLACES = ''
  OMP_STACKSIZE = '70368222510890'
  OMP_WAIT_POLICY = 'PASSIVE'
  OMP_THREAD_LIMIT = '4294967295'
  OMP_MAX_ACTIVE_LEVELS = '2147483647'
  OMP_CANCELLATION = 'FALSE'
  OMP_DEFAULT_DEVICE = '0'
OPENMP DISPLAY ENVIRONMENT END
```

If you do not want to export OpenMP environment variables, you can use the **-env** option to control how **bsub** propagates them. For example, the same results that are shown in Example 8-18 can be achieved by running the following command, but without exporting any variables:

```
$ bsub -m "p9r2n2" -o %J.out -e %J.err -env "all, OMP_DISPLAY_ENV=true,
OMP_NUM_THREADS=20,  OMP_SCHEDULE='static'" ./affinity
```

Example 8-19 shows how the environment variables can be set in a job script to control the OpenMP behavior.

Example 8-19 Exporting OpenMP variables to the IBM Spectrum LSF job script

```
#!/bin/bash

#BSUB -J "openMP example"
#BSUB -o job_%J.out -e job_%J.err
#BSUB -q short
#BSUB -m p9r2n2

export OMP_NUM_THREADS=20
export OMP_SCHEDULE=static
export OMP_DISPLAY_ENV=true

./affinity
```

Considerations for Message Passing Interface programs

Use the **-n** option of the **bsub** command to allocate the number of tasks (or job slots) for the parallel application. Depending on the configuration of IBM Spectrum LSF, you can set job slots in terms of CPUs in the cluster. For example, the following command submits an MPI job with six tasks:

```
$ bsub -n 6 -o %J.out -e %J.err poe ./helloMPI
```

You can select a set of hosts for a parallel job by running the following **bsub** command options:

- Use the **-m** option to select hosts or groups of hosts.
- Use the **-R** option to perform resource-based selection with requirements expressions.
- Using the **-hostfile** option to indicate a host file. Do not use this option with the **-m** or **-R** options.

The following examples show the use of **-m** and **-R** to allocate hosts:

- In the following example, run two tasks of an MPI application on hosts `p8r1n1` and `p8r2n2`:

  ```
  $ bsub -n 2 -m "p9r1n1! p9r2n2" -o %J.out -e %J.err poe ./myAPP
  ```

- In the following example, the "!" symbol indicates that **poe** is first run on `p9r1n1`:

  ```
  $ bsub -n 4 -R "select[ncores==20] same[cpuf]" -o %J.out -e %J.err poe ./myApp
  ```

Run four tasks of an MPI application on hosts with 20 CPU cores (`select[ncores==20]`) if they have same CPU factor (`same[cpuf]`).

The job locality can be specified with the `span` string in a resources requirement expression (**-R** option). One of the following formats can be used to specify the locality:

- `span[hosts=1]`: Set to run all tasks on the same host.
- `span[ptile=`*`n`*`]`: Where *n* is an integer that sets the number of tasks per host.
- `span[block=`*`size`*`]`: Where *size* is an integer that sets the block size.

Enable job task affinity by using the `affinity` string in the resources requirement expression (**-R** option). The affinity applies to CPU or memory, and is defined in terms of processor units that are assigned per task (`core`, `numa`, `socket`, and `task`).

The following example features a 10-task MPI application where five tasks are allocated per host and each has four designated cores with binding by threads:

```
$ bsub -n 10 -R "select[ncores >= 20] span[ptile=5]
affinity[core(4):cpubind=thread]"  -o %J.out -e %J.err
```

Further processor unit specification makes the affinity expression powerful. For more information about affinity expressions in IBM Spectrum LSF, see the Affinity string page.

IBM Spectrum LSF integrates well with IBM Parallel Environment Runtime Edition and supports the running of parallel applications through **poe**. Some configurations in IBM Spectrum LSF are required. For more information, see 5.6.14, "IBM Spectrum LSF integration with Cluster Systems Management" on page 122.

Example 8-20 includes a IBM Spectrum LSF job script to run an MPI program with **poe**. (The IBM Parallel Environment environment variables are going to take effect.)

Example 8-20 IBM Spectrum LSF job script to submit a simple IBM Parallel Environment application

```
#!/bin/bash

#BSUB -J "MPILocalRank"    # Set job name
#BSUB -o lsf_job-%J.out    # Set output file
```

```
#BSUB -e lsf_job-%J.err    # Set error file
#BSUB -q short             # Set queue
#BSUB -n 5                 # Set number of tasks

export MP_INFOLEVEL=1
export LANG=en_US
export MP_RESD=POE
poe ./a.out
```

IBM Spectrum LSF provides a native network-aware scheduling of the IBM Parallel Environment parallel application through the **`-network`** option of the **`bsub`** command. That option encloses the attributes that are listed in Table 8-2.

Table 8-2 IBM Spectrum LSF bsub network attributes for IBM Parallel Environment

Attribute	Description	Values
`type`	Manage network windows reservation.	► `sn_single` (Reserves windows from one network for each task.) ► `sn_all` (Reserves windows from all networks for each task.)
`protocol`	Set the messaging API in use.	► `mpi` ► `shmem` ► `pami`
`mode`	The network type.	► `US` ► `IP`
`usage`	The network adapter usage among processes.	► `shared` ► `dedicated`
`instance`	Number of instances for the reservation window.	Positive integer number

The following command submits a two-task (**`-n 2`**) MPI (`protocol=mpi`) job. It shares the network adapters (`usage=shared`) and reserves windows on all of them.

```
$ bsub -n 2 -R "span[ptile=1]" -network "protocol=mpi:type=sn_all:
instances=2:usage=shared" poe ./myApp
```

Considerations for CUDA programs

You can use graphics processing unit (GPU) resources mapping when requirements expressions are used to allocate hosts and express usage reservation. If IBM Spectrum LSF is configured (see 5.6.14, “IBM Spectrum LSF integration with Cluster Systems Management” on page 122), the following resource fields are available:

- `ngpus`: Total number of GPUs
- `ngpus_shared`: Number of GPUs in share mode
- `ngpus_excl_t`: Number of GPUs in exclusive thread mode
- `ngpus_excl_p`: Number of GPUs in exclusive process mode

The `ngpus` resource field can be used in requirement expressions (**`-R`** option) for learning the number of GPUs that are needed to run a CUDA application. The following command submits a job to any host with one or more GPU:

```
$ bsub -R "select [ngpus > 0]" ./cudaApp
```

In terms of usage, the **ngpus** resource field can reserve GPUs by setting **ngpus_shared** (number of shared), **ngpus_excl_t** (number of GPUs on exclusive thread mode), or **ngpus_excl_p** (number of GPUs on exclusive process mode) resources. Use the **rusage -R** option of the **bsub** command to reserve GPU resources.

In Example 8-21, the job script sets one GPU in shared mode to be used by a `cudaCUDA` application.

Example 8-21 Script to set one GPU in shared mode to be used by a CUDA application

```
#!/bin/bash

#BSUB -J "HelloCUDA"
#BSUB -o helloCUDA_%J.out -e helloCUDA_%J.err
#BSUB -R "select [ngpus > 0] rusage [ngpus_shared=1]"

./helloCUDA
```

Regarding exclusive use of GPUs by parallel applications, set the value of **ngpus_excl_t** or **ngpus_excl_p** to change the run mode properly. The following example runs a parallel two-task (**-n 2**) application in one host (`span[hosts=1]`), where each host reserves two GPUs on exclusive process mode (`rusage[ngpus_excl_p=2]`):

```
$ bsub -n 2 -R “select[ngpus > 0] rusage[ngpus_excl_p=2] span[hosts=1]” poe
./mpi-GPU_app
```

Considerations for OpenSHMEM

You can use the **-network** option of the **bsub** command to set the parallel communication protocol of the application. For an OpenSHMEM application, you can set it by running the following command:

```
$ bsub -n 10 -network ”protocol=shmem” poe ./shmemApp
```

8.4.2 Managing jobs

IBM Spectrum LSF provides a set of commands to manage batch jobs. This section introduces the most common commands.

Modifying a job

The batch job can assume several statuses throughout its lifecycle. The following statuses are the most common ones:

- PEND: Waiting to be scheduled status
- RUN: Running status
- PSUSP, USUSP, or SSUSP: Suspended status
- DONE or EXIT: Terminated status

You can modify the submission parameters of a job during the pending or running statuses. To change them, use the **bmod** command.

You can change most submission parameters by running the **bmod** command. The command can include an option to cancel the option, reset the option to its default value, or override the option.

To override a submission parameter, use the same option as in **bsub**. In the following example, **-o "%J_noop.out"** changes the output file of the job with identifier 209:

```
$ bmod -o "%J_noop.out" 209
Parameters of job <209> are being changed
```

To cancel a submission parameter, append ***n*** to the option. For example, **-Mn** removes the memory limits.

For more information about the **bmod** command, see the bmod page.

Canceling a job

To cancel one or more jobs, use the **bkill** command. This command by default sends the SIGINT, SIGTERM, and SIGKILL signals in sequence on Linux. The time interval can be configured in the `lsb.params` configuration file. In reality, **bkill -s *<signal>*** sends the *<signal>* signal to the job.

The user can cancel only their own jobs. The root and IBM Spectrum LSF administrators can end any job.

In the following example, the job with identifier 217 is ended:

```
$ bkill 217
Job <217> is being terminated
```

Suspending and resuming a job

To suspend one or more unfinished jobs, run the **bstop** command. In Linux, the command sends the SIGSTOP signal to serial jobs and the SIGTSTP signal to parallel jobs. As an alternative, the **bkill -s SIGSTOP** or the **bkill -s SIGTSTP** commands send the stop signal to a job.

In Example 8-22, the job with identifier 220 did not finish (status `RUN`) when it is stopped (`bstop 220`). As a result, bjobs 220 shows that it is now in a suspended status (`USUSP`).

Example 8-22 IBM Spectrum LSF bstop command to suspend a job

```
$ bjobs 220
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
220     wainers RUN   normal     p8r1n1      p9r2n2      serial     Apr 17 16:06
$ bstop 220
Job <220> is being stopped
$ bjobs 220
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
220     wainers USUSP normal     p9r1n1      p9r2n2      serial     Apr 17 16:06
```

To resume a job, use the **bresume** command. In Linux, the command sends a SIGCONT signal to the suspended job. To stop or end a job, run **bkill -s** to send the continue signal **bkill -s CONT**. The following command shows how to resume the job that was stopped in Example 8-22:

```
$ bresume 220
Job <220> is being resumed
```

8.5 IBM Spectrum LSF Job Step Manager

IBM Spectrum LSF Job Step Manager (JSM) is a lightweight job step scheduler that can help you manage available compute resources that are provided through an IBM Spectrum LSF allocation. IBM Spectrum LSF is the high-level resource manager, although JSM manages users' IBM Spectrum LSF allocated resources.

JSM separates the concepts of resource allocation and tasks. This might be a new concept for users who are familiar with launchers that often have complicated methods of determining resources that are based, at least in part, on the number of tasks to be created. Under JSM, each launch is run on a number of resource sets. The resource sets are defined as follows:

- A resource set contains 1 or more tasks.
- A resource set contains 1 or more cores.
- A resource set contains 0 or more GPUs.
- A resource set contains 1 or more bytes of memory.
- All resources in a resource set must be on the same physical node.
- All resource sets that are used by a job step must contain an identical number of each resource. For example, one resource cannot contain three CPUs and another resource contain only one.

The JSM utilities are:

- **jsrun**: Create a job step or reservation.
- **jslist**: List running, completed, or killed job steps.
- **jskill**: Signal a running job step.
- **jswait**: Wait for completion of a job step.

Our Power AC922 nodes are composed of two sockets of 22 CPU cores, six GPUs, and 512 GB of main memory. We submit an interactive IBM Spectrum LSF job asking for two nodes, as shown in Example 8-23.

Example 8-23 LSF job asking for two compute nodes

```
$ bsub -q excl_int  -nnodes 2 -Is /bin/bash
Job <200743> is submitted to queue <excl_int>.
<<Waiting for dispatch ...>>
<<starting on c699launch01>>
```

The job is launched by a launch job with two compute nodes that are assigned. Some example job steps are shown in Table 8-3.

Table 8-3 Sample job steps

Description	Command	Layout
One task per core	**jsrun ./a.out** **jsrun --cpu_per_rs 1 --nrs 88 --np 88 ./a.out**	► One task per core from all assigned nodes.
One task, one GPU, and four cores	**jsrun --cpu_per_rs 4 --gpu_per_rs 1 --nrs 1 --np 1 ./a.out**	► One task, four CPUs, and GPU on one node.
One task per core, three GPUs per task, and one task per node	**jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 hostname ./a.out**	► Two tasks, one per node, one CPU per task, and three GPUs per task.

Description	Command	Layout

Sixty-four MPI tasks and no GPUs	**`jsrun --nrs 64./a.out`**
	`jsrun --nrs 64 --rs_per_host 32 ./a.out`
	`jsrun --nrs 4 --tasks_per_rs 16 --cpu_per_rs 16 --rs_per_host 2 ./a.out`
		► Two nodes, 44 tasks on first node, and 22 tasks on second node. The first node has all its cores occupied, the second node has all its cores on the first socket occupied.
		► Two nodes, 32 tasks on the first node, 32 tasks on the second node. Both nodes have all the cores on the first socket occupied, and 10 cores on the second socket occupied.
		► Two nodes, 1 resource set on each socket, and 16 tasks on each socket. Fully balanced between nodes and sockets. Six free cores on each socket.

Example 8-24 shows the batch submission for the job.

Example 8-24 Full batch submission by using Example 8-23 on page 254

```
$ cat > jobscript.bsub << 'EOF'
#! /bin/bash -
### Begin BSUB Options
#BSUB -J job-name
#BSUB -W 10:00
#BSUB -nnodes 2
#BSUB -alloc_flags "smt4"
#BSUB -o  "output.%J"
### End BSUB Options and begin shell commands

export OMP_NUM_THREADS=4
jsrun --nrs 4 --tasks_per_rs 16 --cpu_per_rs 16 --rs_per_host 2 ./a.out
EOF

$ bsub < jobscript.bsub
Job <202078> is submitted to default queue <excl>.
```

A single IBM Spectrum LSF job can contain multiple job steps. For example, if you have 10 GPU jobs that are each using three GPUs, submit them as a single IBM Spectrum LSF job.

As a more complicated set of job steps, Example 8-25 shows how to run a batch of job steps in one job.

Example 8-25 Full batch submission by using Example 8-24

```
$ cat > jobscript.bsub << 'EOF'
#! /bin/bash -
### Begin BSUB Options
#BSUB -J job-name
#BSUB -W 10:00
```

```
#BSUB -nnodes 2
#BSUB -alloc_flags "smt4"
#BSUB -o  "output.%J"
### End BSUB Options and begin shell commands

jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-1 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-2 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-3 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-4 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-5 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-6 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-7 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-8 &
jsrun --cpu_per_rs 1 --gpu_per_rs 3 --rs_per_host 1 ./a.out-9 &

# List all submitted job steps
jslist

# Wait for all job steps to complete before exiting:
jswait all
EOF

$ bsub < jobscript.bsub
Job <202094> is submitted to default queue <excl>.
```

8.6 Running tasks with IBM Spectrum MPI

Section 7.6.11, “Using IBM Spectrum MPI” on page 222 describes using IBM Spectrum MPI to run jobs in a compute node by running the **mpirun** command. This section shows several other features of running IBM Spectrum MPI jobs by running **mpirun**.

Portable Hardware Locality (hwloc)

A useful API to discover the physical and logical architecture of your POWER8 processor-based server is `hwloc` (Portable Hardware Locality). This API helps identify and display information about NUMA memory nodes, sockets, shared caches, cores, SMT, system attributes, and the locality of I/O.

For example, by using `hwloc` you can use the **--report-bindings** flag with the **mpirun** command with any IBM Spectrum MPI program to fetch a hardware description, as shown in Example 8-26.

Example 8-26 Using the --report-bindings flag with the mpirun command

```
$ [desnesn@c712f7n03 examples]$ /opt/ibm/spectrum_mpi/bin/mpirun -np 1
--report-bindings -pami_noib ./trap.mpi
[c712f7n03:66785] MCW rank 0 bound to socket 0[core 0[hwt 0-7]], socket 0[core
1[hwt 0-7]], socket 0[core 2[hwt 0-7]], socket 0[core 3[hwt 0-7]], socket 0[core
4[hwt 0-7]], socket 0[core 5[hwt 0-7]], socket 0[core 6[hwt 0-7]], socket 0[core
7[hwt 0-7]], socket 0[core 8[hwt 0-7]], socket 0[core 9[hwt 0-7]]:
[BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/
BBBBBBBB][......../......../......../......../......../......../......../......../
......../........]
```

```
Estimate of local_sum of x^2 = 18.000229
Interval [-3.000,3.000]
Using 10000 trapezoids
```

Look at the last line that is presented:

```
[BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/
BBBBBBBB][......../......../......../......../......../......../......../......../
......../........]
```

This line means that two sockets are available, each with 10 cores, and each core with eight hyper-threads. This configuration is compliant with the running of **`nproc`**, which presents a value of 160. Also, the sets of the letter B means that the MPI processes are bound to the first socket.

GPU support

Another valuable feature of IBM Spectrum MPI is that is support running GPU-accelerated applications over CUDA-aware MPI by using the CUDA Toolkit V9.2.

Note: By default, GPU support is turned off. To turn it on, use the **`-gpu`** flag with the **`mpirun`** command.

Sharing work with different hosts in your network

To distribute instances of a IBM Spectrum MPI program by running **`mpirun`**, run the following command:

```
$ mpirun -host h1,h2 prog1
```

This command runs one instance of `prog1` on host `h1` and host `h2`.

However, you can use **`mpirun`** to run jobs in an SPMD manner. For example, if you want host `h2` to receive three instances of `prog1`, run **`mpirun`** as follows:

```
$ mpirun -host h1, h2, h2, h2 prog1
```

You also can perform SPMD by using a host file, as shown in Example 8-27.

Example 8-27 Performing SPMD by using a host file

```
$ cat hosts
c712f7n02.somedomain.ibm.com
c712f7n03.somedomain.ibm.com
$ mpirun -np 2 --host file hosts prog2
```

Also, you can use **`mpirun`** to run jobs in an MPMD manner, which you can perform by using a host file, as shown in the following command:

```
$ mpirun -np 2 prog3 : -np 3 prog4
```

This command includes two instances of `prog3` and three instances of `prog4`.

For more information about how to use **`mpirun`**, use the **`-h`** (help) option with the command, or see the IBM Spectrum MPI Version 10 Release 1.0.2 User's Guide.

9

Measuring and tuning applications

This chapter describes how to measure and tune applications.

The following topics are described in this chapter:

- Effects of basic performance tuning techniques
- General methodology of performance benchmarking
- Sample code for the construction of thread affinity strings
- IBM Engineering and Scientific Subroutine Library performance results
- GPU tuning
- Application development and tuning tools

9.1 Effects of basic performance tuning techniques

This section evaluates the effects of basic performance tuning techniques. It uses the NAS Parallel Benchmarks (NPBs)[1] suite of applications.

The NPB suite was originally used for complex performance evaluation of supercomputers. The developers of the NPB programs distilled the typical computational physics workloads and put the most widespread numerical kernels into their product.

This section uses the Open Multi-Processing (OpenMP) types of NPB benchmarks to achieve the following goals:

- Show the performance variation for the different simultaneous multithreading (SMT) modes.
- Provide guidance for the choice of compilation parameters.

Demonstrate the importance of binding threads to logical processors.The benchmarking used a 44-core IBM Power System AC922 (model 8335-GTW) server based on the POWER9 processor. The processor base frequency was 2.3 GHz. The Linux scaling governors were set to `performance` so that the effective frequency increased to 3.8 GHz.

Each of 16 memory slots was populated with 16 GB DDR4 modules for a total of 512 GB. The server was running Red Hat Enterprise Linux for IBM Power Little Endian V7.5. The operating system (OS) was installed in a non-virtualized mode. The Linux kernel version was 4.14.0-49.15.1.el7a. Version 16.1.0 of the IBM Xpertise Library (XL) Fortran compiler was used to compile the sources.

Note: The performance numbers that are shown in this section must not be treated as the official results. They are provided to demonstrate the possible effect of various performance tuning techniques on application performance. The results that are obtained in different hardware and software environments or with other compilation parameters can vary widely from the numbers that are shown here.

The size of the problems in the NPB benchmarking suite is predefined by the developers. The example uses the benchmarks of class C. The source code of the NPB suite was not changed. The code generation was controlled by setting compilation parameters in a `make.def` file, as shown in Example 9-1.

Example 9-1 NPB: A sample make.def file for the -O3 parameter set

```
F77 = xlf_r -qsmp=noauto:omp -qnosave
FLINK = $(F77)
FFLAGS = -O3 -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
FLINKFLAGS = $(FFLAGS)
CC = xlc_r -qsmp=noauto:omp
CLINK = $(CC)
C_LIB = -lm
CFLAGS = $(FFLAGS)
CLINKFLAGS = $(CFLAGS)
UCC = xlc_r
BINDIR = ../O3
RAND = randi8
```

[1] The NPB suite was developed by NASA Advanced Supercomputing (NAS) Division. For more information, see NAS Parallel Benchmarks.

```
WTIME = wtime.c
MACHINE = -DIBM
```

It is not feasible to cover all the compilation parameters, so the tests varied only the level of optimization. The following parameters were common for all builds:

- **`-qsmp=noauto:omp`**
- **`-qnosave`**
- **`-qmaxmem=-1`**
- **`-qarch=auto`**
- **`-qtune=auto:balanced`**

The example considers four sets of compilation parameters. In addition to the previous compilation parameters, the remaining parameters are listed in Table 9-1. The column "Option set name" lists shortcuts that were used to reference each set of compilation parameters that are presented in the "Compilation parameters" columns.

Table 9-1 NPB: Compilation parameters that were used to build the NPB suite executable files

Option set name	Compilation parameters	
	Varying	Common

-O2	`-O2`	**`-qsmp=noauto:omp`**
		`-qnosave`
		`-qmaxmem=-1`
		`-qarch=auto`
		`-qtune=auto:balanced`

-O3	`-O3`	
-O4	`-O4`	
-O5	`-O5`	

All of the runs were performed with the following environment variables:

- **`export OMP_DYNAMIC="FALSE"`**
- **`export OMP_SCHEDULE="static"`**

The benchmarks were submitted through IBM Spectrum Load Sharing Facility (IBM Spectrum LSF) by using **`jsrun`** as the job launcher. IBM Spectrum LSF configured the node for the needed SMT level, and **`jsrun`** made sure to pack the requested OpenMP threads into all available CPUs in each core.

Example 9-2 shows how to submit a job to run on 22 CPU cores with SMT-2 by using 44 virtual CPUs.

Example 9-2 IBM Spectrum LSF job submission for running benchmarks with 22 cores in SMT-2 mode

```
$ cat jobscript
#! /bin/bash
#BSUB -q excl
#BSUB -J NPB-22cpu-smt2
#BSUB -nnodes 1
#BSUB -alloc_flags "smt2"
#BSUB -o NPB-22cpu-smt2.%J
### End BSUB Options and begin shell commands

export OMP_DYNAMIC="FALSE"
export OMP_SCHEDULE="static"

# 22 cpus with SMT2 = 44 available cpus.
export OMP_NUM_THREADS=44
```

```
cd /home/janfrode/code/NPB3.3.1/NPB3.3-OMP
for O in 02 03 04 05 ; do
        for i in bt cg ep ft is lu mg sp ua ; do
                echo ========= ./${O}/${i}.C.x  ============
                jsrun -n 1 -a 1 -c 22 -r 1 -d cyclic -b rs ./${O}/${i}.C.x
         done
done

$ bsub < jobscript
Job <195755> is submitted to queue <excl>.
```

Note: In several cases, performance results that are presented deviate significantly from the general behavior within the same plot. The plot bars with deviations can be ignored because the tests can experience unusual conditions (OS jitters, parasitic external workload, and so on).

9.1.1 Performance effect of a rational choice of SMT mode

The POWER9 processor-based core can run instructions from up to four application threads simultaneously. This capability is known as *SMT*. The POWER9 processor-based architecture supports the following three multithreading levels:

- ST (single-thread)[2]
- SMT-2 (two-way multithreading)
- SMT-4 (four-way multithreading)

The SMT mode, which can be used to obtain the optimal performance, depends on the characteristics of the application. Compilation parameters and mapping between application threads and logical processors can also affect the timing.

The reasons behind a conscious choice of an SMT mode

Execution units of a core are shared by all logical processors of a core (two logical processors in SMT-2 mode and four logical processors in SMT-4 mode). Execution units are available with multiple instances (for example, load and store units and fixed-point units) and single instances (for example, branch execution unit).

In ideal conditions, application threads do not compete for execution units. This configuration results in each of the four logical processors of a core that is running in SMT-4 mode being almost as fast as a core that is running in ST mode.

Depending on the application threads instruction flow, some execution units become fully saturated with instructions that come from different threads. As a result, the progress of the dependent instructions is postponed. This postponement limits the overall performance of the four logical processors of a core that is running in SMT-4 to the performance of a core that is running in ST mode.

It is also possible for even a single thread to saturate fully the resources of a whole core. Therefore, adding threads to a core can result in performance degradation. For example, see the performance of the `mg.C` benchmark that is shown in Figure 9-9 on page 268.

[2] Single-thread mode is referred sometimes as SMT-1.

Note: Generally, the performance of each logical processor of a core that is running in SMT-2 or SMT-4 mode is not equivalent to the performance of a core that is running in ST mode. The performance of each logical processor of a core is influenced by all other logical processors in a core. The influence comes from the application threads instruction flow.

Performance effect of SMT mode on NPB benchmarks

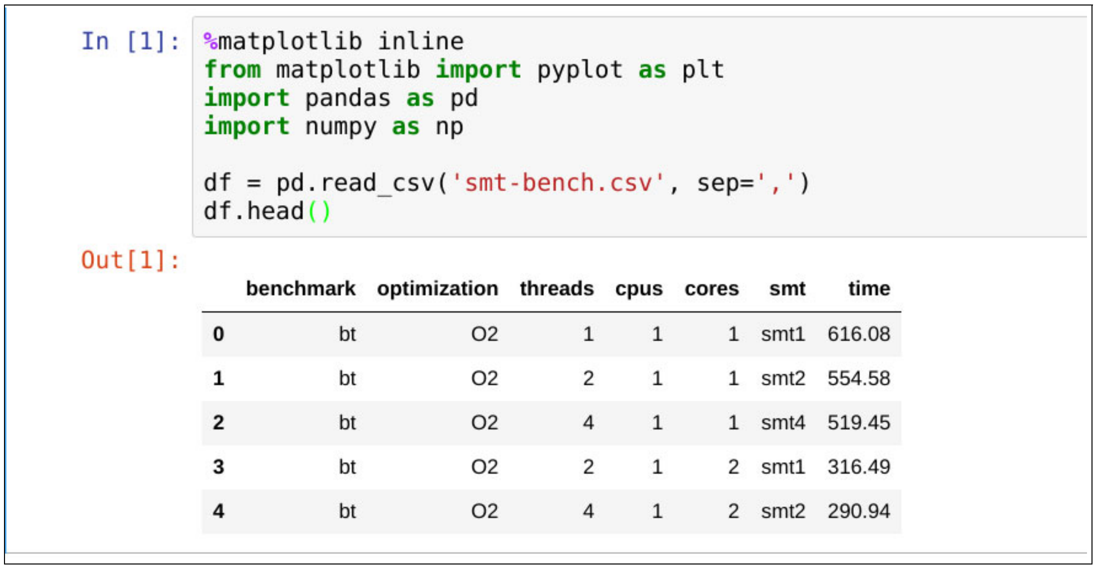
The bar charts that are shown in Figure 9-3 on page 265, Figure 9-4 on page 266, Figure 9-5 on page 266, Figure 9-6 on page 267, Figure 9-7 on page 267, Figure 9-8 on page 268, Figure 9-9 on page 268, Figure 9-10 on page 269, and Figure 9-11 on page 269 show the performance benefits that come from a rational choice of SMT mode for applications from the NPB suite (`bt.C`, `cg.C`, `ep.C`, `ft.C`, `is.C`, `lu.C`, `mg.C`, `sp.C`, and `ua.C`). The performance of the applications was measured for each combination of the following options:

- Four sets of compiler parameters, as listed in Table 9-1 on page 261.
- Six core layouts:
  - 1, 2, 5, 10, and 22 cores from one socket
  - A total of 44 cores from both sockets

The plots are organized according to the following scheme:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a particular set of compiler options.
- The horizontal axis lists core layouts. We packed cores into as few sockets as possible so that for 1 - 22 cores we used only one socket, and for 44 cores we used two sockets.
- The vertical axis shows the performance gain as measured in percentage relative to a baseline. As a baseline, we choose the slowest SMT mode for that particular application run.

To explore the results of the benchmarks, we log the settings and results of every run to a CSV file, and use a Jupyter Notebook to work with the data, as shown in Figure 9-1.



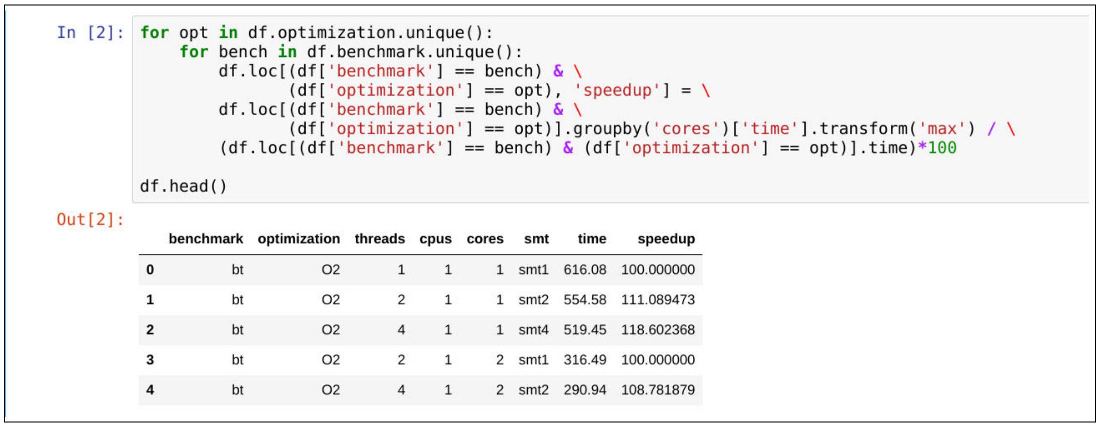
In [1]:
```
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np

df = pd.read_csv('smt-bench.csv', sep=',')
df.head()
```

Out[1]:

	benchmark	optimization	threads	cpus	cores	smt	time
0	bt	O2	1	1	1	smt1	616.08
1	bt	O2	2	1	1	smt2	554.58
2	bt	O2	4	1	1	smt4	519.45
3	bt	O2	2	1	2	smt1	316.49
4	bt	O2	4	1	2	smt2	290.94

Figure 9-1 Exploring the benchmark results

Now, we can add a “speedup” column with the calculation of relative speedup compared to slowest SMT mode, as shown in Figure 9-2.



```
In [2]: for opt in df.optimization.unique():
            for bench in df.benchmark.unique():
                df.loc[(df['benchmark'] == bench) & \
                       (df['optimization'] == opt), 'speedup'] = \
                df.loc[(df['benchmark'] == bench) & \
                       (df['optimization'] == opt)].groupby('cores')['time'].transform('max') / \
                (df.loc[(df['benchmark'] == bench) & (df['optimization'] == opt)].time)*100

        df.head()
Out[2]:
```

	benchmark	optimization	threads	cpus	cores	smt	time	speedup
0	bt	O2	1	1	1	smt1	616.08	100.000000
1	bt	O2	2	1	1	smt2	554.58	111.089473
2	bt	O2	4	1	1	smt4	519.45	118.602368
3	bt	O2	2	1	2	smt1	316.49	100.000000
4	bt	O2	4	1	2	smt2	290.94	108.781879

Figure 9-2 Adding the speedup column

Figure 9-3 on page 265, Figure 9-4 on page 266, Figure 9-5 on page 266, Figure 9-6 on page 267, Figure 9-7 on page 267, Figure 9-8 on page 268, Figure 9-9 on page 268, Figure 9-10 on page 269, and Figure 9-11 on page 269 create plots showing how the choice of SMT mode affects the performance. The complete Jupyter source code for these plots is listed in Example 9-3.

Example 9-3 Source code for Jupyter Notebook data exploration

```
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
import numpy as np
import seaborn
seaborn.set()
from itertools import cycle, islice
my_colors = list(islice(cycle(['#003ac7',  '#00b89f', '#0063ff', '#008471']),
None, len(df)))


df = pd.read_csv('smt-bench.csv', sep=',')
df.head() = {

# Next cell

for opt in df.optimization.unique():
    for bench in df.benchmark.unique():
        df.loc[(df['benchmark'] == bench) & \
               (df['optimization'] == opt), 'speedup'] = \
        df.loc[(df['benchmark'] == bench) & \
               (df['optimization'] ==
opt)].groupby('cores')['time'].transform('max') / \
        (df.loc[(df['benchmark'] == bench) & (df['optimization'] ==
opt)].time)*100

df.head()

# Next cell
```

```
for bench in df.benchmark.unique():
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,5))
    fig.suptitle("Benchmark: " + bench + ".C")

    for opt in df.optimization.unique():
        if opt == "O2":
            i=0
            j=0
        if opt == "O3":
            i=0
            j=1
        if opt == "O4":
            i=1
            j=0
        if opt == "O5":
            i=1
            j=1
        df2=df.loc[(df['benchmark'] == bench) & (df['optimization'] == opt)]
        plot = df2.pivot(index="cores", columns="smt").plot(kind="bar",
color=my_colors, sharex='col',  width=0.8, ax=axes[i,j],
ylim=(0.95*min(df2.speedup), 1.05*max(df2.speedup)), y="speedup", title=opt)
        plot.legend(loc=1, prop={'size': 8})
    fig.tight_layout()
    fig.show()
```

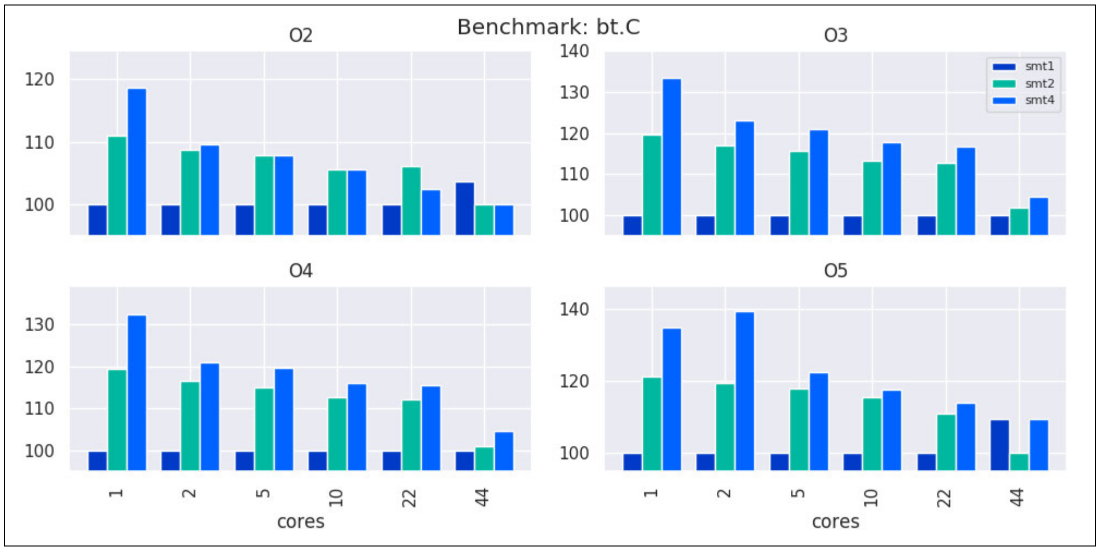


Figure 9-3   Performance benefits from a rational choice of SMT mode for the bt.C benchmark

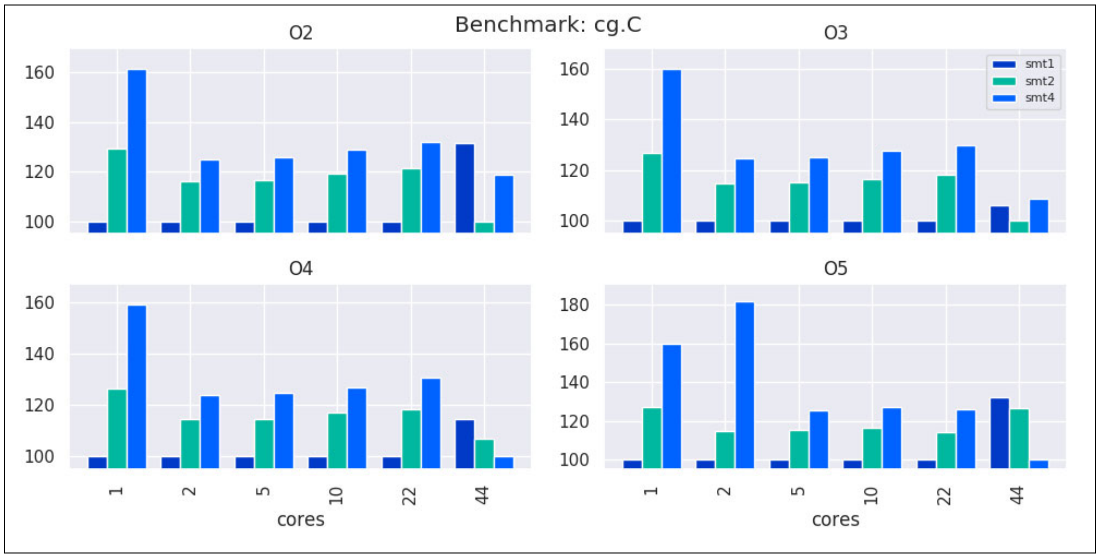


Figure 9-4 Performance benefits from a rational choice of SMT mode for the cg.C benchmark

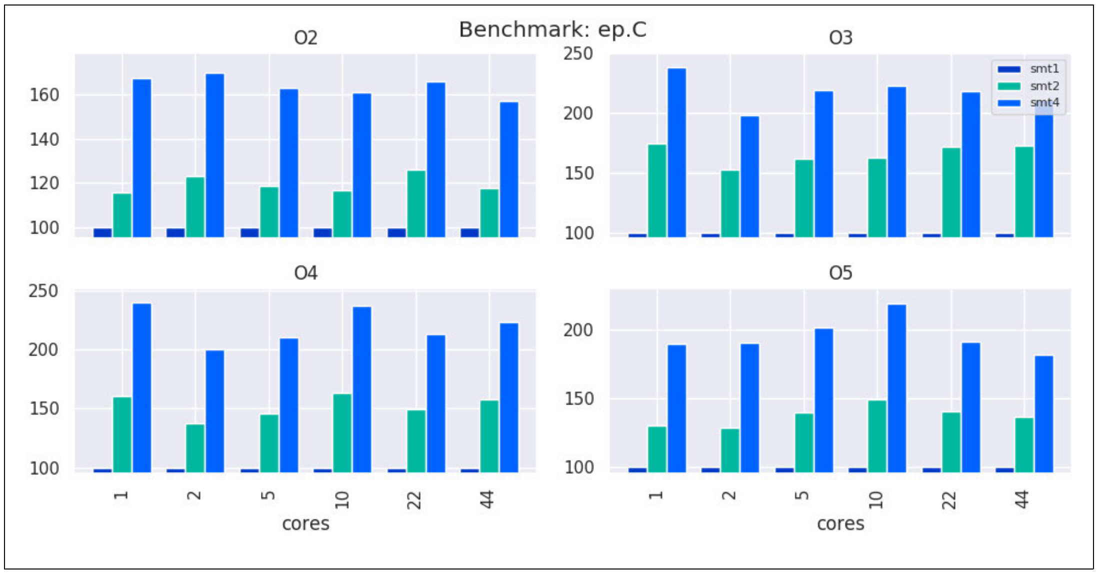


Figure 9-5 Performance benefits from a rational choice of SMT mode for the ep.C benchmark

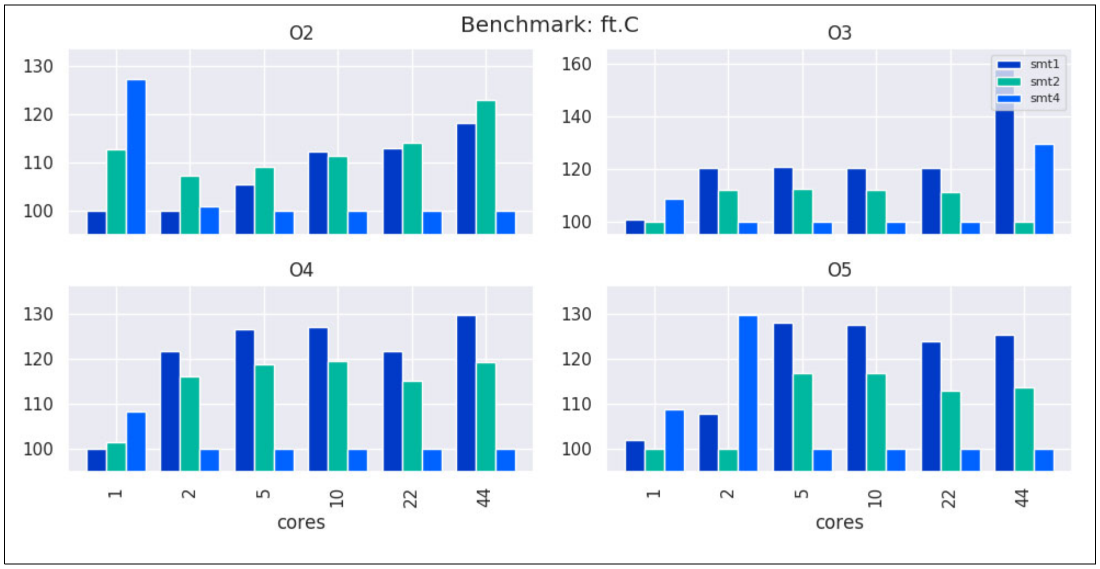


Figure 9-6 Performance benefits from a rational choice of SMT mode for the ft.C benchmark

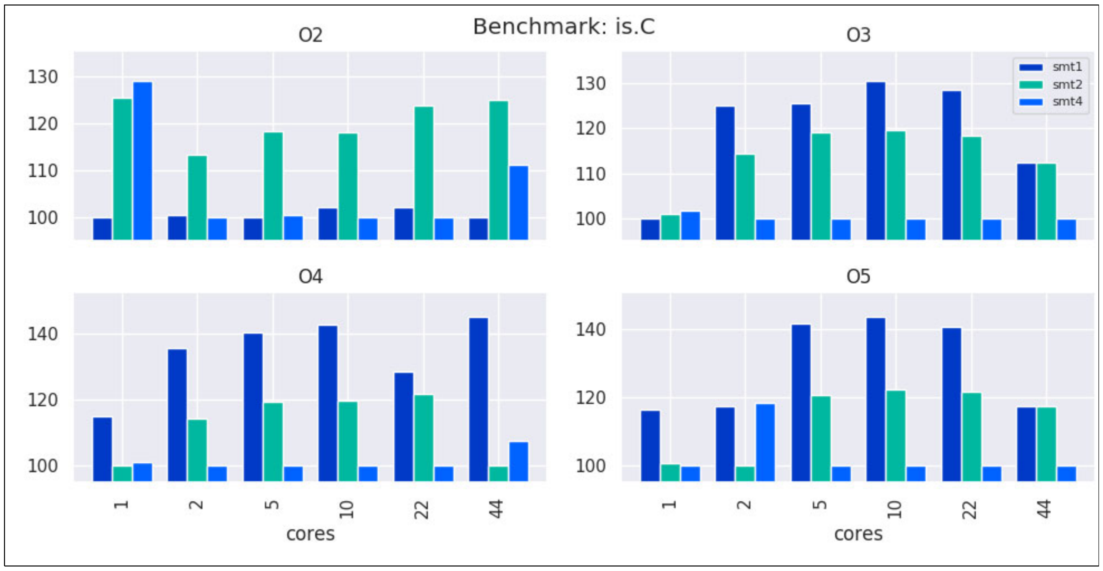


Figure 9-7 Performance benefits from a rational choice of SMT mode for the is.C benchmark

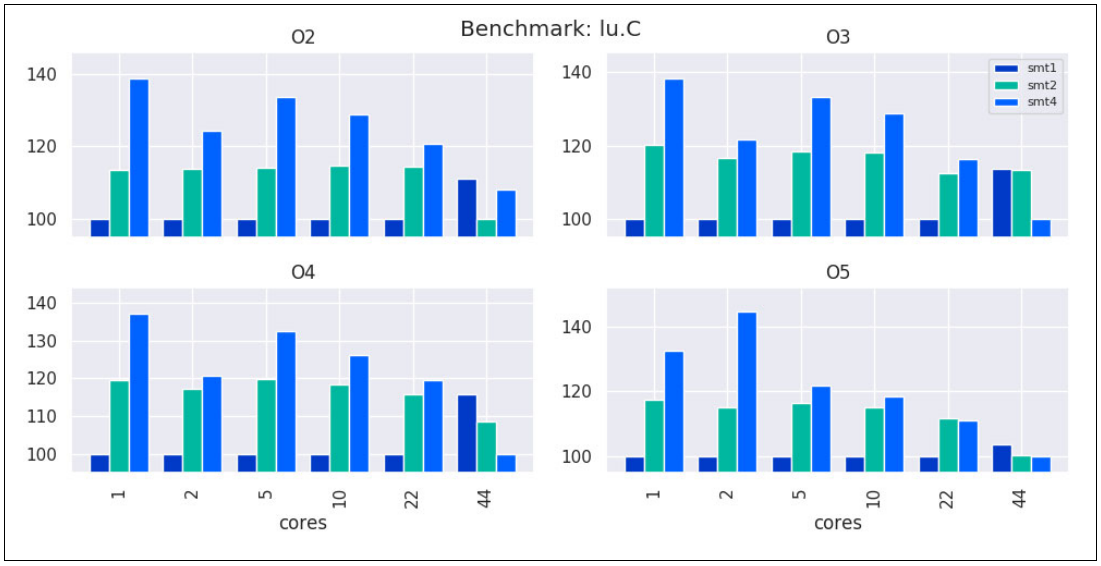


Figure 9-8 Performance benefits from a rational choice of SMT mode for the lu.C benchmark

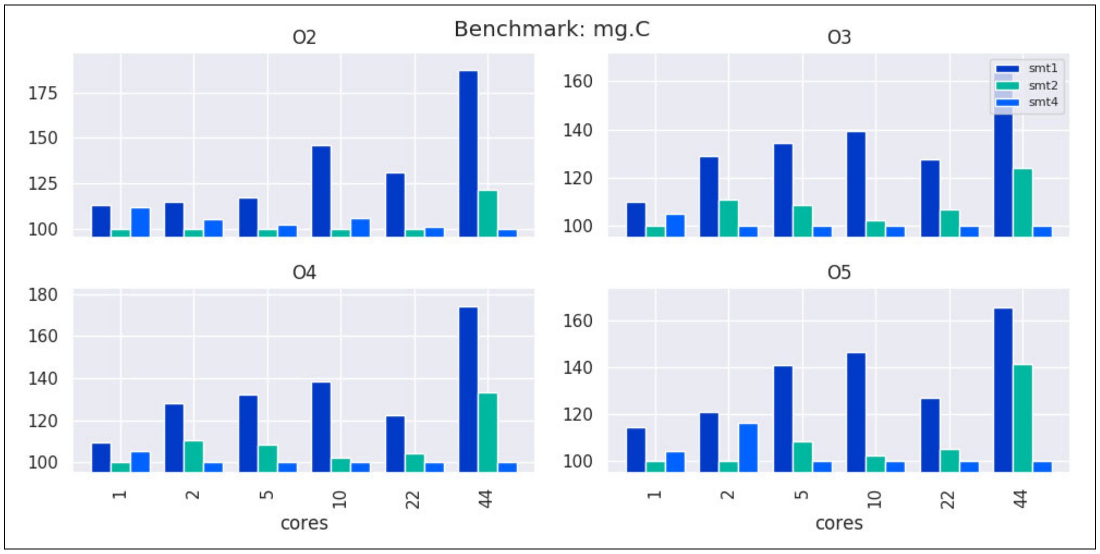


Figure 9-9 Performance benefits from a rational choice of SMT mode for the mg.C benchmark

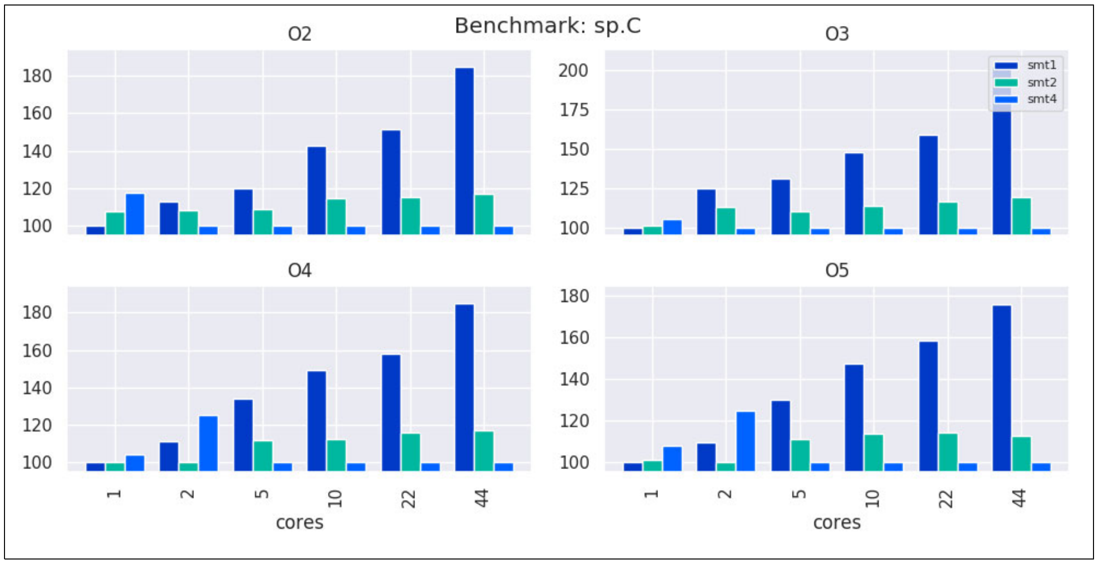


Figure 9-10 Performance benefits from a rational choice of SMT mode for the sp.C benchmark

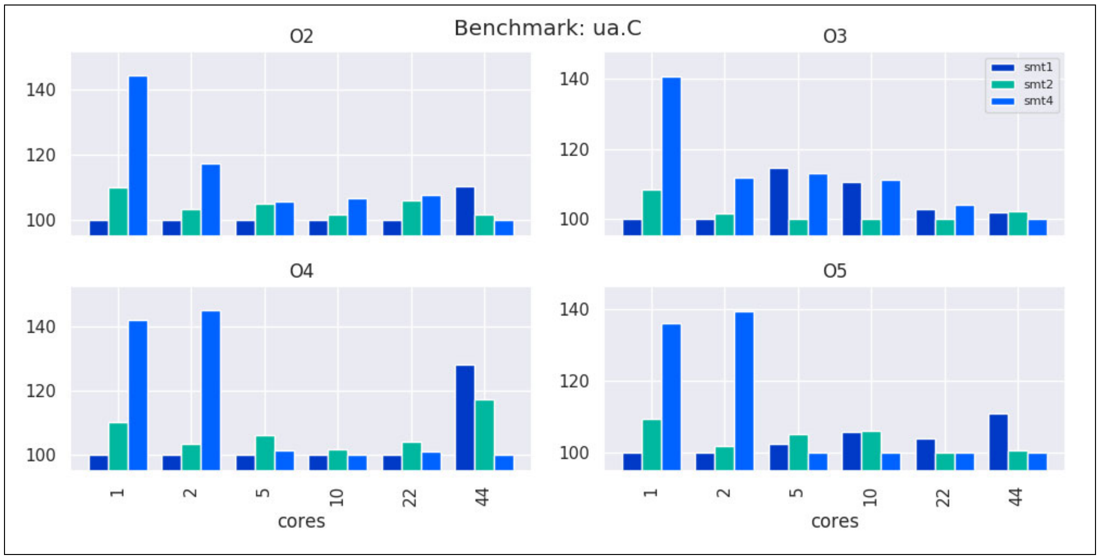


Figure 9-11 Performance benefits from a rational choice of SMT mode for the ua.C benchmark

Choosing the SMT mode for computing nodes

Many NPB applications benefit from SMT-4 mode. Therefore, from the general system management perspective, it can be unwise to restrict users to lower SMT values. The administrator should consider the following recommendations:

- Run the system in SMT-4 mode by default.
- Let the batch scheduler turn on SMT mode on a per job basis.
- Use a processor core as a unit of hardware resource allocation.[3]

[3] This recommendation is targeted at applications that are limited only by the computing power of a processor. It does not account for interaction with the memory subsystem and other devices. At some supercomputing sites and user environments, it can be reasonable to use a socket or even a server as a unit of hardware resource allocation.

By using the physical processor as a unit of hardware resource allocation, you ensure that no other applications use the idle logical processors of a physical core that is assigned to the user. Before the users run a productive workload, they must run several benchmarks for an application of their choice. The benchmarks are necessary to determine a favorable number of software threads to run at each core. After the value is identified, that number must be accounted for when users arrange productive workloads.

If possible, recompile the application with the **`-qtune=pwr9:smt`*`X`*** option of the IBM XL compiler (where *X* is 1, 2, or 4, depending on the SMT mode), and repeat the benchmark.

Reasoning for Message Passing Interface applications

The same logic holds for applications that use Message Passing Interface (MPI). For programs that are based on OpenMPI, seek a favorable number of threads to run on each core. Similarly, for an application that is created with MPI, find a favorable number of MPI processes to run on each core.

9.1.2 Effect of optimization options on performance

Various compiler options affect the performance characteristics of produced binary code. However, not all of the optimization options are equally suited for all types of workloads. Compilation parameters that result in good performance for one type of application might not perform equally well for other types of computations. Choose a favorable set of compiler options that is based on the timing of a particular application.

The bar charts that are shown in Figure 9-12 on page 271, Figure 9-13 on page 271, Figure 9-14 on page 272, Figure 9-15 on page 272, Figure 9-16 on page 272, Figure 9-17 on page 272, Figure 9-18 on page 273, Figure 9-19 on page 273, and Figure 9-20 on page 273 compare the performance benefits that come from the rational choice of compiler options for the same applications from the NPB suite (`bt.C`, `cg.C`, `ep.C`, `ft.C`, `is.C`, `lu.C`, `mg.C`, `sp.C`, and `ua.C`) that are described in 9.1.1, “Performance effect of a rational choice of SMT mode” on page 262.

Again, the four sets of compiler options that are considered are listed in Table 9-1 on page 261. The application threads are bound to 1, 2, 5, 10, or 22 cores of one socket or 44 cores of two sockets.

The organization of the plots is similar to the scheme that was described in 9.1.1, “Performance effect of a rational choice of SMT mode” on page 262:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a particular SMT mode.
- The horizontal axis lists the number of cores that is used.
- The vertical axis shows the performance gain as measured in percentage relative to a baseline. As a baseline, we chose a compiler option set that is less favorable for that particular application run.

Again, we must add a column to our table (optSpeedup) that shows the relative speedup between the various optimizations, as shown in Example 9-4, and generate plots showing the effects of compiler optimization levels for the NPB applications.

Example 9-4 Source code for adding the optSpeedup column and generating new plots

```
for smt in df.smt.unique():
    for bench in df.benchmark.unique():
       df.loc[(df['benchmark'] == bench) & (df['smt'] == smt), 'optSpeedup'] = \
         df.loc[(df['benchmark'] == bench) & (df['smt'] ==
smt)].groupby('cores')['time'].transform('max') / \
        (df.loc[(df['benchmark'] == bench) & (df['smt'] == smt)].time)*100

for bench in df.benchmark.unique():
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(10,3))
    fig.suptitle("Benchmark: " + bench + ".C")

    for smt in df.smt.unique():
        if smt == "smt1":
            i=0
        if smt == "smt2":
            i=1
        if smt == "smt4":
            i=2
        df2=df.loc[(df['benchmark'] == bench) & (df['smt'] == smt)]
        plot = df2.pivot(index="cores", columns="optimization").plot(kind="bar", \
                color=my_colors, width=0.8, ax=axes[i],
ylim=(0.95*min(df2.optSpeedup), \
                1.05*max(df2.optSpeedup)), y="optSpeedup", title=smt)
        plot.legend(loc=1, prop={'size': 8})
    fig.tight_layout()
    fig.show()
```

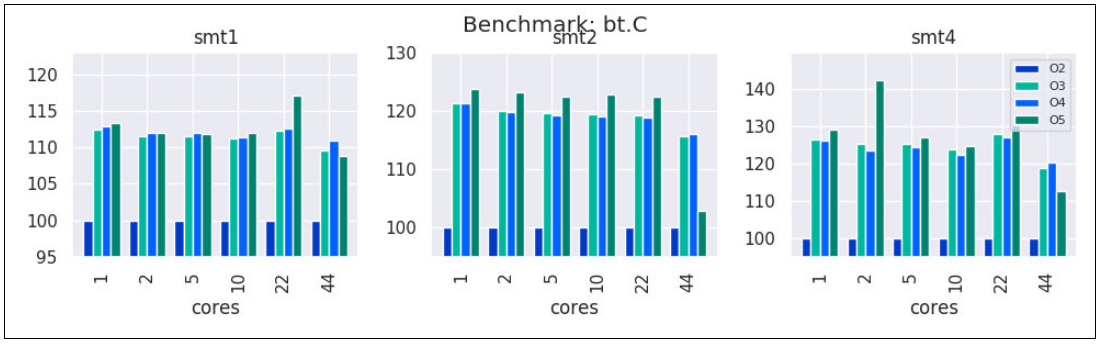


Figure 9-12 Performance gain from a rational choice of compiler options for the bt.C benchmark

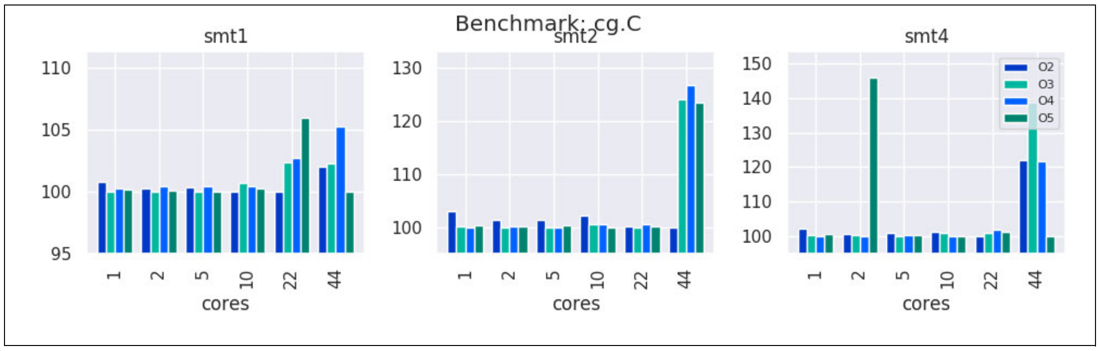


Figure 9-13 Performance gain from a rational choice of compiler options for the cg.C benchmark

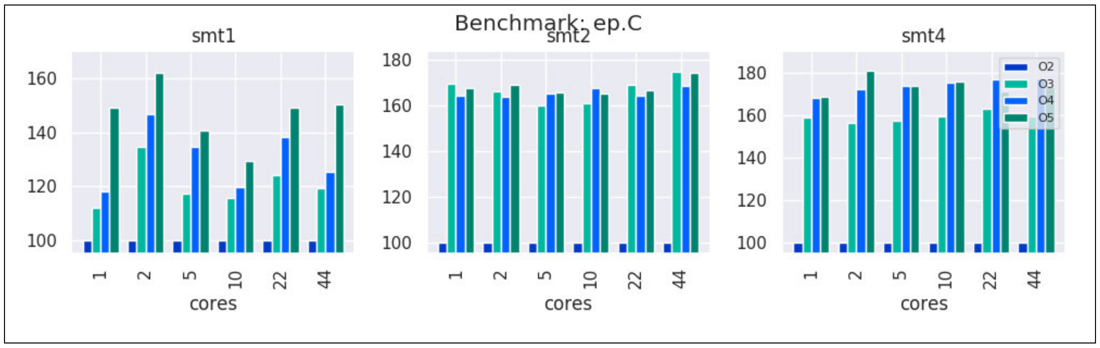


Figure 9-14 Performance gain from a rational choice of compiler options for the ep.C benchmark

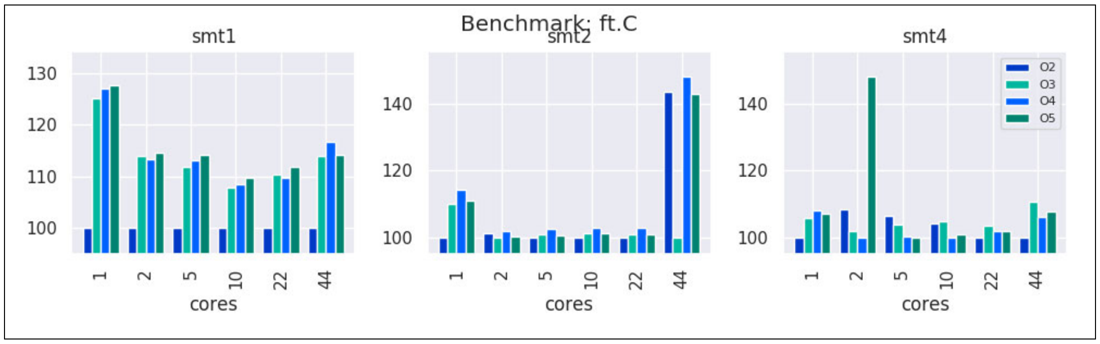


Figure 9-15 Performance gain from a rational choice of compiler options for the ft.C benchmark

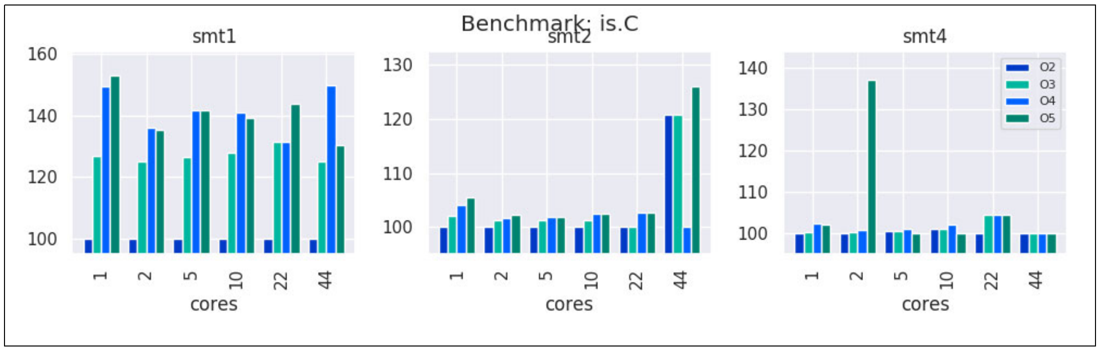


Figure 9-16 Performance gain from a rational choice of compiler options for the is.C benchmark

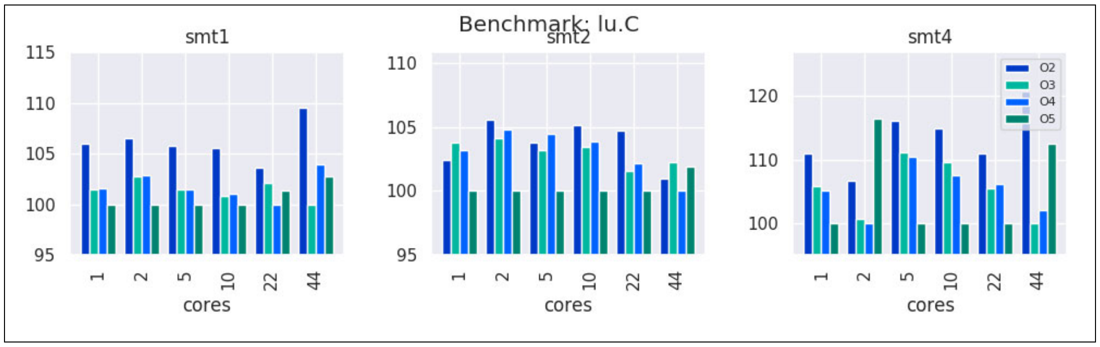


Figure 9-17 Performance gain from a rational choice of compiler options for the lu.C benchmark

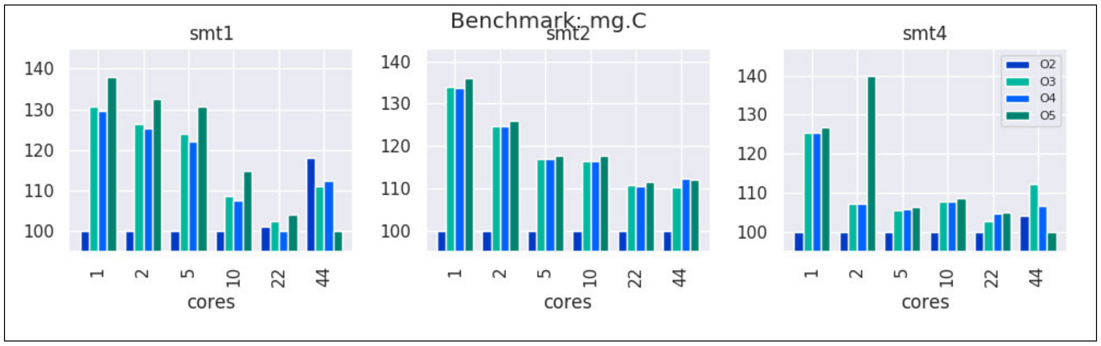


Figure 9-18 Performance gain from a rational choice of compiler options for the mg.C benchmark

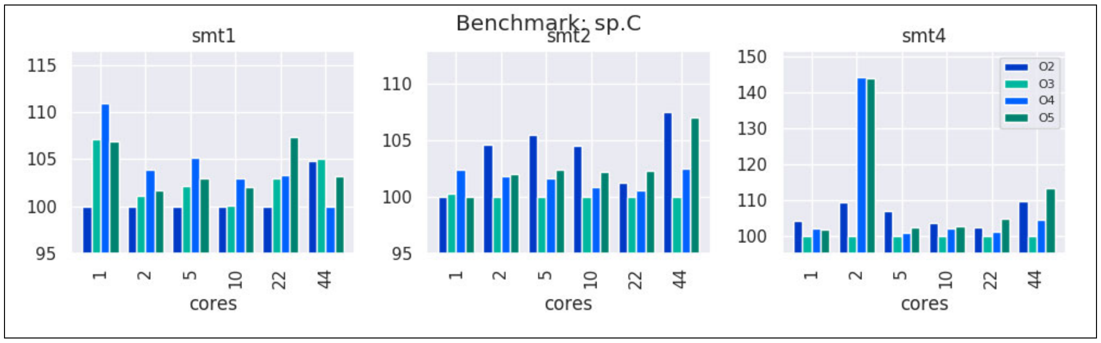


Figure 9-19 Performance gain from a rational choice of compiler options for the sp.C benchmark

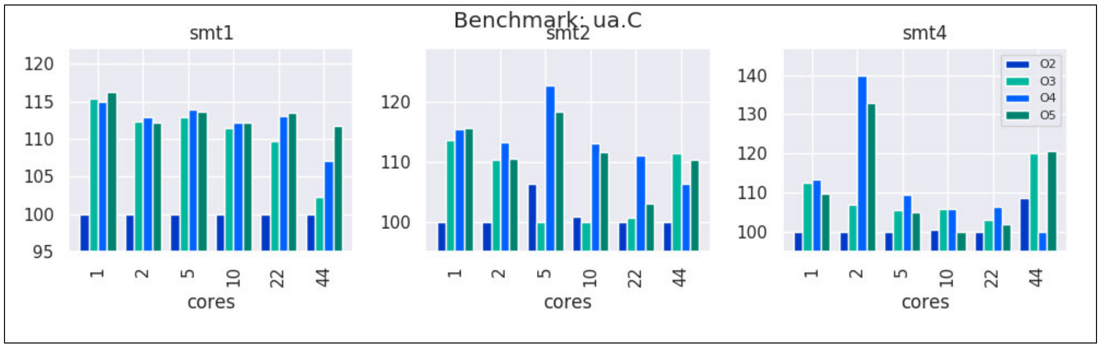


Figure 9-20 Performance gain from a rational choice of compiler options for the ua.C benchmark

9.1.3 Favorable modes and options for applications from the NPB suite

Table 9-2 lists SMT modes and compiler optimization options that are favorable for most of the runs that are described in 9.1.1, “Performance effect of a rational choice of SMT mode” on page 262 and 9.1.2, “Effect of optimization options on performance” on page 270. The row headers (ST, SMT-2, and SMT-4) designate SMT modes. The column headers (-O2, -O3, -O4, and -O5) refer to sets of compiler options that are listed in Table 9-1 on page 261.

Table 9-2 Favorable modes and options for applications from the NPB suite

Mode	-O2	-O3	-O4	-O5
ST	—	is.C and mg.C	is.C	—
SMT-2	is.C	—	—	—
SMT-4	—	ep.C	—	—

Favorable SMT modes can vary from ST to SMT-4 and favorable compilation options can vary from -O2 to -O5.

It is difficult to know before experimenting which SMT mode and which set of compilation options are favorable for a user application. Therefore, establish benchmarks before conducting production runs. For more information, see 9.2, “General methodology of performance benchmarking” on page 275.

9.1.4 Importance of binding threads to logical processors

The OS can migrate application threads between logical processors if a user does not explicitly specify thread affinity. As described in 8.1, “Controlling the running of multithreaded applications” on page 228, a user can specify the binding of application threads to a specific group of logical processors. One option for binding threads is to use system calls in a source code. The other option is to set environment variables that control threads affinity.

For technical computing workloads, you want to ensure that the application threads are bound to logical processors. Thread affinity often helps to leverage the POWER Architecture memory hierarchy and reduce the impact that is related to the migration of threads by an OS.

To demonstrate the importance of this technique, we chose the `mg.C` test from the NPB suite as an example. The performance of the `mg.C` application peaks at SMT-1 (see Table 9-2). For this benchmark, you can expect a penalty if an OS puts more than one thread on each core.

Figure 9-21 shows the performance improvement that was obtained by binding application threads to logical processors. The baseline corresponds to runs without affinity. The bars show the relative gain that was obtained after the assignment of software threads to hardware threads. As SMT mode increases, the OS has more freedom in scheduling threads. As the result, the effect of thread binding becomes more pronounced for higher values of threads and SMT mode.

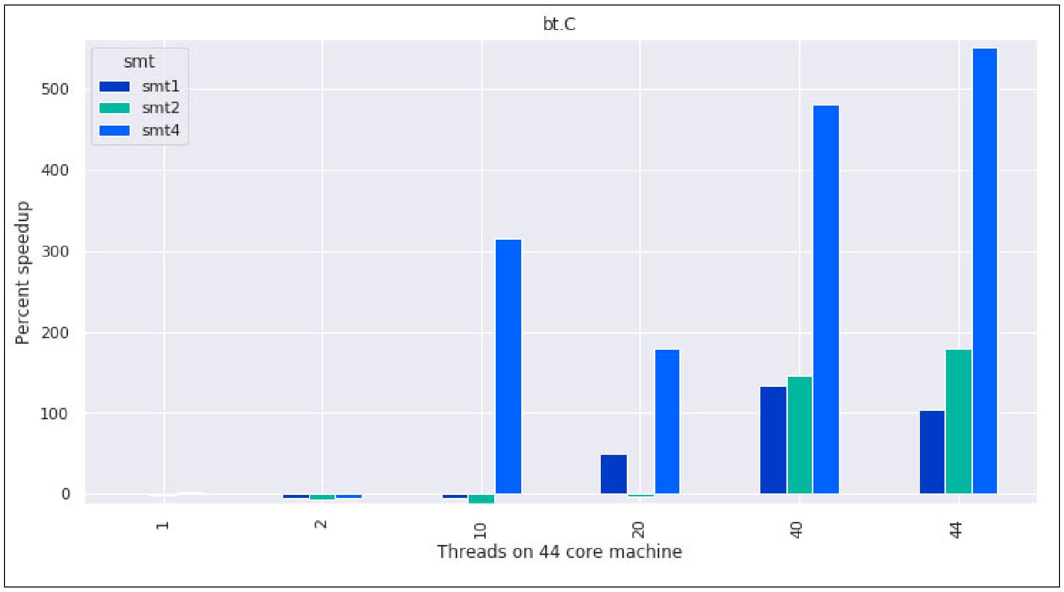


Figure 9-21 Performance improvement for an application when thread binding is used

When running with 44 threads, we get twice the performance of an unbound run on SMT-1, and 6x the performance in SMT-4 mode. The reason for the extreme speedup in this case is that with thread binding we can ensure that each core is running only one thread, although in the unbound mode we have some cores running multiple threads and competing for CPU resources during the time other cores are idle.

9.2 General methodology of performance benchmarking

This section describes how to evaluate the performance of an application on a system with massively multithreaded processors. It also provides some hints about how to take advantage of the performance of an application without access to the source code.

This example assumes that the application is thread-parallel and does not use MPI[4] or General Purpose GPUs (GPGPUs)[5]. However, the generalization of the methodology to a broader set of applications is relatively straightforward.

For the simplicity of table structure throughout this section, the examples make the following assumptions about the configuration of a computing server:

- The server is a two-socket system with fully populated memory slots.
- The server has 22 cores per socket (44 cores in total).

The computing server reflects the architecture of the Power AC922 (8335-GTW) server.

[4] MPI is a popular message-passing application programmer interface that is used for parallel programming.
[5] GPGPU is the use of GPUs for solving compute-intensive data parallel problems.

This section also describes performance benchmarking and summarizes the methodology in a step-by-step instruction form. For more information, see 9.2.11, “Summary” on page 283.

9.2.1 Defining the purpose of performance benchmarking

Before starting performance benchmarking, clarify the purpose of this activity. A clearly defined purpose of benchmarking is helpful in creating a plan of benchmarking and defining the success criteria.

The purpose of performance benchmarking looks different from each of the following two points of view:

- Performance benchmarking that is carried out by an *application developer*.
- Performance benchmarking that is done by an *application user* or a *system administrator*.

Benchmarking by application developers

From the perspective of an application developer, the performance benchmarking is part of the software development process. An application developer uses performance benchmarking to pursue the following goals:

- Comparing the code’s performance with the performance model.
- Identifying bottlenecks in the code that limit its performance (this process is typically done by using profiling).
- Comparing the code’s scalability with the scalability model.
- Identifying parts of the code that prevent scaling.
- Tuning the code to specific computer hardware.

A software developer performs a performance benchmarking of an application to understand how to improve application performance by changing an algorithm. Performance benchmarking viewed from an application developer perspective has complex methodologies. These methodologies are not covered in this publication.

Benchmarking by application users and system administrators

From the perspective of an *application user* or a *system administrator,* performance benchmarking is a necessary step before using an application for production runs on a computing system that is available for them. Application users and system administrators make the following assumptions:

- An application will be used on a computing system many times in the future.
- It makes sense to invest time into performance benchmarking because the time will be made up by faster completion of production runs in the future.
- Modification of an application source code is out of their scope.

  This statement implies that application users and system administrators are limited to the following choices to tune the performance:

  – Different compilers
  – Compiler optimization options
  – SMT mode
  – Number of computing cores
  – Runtime system parameters and environment variables
  – OS parameters

Essentially, applications users and system administrators are interested in how to make an application solve problems as fast as possible without changing the source code.

System administrators also need performance benchmarking results to determine hardware allocation resources when configuring a computing system. For more information, see "Choosing the SMT mode for computing nodes" on page 269.

This book considers performance benchmarking from the perspective of an application user or a system administrator.

9.2.2 Benchmarking plans

Complete the following steps for your benchmarking project:

1. A kick-off planning session with the experts
2. Workshop with the experts
3. Benchmarking
4. Session to discuss the intermediate results with the experts
5. Benchmarking
6. Preparation of the benchmarking report
7. Session to discuss the final results with the experts

You also must prepare the following lists:

- Applications to be benchmarked
- Persons who are responsible for specific applications

Planning and discussion sessions are ideally face-to-face meetings between the benchmarking team and Power Architecture professionals. A workshop with the Power Architecture experts is a form of knowledge transfer and educational activity with hands-on sessions.

The specific technical steps that you perform when benchmarking individual applications are described next.

For an example of the technical part of a plan, see 9.2.11, "Summary" on page 283.

9.2.3 Defining the performance metric and constraints

The *performance metric* provides a measurable indicator of application performance. Essentially, a performance metric is a number that can be obtained in a fully automated manner. The most commonly used performance metrics include the following examples:

- Time of application run.
- Number of operations that is performed by an application in a unit of time.

Some applications impose constraints in addition to the performance metric. Typically, the violation of a constraint means that running the application in such conditions is unacceptable. Constraints include the following examples:

- Latency of individual operations (for example, the running of a specific ratio of operations takes no longer than a specific time threshold).
- Quality metric of the results that are produced by the application does not fall under a specified threshold (for example, a video surveillance system drops no more than a specified number of video frames in a unit of time).

9.2.4 Defining the success criteria

Satisfying a success criteria is a formal reason to finalize performance benchmarking. Usually, success criteria is based on the performance results. Typically, *success criteria* is a numeric threshold that provides a definition of an acceptable performance (see 9.2.3, “Defining the performance metric and constraints” on page 277).

The performance benchmarking can result in the following outcomes:

- The success criteria are satisfied. This result means that the process of performance tuning can be finalized.
- The application does not scale as expected (see “Probing scalability” on page 280). This result indicates that you must reconsider the success criteria.
- The success criteria are not satisfied. Use the following techniques to solve the problem:
  - Discuss the performance tuning options that you tried and the results that you obtained with the experts. For this purpose, keep a verbose benchmarking log. For more information, see “Keeping the log of benchmarking” on page 279.
  - Engage the experts to perform deep performance analysis.
  - Seek the help of software developers to modify the source code.

Performance of a logical processor versus performance of a core

In most cases, there is no sense in defining success criteria based on the performance of a logical processor or a core taken in isolation. As described in 9.1.1, “Performance effect of a rational choice of SMT mode” on page 262, the POWER9 core can run instructions from multiple application threads simultaneously. Therefore, the whole core is the minimal entity for discussing performance. For more information, see “Choosing the SMT mode for computing nodes” on page 269.

Aggregated performance statistics for poorly scalable applications

Similarly, some applications are not designed to scale up to a whole computing server. For example, an application can violate constraints under a heavy load (see 9.2.3, “Defining the performance metric and constraints” on page 277). In such situation, it makes sense to run several instances of an application on a computing node and collect aggregated performance statistics. With this technique, you can evaluate the performance of a whole server that is running a particular workload.

9.2.5 Correctness and determinacy

Before you start performance benchmarking, check whether the application works correctly and produces deterministic results. If the application produces undeterministic results by design (for example, the application implements the Monte-Carlo method or other stochastic approach), you have at least two options:

- Modify the application by making its output deterministic (for example, fix the seed of a random number generator).
- Develop a reliable approach for measuring performance of an undeterministic application (this process can require many more runs than for a deterministic application).

During each stage of the performance tuning, verify that the application still works correctly and produces deterministic results (for example, by using regression testing).

9.2.6 Keeping the log of benchmarking

Preserve the history and output of all commands that are used in the preparation of the benchmark and during the benchmark. This log includes the following files:

- Makefiles
- Files with the output of the **make** command
- Benchmarking scripts
- Files with the output of benchmarking scripts
- Files with the output of individual benchmarking runs

A benchmarking script is a shell script that includes the following commands:

- A series of commands that capture the information about the environment
- A sequence of commands that run applications under various conditions

Example 9-5 shows several commands that you might want to include in the beginning of a benchmarking script to capture the information about the environment.

Example 9-5 Example of the beginning of a benchmarking script

```
#!/bin/bash -x
uname -a
tail /proc/cpuinfo
numactl -H
ppc64_cpu --smt
ppc64_cpu --cores-present
ppc64_cpu --cores-on
gcc --version
xlf -qversion
env
export OMP_DISPLAY_ENV="VERBOSE"
```

Typically, a benchmarking script combines multiple benchmarking runs in a form of loops over a number of sockets, cores, and SMT modes. By embedding the commands that you use to run an application into a script, you keep the list of commands along with their outputs.

Running a benchmarking script

If you do not use a job scheduler to submit a benchmarking script, run a benchmarking script inside a session that is created by running the **screen** command. Doing so gives protection against network connection issues and helps to keep applications running, even during a network failure.

One option to run a benchmarking script is to run the following command:

```
./benchmarking_script_name.sh 2>&1 | tee specific_benchmarking_scenario.log
```

This command combines the standard output and the standard error streams, but the **tee** command writes the combined stream to a workstation and a specified file.

Choosing names for log files

It is a best practice to write the output of individual benchmarking runs to separate files with well-defined descriptive names. For example, we found the following format useful for the purposes of the benchmarking runs that are described in 9.1, “Effects of basic performance tuning techniques” on page 260:

```
$APP_NAME.t$NUM_SMT_THREADS.c$NUM_CORES.s$NUM_SOCKETS.log
```

This format facilitated the postprocessing of the benchmarking logs by running the **sed** command.

9.2.7 Probing scalability

Probing scalability is a necessary step of the performance benchmarking for the following reasons:

- It evaluates the behavior of an application when the number of computing resources increases.
- It helps to determine the favorable number of processor cores to use for production runs.

To be concise, the ultimate purpose of probing the scalability is to check whether an application is scalable.

Note: In the performance benchmarking within a one-node system, the scalability is probed only in a *strong* sense, also known as a *speed-up* metric. This metric is obtained by fixing the size of a problem and varying the number of processor cores. For a multi-node system, the complimentary metric is a scalability in a *weak* sense. It is obtained by measuring performance when the amount of computing work per node is fixed.

Before performing runs, the user answers the following questions:

- Does the application has some specific limits on the number of logical processors it can use?

  (For example, some applications are designed to run with the number of logical processors that is a power of two only: 1, 2, 4, 8, and so on).
- Is the scalability model of the application available?

  The model can be purely analytical (for example, the application vendor can claim a linear scalability) or the model can be based on the results of previous runs.
- What is the scalability criteria for the application? That is, what is the numeric threshold that defines the “yes” or “no” answer to the following formal question: “Given $N > N_{\text{base}}$ logical processors, are you ready to agree that the application is scalable on N logical processors in respect to N_{base} logical processors?”

If available, the scalability model helps you to choose the scalability criteria.

The next step is to compile the application with a basic optimization level. For example, you can use the following options:

- **-O3 -qstrict** (with IBM XL Compilers)
- **-O3** (with GNU Compiler Collection (GCC)[6] and IBM Advance Toolchain)

For more information about IBM XL and GCC, see 7.1, “Compiler options” on page 162.

For scalability probing, choose the size of a problem that is large enough to fit the memory of a server. In contrast, solving a problem of a small size often is a waste of computing cores. If you must process multiple small problems, run multiple one-core tasks simultaneously on one node.

6 GCC contains various compilers, including C, C++, and Fortran.

When you decide on a problem size, run the application in ST mode with a different number of logical processors. Complete the “Performance, Meets the scalability model? (yes/no)”, and “Meets the scalability criteria? (yes/no)” rows, as shown in Table 9-3.

Table 9-3 Scalability probing measurements table

Number of cores	1	2	3	…	22	...	32	36	40	44
Number of sockets	1	1	1	…	1	...	2	2	2	2
Number of cores per socket	1	2	3	…	22	...	16	18	20	22
Performance										
Meets the scalability model? (yes/no)	—									
Meets the scalability criteria? (yes/no)	—									

Note: Bind application threads to logical processors. Failing to do so typically produces worse results (see 9.1.4, “Importance of binding threads to logical processors” on page 274).

When probing scalability, vary the number of cores and run each core in ST mode. Therefore, the total number of threads that is used can be equal to the number of cores.

Depending on the results of the benchmarking, you receive one of the following outcomes:

- The application is scalable up to 44 cores.
- The application is scalable within the socket (up to 22 cores).
- The application is not scalable within the socket.

If the scalability of the application meets your expectations, proceed to the next step and evaluate the performance by using the maximum number of cores that you determined in this section.

If the application does not meet your scalability expectations, clarify why and repeat probing scalability until you are satisfied. Only after that process can you proceed to the next step.

9.2.8 Evaluation of performance on a favorable number of cores

The performance of an application heavily depends on the choice of SMT mode and compilation modes. For more information, see Figure 9-12 on page 271. It is difficult to know beforehand which set of compiler options and number of hardware threads per core can be a favorable choice for an application. Therefore, you must run the application with several compilation options or SMT modes and select the most appropriate combination.

In 9.2.7, “Probing scalability” on page 280, the maximum reasonable number of cores to be used by an application was determined. The next step is to use that number of cores to run the application with different sets of compiler options and SMT modes.

You must try each of four SMT modes for several sets of compiler options. If you are limited on time, try fewer sets of compiler options, but go through all SMT modes. Enter the results of your performance measurements in Table 9-4 (the table is similar to the Table 9-2 on page 274). Your actual version of a table can include other columns and different headings of the columns, depending on the sets of compiler options you choose.

Table 9-4 Performance measurements results table

Mode	-O2	-O3	-O4	-O5
ST				
SMT-2				
SMT-4				

Optionally, you can repeat the step that is described in 9.2.7, “Probing scalability” on page 280 with the compiler options that give the highest performance.

9.2.9 Evaluation of scalability

The core of the IBM POWER9 processor is a multithreaded processor. Therefore, measuring the performance of a single application thread is not recommended. There is more meaning in application performance when a whole core is used. The application performance depends on the SMT mode of a core, as described in “The reasons behind a conscious choice of an SMT mode” on page 262.

Before the scalability of an application is evaluated, select a favorable SMT mode and compilation options, as described in 9.2.8, “Evaluation of performance on a favorable number of cores” on page 281. Then, use that SMT mode and vary the number of cores to get the scalability characteristics. Enter the results of the measurements in the “Performance” row of Table 9-5 (the table is similar to Table 9-3 on page 281). Compute values for the “Speed-up” row based on the performance results. The number of columns in your version of the table depends on the maximum number of cores that you obtained when you probed the scalability. For more information, see 9.2.7, “Probing scalability” on page 280.

Table 9-5 Scalability evaluation results table

Number of cores	1	2	3	…	22	...	32	36	40	44
Number of sockets	1	1	1	…	1	...	2	2	2	2
Number of cores per socket	1	2	3	…	22	...	16	18	20	22
Performance										
Speed-up	—									

Note: When we probed the scalability, we were running the cores in ST mode. In the scalability evaluation step, we ran the cores in an SMT mode that we determined in the performance evaluation step. The total number of threads in each run of the scalability evaluation is equal to the number of used cores that are multiplied by the number of threads per core in a chosen SMT mode.

9.2.10 Conclusions

As a result of the benchmark, you have the following information for each application:

- The level of the application scalability in ST mode.
- A favorable SMT mode and compiler options that deliver the highest performance.
- Scalability characteristics in the most favorable SMT mode.

9.2.11 Summary

The process that is used to facilitate your benchmarking includes the following steps:

1. Define the purpose of the benchmarking (see 9.2.1, “Defining the purpose of performance benchmarking” on page 276) and choose which of the following options you will use for performance tuning:
   - Choice between different compilers.
   - Choice of compiler optimization options.
   - Choice of an SMT mode.
   - Choice of the number of computing cores.
   - Choice of runtime system parameters and environment variables.
   - Choice of OS parameters.
2. Create the benchmarking plan (see 9.2.2, “Benchmarking plans” on page 277).
3. Define the performance metric (see 9.2.3, “Defining the performance metric and constraints” on page 277).
4. Define the success criteria (see 9.2.4, “Defining the success criteria” on page 278).
5. Verify that the application works correctly (see 9.2.5, “Correctness and determinacy” on page 278).
6. Probe the scalability (see 9.2.7, “Probing scalability” on page 280) to determine the limits of the application scalability:
   a. Complete the questionnaire on a scalability model and scalability criteria (see the bulleted list on page 280).
   b. Complete a results table (see Table 9-3 on page 281).
7. Discover the favorable SMT mode and compilation options (see 9.2.8, “Evaluation of performance on a favorable number of cores” on page 281) by completing Table 9-4 on page 282.
8. Evaluate the scalability (see 9.2.9, “Evaluation of scalability” on page 282) by completing Table 9-5 on page 282.

9.3 Sample code for the construction of thread affinity strings

Example 9-6 provides the source code `t_map.c` for the program `t_map`. You can use this small utility to construct a text string that describes the mapping of OpenMP threads of an application to logical processors. Text strings of this kind are intended to be assigned to OpenMP thread affinity environment variables.

Example 9-6 Source code t_map.c (in C programming language) for the program t_map

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

```
#define MAX_TPC 4 // Threads per core (max SMT mode is SMT4)
#define MAX_CPS 22 // Cores per socket
#define MAX_SPS 2
#define MAX_THR (MAX_TPC * MAX_CPS * MAX_SPS)

void Print_Map(int sps, int cps, int tpc, int base) {
  const int maps[MAX_TPC][MAX_TPC] = {
    { 0                     },
    { 0,          4         },
    { 0,    2,    4         },
    { 0,    2,    4,    6   },
    { 0, 1, 2,    4,    6   },
    { 0, 1, 2,    4, 5, 6   },
    { 0, 1, 2, 3, 4, 5, 6   },
    { 0, 1, 2, 3, 4, 5, 6, 7 }
  };

  const int sep = ',';

  int thread, core, socket;

  int tot = sps * cps * tpc;
  int cur = 0;

  for (socket = 0; socket < sps; ++socket) {
    for (core = 0; core < cps; ++core) {
      for (thread = 0; thread < tpc; ++thread) {
        int shift = socket * MAX_CPS * MAX_TPC +
                            core * MAX_TPC;
        shift += base;
        ++cur;
        int c = (cur != tot) ? sep : '\n';
        printf("%d%c", shift + maps[tpc-1][thread], c);
      }
    }
  }
}

void Print_Usage(char **argv) {
    fprintf(stderr, "Usage: %s "
        "threads_per_core=[1-%d] "
        "cores_per_socket=[1-%d] "
        "sockets_per_system=[1-%d] "
        "base_thread=[0-%d]\n",
        argv[0], MAX_TPC, MAX_CPS, MAX_SPS, MAX_THR-1);
}

int main(int argc, char **argv) {
  const int num_args = 4;

  if (argc != num_args+1) {
    fprintf(stderr, "Invalid number of arguments (%d). Expecting %d "
        "arguments.\n", argc-1, num_args);
    Print_Usage(argv);
    exit(EXIT_FAILURE);
  }

  int tpc = atoi(argv[1]);
  int cps = atoi(argv[2]);
  int sps = atoi(argv[3]);
  int base = atoi(argv[4]);

  if (tpc < 1 || tpc > MAX_TPC ||
      cps < 1 || cps > MAX_CPS ||
      sps < 1 || sps > MAX_SPS) {
    fprintf(stderr, "Invalid value(s) specified in the command line\n");
    Print_Usage(argv);
    exit(EXIT_FAILURE);
  }

  int tot = sps * cps * tpc;

  if (base < 0 || base+tot-1 >= MAX_THR) {
    fprintf(stderr, "Invalid value specified for the base thread (%d). "
        "Expected [0, %d]\n", base, MAX_THR-tot);
    Print_Usage(argv);
    exit(EXIT_FAILURE);
  }

  Print_Map(sps, cps, tpc, base);

  return EXIT_SUCCESS;
}
```

Note: Example 9-6 on page 283 lists a revised version of the code that originally appeared in Appendix D, "Applications and performance", of *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263. The code was adjusted to better fit the architecture of Power AC922 servers.

To use the tool, first you must compile the code by using the C compiler of your choice. For example, run the following command with the **gcc** compiler:

```
$ gcc -o t_map t_map.c
```

If you run the tool without any arguments, you are provided a brief hint about the usage, as shown in the following example:

```
$ ./t_map
Invalid number of arguments (0). Expecting 4 arguments.
Usage: ./t_map threads_per_core=[1-4] cores_per_socket=[1-22]
sockets_per_system=[1-2] base_thread=[0-176]
```

The utility needs you to specify the following amounts and locations of resources that you want to use:

- Number of threads per core
- Number of cores per socket
- Number of sockets
- The initial logical processor number (counting from zero)

Example 9-7 shows how to generate a thread mapping string for an OpenMP application that uses the following resources of a 22-core Power AC922 server:

- Twenty OpenMP threads in total
- Two threads on each core
- Ten cores on each socket
- Only the second socket

Example 9-7 Sample command for the generation of a thread mapping string

```
$ ./t_map 2 10 1 80
80,84,84,88,88,92,92,96,96,100,100,104,104,108,108,112,112,116,116,120
```

The runtime system of an OpenMP application obtains the thread mapping string from an environment variable. You must use different OpenMP thread affinity environment variables, depending on the compiler you use for your OpenMP application. Table 9-6 lists references for OpenMP thread affinity environment variables.

Table 9-6 OpenMP thread affinity environment variables

Compiler family	Some compilers from the compiler family	OpenMP thread affinity environment variable

GCC	`gcc`
	`g++`
	`gfortran`
		`GOMP_CPU_AFFINITY`

IBM XL compilers	`xlc_r`
	`xlc++_r`
	`xlf2008_r`
		`XLSMPOPTS`, suboption `procs`

The information in Table 9-6 also applies to the OpenMP applications that are built with the derivatives of GCC and IBM XL compilers (for example, MPI wrappers).

Example 9-8 shows how to assign values to OpenMP thread environment variables to implement the scenario in Example 9-7.

Example 9-8 Assigning values to the OpenMP thread affinity environment variables

```
$ export XLSMPOPTS=procs="`t_map 2 10 1 80`"
$ echo $XLSMPOPTS
procs=80,84,84,88,88,92,92,96,96,100,100,104,104,108,108,112,112,116,116,120
$ export GOMP_CPU_AFFINITY="`t_map 2 10 1 80`"
$ echo $GOMP_CPU_AFFINITY
80,84,84,88,88,92,92,96,96,100,100,104,104,108,108,112,112,116,116,120
```

Note: Example 9-8 implies that the `t_map` program is in your `PATH`. You must specify the full path to the tool if the OS does not find it in your `PATH`.

Example 9-8 shows the value of the environment variables with the **`echo`** command to demonstrate the result of the assignment. This command does not affect the affinity.

9.4 IBM Engineering and Scientific Subroutine Library performance results

The IBM Engineering and Scientific Subroutine Library (IBM ESSL) library includes the implementation of the famous double precision general matrix multiplication (DGEMM) routine, which is used in a large spectrum of libraries, benchmarks, and other IBM ESSL routines. Therefore, its performance is significant.

DGEMM implements the following formula:

$$C = \alpha[A] \cdot [B] + \beta[C]$$

Alpha and *beta* are real scalar values, and *A*, *B*, and *C* are matrixes of conforming shape.

Example 9-9 features a sample Fortran program with multiple calls of DGEMM for different sizes of input matrixes.

Example 9-9 IBM ESSL Fortran example source code dgemm_sample.f

```
      program dgemm_sample
      implicit none

      real*8 diff
      integer n, m, k
      integer maxn
      integer step
      integer i
      real*8,allocatable :: a(:,:), b(:,:), c(:,:)
      real*8,allocatable :: at(:,:), bt(:,:), ct(:,:)
      real*8 rmin
      real*8 seed1, seed2, seed3
      real*8 dtime, mflop
      real*8 flop
      integer tdummy, tnull, tstart, tend, trate, tmax

      maxn = 20000
      step = 1000

      seed1 = 5.0d0
      seed2 = 7.0d0
      seed3 = 9.0d0
      rmin = -0.5d0

      call system_clock(tdummy,trate,tmax)
      call system_clock(tstart,trate,tmax)
      call system_clock(tend,trate,tmax)
      tnull = tend - tstart

      allocate( at(maxn, maxn) )
      allocate( bt(maxn, maxn) )
      allocate( ct(maxn, maxn) )
      allocate( a(maxn, maxn) )
      allocate( b(maxn, maxn) )
      allocate( c(maxn, maxn) )

      call durand(seed1, maxn*maxn, at)
      call dscal(maxn*maxn, 1.0d0, at, 1)
      call daxpy(maxn*maxn, 1.0d0, rmin, 0, at, 1)

      call durand(seed2, maxn*maxn, bt)
      call dscal(maxn*maxn, 1.0d0, bt, 1)
      call daxpy(maxn*maxn, 1.0d0, rmin, 0, bt, 1)

      call durand(seed3, maxn*maxn, ct)
      call dscal(maxn*maxn, 1.0d0, ct, 1)
      call daxpy(maxn*maxn, 1.0d0, rmin, 0, ct, 1)

      do i = 1, maxn/step
        n = i*step
        m = n
        k = n

        flop = dfloat(n)*dfloat(m)*(2.0d0*(dfloat(k)-1.0d0))

        call dcopy(n*k, at, 1, a, 1)
        call dcopy(k*m, bt, 1, b, 1)
        call dcopy(n*m, ct, 1, c, 1)

        call system_clock(tstart,trate,tmax)
        call dgemm('N','N',m,n,k,1.0d0,a,n,b,k,1.0d0,c,n);
        call system_clock(tend,trate,tmax)

        dtime = dfloat(tend-tstart)/dfloat(trate)
        mflop = flop/dtime/1000000.0d0

        write(*,1000) n, dtime, mflop

      enddo

 1000 format(I6,1X,F10.4,1X,F14.2)

      end program dgemm_sample
```

The commands that are shown Example 9-10 compile run this program by using different types of IBM ESSL libraries (serial, symmetric multiprocessor (SMP), and SMP Compute Unified Device Architecture (CUDA)). For SMP runs, it uses 20 SMP threads with each thread bound to a different POWER8 physical core.

Example 9-10 Compilation and running of dgemm_sample.f

```
echo "Serial run"
xlf_r -O3 -qnosave dgemm_sample.f -lessl -o dgemm_fserial
./dgemm_fserial

echo "SMP run"
export XLSMPOPTS=startproc=4:stride=4:parthds=20:spins=0:yields=0

xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmp -o dgemm_fsmp
./dgemm_fsmp
```

```
export ESSL_CUDA_HYBRID=yes
xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmp -o dgemm_fsmp
./dgemm_fsmp
export ESSL_CUDA_HYBRID=yes
echo "SMP CUDA run with 6 GPUs hybrid mode"
xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmpcuda -lcublas -lcudart
-L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -o dgemm_fcuda
./dgemm_fcuda
echo "SMP CUDA run with 6 GPUs non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
export CUDA_VISIBLE_DEVICES=0,1,2,3,4
export ESSL_CUDA_HYBRID=yes
echo "SMP CUDA run with 5 GPUs hybrid mode (1st, 2nd, 3rd, 4th, 5th)"
./dgemm_fcuda
echo "SMP CUDA run with 5 GPUs non-hybrid mode (1st, 2nd, 3rd, 4th, 5th)"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
export CUDA_VISIBLE_DEVICES=0,1,2,3
export ESSL_CUDA_HYBRID=yes
echo "SMP CUDA run with 4 GPUs hybrid mode (1st, 2nd, 3rd, 4th)"
./dgemm_fcuda
echo "SMP CUDA run with 4 GPUs non-hybrid mode (1st, 2nd, 3rd, 4th)"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1,2
./dgemm_fcuda
echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
echo "SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1
./dgemm_fcuda
echo "SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
echo "SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=1,2
./dgemm_fcuda
echo "SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (1st) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (1st) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
```

```
echo "SMP CUDA run with 1 GPU (2nd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=1
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (2nd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=2
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (4th) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=3
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (4th) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (5th) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=4
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (5th) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (6th) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=4
./dgemm_fcuda
echo "SMP CUDA run with 1 GPU (6th) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
```

Figure 9-22 shows the performance in floating point operations per second when sizing matrixes for different calls.

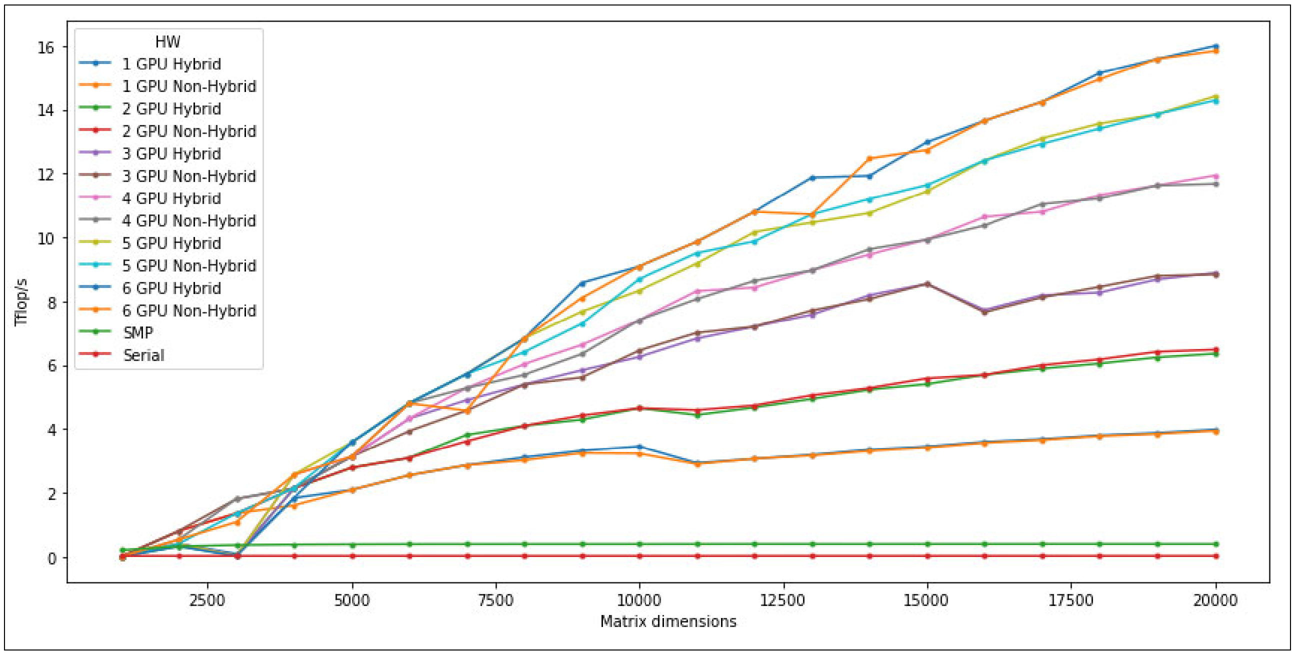


Figure 9-22 DGEMM performance results for different types of IBM ESSL libraries

The chart shows that the GPUs provide an advantage in performance around the 3000 - 5000 problem size. A smaller size is not enough to run the computation in the GPU, and it is better to run the computation in the CPU by using the IBM ESSL SMP library.

We can determine what effects occur when we select different GPUs. In Figure 9-23, we plot the performance of running on any one GPU.

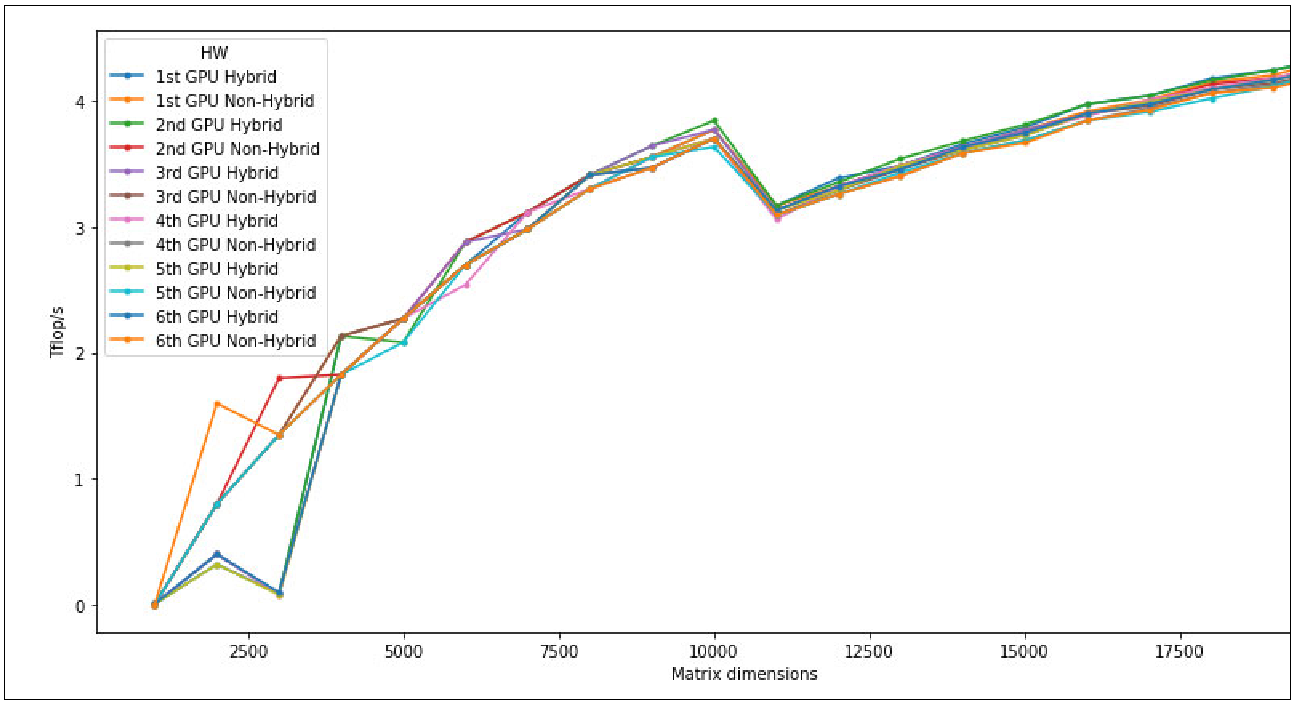


Figure 9-23 IBM ESSL SMP CUDA running with one GPU

For small matrix dimensions, there seems to be an extra cost for running on the GPUs that are attached to the second socket[7]. For larger dimensions, this extra impact is no longer noticeable, and any GPU gives the same performance within +/- 3%. Run your program on different GPUs to find the environment with the best performance results.

9.5 GPU tuning

This section explains the different GPU computing modes and how to manage shared access to GPUs by multi-process applications through the Multi-Process Service (MPS).

9.5.1 GPU processing modes

The NVIDIA Volta GPU has different computing modes, as shown in Table 9-7.

Table 9-7 NVIDIA Volta GPU modes

Mode	Name	Description
0	DEFAULT	Multiple threads can use **`cudaSetDevice()`** with this device.
1	EXCLUSIVE THREAD	Only one thread may use **`cudaSetDevice()`** with this device.
2	PROHIBITED	No threads can access the GPU.
3	EXCLUSIVE PROCESS	Only one context is allowed per device and is usable from multiple threads concurrently.

Configure these modes by running the following command:

```
nvidia-smi -c $mode
```

9.5.2 CUDA Multi-Process Service

MPS is client/server runtime implementation of the CUDA application programming interface (API) that helps multiple CUDA processes to share GPUs. MPS takes advantage of parallelism between MPI tasks to improve GPU utilization.

MPS contains the following components:

- Control daemon process

  Coordinates connections between servers and clients, and starts and stops the server.

- Client run time

  Any CUDA application can use the MPS client run time from the CUDA driver library.

- Server process

  This connection is the shared clients' shared connection to the GPU and provides concurrency between clients.

[7] The CPU threads were in this case bound to the first socket.

MPS is recommended for jobs that do not generate enough work for GPUs. If you have multiple runs concurrently, MPS helps to balance the workload of the GPU and increases its utilization.

Also, MPI programs that have different MPI-tasks but share a GPU can see performance improvements by using MPS to control access to the GPU between tasks.

Note: the Volta MPS server supports 48 client CUDA contexts per-device.

A recommendation is to use MPS with the **`EXCLUSIVE_PROCESS`** mode to be sure that only the MPS server runs on GPUs.

The CUDA program works by using MPS if the MPS control daemon runs in the system. The program at startup attempts to connect to the MPS control daemon, which creates an MPS server or reuses a server if it has the same user ID as the user who started the job. Therefore, each user has its own MPS server.

The MPS server creates the shared GPU context for the different jobs in the system, manages its clients, and calls to GPU on behalf of them. An overview of this process is shown in Figure 9-24.

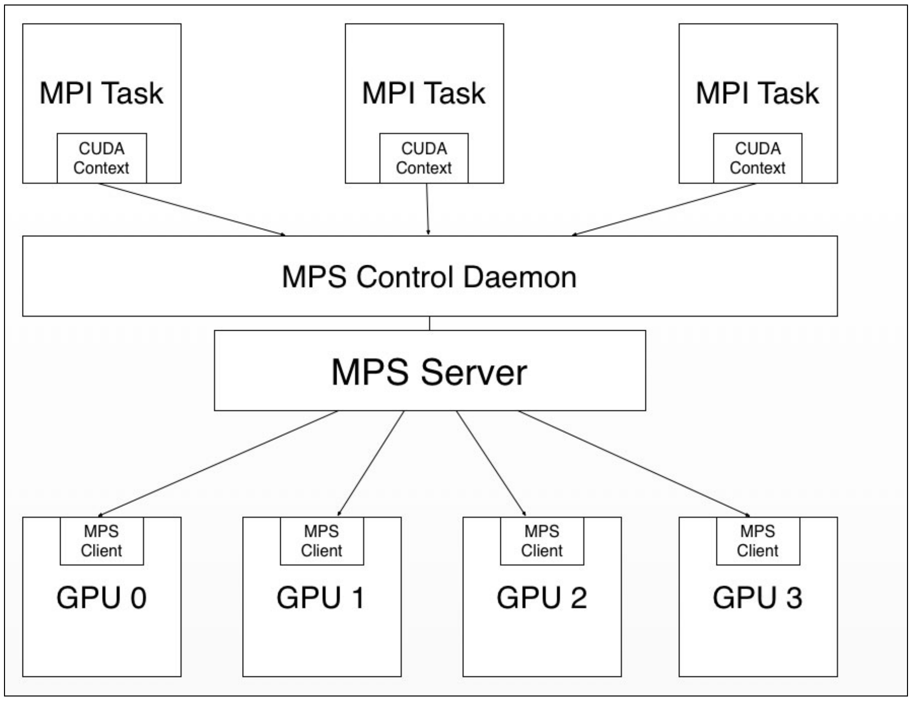


Figure 9-24 NVIDIA MPS

To run MPS, you need to run the following commands as root:

1. (This step is optional.) Set the **`CUDA_VISIBLE_DEVICES`** environment variable to inform the system which GPUs will be used. To use all GPUs, start from step 2 on page 294 and issue the following command:

```
export CUDA_VISIBLE_DEVICES=0,1 #Use only first and second GPUs
```

2. Change the compute mode for all GPUs or to specific GPUs, which are chosen in the first step:

   ```
   nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
   nvidia-smi -i 1 -c EXCLUSIVE_PROCESS
   ```

3. Start the daemon:

   ```
   nvidia-cuda-mps-control -d
   ```

To stop the MPS daemon, run the following command as root:

```
echo quit | nvidia-cuda-mps-control
```

Example 9-11 shows the output for starting and stopping MPS for two GPUs.

Example 9-11 Start and stop commands of the MPS for two GPUs

```
# nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 00000004:04:00.0.
All done.
# nvidia-smi -i 1 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 00000004:05:00.0.
All done.
# nvidia-cuda-mps-control -d
# echo quit | nvidia-cuda-mps-control
```

If the system has running jobs, running the **`nvidia-smi`** command provides output similar to the output that is shown in Example 9-12.

Example 9-12 The nvidia-smi output for a system with MPS

```
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0     23715    C   nvidia-cuda-mps-server                          89MiB |
|    1     23715    C   nvidia-cuda-mps-server                          89MiB |
|    2     23715    C   nvidia-cuda-mps-server                          89MiB |
|    3     23715    C   nvidia-cuda-mps-server                          89MiB |
+-----------------------------------------------------------------------------+
```

For more information about the CUDA MPS, see the *Multi-Process Service* documentation from NVIDIA.

9.6 Application development and tuning tools

Many tools for parallel program development and tuning are available for the cluster environment that is described in this book. Because different development models are allowed, more than one tool is often needed to develop the application.

This section describes some tools that were developed by IBM, and other tools that are useful but are maintained by third-party companies or open source communities.

Table 9-8 lists the tools and development models.

Table 9-8 Tools for development and tuning of parallel applications

Development model	Provider	Tool
MPI	IBM Parallel Performance Toolkit	Call graph analysis
		I/O Profiling (MPI-IO)
		MPI Profiling
		Hardware Performance Monitor (HPM)
	RogueWave Software	TotalView
	Allinea	DDT
Hybrid of MPI and OpenMP	Parallel Performance Toolkit	OpenMP profiling
CUDA	CUDA Toolkit	CUDA-MEMCHECK
		CUDA Debugger
		nvprof
		Nsight Eclipse Edition
Hybrid of MPI and CUDA	Parallel Performance Toolkit	GPU Performance Monitor
MPI, OpenMP, Parallel Active Messaging Interface (PAMI), and OpenSHMEM	Eclipse Parallel Tools Platform (Eclipse PTP)	Integrated development environment (IDE)

9.6.1 Parallel Performance Toolkit

IBM Parallel Performance Toolkit V2.4 provides a set of tools (see Table 9-8) and libraries to help with the development of applications that are written in C, C++, or Fortran by using pure MPI or hybrid with OpenMP and CUDA.

Note: IBM Parallel Environment Developer Edition was rebranded to Parallel Performance Toolkit since Version 2.3. Although it is renamed, most of components and usage documentation that is available in other IBM Redbooks publications and the IBM Knowledge Center are up-to-date with this new version.

This toolkit has the following main components:

- Parallel Performance Toolkit: Provides back-end runtime and development libraries and command-line tools.
- hpctView: Provides a GUI front end to the Parallel Performance Toolkit. Also, hpctView plug-ins to Eclipse PTP are available.

The fluxogram (see Figure 9-25) sees the analysis cycle of a parallel application by using the toolkit. As general rule, it involves the following phases:

- Instrument: An application executable file is designed to use a specific tool of the HPC Toolkit.
- Profile: Run the application to record execution data.
- Visualize: Visualize the resulting profile by using command-line or GUI (hpctView) tool.

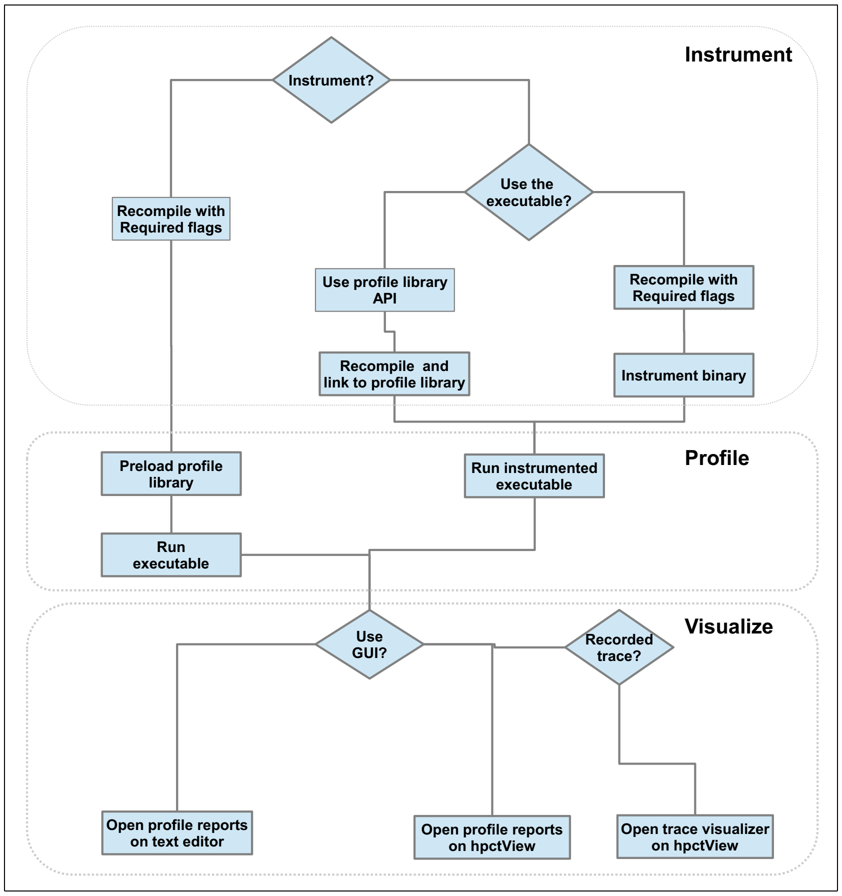

Instrument
Instrument?
Use the executable?
Recompile with Required flags
Use profile library API
Recompile with Required flags
Recompile and link to profile library
Instrument binary
Profile
Preload profile library
Run instrumented executable
Run executable
Visualize
Use GUI?
Recorded trace?
Open profile reports on text editor
Open profile reports on hpctView
Open trace visualizer on hpctView

Figure 9-25 Fluxogram shows how to use IBM Parallel Performance Toolkit

Instrumentation is an optional step for some tools. If you need a fine-grained collection of runtime information, use any of the following instrument mechanisms that are provided:

- Binary instrumentation: Use the **hpctInst** command or hpcView (GUI) to place probes on the executable file. It does not require you to rebuild the application.
- Profiling library API: Place calls to library methods to control profiling at the source level. It does require you to recompile the application and link it to the profiling library.

In the profile phase, you run an instrumented executable file with a relevant work load so that useful data is collected. For situations where the binary file is not instrumented, you must preinstall at run time the corresponding profiling tool library and use environment variables to control the behavior of the library. Then, the library takes over some method calls to produce the profile data.

To visualize the data that is produced by the profiling tools, use hpctView for the graphical visualization or Linux commands, such as **`cat`**, to inspect the report files.

Using hpctView to perform the instrument-profile-visualize analysis cycle is convenient because it permits fine-grained instrumentation at file, function, MPI call, or region levels. It also provides other means to profile the parallel application, and easy visualization of profile data and graphical tracing.

The Parallel Performance Toolkit tools are briefly described in the next sections, which show the many ways to profile and trace a parallel application. For more information about those topics, go to IBM Knowledge Center and click **Select a product** → **Parallel Performance Toolkit** → **Parallel Performance Toolkit 2.4.0**.

Application profiling and tracing tool

Often, the first task in performance tuning an application involves some sort of call graph analysis. For that purpose, the toolkit includes the application profiling and tracing tool, which is used to visualize GNU profile (`gprof`) files at a large scale.

To use the tool, you must compile the application with the **`-pg`** compiler flag, and then run the application to produce the profile data. Use a workload that is as close as possible to the production scenario so that good profile data can be generated. As an alternative, the application can be profiled by using Oprofile and converting the collected data by using the **`opgprof`** tool. However, this approach does not scale well for Single Program Multiple Data (SPMD) programs.

The following example shows how to prepare a Fortran 77 application to generate **`gprof`** data:

```
$ mpif77 -g -c compute_pi.f
$ mpi77 -g -pg -o compute_pi compute_pi.o
```

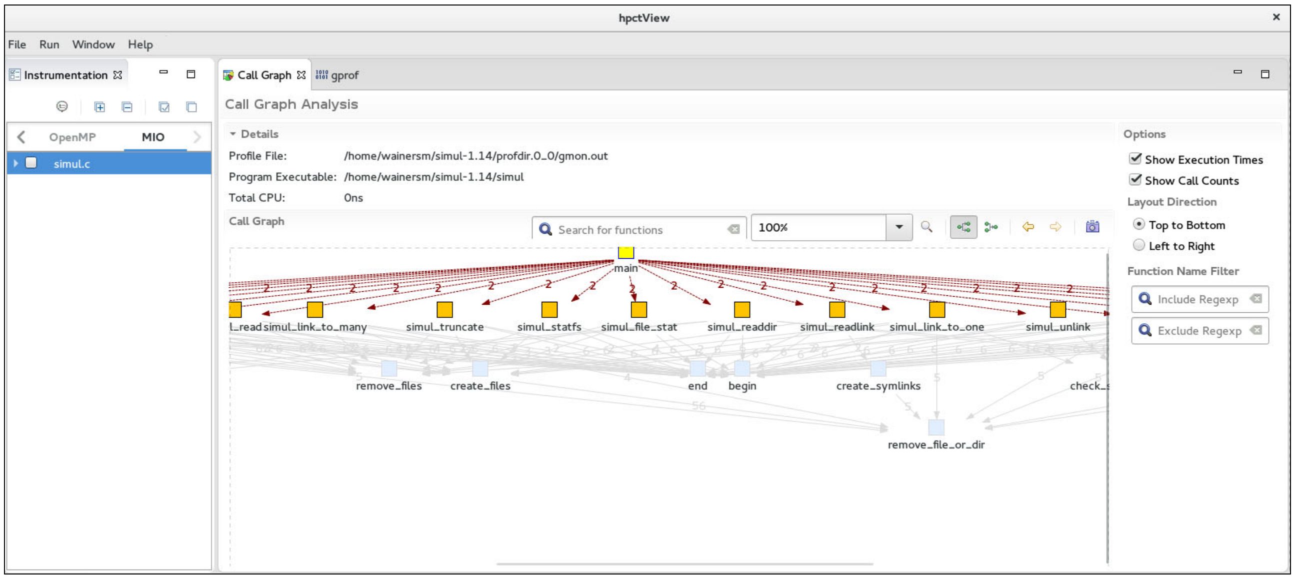
After the application is run, one `gmon.out` profile file is built per task.

Because MPI applications generate several performance files (one per rank), hpctView provides a visualization mode specifically to combine the various files and ease data navigation.

To load the `gmon.out` files into hpctView, complete the following steps:

1. Import the profiled application executable file by clicking **File** → **Open Executable**.
2. On the menu bar, select **File** → **Load Call Graph (gmon.out)**.
3. In the “Select gmon.out Files” window, browse the remote file system to select the `gmon.out` files that you want to load. Click **OK**.

The hpctView callgraph visualizer opens, as shown in Figure 9-26.



Call Graph Analysis

Details

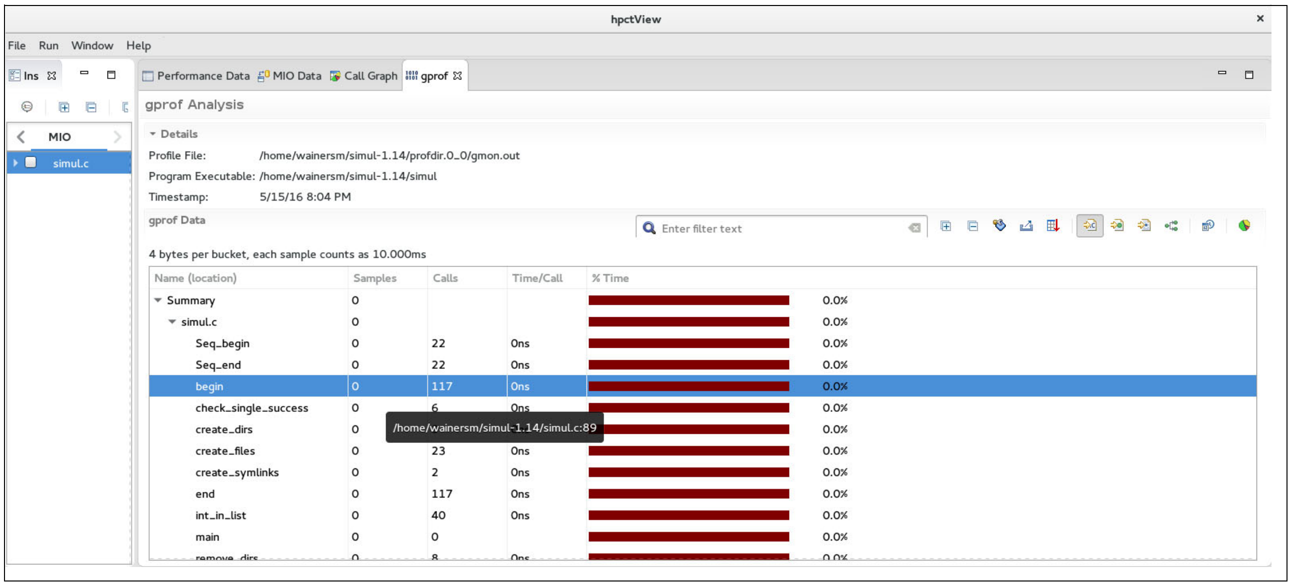
Profile File: /home/wainersm/simul-1.14/profdir.0_0/gmon.out
Program Executable: /home/wainersm/simul-1.14/simul
Total CPU: 0ns

Call Graph

Options
[✓] Show Execution Times
[✓] Show Call Counts
Layout Direction
[✓ Top to Bottom] Left to Right
Function Name Filter
Include Regexp
Exclude Regexp

Figure 9-26 The hpctView callgraph visualizer with example gprof data that is loaded on the callgraph tool

Information about the called method number of calls, average amount of time that is spent in each call, and the total percentage of time are presented in the gprof tab, as shown in Figure 9-27.



gprof Analysis

Details

Profile File: /home/wainersm/simul-1.14/profdir.0_0/gmon.out
Program Executable: /home/wainersm/simul-1.14/simul
Timestamp: 5/15/16 8:04 PM

gprof Data

4 bytes per bucket, each sample counts as 10.000ms

Name (location)	Samples	Calls	Time/Call	% Time
Summary	0			0.0%
simul.c	0			0.0%
Seq_begin	0	22	0ns	0.0%
Seq_end	0	22	0ns	0.0%
begin	0	117	0ns	0.0%
check_single_success	0	6	0ns	0.0%
create_dirs	0	[illegible]	[illegible]	0.0%
create_files	0	23	0ns	0.0%
create_symlinks	0	2	0ns	0.0%
end	0	117	0ns	0.0%
int_in_list	0	40	0ns	0.0%
main	0	0		0.0%
remove_dirs	0	8	0ns	0.0%

/home/wainersm/simul-1.14/simul.c:89

Figure 9-27 The hpctView callgraph visualizer with an example of gprof loaded

Message Passing Interface profiling and tracing

Performance analysis of MPI applications usually involves understanding many aspects of messages that are exchanged between the tasks, such as communication patterns, synchronization, and the data that is moved. To assist with application analysis, use the Parallel Performance Toolkit, which provides tools for profiling, tracing, and visualization of the produced data.

The following profiles and reporting information (per tasks) are available:

- Number of times that MPI routines are run.
- Total wall time that is spent in each MPI routine.
- Average transferred bytes on message passing routines.
- Highlight the tasks with minimal, median, and maximum communication time. This information is provided only in the report of task 0.

You can use the profile and trace data to perform many types of analysis, such as communication patterns, hot spots, and data movements.

Example 9-13 shows the textual report for task 0 of an MPI program. The last section of the report (Communication summary for all tasks) shows the minimum (task 7), median (task 4), and maximum (tasks 5) communication time that was spent on tasks.

Example 9-13 HPC Toolkit: Message Passing Interface profiling report for task 0

```
$ cat hpct_0_0.mpi.txt
-----------------------------------------------------------------
MPI Routine                  #calls   avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                     1          0.0          0.000
MPI_Comm_rank                     1          0.0          0.000
MPI_Bcast                         1          4.0          0.000
MPI_Barrier                       1          0.0          0.000
MPI_Allreduce                     4         26.0          0.000
-----------------------------------------------------------------
total communication time = 0.000 seconds.
total elapsed time       = 2.725 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Bcast                    #calls   avg. bytes      time(sec)
                                  1          4.0          0.000

MPI_Allreduce                #calls   avg. bytes      time(sec)
                                  3          8.0          0.000
                                  1         80.0          0.000

-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 0.000 sec for task 7
  median  communication time = 0.001 sec for task 4
  maximum communication time = 0.002 sec for task 5
```

Example 9-14 lists the files that are generated by the tool. Although the parallel job that is used in this example has eight tasks, only the reports of tasks 0, 4, 5, and 7 are available. The MPI profiling tool by default generates reports for the four most significant tasks according to the communication time criteria: Task 0 (an aggregate of all tasks) and tasks with minimum, median, and maximum communication time.

Example 9-14 Listing the files that are generated by the tool

```
$ ls
hpct_0_0.mpi.mpt  hpct_0_0.mpi.txt  hpct_0_0.mpi.viz  hpct_0_4.mpi.txt
hpct_0_4.mpi.viz  hpct_0_5.mpi.txt  hpct_0_5.mpi.viz  hpct_0_7.mpi.txt
hpct_0_7.mpi.viz
```

Using the same parallel job as an example, the task with maximum communication time is 5, as shown in Example 9-15.

Example 9-15 Contents of the profiling tasks

```
$ cat hpct_0_5.mpi.txt
-----------------------------------------------------------------
MPI Routine                  #calls   avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                     1          0.0          0.000
MPI_Comm_rank                     1          0.0          0.000
MPI_Bcast                         1          4.0          0.000
MPI_Barrier                       1          0.0          0.000
MPI_Allreduce                     4         26.0          0.001
-----------------------------------------------------------------
total communication time = 0.002 seconds.
total elapsed time       = 2.725 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Bcast                    #calls   avg. bytes      time(sec)
                                  1          4.0          0.000

MPI_Allreduce                #calls   avg. bytes      time(sec)
                                  3          8.0          0.001
                                  1         80.0          0.000
```

Along with the profile data, the tool records MPI routines calls over time that can be used within the hpctView trace visualizer. This information is useful for analyzing the communication patterns within the parallel program.

The easiest way to profile your MPI application is to load the trace library before the run by exporting the **LD_PRELOAD** environment variable. However, this variable can produce too much data from large programs.

Using the hpctView instrumentation-run-visualize analysis cycle from within hpctView is also a convenient way to profile the application because it permits fine-grained instrumentation at levels of files, functions, MPI routines, or code regions.

The examples that are shown in this section were generated by using the script that is shown in Example 9-16. To use the MPI profile preinstalled library, compile the parallel program by using the **`-g -Wl,--hash-style=sysv -emit-stub-syms`** flags.

Example 9-16 Script to profile the Message Passing Interface with HPC Toolkit

```
#!/bin/bash
#
# Use it as: $ MP_PROCS=<num> poe ./hpct_mpiprofile.sh <app> <args>
#

. /opt/ibmhpc/ppedev.hpct/env_sh

#
# HPCT MPI Trace control variables
#

## Uncomment to set maximum limit of events traced.
## Default is 30000.
#MAX_TRACE_EVENTS=

## Uncomment to generate traces for all ranks.
## Default are 4 tasks: task 0 and tasks with maximum, minimum and median communication time.
#OUTPUT_ALL_RANKS=yes

## Uncomment to enable tracing of all tasks.
## Default are tasks from 0 to 255 ranks.
#TRACE_ALL_TASKS=yes

## Uncomment to set maximum limit of rank traced.
## Default are 0-255 ranks or all if TRACE_ALL_TASKS is set.
#MAX_TRACE_RANK=

## Uncomment to set desired trace back level. Zero is level where MPI function is called.
## Default is the immediate MPI function's caller.
#TRACEBACK_LEVEL=

# Preload MPI Profile library
LD_PRELOAD=/opt/ibmhpc/ppedev.hpct/lib64/preload/libmpitrace.so

$@
```

Message Passing Interface I/O profiling

Characterization and analysis of I/O activities is an important topic of performance improvement, especially regarding parallel applications that demand critical usage of storage. The Message Passing Interface I/O (MPI-IO) tool consists of a profiler and trace recorder for collecting information about I/O system calls that are carried out by the parallel program to access and manipulate files.

The tool records information about I/O events that are triggered when the parallel application accesses files. General statistics per file are calculated out of the events that are probed; for example, the number of times each operation occurred and total time spent. Also, depending on the event, it collects the following information:

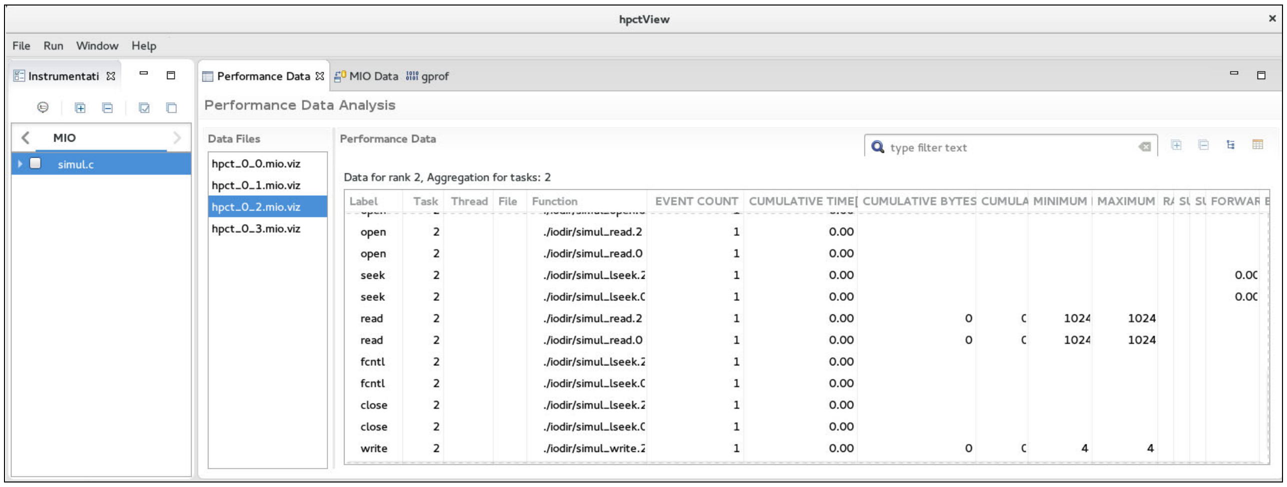
- Total of bytes requested
- Total of bytes delivered
- Minimum requested size in bytes
- Maximum requested size in bytes
- Rate in bytes
- Suspend wait count
- Suspend wait time
- Forward seeks average
- Backward seeks average

In particular, for read and write events, the tool can trace operations. For example, you can access start offset, number of bytes, and end offset operations.

To use MPI-IO, compile the application by using the **`-g -Wl,--hash-style=sysv -Wl,--emit-stub-syms`** flags to prepare the binary file for instrumentation.

You also can use hpctView to easily cycle instrumentation, execution, and visualization of information that is implemented by MPI-IO tool. The tool abstracts the many environment variables that can be used to control type and amount of information that is gathered.

Figure 9-28 shows the hpctView visualization mode for data that is generated by using the MPI-IO tool. More information about each I/O system call is provided on a per-task basis.



Performance Data Analysis

Data Files: hpct_0_0.mio.viz, hpct_0_1.mio.viz, hpct_0_2.mio.viz, hpct_0_3.mio.viz

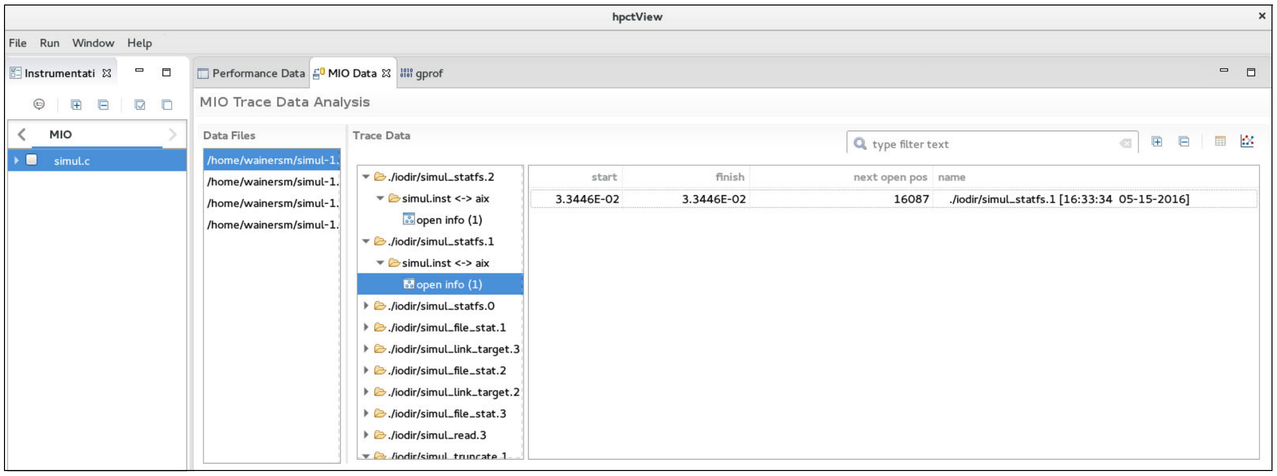
Performance Data

Data for rank 2, Aggregation for tasks: 2

Label	Task	Thread	File	Function	EVENT COUNT	CUMULATIVE TIME[	CUMULATIVE BYTES	CUMULA	MINIMUM		MAXIMUM	R/	SU	SU	FORWAR	E
open	2			./iodir/simul_read.2	1	0.00									
open	2			./iodir/simul_read.0	1	0.00									
seek	2			./iodir/simul_lseek.2	1	0.00								0.0C	
seek	2			./iodir/simul_lseek.C	1	0.00								0.0C	
read	2			./iodir/simul_read.2	1	0.00	0	C	1024	1024					
read	2			./iodir/simul_read.0	1	0.00	0	C	1024	1024					
fcntl	2			./iodir/simul_lseek.2	1	0.00									
fcntl	2			./iodir/simul_lseek.C	1	0.00									
close	2			./iodir/simul_lseek.2	1	0.00									
close	2			./iodir/simul_lseek.C	1	0.00									
write	2			./iodir/simul_write.2	1	0.00	0	C	4	4					

Figure 9-28 hpctView: Message Passing Interface I/O tool profiling visualization

Another view of hpctView displays the I/O trace data, as shown in Figure 9-29.



```
/home/wainersm/simul-1.
/home/wainersm/simul-1.
/home/wainersm/simul-1.
/home/wainersm/simul-1.
```

```
./iodir/simul_statfs.2
    simul.inst <-> aix
        open info (1)
./iodir/simul_statfs.1
    simul.inst <-> aix
        open info (1)
./iodir/simul_statfs.0
./iodir/simul_file_stat.1
./iodir/simul_link_target.3
./iodir/simul_file_stat.2
./iodir/simul_link_target.2
./iodir/simul_file_stat.3
./iodir/simul_read.3
./iodir/simul_truncate.1
```

start	finish	next open pos	name
3.3446E-02	3.3446E-02	16087	./iodir/simul_statfs.1 [16:33:34 05-15-2016]

Figure 9-29 hpctView: Message Passing Interface I/O tool trace visualization

Hardware Performance Monitor

The HPM tool is an analog tool that provides easy SPMD programs profiling on Linux on Power Systems. By using HPM, you can profile MPI programs regarding any of the hardware events available or obtain any of the predefined metrics that are most commonly used in performance analysis.

The Processor Monitor Unit (PMU) of the POWER8 processor is a part of the processor that is dedicated to recording hardware events. It has six Performance Monitor Counter (PMC) registers: Registers 0 - 3 can be programmed to count any of the more than 1000 events that are available, register 4 can count run instructions that are completed, and register 5 can count run cycles.

These events are important because they can reveal performance issues, such as pipeline bubbles, inefficient use of caches, and the high ratio of branch misprediction from the perspective of the processor. Metrics for performance measurements can also be calculated from hardware events, such as instructions per cycle, millions of instructions per second (MIPS), and memory bandwidth.

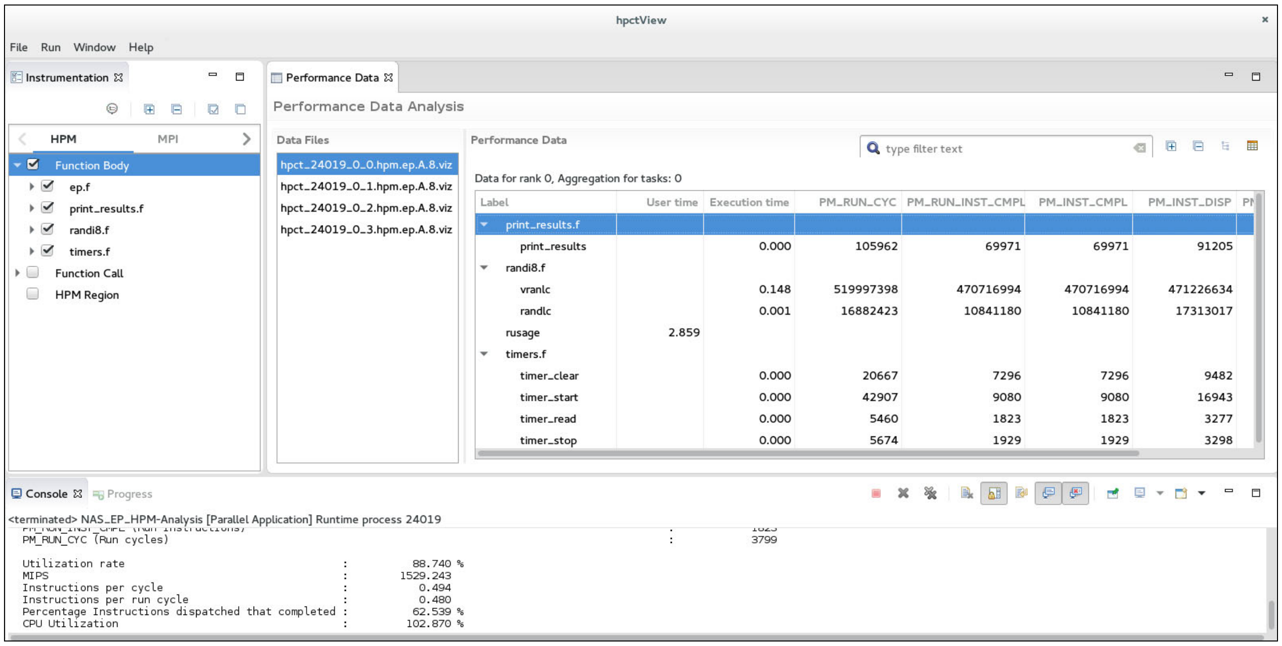
For single programs, tools such as Oprofile and Perf provide access to system-wide or application profiling of those hardware events on POWER8 processors. Parallel SPMD programs are often difficult to profile with these traditional tools because they are designed to deal with a single process (whether multi-threaded or not).

For more information about Oprofile, see the following resources:

- Oprofile tool website
- Perf tool Wiki webpage

For more information about predefined metrics, see the Derived metrics defined for POWER8 architecture page.

Figure 9-30 shows an HPM report that is opened by using hpctView.



Performance Data

Data for rank 0, Aggregation for tasks: 0

Label	User time	Execution time	PM_RUN_CYC	PM_RUN_INST_CMPL	PM_INST_CMPL	PM_INST_DISP
print_results.f						
print_results		0.000	105962	69971	69971	91205
randi8.f						
vranlc		0.148	519997398	470716994	470716994	471226634
randlc		0.001	16882423	10841180	10841180	17313017
rusage	2.859					
timers.f						
timer_clear		0.000	20667	7296	7296	9482
timer_start		0.000	42907	9080	9080	16943
timer_read		0.000	5460	1823	1823	3277
timer_stop		0.000	5674	1929	1929	3298

```
<terminated> NAS_EP_HPM-Analysis [Parallel Application] Runtime process 24019
  PM_RUN_CYC (Run cycles)                                                     :          3799

  Utilization rate                               :          88.740 %
  MIPS                                           :        1529.243
  Instructions per cycle                         :           0.494
  Instructions per run cycle                     :           0.480
  Percentage Instructions dispatched that completed :       62.539 %
  CPU Utilization                                :         102.870 %
```

Figure 9-30 hpctView: HPC tool visualization

GPU Performance Monitoring

The GPU Performance Monitoring (GPM) tool profiles and traces hardware events that are triggered in the GPU on hybrid MPI with CUDA C programs. Similar to HPM, GPM profiles raw hardware events and provides a set of predefined metrics that are calculated out of those events.

The HPM tool can profile raw hardware events and obtain metrics from a predefined list. The **gpmlist** command is the interface that is used to fetch the list of events and supported metrics. Example 9-17 shows a sample of the list of events that are available for the NVIDIA P100 GPU.

Example 9-17 How to list events that are available to the GPM profile tool

```
$ /opt/ibmhpc/ppedev.hpct/bin/gpmlist -d p100 -e -l
Domain 0
    83886081 tex0_cache_sector_queries - queries
    83886082 tex1_cache_sector_queries - queries
    83886083 tex0_cache_sector_misses - misses
    83886084 tex1_cache_sector_misses - misses
  Domain 1
    83886182 active_cycles - active_cycles
    83886183 elapsed_cycles_sm - elapsed_cycles_sm
  Domain 2
    83886085 fb_subp0_read_sectors - sectors
    83886086 fb_subp1_read_sectors - sectors
    83886087 fb_subp0_write_sectors - sectors
    83886088 fb_subp1_write_sectors - sectors
<... Output Omitted ...>
  Domain 3
    83886134 gld_inst_8bit - gld_inst_8bit
    83886135 gld_inst_16bit - gld_inst_16bit
    83886136 gld_inst_32bit - gld_inst_32bit
    83886137 gld_inst_64bit - gld_inst_64bit
    83886138 gld_inst_128bit - gld_inst_128bit
<... Output Omitted ...>
  Domain 4
    83886166 active_cycles_in_trap - active_cycles_in_trap
  Domain 5
    83886144 prof_trigger_00 - pmtrig0
    83886145 prof_trigger_01 - pmtrig1
<... Output Omitted ...>
```

Example 9-18 shows a sample of the metrics that are available for the NVIDIA P100 GPU.

Example 9-18 How to list metrics that are available to the GPM profile tool

```
$ /opt/ibmhpc/ppedev.hpct/bin/gpmlist -d p100 -m -l
    19922945 inst_per_warp - Instructions per warp
        83886156 inst_executed - inst_executed
        83886152 warps_launched - warps_launched
    19922946 branch_efficiency - Branch Efficiency
        83886177 branch - branch
        83886176 divergent_branch - branch
    19922947 warp_execution_efficiency - Warp Execution Efficiency
        83886157 thread_inst_executed - thread_inst_executed
        83886156 inst_executed - inst_executed
<... Output Omitted ...>
```

As with other HPC Toolkit tools, GPM is also flexible regarding the instrument-profile-visualize cycle possibilities. It can also be used with the HPM tool.

The following environment variables are used to configure the profiler:

- **GPM_METRIC_SET**: Sets the metrics to be profiled.
- **GPM_EVENT_SET**: Sets the hardware events to be profiled. Cannot be exported with **GPM_METRIC_SET**.
- **GPM_VIZ_OUTPUT=y**: Enables the creation of visualization files that are used by the hpctView visualizer (disabled by default).
- **GPM_STDOUT=n**: Suppresses the profiler messages to standard output (`stdout`). Sends messages to `stdout` by default.
- **GPM_ENABLE_TRACE=y**: Turns on trace mode (is off by default).

Example 9-19 shows a profiling session of a hybrid MPI and CUDA C application that is named `a.out`. The lines 1 - 26 are the content of the **gpm.sh** script, which exports the GPM control variables (lines 13 - 15) that instruct the system to calculate the `sm_efficiency` metric. Then, they evoke the wrapper that is named `gpm_wrap.sh`. In turn, the wrapper (lines 29 - 35) script preinstalls (line 33) the GPM library. The remained lines are messages that are printed by GPM on standard output.

Example 9-19 Profiling with the GPM tool

```
  1 $ cat gpm.sh
  2
  3 #!/bin/bash
  4
  5
```

```

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-7.5/lib64:/opt/ibmhpc/pecurrent/mpich/gnu/lib64:/opt/ibmhpc/pecurrent/gnu/lib64/



# GPU Performance Monitor - variables

export GPM_METRIC_SET=sm_efficiency

export GPM_VIZ_OUTPUT=y



export MP_CUDA_AWARE=yes

export MP_RESD=poe

export MP_PROCS=2

poe ./gpm_wrap.sh

$ cat gpm_wrap.sh

#!/bin/bash



export LD_PRELOAD=/opt/ibmhpc/ppedev.hpct/lib64/preload/libgpm.so:$LD_PRELOAD

./a.out

$ ./gpm.sh



 GPM (IBM HPC Toolkit for PE Developer Edition) results



Device 0 (Tesla K80):

------------------------------------------------

  Symbol Name: cudaMalloc

  ----------------------------------------------

             Memory Kind: cudaMalloc

            Bytes Copied: 4096



```

```
Symbol Name: cudaMemcpy

----------------------------------------------

           Memory Kind: cudaMemcpyHostToDevice

          Bytes Copied: 4096



Kernel Name: _Z12vecIncKernelPii

---------------------------------------------

       GPU Kernel Time: 0.000005 seconds

          W/clock Time: 0.012338 seconds

         sm_efficiency: 31.155739%



Symbol Name: cudaMemcpy

----------------------------------------------

           Memory Kind: cudaMemcpyDeviceToHost

          Bytes Copied: 4096



Totals for device 0

----------------------------------------------

        Total Memory Copied: 8192 bytes

      Total GPU Kernel Time: 0.000005 seconds

         Total W/clock Time: 0.012338 seconds

              sm_efficiency: 31.155739%

```

As a result, the tool reports the calculated metric or counted event on per-task files. Example 9-20 shows the report that is generated for execution in Example 9-19 on page 305.

Example 9-20 Report that is generated by the GPM tool

```
$ cat hpct_0_0.gpm.a.out.txt

 GPM (IBM HPC Toolkit for PE Developer Edition) results

  Totals for device 0
  ----------------------------------------------
```

```
   Total Memory Copied: 8192 bytes
 Total GPU Kernel Time: 0.000005 seconds
    Total W/clock Time: 0.012338 seconds
         sm_efficiency: 31.155739%
```

For more information about the GPM tool, see the Using GPU hardware counter profile page.

IBM HPC Toolkit hpctView

The hpctView is a front-end view for the Parallel Performance Toolkit runtime and command-line tools. It provides a GUI to instrument the parallel executable file, which you can run to collect data and visualize information from your development workstation (desktop or notebook).

The tool can be run from the developer workstation. Version 2.3 is supported on Mac OS 10.9 (Mavericks) or later, Microsoft Windows 64-bit, and any 64-bit Linux distribution.

You can find hpctView as a `.tar` file in the toolkit's distribution media. Then, proceed as shown in the following example to install it and run it on Linux workstation:

```
$ tar xvf hpctView-2.4.0-1-linux-gtk-x86_64.tar.gz
$ cd hpctView
$ ./hpctView &
```

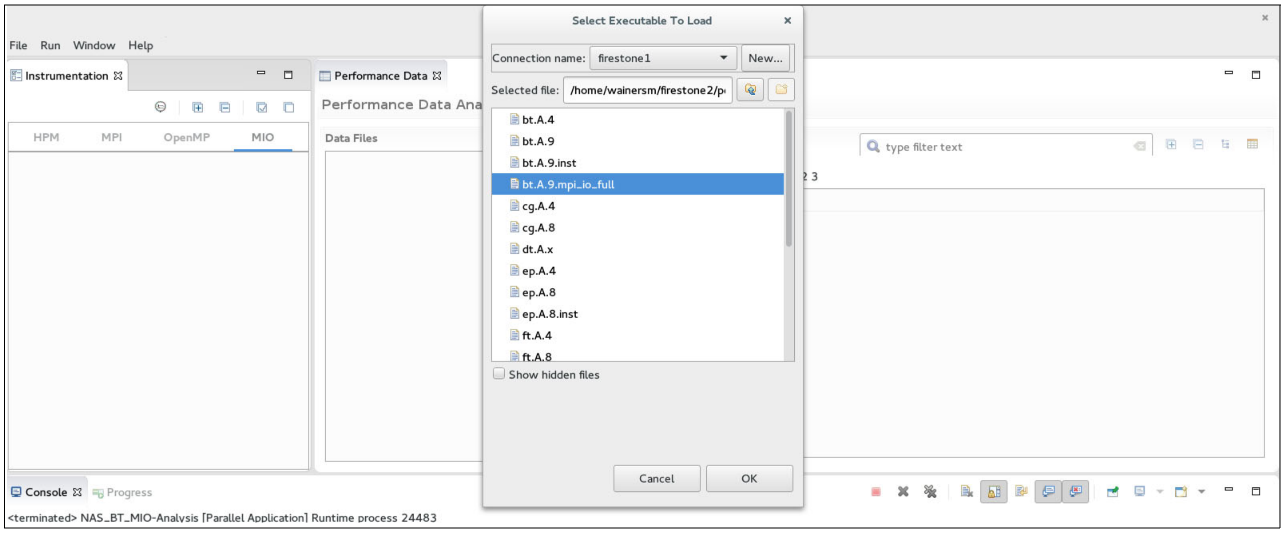
The hpctView application implements the complete cycle of instrument-run-visualize (see 9.6.1, "Parallel Performance Toolkit" on page 295) that is required to use the Parallel Performance Toolkit tools. As an alternative, profile and trace data files can be generated by using the command-line tools and libraries and then loaded into hpctView for visualization and analysis.

In addition to the hpctView stand-alone application, the toolkit includes plug-ins to enable hpctView within Eclipse IDE. For more information, see 9.6.3, "Eclipse for Parallel Application Developers" on page 310.

The first step to profile and trace an application is to prepare the program execution. To instrument the binary file, complete the following steps in hpctView:

1. In the Instrumentation pane (left side pane), select the profiler for which you want to instrument the binary. The options are HPM, MPI, OpenMP, and MPI-IO.

2. Click **File** → **Open Executable**. A window opens that includes a connection to the remote machine. Select the file to be instrumented, as shown Figure 9-31.



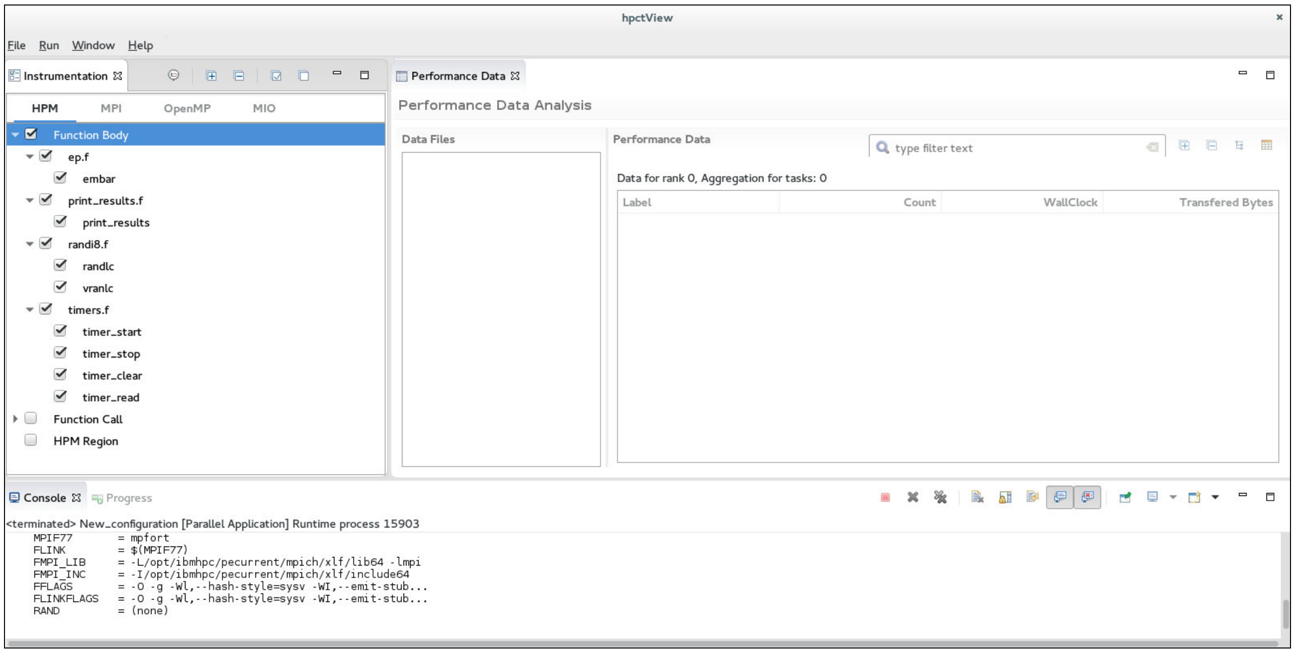
```
<terminated> NAS_BT_MIO-Analysis [Parallel Application] Runtime process 24483
```

Figure 9-31 hpctView: Selecting the binary file for instrumentation

3. Click **New** to create a connection to the remote system where the binary file is hosted if the connection does not exist.

4. Select the binary file. Click **OK**.

   The binary file content is analyzed and sections that correspond to the instrumentation portions of the application code are listed in tree format in the Instrumentation pane. Different instrumentation options are displayed for each type of profiling tool. For example, Figure 9-32 shows the options (Function body, Function Call, and HPM Region) for the instrumentation of a binary file that will be used with the HPM tool. Select the instrumentation points according to your needs.



```
<terminated> New_configuration [Parallel Application] Runtime process 15903
    MPIF77      = mpfort
    FLINK       = $(MPIF77)
    FMPI_LIB    = -L/opt/ibmhpc/pecurrent/mpich/xlf/lib64 -lmpi
    FMPI_INC    = -I/opt/ibmhpc/pecurrent/mpich/xlf/include64
    FFLAGS      = -O -g -Wl,--hash-style=sysv -WI,--emit-stub...
    FLINKFLAGS  = -O -g -Wl,--hash-style=sysv -WI,--emit-stub...
    RAND        = (none)
```

Figure 9-32 hpctView: Binary file instrumentation

5. Click the instrument executable icon in the Instrumentation pane. If the operation succeeds, a file that is named `<binary>.inst` is saved in the same folder as the original binary file.

From within hpctView, you can then start the instrumented binary file by using IBM Spectrum Message Passing Interface (IBM Spectrum MPI) or submitting a batch job to IBM Spectrum LSF. The Parallel Performance Toolkit provides many environment variables to control the tools behavior (for example, the amount of data to be collected).

After the instrumented binary file is run, a view of the corresponding tool automatically opens. Now, you can use the many options that are available to analyze the data that is collected, for example, the interval trace viewer.

For more information about using hpctView, see the Using the hpctView application page.

9.6.2 Parallel application debuggers

The GNU Debugger (GDB) is a debugger for serial and multi-threaded application. It also can be used to debug parallel SPMD programs. By using the IBM Spectrum MPI, the GDB can be attached to individual MPI processes or start many debug instances by running the **`mpirun`** command. Both techniques require a complex setup, do not scale well, and often do not deliver suitable results.

Using specialized debuggers for parallel applications is recommended. The following products support debugging of C, C++, and Fortran applications in Power Systems servers:

- Allinea DDT debugger
- RogueWave TotalView

These parallel debuggers are tightly integrated with IBM Spectrum MPI. Using the **`mpirun --debug`** command starts DDT or TotalView if they are reachable in the system's PATH.

For more information about DDT and TotalView integration with IBM Spectrum MPI, see the following pages:

- Debugging applications with the Allinea DDT debugger and IBM Spectrum MPI
- Debugging applications with the TotalView debugger and IBM Spectrum MPI

9.6.3 Eclipse for Parallel Application Developers

Eclipse for Parallel Application Developers provides an entire IDE. It bundles some open source projects under the Eclipse umbrella, such as C/C++ Development Tools (CDT) and Eclipse PTP. It provides a complete environment for coding, compiling, starting, analyzing, and debugging applications that are written in C, C++, and Fortran. It uses MPI, OpenMP, PAMI, and OpenSHMEM.

The specialized editors feature syntax highlighting, code assistance, place markers for compiler errors, auto-completion, and many other features to improve productivity. They also include static analysis tools that are used to identify common coding mistakes, such as MPI barriers mismatch.

You can build programs by using Eclipse managed makefiles (integrated with IBM XL and GNU compilers), Makefile, Autotools, and custom scripts and commands. The integration with IBM XL and GNU compilers also includes build output parsers that can correlate errors to source code files.

Remotely started applications on IBM Spectrum MPI (OpenMPI) or IBM Spectrum LSF are supported.

The Eclipse for Parallel Application Developers parallel debugger provides specific debugging features for parallel applications that distinguish it from the Eclipse debugger for serial applications. Its parallel debugger can also start and remotely debug parallel applications through OpenMPI and IBM Spectrum LSF.

For more information about the topics that are introduced in this section, see Parallel Tools Platform (PTP) User Guide.

To install Eclipse, download its `.tar` file and then decompress it on your development workstation. For more information, see **Eclipse for Parallel Application Developers**. At the time of writing, the latest Eclipse version was 4.6.2 (also known as Eclipse Neon.2).

Remote synchronized development model

The Eclipse for Parallel Application developers implemented a remote development working model in which the code editing is done locally within a workbench (GUI). However, other tasks that are required to be performed on the server side, such as build, start, debug, and profile, are carried out remotely.

The model uses a synchronized project type to keep local and remote copies of the working directory updated so that code programming is minimally affected by network latency or slow response time in the editor. The C, C++, and Fortran editors appear as though you are developing locally on the Power Systems machine even though all of the resources are on the server side. By using 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.

Parallel debugger

The Eclipse PTP parallel debugger extends the default C/C++ Eclipse debugger for serial application to scale parallel programs. It combines the information that is obtained from the many processes and threads that a parallel job is composed of into a single viewer. Therefore, you can browse the entities separately, if needed.

Debugging operations can be carried out on any arbitrary subset of processes within a parallel job; for example, as step in and out, and stop at breakpoint. A different type of breakpoint, which is called a *parallel breakpoint*, can be applied in these collections of processes.

The debugger can start and remotely debug parallel programs by using several types of workload managers (WLMs), including the OpenMPI and IBM Spectrum LSF.

Using IBM hpctView within Eclipse

The IBM Parallel Performance Toolkit provides plug-ins for hpctView (see9.6.1, "Parallel Performance Toolkit" on page 295) that you can install on top of Eclipse.

Find the plug-ins in the Parallel Performance Toolkit distribution media. Then, decompress the file by running the following commands:

- **`$ mkdir ppedev_update-2.3.0`**
- **`$ unzip ppedev_update-2.3.0-0.zip -d ppedev_update-2.3.0`**

Next, install the plug-ins by using the Eclipse installation software by completing the following steps:

1. From the tool bar menu, select **Help** → **Install New Software**.
2. Click **Add** to create a repository location from where Eclipse can find plug-ins to be installed.
3. Enter the location where the hpctView update site files were decompressed. Click **OK** to add the repository configuration.
4. In the selection tree that appears, select its root element to install all of the plug-ins. Click **Next** and then `Next` again.
5. Read and accept the license agreement to continue the installation.
6. Click **Finish** to install the plug-ins.

For more information about installing the plug-ins, see Updating and installing software. In the left side navigation tree at the website, click **Workbench User Guide** → **Tasks** → **Updating and installing software**.

To use hpctView, open its perspective within Eclipse by completing the following steps:

1. Click **Window** → **Perspective** → **Open Perspective** → **Other**.
2. In the perspective window that opens, select **HPCT** and click **OK**.

9.6.4 NVIDIA Nsight Eclipse Edition for CUDA C/C++

The NVIDIA Nsight Eclipse Edition is a complete IDE for development of CUDA C and C++ applications that run on NVIDIA GPUs. It provides developers with an Eclipse-based GUI that includes tools for a complete cycle of development: Code writing, compiling, debugging, and profiling and tuning.

The following development models are available:

- Local: The complete development cycle is carried out at the machine where Nsight is running. The program that is produced is targeted for the local host architecture and GPU.
- Remote: The complete development cycle is carried out at a remote host where Nsight is running.

In the next sections, we describe the basic use of Nsight for remote development of CUDA C applications for Power Systems servers. The examples that are described in this section use a workstation with Linux Fedora 23 64-bit that is running the NVIDIA Nsight Eclipse Edition that is included with the CUDA Toolkit 8.0.

Not all of the Nsight functions are described in this book. Instead, we show how to create, compile, and debug projects. For more information, see the Nsight Eclipse Edition website.

Running NVIDIA Nsight Eclipse Edition

To start the Nsight GUI, run the following commands:

- `$ export PATH=/usr/local/cuda-8.0/bin:$PATH`
- `$ nsight &`

Creating a project for remote development

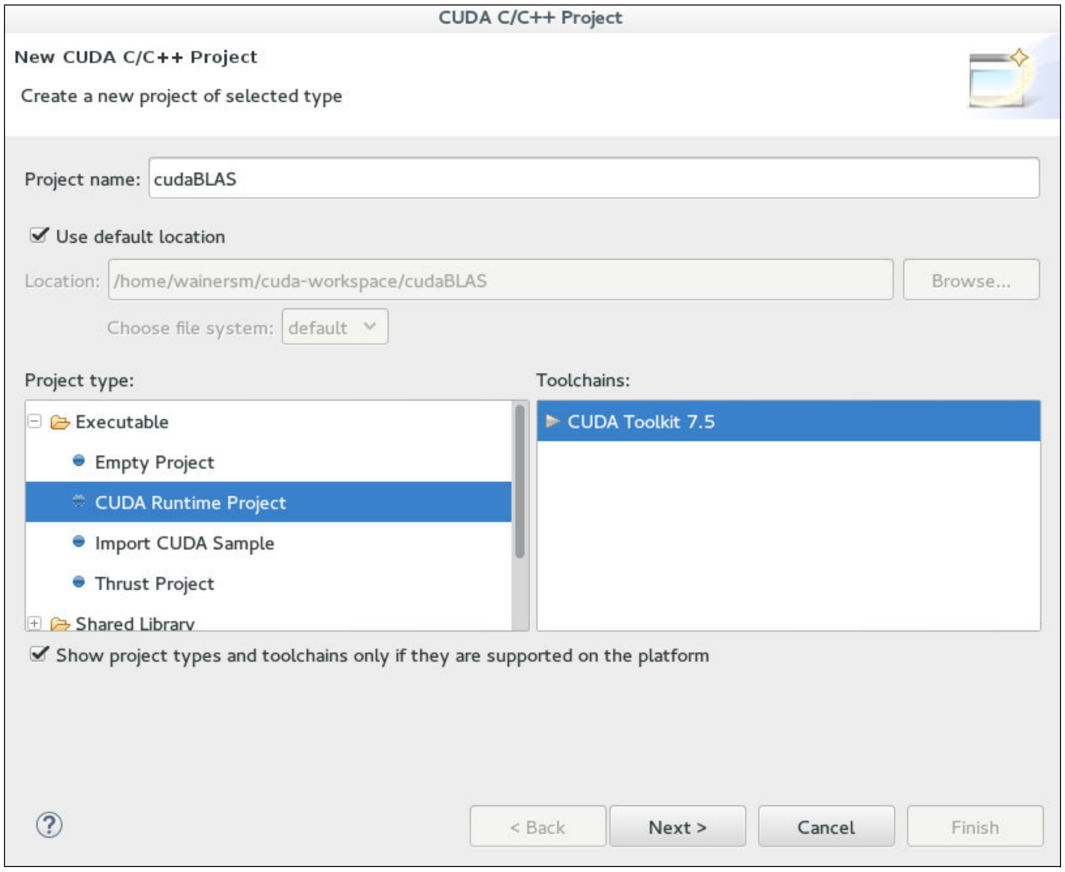
Before you begin, check the requirements and complete the following steps:

1. Ensure that the machine (usually the cluster login node) you are going to use for remote development has Git installed. Nsight uses **`git`** commands to synchronize local and remote copies of the project files.
2. Connect to the remote machine to configure the user name and email that is used by Git by running the following commands:
   - **`$ git config --global user.name "Jan-Frode Myklebust"`**
   - **`$ git config --global user.email "jan-frode@no.ibm.com"`**
3. Ensure that the directory that is going to hold the project files in the remote machines is created by running the following command:

   ```
   $ mkdir -p ~/cudaBLAS
   ```

In the Nsight GUI, to create a CUDA C/C++ project, complete the following steps:

1. On the main menu toolbar, click **File** → **New** → **CUDA C/C++ Project** to open the window that is shown in Figure 9-33.



CUDA C/C++ Project

New CUDA C/C++ Project

Create a new project of selected type

Project name: cudaBLAS

✓ Use default location

Location: /home/wainersm/cuda-workspace/cudaBLAS

Choose file system: default

Project type:
- Executable
  - Empty Project
  - CUDA Runtime Project
  - Import CUDA Sample
  - Thrust Project
- Shared Library

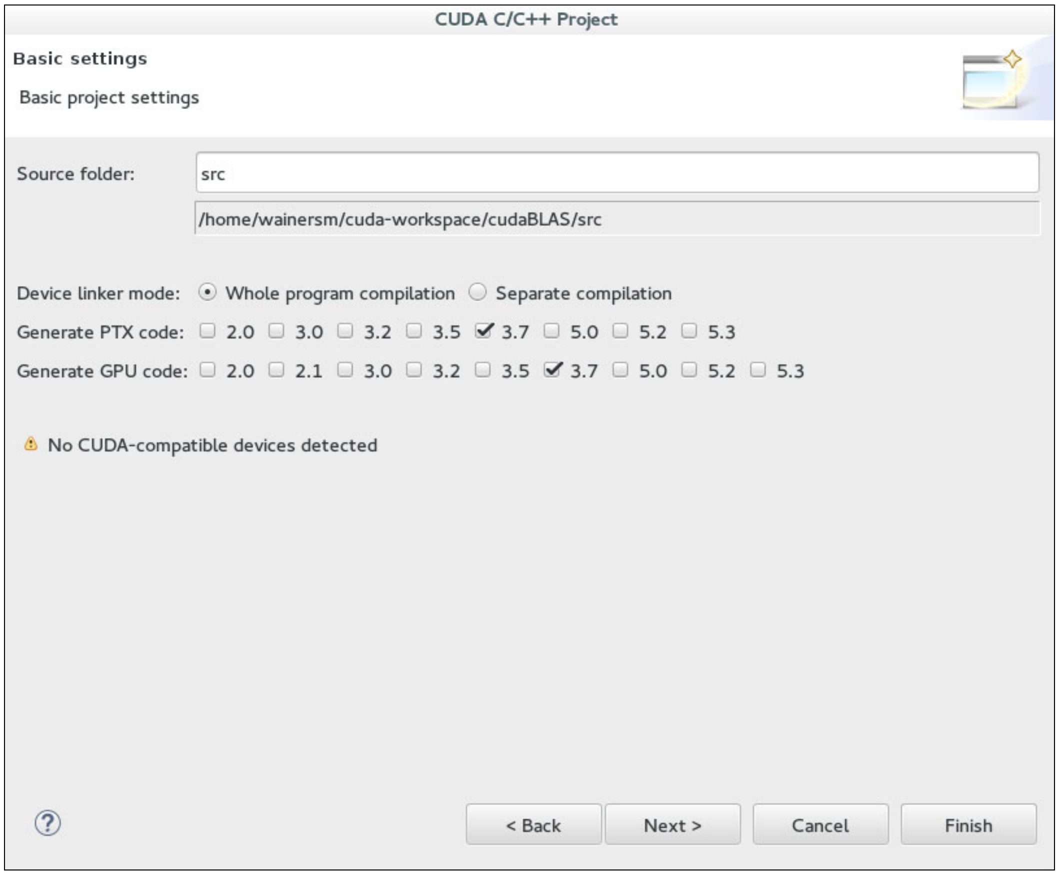
Toolchains:
- CUDA Toolkit 7.5

✓ Show project types and toolchains only if they are supported on the platform

Figure 9-33 Nsight: New CUDA C/C++ project wizard

2. Click **Next** and then click **Next** again.

3. Select **3.7** for the **Generate PTX code** and **Generate GPU code** options, as shown in Figure 9-34. Click **Next**.


CUDA C/C++ Project

Basic settings

Basic project settings

Source folder: src

/home/wainersm/cuda-workspace/cudaBLAS/src

Device linker mode: [✓ Whole program compilation] Separate compilation

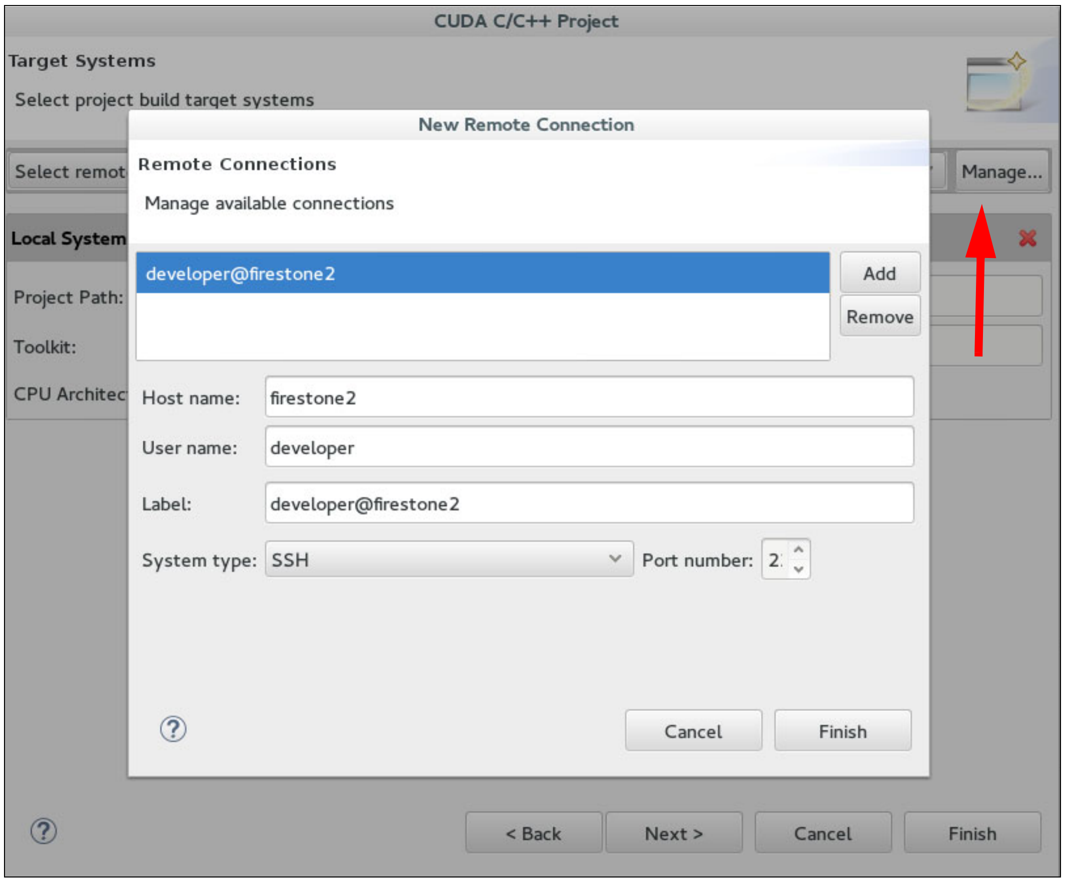
Generate PTX code: 2.0 3.0 3.2 3.5 [✓ 3.7] 5.0 5.2 5.3

Generate GPU code: 2.0 2.1 3.0 3.2 3.5 [✓ 3.7] 5.0 5.2 5.3

No CUDA-compatible devices detected

Figure 9-34 Nsight: New CUDA C/C++ project basic settings

4. Click **Manage** (as shown in Figure 9-35) to configure a new connection to the remote machine.



developer@firestone2

Host name: firestone2

User name: developer

Label: developer@firestone2

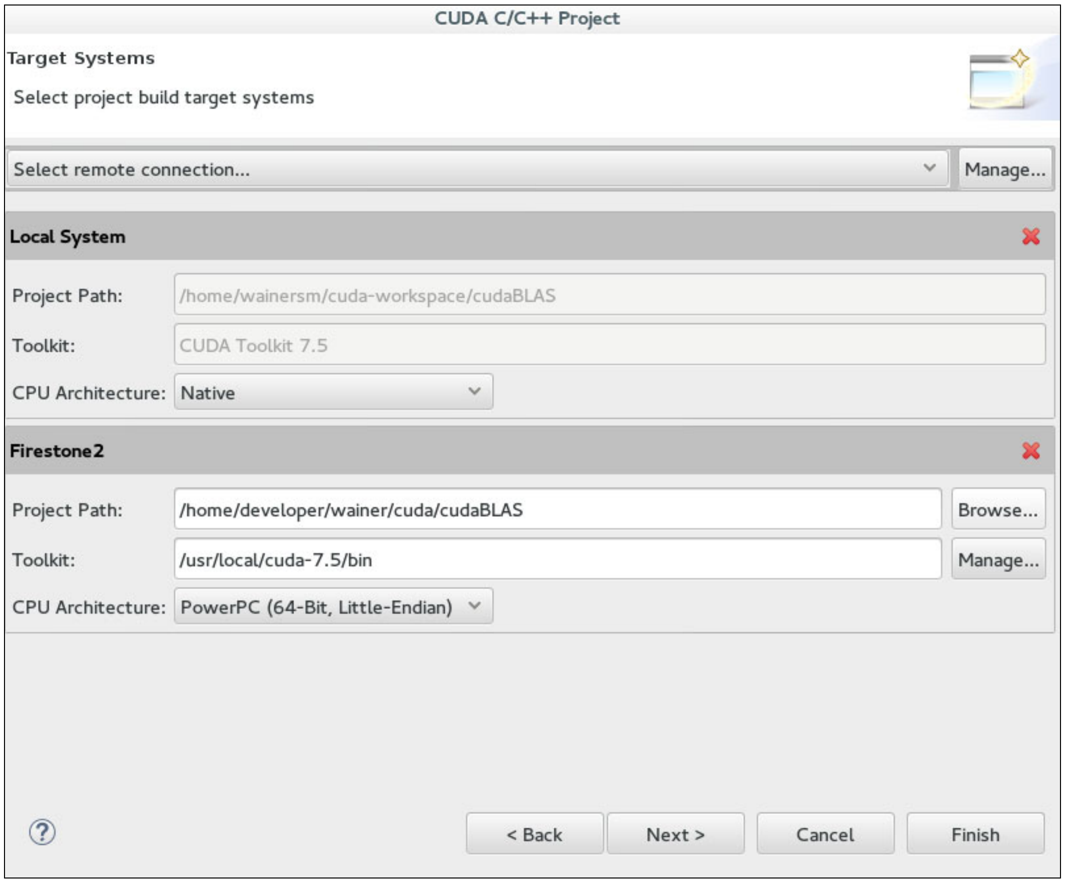
System type: SSH

Port number: 2:

Figure 9-35 Nsight: New CUDA C/C++ project remote connection setup

5. As shown in Figure 9-35, enter the connection information fields (host name and user name). Click **OK**.

6. Enter the Project path and Toolkit information and complete the CPU Architecture fields for the newly created connection (see Figure 9-36). You must remove the Local System configuration. Click **Next** and then **Finish**.


CUDA C/C++ Project

Target Systems

Select project build target systems

Select remote connection...

Local System

Project Path: /home/wainersm/cuda-workspace/cudaBLAS

Toolkit: CUDA Toolkit 7.5

CPU Architecture: Native

Firestone2

Project Path: /home/developer/wainer/cuda/cudaBLAS

Toolkit: /usr/local/cuda-7.5/bin

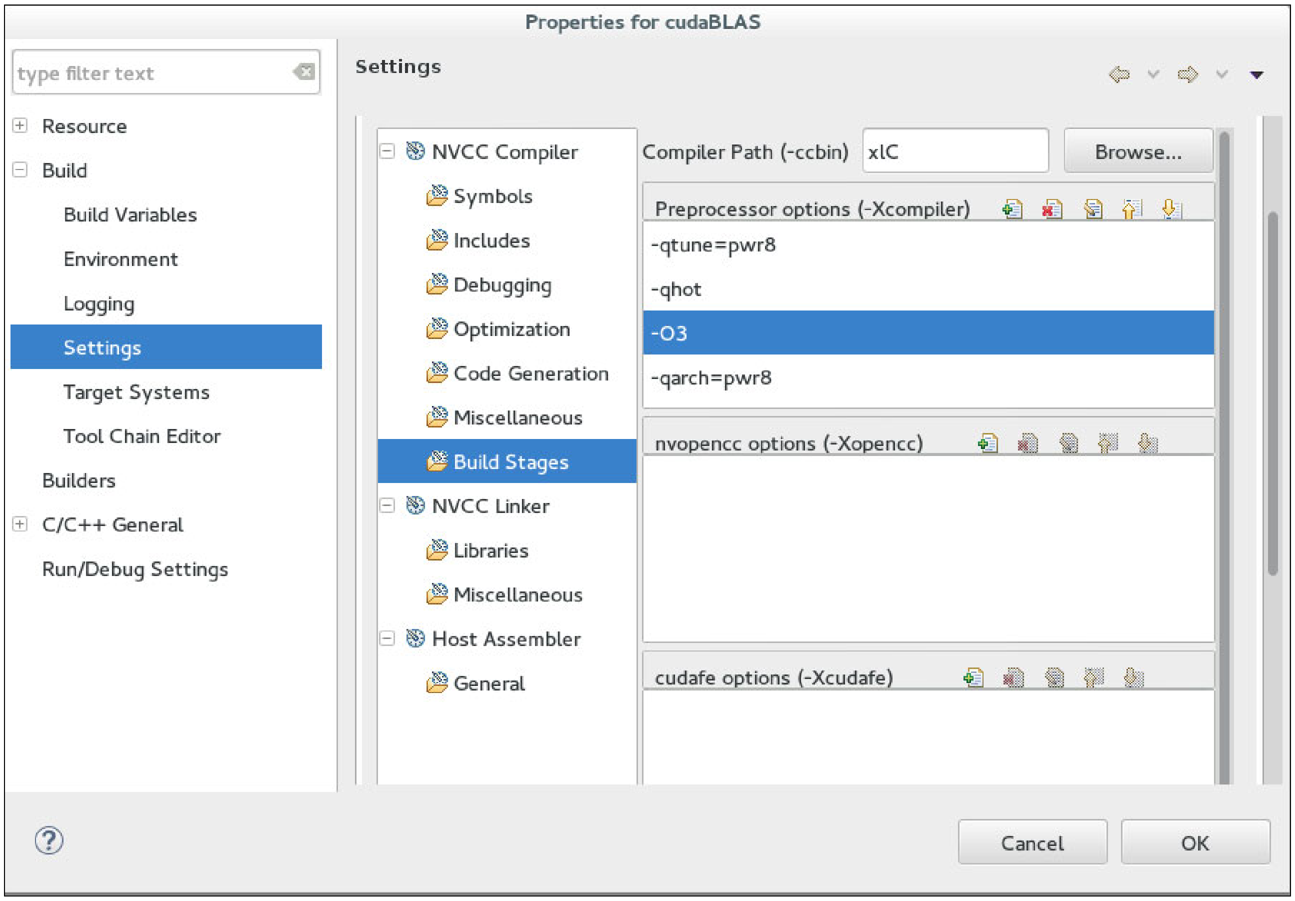
CPU Architecture: PowerPC (64-Bit, Little-Endian)

Figure 9-36 Nsight: New CUDA C/C++ project further remote configuration setup

After you successfully create the project, complete the following steps to configure the Nsight to automatically synchronize the local and remote project files:

1. Create a source code file (for example, `main.c`).
2. Right-click the project in the Project Explorer pane and select **Synchronize** → **Set active**. Then, choose the option that matches the name of the connection configuration to the remote machine.
3. Right-click the project again and select **Synchronize** → **Sync Active now** to perform the first synchronization between local and remote folders. (A synchronization can be performed at any time.) Nsight is configured to perform a synchronization after or before some tasks (for example, before compiling the project).

4. As an optional step, set the host compiler that is evoked by **`nvcc`**. Right-click the project name and select **Properties**. Expand **Build**, and then choose **Settings**. Click the **Tool Settings** tab and select **Build Stages**. Enter the necessary information in the Compiler Path and Preprocessor options fields (see Figure 9-37). Change the content of the **Compiler Path** under the Miscellaneous section of the NVCC Linker window.



Compiler Path (-ccbin): xlC

Preprocessor options (-Xcompiler):

```
-qtune=pwr8
-qhot
-O3
-qarch=pwr8
```

Figure 9-37 Nsight: New CUDA C/C++ project build settings

After completing these steps, the new project is ready for CUDA C/C++ programming. You can use many of the features that are provided by the Nsight C and C++ code editor, such as syntax highlighting, code completion, static analysis, and error markers.

Compiling the application

To build the application, complete the following steps:

1. Right-click the project in the Project Explorer pane and select **Run As** → **Remote C/C++ Application**.
2. Check that all the information is correct and change any information, if needed. Then, click **Run**.

Debugging the application

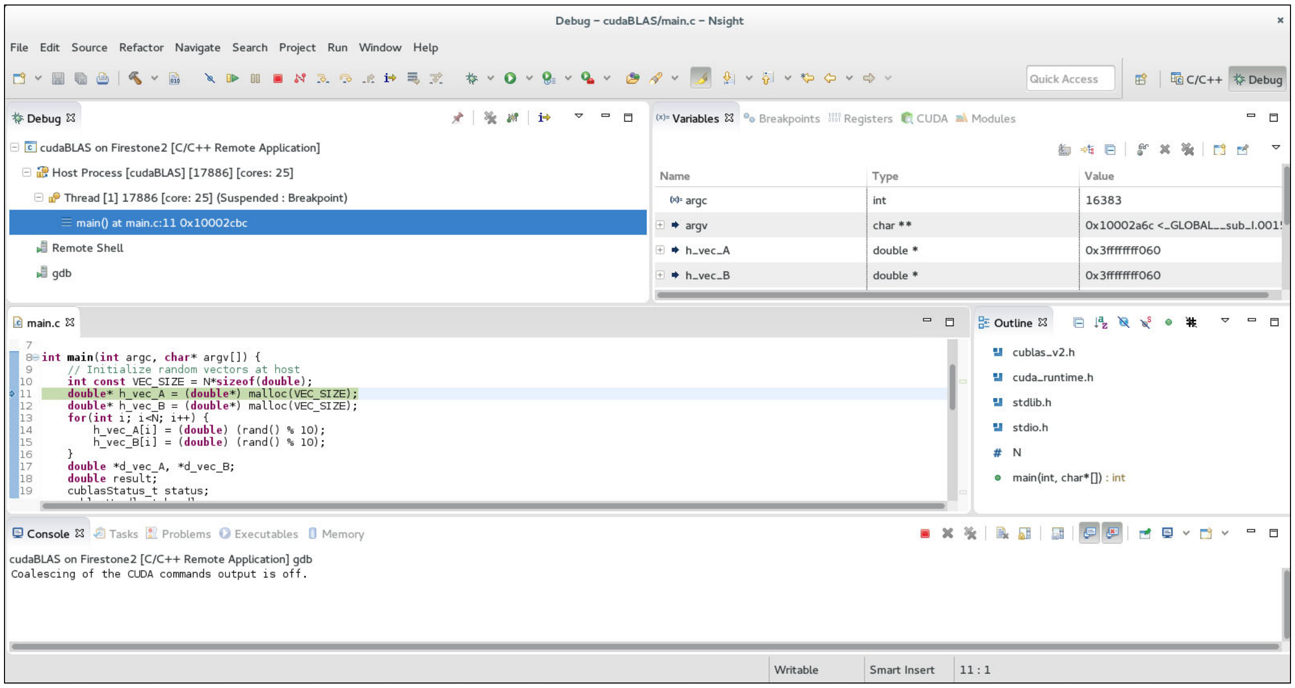
The steps to start the Nsight debugger are similar to the steps that are used to run the application. However, the executable file is started by running the **`cuda-gdb`** command, which in turn is backed by the GNU GDB debugger tool.

For more information about the **cuda-gdb** command, see 9.6.5, “Command-line tools for CUDA C/C++” on page 318.

To debug the application, complete the following steps:

1. Right-click project name in the Project Explorer pane and select **Debug As** → **Remote C/C++ Application**.
2. A window opens in which you are prompted for permission to open the Eclipse Debug perspective. Click **Yes**.

By default, the debugger stops at the first instruction on the main method. Figure 9-38 shows the Nsight debugger.



cudaBLAS on Firestone2 [C/C++ Remote Application]
Host Process [cudaBLAS] [17886] [cores: 25]
Thread [1] 17886 [core: 25] (Suspended : Breakpoint)
main() at main.c:11 0x10002cbc
Remote Shell
gdb

Name	Type	Value
argc	int	16383
argv	char **	0x10002a6c <_GLOBAL__sub_I.001!
h_vec_A	double *	0x3fffffff060
h_vec_B	double *	0x3fffffff060

```
int main(int argc, char* argv[]) {
    // Initialize random vectors at host
    int const VEC_SIZE = N*sizeof(double);
    double* h_vec_A = (double*) malloc(VEC_SIZE);
    double* h_vec_B = (double*) malloc(VEC_SIZE);
    for(int i; i<N; i++) {
        h_vec_A[i] = (double) (rand() % 10);
        h_vec_B[i] = (double) (rand() % 10);
    }
    double *d_vec_A, *d_vec_B;
    double result;
    cublasStatus_t status;
```

cublas_v2.h
cuda_runtime.h
stdlib.h
stdio.h
N
main(int, char*[]) : int

```
cudaBLAS on Firestone2 [C/C++ Remote Application] gdb
Coalescing of the CUDA commands output is off.
```

Figure 9-38 Nsight: CUDA C/C++ debugger

9.6.5 Command-line tools for CUDA C/C++

The NVIDIA Toolkit 8.0 provides the following tools for debugging problems on CUDA C and C++ applications.

CUDA-MEMCHECK

The CUDA-MAMCHECK tool detects memory-related flaws on programs. It dynamically instruments the program executable file at run time and can check allocation and deallocation and accesses on global and shared memories.

In addition to memory checking, CUDA-MEMCHECK also can inspect the application to detect the following types of errors:

- Race condition: Checks for the race condition for access of shared and local variables. Uses the **--tool racecheck** option.
- initcheck: Checks for use of non-initialized memory. Uses the **--tool initcheck** option.
- synccheck: Checks for synchronization errors. Uses the **--tool synccheck** option.

For complete visualization of the source code file and the line number where the detected problems occur, the binary file must be compiled by using the **`nvcc -g -lineinfo`** options.

For more information about the CUDA-MEMCHECK tool, see the CUDA-MEMCHECK user's manual.

CUDA GDB

The CUDA application portions of the code that is run on CPU (host) and GPUs (devices), and programs often have thousands of CUDA threads that are spread across many GPUs. Although traditional debuggers work effectively with CPU code, they are not built to deal with both.

The **`cuda-gdb`** tool is an implemented debugger extension of GNU GDB that is designed to deal with the scenario that the CUDA hybrid model imposes.

The nvprof tool

The **`nvprof`** tool is an extensive command-line profiling tool that is distributed with CUDA Toolkit. It is flexible, and profile all processes in the entire system, only a specific application, or only some elements of it (for example, kernels, streams, and contexts). Also, you may select which GPU device must be profiled (the default is all the GPUs on the system).

By default, **`nvprof`** timing profiles the calls to kernel functions and CUDA API, as shown in Example 9-21, "Time profiling kernels and CUDA API" on page 319.

Example 9-21 Time profiling kernels and CUDA API

```
$ nvprof --log-file profile.out ./MonteCarloMultiGPU 2&> /dev/null
$ cat profile.out
==73495== NVPROF is profiling process 73495, command: ./MonteCarloMultiGPU 2
==73495== Profiling application: ./MonteCarloMultiGPU 2
==73495== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 79.29%  97.066ms         4  24.267ms  24.169ms  24.396ms
MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
 20.67%  25.306ms         4  6.3264ms  6.2893ms  6.3711ms
rngSetupStates(curandStateXORWOW*, int)
  0.03%  37.344us         4  9.3360us  8.9600us  10.048us  [CUDA memcpy HtoD]
  0.01%  12.928us         4  3.2320us  2.8480us  3.5840us  [CUDA memcpy DtoH]

==73495== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 85.47%  547.62ms         4  136.90ms  131.25ms  143.30ms  cudaStreamCreate
  3.77%  24.160ms         4  6.0401ms  195.00us  22.889ms  cudaEventSynchronize
  2.03%  12.999ms        12  1.0832ms  1.0394ms  1.1578ms  cudaMalloc
  1.84%  11.818ms        16  738.65us  714.22us  818.61us  cudaGetDeviceProperties
  1.19%  7.6318ms         4  1.9079ms  1.8529ms  1.9930ms  cudaMallocHost
  1.18%  7.5904ms         4  1.8976ms  1.8906ms  1.9076ms  cudaHostAlloc
  1.04%  6.6817ms         8  835.22us  17.319us  1.6565ms  cudaLaunch
  0.98%  6.2936ms         4  1.5734ms  14.881us  6.2399ms  cudaDeviceSynchronize
  0.86%  5.5070ms         8  688.37us  673.41us  748.69us  cudaFreeHost
  0.66%  4.2604ms         4  1.0651ms  1.0615ms  1.0672ms  cuDeviceTotalMem
  0.48%  3.0439ms       364  8.3620us     394ns  303.12us  cuDeviceGetAttribute
  0.38%  2.4584ms        12  204.87us  194.19us  230.03us  cudaFree
  0.04%  251.96us         4  62.989us  61.403us  64.379us  cuDeviceGetName
  0.02%  132.76us         8  16.594us  11.935us  31.873us  cudaMemcpyAsync
  0.01%  77.114us        28  2.7540us  1.6600us  15.832us  cudaSetDevice
  0.01%  53.252us         4  13.313us  12.214us  16.493us  cudaStreamDestroy
  0.00%  23.942us         4  5.9850us  5.1930us  7.5630us  cudaEventRecord
  0.00%  21.598us        28     771ns     662ns  1.2580us  cudaSetupArgument
  0.00%  19.164us         4  4.7910us  4.4390us  5.1540us  cudaEventCreate
  0.00%  17.214us         4  4.3030us  2.7130us  5.3020us  cudaEventDestroy
  0.00%  8.7520us         8  1.0940us     744ns  1.6830us  cudaConfigureCall
  0.00%  7.1430us        12     595ns     422ns     795ns  cuDeviceGet
  0.00%  7.1380us         8     892ns     707ns  1.1000us  cudaGetLastError
  0.00%  6.2230us         3  2.0740us     719ns  4.6540us  cuDeviceGetCount
  0.00%  4.3670us         1  4.3670us  4.3670us  4.3670us  cudaGetDeviceCount
```

Time profiling is a good start point towards improving an application. As shown in Example 9-21, "Time profiling kernels and CUDA API" on page 319, it reveals that the `MonteCarloOneBlockPerOption` kernel ran in a total of 97.066 ms (79.29%), although `cudaStreamCreate API` calls used 547.62 ms (85.47%).

Another way to view the application behavior that is shown in Example 9-21, "Time profiling kernels and CUDA API" on page 319 is to determine the execution critical path that is combined with the information that is gathered from CPU (CUDA API calls) and GPU (kernel evocations) executions. The **nvprof** dependency analysis mode[8] produces a report for the critical path when the option **--dependency-analysis** is used (see Example 9-22).

Example 9-22 Reporting the execution critical path

```
$ nvprof --log-file profile.out --dependency-analysis ./MonteCarloMultiGPU 2&>
/dev/null
$ cat profile.out
==73532== NVPROF is profiling process 73532, command: ./MonteCarloMultiGPU 2
==73532== Profiling application: ./MonteCarloMultiGPU 2
==73532== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 79.32%  97.266ms         4  24.316ms  24.010ms  24.473ms
MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
 20.64%  25.310ms         4  6.3274ms  6.2963ms  6.3661ms
rngSetupStates(curandStateXORWOW*, int)
  0.03%  36.544us         4  9.1360us  8.7680us  9.6000us  [CUDA memcpy HtoD]
  0.01%  12.992us         4  3.2480us  2.8160us  3.6800us  [CUDA memcpy DtoH]

==73532== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 86.40%  590.17ms         4  147.54ms  138.74ms  162.78ms  cudaStreamCreate
  3.51%  23.995ms         4  5.9988ms  38.611us  23.186ms  cudaEventSynchronize
  1.92%  13.088ms        12  1.0907ms  1.0670ms  1.1258ms  cudaMalloc
  1.70%  11.595ms        16  724.69us  713.95us  737.43us  cudaGetDeviceProperties
  1.13%  7.6917ms         4  1.9229ms  1.9016ms  1.9583ms  cudaMallocHost
  1.12%  7.6606ms         4  1.9152ms  1.8882ms  1.9460ms  cudaHostAlloc
  0.97%  6.6572ms         8  832.15us  17.940us  1.6505ms  cudaLaunch
  0.92%  6.2873ms         4  1.5718ms  13.461us  6.2349ms  cudaDeviceSynchronize
  0.81%  5.5394ms         8  692.43us  675.34us  759.93us  cudaFreeHost
  0.62%  4.2598ms         4  1.0650ms  1.0618ms  1.0691ms  cuDeviceTotalMem
  0.45%  3.0713ms       364  8.4370us     390ns  308.84us  cuDeviceGetAttribute
  0.36%  2.4474ms        12  203.95us  193.04us  230.29us  cudaFree
```

[8] New in **nvprofin** NVIDIA CUDA Toolkit 8.0.

```
  0.04%  254.18us         4  63.544us  61.981us  65.150us  cuDeviceGetName
  0.02%  135.26us         8  16.907us  11.896us  31.121us  cudaMemcpyAsync
  0.01%  73.833us        28  2.6360us  1.5100us  15.121us  cudaSetDevice
  0.01%  52.310us         4  13.077us  11.761us  16.365us  cudaStreamDestroy
  0.00%  24.634us         4  6.1580us  5.3050us  7.7970us  cudaEventRecord
  0.00%  21.074us        28     752ns     666ns  1.1720us  cudaSetupArgument
  0.00%  18.341us         4  4.5850us  4.2260us  5.2930us  cudaEventCreate
  0.00%  15.729us         4  3.9320us  2.6210us  5.1060us  cudaEventDestroy
  0.00%  8.5760us         8  1.0720us     801ns  1.9340us  cudaConfigureCall
  0.00%  7.3530us        12     612ns     441ns     844ns  cuDeviceGet
  0.00%  6.8210us         8     852ns     689ns  1.0040us  cudaGetLastError
  0.00%  6.4790us         3  2.1590us     681ns  4.9340us  cuDeviceGetCount
  0.00%  4.4320us         1  4.4320us  4.4320us  4.4320us  cudaGetDeviceCount

==73532== Dependency Analysis:
Critical path(%)  Critical path  Waiting time  Name
          83.03%   590.169908ms           0ns  cudaStreamCreate
           4.14%    29.393820ms           0ns  <Other>
           3.38%    24.010192ms           0ns
MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
           1.84%    13.088064ms           0ns  cudaMalloc
           1.63%    11.595084ms           0ns  cudaGetDeviceProperties
           1.08%     7.691710ms           0ns  cudaMallocHost
           1.08%     7.660636ms           0ns  cudaHostAlloc
           0.89%     6.296326ms           0ns  rngSetupStates(curandStateXORWOW*,
int)
           0.78%     5.539428ms           0ns  cudaFreeHost
           0.70%     4.998013ms           0ns  cudaLaunch
           0.60%     4.259809ms           0ns  cuDeviceTotalMem_v2
           0.43%     3.071325ms           0ns  cuDeviceGetAttribute
           0.34%     2.447406ms           0ns  cudaFree
           0.04%   254.177000us           0ns  cuDeviceGetName
           0.02%   108.747000us           0ns  cudaMemcpyAsync
           0.01%    58.902000us           0ns  cudaSetDevice
           0.01%    52.310000us           0ns  cudaStreamDestroy_v5050
           0.00%    19.329000us           0ns  cudaEventRecord
           0.00%    18.341000us           0ns  cudaEventCreate
           0.00%    17.468000us           0ns  cudaSetupArgument
           0.00%    15.729000us           0ns  cudaEventDestroy
           0.00%     9.600000us           0ns  [CUDA memcpy HtoD]
           0.00%     7.747000us           0ns  cudaConfigureCall
           0.00%     7.353000us           0ns  cuDeviceGet
           0.00%     6.479000us           0ns  cuDeviceGetCount
           0.00%     5.051000us           0ns  cudaGetLastError
           0.00%     4.432000us           0ns  cudaGetDeviceCount
           0.00%     3.680000us           0ns  [CUDA memcpy DtoH]
           0.00%            0ns   23.940847ms  cudaEventSynchronize
           0.00%            0ns    6.228596ms  cudaDeviceSynchronize
```

Example 9-23 shows application information by using the **nvprof** flag **--cpu-profiling**, which enables capture operations that are run in the CPU.

Example 9-23 Collecting information about operations that run in the CPU

```
$ nvprof --log-file profile.out --cpu-profiling on --cpu-profiling-thread-mode
separated ./MonteCarloMultiGPU 2&> /dev/null
$ cat profile.out

======== CPU profiling result (bottom up):
Time(%)      Time  Name
======== Thread 17592186332928
 65.66%     650ms  cuDevicePrimaryCtxRetain
 65.66%     650ms  |
cudart::contextStateManager::initPrimaryContext(cudart::device*)
 65.66%     650ms  |   cudart::contextStateManager::initDriverContext(void)
 65.66%     650ms  |
cudart::contextStateManager::getRuntimeContextState(cudart::contextState**, bool)
 65.66%     650ms  |       cudart::doLazyInitContextState(void)
 65.66%     650ms  |         cudart::cudaApiStreamCreate(CUstream_st**)
 65.66%     650ms  |           cudaStreamCreate
 65.66%     650ms  |             _ZL11multiSolverP11TOptionPlani
 17.17%     170ms  cuInit
 17.17%     170ms  | cudart::__loadDriverInternalUtil(void)
 17.17%     170ms  |   __pthread_once
 17.17%     170ms  |     cudart::cuosOnce(int*, void (*) (void))
 17.17%     170ms  |       cudart::globalState::initializeDriver(void)
 17.17%     170ms  |         cudaGetDeviceCount
 17.17%     170ms  |           main
  3.03%      30ms  ???
  3.03%      30ms  | getcontext@@GLIBC_2.17
  2.02%      20ms  cuMemFreeHost
  2.02%      20ms  | cudart::driverHelper::freeHost(void*)
  2.02%      20ms  |   cudart::cudaApiFreeHost(void*)
  2.02%      20ms  |     cudaFreeHost
  2.02%      20ms  |       closeMonteCarloGPU
  2.02%      20ms  cuLaunchKernel
  2.02%      20ms  | _ZN6cudartL19cudaApiLaunchCommonEPKvb
  2.02%      20ms  |   cudaLaunch
  2.02%      20ms  |     cudaError cudaLaunch<char>(char*)
  2.02%      20ms  cuDeviceGetAttribute
  2.02%      20ms  | cudart::device::updateDeviceProperties(void)
  2.02%      20ms  |   cudart::cudaApiGetDeviceProperties(cudaDeviceProp*, int)
  2.02%      20ms  |     cudaGetDeviceProperties
  1.01%      10ms  |       _ZL11multiSolverP11TOptionPlani
  1.01%      10ms  |       adjustGridSize(int, int)
  1.01%      10ms  random
  1.01%      10ms  | rand
  1.01%      10ms  |   randFloat(float, float)
  1.01%      10ms  |     main
  1.01%      10ms  |       generic_start_main.isra.0
  1.01%      10ms  cuMemHostAlloc
  1.01%      10ms  | cudart::driverHelper::mallocHost(unsigned long, void**,
unsigned int)
  1.01%      10ms  |   cudart::cudaApiHostAlloc(void**, unsigned long, unsigned
int)
```

```
  1.01%      10ms  |     cudaHostAlloc
  1.01%      10ms  |       _ZL14cudaMallocHostPPvmj
  1.01%      10ms  __exp_finite
  1.01%      10ms  | exp
  1.01%      10ms  |   BlackScholesCall
  1.01%      10ms  |     main
  1.01%      10ms  |       generic_start_main.isra.0
  1.01%      10ms  cuDeviceTotalMem_v2
  1.01%      10ms  | cudart::deviceMgr::enumerateDevices(void)
  1.01%      10ms  |   cudart::globalState::initializeDriverInternal(void)
  1.01%      10ms  |     cudart::globalState::initializeDriver(void)
  1.01%      10ms  |       cudaGetDeviceCount
  1.01%      10ms  |         main
  1.01%      10ms  ???
  1.01%      10ms  | cudart::contextState::loadCubin(bool*, void**)
  1.01%      10ms  | |
cudart::globalModule::loadIntoContext(cudart::contextState*)
  1.01%      10ms  | |   cudart::contextState::applyChanges(void)
  1.01%      10ms  | |
cudart::contextStateManager::getRuntimeContextState(cudart::contextState**, bool)
  1.01%      10ms  | |       cudart::doLazyInitContextState(void)
  1.01%      10ms  | |         cudart::cudaApiStreamCreate(CUstream_st**)
  1.01%      10ms  | |           cudaStreamCreate
  1.01%      10ms  | |             _ZL11multiSolverP11TOptionPlani
  1.01%      10ms  cuMemAlloc_v2
  1.01%      10ms  | cudart::driverHelper::mallocPtr(unsigned long, void**)
  1.01%      10ms  |   cudart::cudaApiMalloc(void**, unsigned long)
  1.01%      10ms  |     cudaMalloc
  1.01%      10ms  |       initMonteCarloGPU
  1.01%      10ms  __log_finite
  1.01%      10ms  | log
  1.01%      10ms  |   BlackScholesCall
  1.01%      10ms  |     main
  1.01%      10ms  |       generic_start_main.isra.0
  1.01%      10ms  cuEventSynchronize
  1.01%      10ms    cudart::cudaApiEventSynchronize(CUevent_st*)
  1.01%      10ms      cudaEventSynchronize
  1.01%      10ms        _ZL11multiSolverP11TOptionPlani
======== Thread 21990246248880
 86.87%     860ms  ???
 86.87%     860ms  | start_thread
 86.87%     860ms  | | clone
======== Thread 23089793855920
 68.69%     680ms  ???
 68.69%     680ms  | start_thread
 68.69%     680ms  | | clone
======== Thread 24189306532272
 52.53%     520ms  ???
 52.53%     520ms  | start_thread
 52.53%     520ms  | | clone
======== Thread 25288817111472
 36.36%     360ms  ???
 36.36%     360ms  | start_thread
 36.36%     360ms  | | clone
======== Thread 17592299352496
```

```
 90.91%     900ms  ???
 90.91%     900ms  | start_thread
 90.91%     900ms  | | clone

======== Data collected at 100 Hz frequency
```

Some tools, such as **oprofile** and **perf**, can obtain data out of the PMU in the CPU. Also, **nvprof** can profile hardware events that are triggered by the GPU and calculate metrics. Use the **--events** or **--metrics** flags to pass a list of events or metrics to the profiler.

Although some available metrics calculate aspects of the GPU efficiency, others can reveal bottlenecks in specific units (for example, the case of metrics from events involving NVLink communication). Example 9-24 shows the receive (**nvlink_receive_throughput**) and transmit (**nvlink_transmit_throughput**) throughput metrics for each GPU in the system.

Example 9-24 NVIDIA nvprof: Report metrics for NVLink

```
$ nvprof --log-file profile.out --metrics
nvlink_transmit_throughput,nvlink_receive_throughput  ./MonteCarloMultiGPU 2&>
/dev/null
$ cat profile.out
==73977== NVPROF is profiling process 73977, command: ./MonteCarloMultiGPU 2
==73977== Profiling application: ./MonteCarloMultiGPU 2
==73977== Profiling result:
==73977== Metric result:
Invocations                               Metric Name
Metric Description         Min         Max         Avg
Device "Tesla P100-SXM2-16GB (0)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1               nvlink_transmit_throughput               NVLink
Transmit Throughput  1.1570MB/s  1.1570MB/s  976.56KB/s
          1                nvlink_receive_throughput                NVLink
Receive Throughput  681.24KB/s  681.24KB/s  0.00000B/s
   Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1               nvlink_transmit_throughput               NVLink
Transmit Throughput  567.64KB/s  567.64KB/s  0.00000B/s
          1                nvlink_receive_throughput                NVLink
Receive Throughput  285.74KB/s  285.74KB/s  0.00000B/s
Device "Tesla P100-SXM2-16GB (1)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1               nvlink_transmit_throughput               NVLink
Transmit Throughput  177.71KB/s  177.71KB/s  0.00000B/s
          1                nvlink_receive_throughput                NVLink
Receive Throughput  244.35KB/s  244.35KB/s  0.00000B/s
   Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1               nvlink_transmit_throughput               NVLink
Transmit Throughput  308.39KB/s  308.39KB/s  0.00000B/s
          1                nvlink_receive_throughput                NVLink
Receive Throughput  190.42KB/s  190.42KB/s  0.00000B/s
Device "Tesla P100-SXM2-16GB (2)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1               nvlink_transmit_throughput               NVLink
Transmit Throughput  177.16KB/s  177.16KB/s  0.00000B/s
```

```
          1                 nvlink_receive_throughput                NVLink
Receive Throughput  231.30KB/s  231.30KB/s  0.00000B/s
    Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  568.43KB/s  568.43KB/s  0.00000B/s
          1                 nvlink_receive_throughput                NVLink
Receive Throughput  284.86KB/s  284.86KB/s  0.00000B/s
Device "Tesla P100-SXM2-16GB (3)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  1.1607MB/s  1.1607MB/s  976.56KB/s
          1                 nvlink_receive_throughput                NVLink
Receive Throughput  703.26KB/s  703.26KB/s  0.00000B/s
    Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  309.12KB/s  309.12KB/s  0.00000B/s
          1                 nvlink_receive_throughput                NVLink
Receive Throughput  186.25KB/s  186.25KB/s  0.00000B/s
```

Use the **`--query-metrics`** or **`--query-events`** flags to list all of the metrics and events that are available for profiling.

Another useful function of **`nvprof`** is to generate traces of the application run time. Those traces can be then imported in to the NVIDIA Visual Profiler tool. For more information, see the NVIDIA Visual Profiler website.

For more information about the **`nvprof`** tool, see the Profiler User's Guide.

A

Additional material

This book refers to additional material that can be downloaded from the internet as described in the following sections.

Locating the web material

The web material that is associated with this book is available in softcopy on the internet from the IBM Redbooks web server:

`ftp://www.redbooks.ibm.com/redbooks/SG248422`

Alternatively, you can go to the IBM Redbooks website:

ibm.com/redbooks

Search for SG248422, select the title, and then click **Additional materials** to open the directory that corresponds with the IBM Redbooks form number, SG248422.

Using the web material

The additional web material that accompanies this book includes the following files:

File name	*Description*
smn_md.cfg.zip	Kickstart file

System requirements for downloading the web material

The web material requires the following system configuration:

Hard disk space:	100 MB minimum
Operating System:	Windows, Linux, or macOS
Processor:	i3 minimum
Memory:	1024 MB minimum

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material compressed file into this folder.

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publication provides more information about the topics in this document.

- *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263

You can search for, view, download, or order these documents and other Redbooks, Redpapers, web docs, drafts, and additional materials at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Extreme Cluster and Cloud Administration Toolkit (xCAT)

  http://xcat.org
- Mellanox Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)

  http://www.mellanox.com/page/products_dyn?product_family=261&mtag=sharp
- NVIDIA Compute Unified Device Architecture (CUDA) Toolkit

  http://developer.nvidia.com/cuda-downloads
- SUMMIT

  http://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
- Tesla graphics processing units (GPUs) NVIDIA drivers

  http://www.nvidia.com/drivers

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

IBM High-Performance Computing Insights with IBM Power

SG24-8422-00

ISBN 0738457450

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8422-00

ISBN 0738457450

Printed in U.S.A.