APPLICATION USER GUIDE
SCIENTIFIC COLLABORATION PLATFORM
VERSION 17.10-1

Nicolas TALLET
[nicolas.tallet@fr.ibm.com]

1. General Information

Ouessant constitutes the technical platform for the Scientific Collaboration around OpenPOWER technologies established between GENCI on one side, and IBM, Mellanox and NVIDIA on the other side.

The platform is based on IBM Power System S822LC compute nodes, with the following specifications:

- 12 IBM Power System S822LC « Minsky » compute nodes.
- 20 physical cores, and up to 160 logical cores (hardware threads) per node.
- 4 NVIDIA Tesla P100 GPU per node.
- Peak computing performance:
    - 470 GFlops for CPU subsystems.
    - 21 TFlops for GPU subsystems.
- 128 GB of memory per node, i.e. 6 GB / physical core.
- Dedicated IBM Spectrum Scale (GPFS) filesystem with 128 TB shared storage capacity.

The applications to be executed on Ouessant must be able to take advantage of the GPU accelerators in order to fully benefit from the OpenPOWER architecture.

Several programming models are available to achieve GPU offloading:

- OpenMP directives.
- OpenACC directives.
- CUDA API.
- Existing GPU-enabled runtimes: StarPU, Kokkos..

These various acceleration models are being studied as part of the the Scientific Collaboration activities.

2. QUICKSTART

ONE PAGE REFERENCE FOR IMPATIENT USERS ☺

2.1. MAIN SOFTWARE MODULES

Category	Software Package	Module
Compilers	CUDA Toolkit	cuda
	GCC	gcc
	Advance Toolchain	at
	IBM XL C/C++	xlc
	IBM XL Fortran	xlf
	LLVM Clang	llvm/clang
	LLVM XLFlang	llvm/xlflang
	PGI Accelerator	pgi
MPI Library	IBM Spectrum MPI	smpi
Scientific Libraries	IBM ESSL	essl
	IBM PESSL	pessl
Performance Analysis	IBM Parallel Performance Toolkit for POWER	ppedev

2.2. COMPILATION ENVIRONMENT SELECTION

- IBM XL C/C++ & IBM XL Fortran:

```
$ ml xlc xlf
$ which xlc xlf
/opt/ibm/xlC/13.1.5/bin/xlc
/opt/ibm/xlf/15.1.5/bin/xlf
```

- PGI Accelerator C/C++/Fortran:

```
$ ml pgi
$ which pgcc pgfortran
/opt/pgi/linuxpower/16.10/bin/pgcc
/opt/pgi/linuxpower/16.10/bin/pgfortran
```

- IBM XL C/C++ & IBM XL Fortran + IBM Spectrum MPI:

```
$ ml xlc xlf smpi
$ mpicc -show
xlc_r -I/opt/ibm/spectrum_mpi/include -pthread -L/opt/ibm/spectrum_mpi/lib -lmpi_ibm
$ mpif90 -show
xlf90_r -I/opt/ibm/spectrum_mpi/include -qthreaded -I/opt/ibm/spectrum_mpi/lib/XL -
L/opt/ibm/spectrum_mpi/lib -lmpiprofilesupport -lmpi_usempi -lmpi_mpifh -lmpi_ibm
```

- PGI Accelerator C/C++/Fortran + IBM Spectrum MPI

```
$ ml pgi smpi
$ mpicc -show
pgcc -I/opt/ibm/spectrum_mpi/include -lpthread -L/opt/ibm/spectrum_mpi/lib -lmpi_ibm
$ mpif90 -show
pgf90 -I/opt/ibm/spectrum_mpi/include -lpthread -I/opt/ibm/spectrum_mpi/lib/PGI -
L/opt/ibm/spectrum_mpi/lib -lmpiprofilesupport -lmpi_usempi -lmpi_mpifh -lmpi_ibm
```

2.3. JOB SUBMISSION

- Interactive Jobs:
    - Choose Target Queue:

Queue Name	Target Nodes	Purpose
interactive-firestone	ouessantf03	Interactive queue on Firestone node
interactive-minsky	ouessantm01	Interactive queue on Minsky node

    - Open Session in Interactive Queue:

```
$ bsub -Ip -n 10 -q interactive-minsky /bin/bash
Job <11858> is submitted to queue <interactive-minsky>.
<<Waiting for dispatch ...>>
<<Starting on ouessantm01>>
$ hostname
ouessantm01
```

Note#1: It is required to specify the requested number of slots if a parallel execution is expected to take place inside the interactive session.

Note #2: When using IBM Spectrum MPI inside an interactive session, it is necessary to specify the following option to the 'mpirun' / 'mpiexec' command: '-pami_noib'.

- Non-Interactive Jobs:
    - Build Job Submission File:

```
#!/bin/bash

#BSUB –a p8aff(1,1,1,balance)
#BSUB -cwd /pwrhome/login/myjob
#BSUB -e myjob.%J.log
#BSUB -gpu "num=4:mode=exclusive_process:mps=no:j_exclusive=yes"
#BSUB -J MyJob
#BSUB -n 8
#BSUB –o myjob.%J.log
#BSUB -R "span[ptile=4]"
#BSUB -W 01:00
#BSUB -x

# Uncomment To Start NVIDIA MPS If Required
##BSUB -env “all,LSB_START_JOB_MPS=Y”

ml xlc xlf smpi

mpirun -display-allocation -prot -report-bindings /pwrlocal/ibmccmpl/bin/task_prolog.sh -devices auto
/pwrhome/login/myjob/myjob.bin
```

    - Submit Job:

```
$ bsub < job.sh
```

The following table summarizes the Spectrum LSF specific settings for several standard execution configurations :

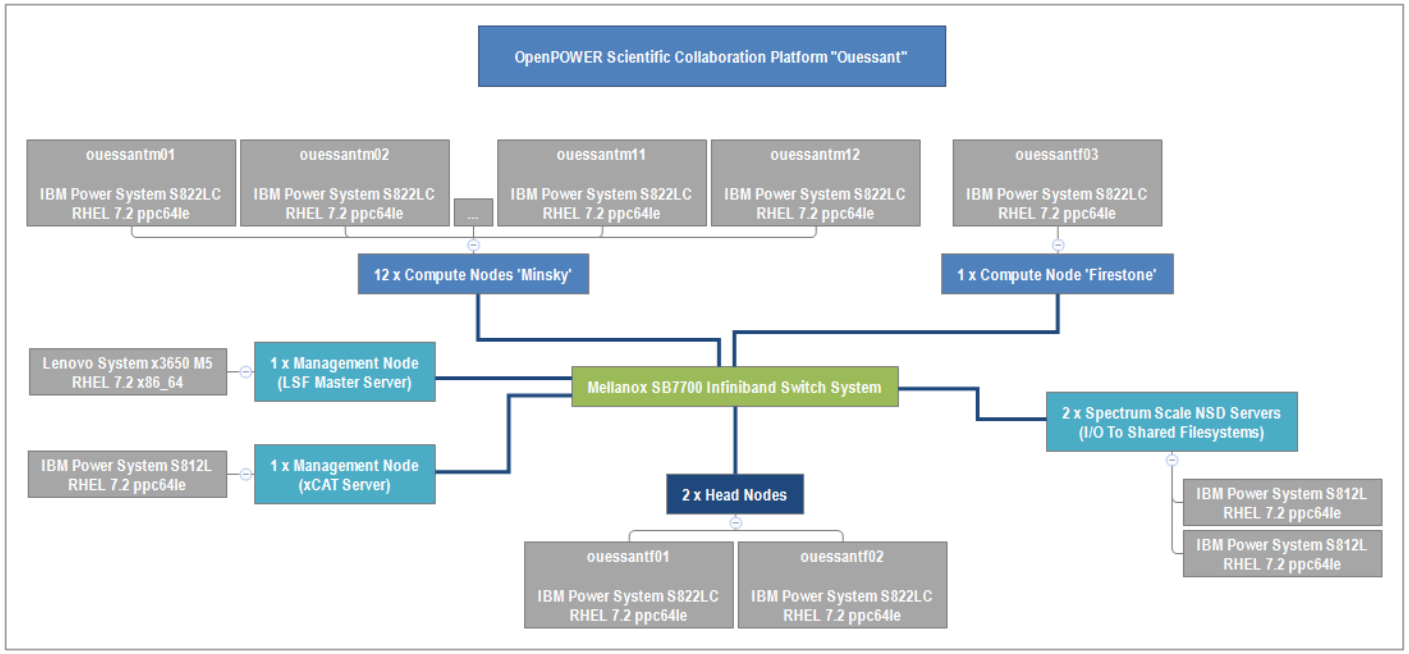
Execution Configuration	Spectrum LSF Directives
2 MPI Tasks x 10 Threads Per Task	#BSUB -a p8aff(10,4,1,balance) #BSUB -n 2 #BSUB -R "span[ptile=2]"
4 MPI Tasks x 5 Threads Per Task	#BSUB -a p8aff(5,4,1,balance) #BSUB -n 4 #BSUB -R "span[ptile=4]"
10 MPI Tasks x 2 Threads Per Task	#BSUB -a p8aff(2,4,1,balance) #BSUB -n 10 #BSUB -R "span[ptile=10]"
20 MPI Tasks x 1 Thread Per Task	#BSUB -a p8aff(1,4,1,balance) #BSUB -n 20 #BSUB -R "span[ptile=20]"
20 MPI Tasks x 2 Threads Per Task	#BSUB -a p8aff(2,4,2,balance) #BSUB -n 20 #BSUB -R "span[ptile=20]"
20 MPI Tasks x 4 Threads Per Task	#BSUB -a p8aff(4,8,4,balance) #BSUB -n 20 #BSUB -R "span[ptile=20]"

3. Computation Environment

3.1. Hardware Configuration

3.1.1. Architecture

The platform is made of the following hardware components:



The systems constituting the platform have the following role in this environment:

System	Type	Role
ouessantf[01-02]	IBM Power System S822LC 8335-GTA ("Firestone")	Login Node
ouessantf03	IBM Power System S822LC 8335-GTA ("Firestone")	Interactive Compute Node
ouessantm01	IBM Power System S822LC 8335-GTB ("Minsky")	Interactive Compute Node
ouessantm[02-12]	IBM Power System S822LC 8335-GTB ("Minsky")	Compute Node

3.1.2. IBM POWER SYSTEM S822LC "FIRESTONE" & "MINSKY" COMPUTE NODES

The IBM Power System S822LC compute nodes have the following technical characteristics:

	8335-GTA Firestone	8335-GTB Minsky
# CPUs	2	2
Model	POWER8	POWER8
Physical Cores	10	10
Logical Cores / Physical Threads	10-80	10-80
Clock Frequency: Nominal	2.93 GHz	2.86 GHz
Clock Frequency: Max. (Turbo)	3.49 GHz	4.03 GHz
Memory	128 GB	128 GB
Memory Controllers	2	2
Memory Channels	4	4
L3 Cache	8 MB	8 MB
L4 Cache	16 MB / Buffer Chip	16 MB / Buffer Chip
Memory Bandwidth	230 GB/s	230 GB/s
Peak Performance (Double Precision)	235 GFlops	235 GFlops
# GPUs	2	4
Model	Tesla K80	Tesla P100
CUDA Cores	4992	3584
Memory	24 GB	16 GB
Peak Performance (Double Precision)	1.87 Tflops	5.3 TFlops

Simultaneous Multi-Threading (SMT)

The S822LC compute nodes can be configured in 4 distinct SMT modes. Each SMT mode defines the number of logical cores (hardware threads) available on the node:

SMT Mode	# Logical Cores (# Physical Threads)
1 (Off)	20
2	40
4	80
8	160

SMT mode configuration does not require the system to be restarted; therefore, SMT mode configuration can be performed on-the-fly.

3.1.3. EDR INFINIBAND INTERCONNECTION

High performance interconnection between the compute nodes is achieved through an EDR Infiniband network (100 Gb/s).

3.2. REMOTE ACCESS

A SSH session can be opened on the platform through the following hostname:

Hostname
ouessant.idris.fr

This hostname is redirected by default to the ouessantf01 login node. Redirection automatically switches to a different login node in case this one is temporarily unavailable.

3.3. FILESYSTEMS

The following filesystems are available on the platform:

Path	Purpose	Access Rights	User Group Quota
/pwrhome/<userid> /linkhome/<userid>	Home Directories	Read + Write	10 GB 150K inodes
/pwrlocal	Shared Software	Read	-
/pwrwork/<category>/<groupid>	Group Workspace	Read + Write	1 TB 300000 inodes

The current level of space usage with respect to the predefined quota can be displayed through the following commands:

- For Home Directories:

```
$ quota_u
```

- For Shared Workspace:

```
$ quota_u -w
```

The output of these commands looks like:

```
********************************************************************************
*    Quotas du groupe usergroup sur le HOME de la machine  Ouessant            *
*    Frequence MAJ : Toutes les 20 mns - Derniere mise a jour : Jan 13 12:12   *
********************************************************************************
*  Type                  Consomme          Quota          %                    *
*  ----                  --------          -----          -                    *
*  Espace(Mio)               7660          10240      74.81%                   *
*  Inodes(fic.+rep.)        79957         150000      53.30%                   *
********************************************************************************
*                                                                              *
*  Ratio(Mio/inode) :    0.10                                                  *
*                                                                              *
********************************************************************************
*             Occupation  des   membres  de   votre  groupe                    *
********************************************************************************
                                       Espace       Occupe       Nb.      Inodes
 Login                                  (Mio)        (%)       inodes      (%)

userid1 LASTNAME_Firstname              6697        87.44%      36740     45.95%
userid2 LASTNAME_Firstname               949        12.40%      42676     53.37%
```

3.4. SOFTWARE STACK

The following software packages are available on the platform:

Category	Software Package	Version
Compilers	CUDA Toolkit	8.0
	GCC [Standard]	4.8
	GCC [Advance Toolchain 9.0]	5.3
	GCC [Advance Toolchain 10.0]	6.3
	GCC [Advance Toolchain 11.0]	7.2
	IBM XL C/C++	13.1.4 13.1.5 13.1.6 (Beta)
	IBM XL Fortran	15.1.4 15.1.5 15.1.6 (Beta)
	LLVM Clang	Beta
	LLVM XLFlang	Beta
	PGI Accelerator	17.4 17.5 17.7
MPI Library	IBM Spectrum MPI	10.1
Scientific Libraries	IBM ESSL	5.5
	IBM PESSL	5.3
Performance Analysis	IBM Parallel Performance Toolkit for POWER	2.3

Additional scientific libraries have be made available upon users request: FFTW, HDF5, LAPACK, ScaLAPACK…

Note : IBM XL C/C++ is fully compatible with the C++ 11 standard.

Note : IBM Spectrum MPI (based on Open MPI 2.0) has become the default choice for MPI library starting from Q4 2016. IBM Parallel Environment has therefore been discontinued.

3.5. LMOD ENVIRONNEMENT MODULES

3.5.1. PRINCIPLE

Lmod provides a simple and reliable way to dynamically load / unload the various software components available on the platform.

Lmod takes care of performing the user environment dynamic configuration by automatically altering the associated environment variables (LD_LIBRARY_PATH, MANPATH, PATH…).

3.5.2. USAGE

Lmod usage is based on the following command:

Command	Action
ml avail module avail	Display available modules
ml whatis <module> module whatis <module>	Display module information
ml list module list	Display currently-loaded modules
ml <module> module load <module>	Load specified module
ml -<module> ml unload <module> module unload <module>	Unload specified module
ml switch <mod1> <mod2> module switch <mod1> <mod2>	Substitue mod1 by mod2
ml purge module purge	Unload all currently-loaded modules (fully purge user environment)

3.5.3. MODULES HIERARCHY

Modules are spread into a three-level hierarchy:

Level	Purpose
level0	Compilers, Basic Tools
level1	Software depending on a given compiler Example : Serial libraries
level2	Software depending on a given compiler AND a given MPI library Example : Parallel libraries

The module hierarchy sets the dependencies existing between the software packages, and defines the sequence for loading modules by the user. More precisely :

- Only level-0 modules are available to the user at first.
- Level-1 modules become available once the user has loaded a compiler module.
- Level-2 modules become available once the user has loaded a MPI library module.

4. COMPILATION

4.1. COMPILERS

The invocation commands of the compilers are the following:

Compiler	C	C++	Fortran
GCC	gcc	g++	gfortran
IBM XL	xlc_r	xlC_r	xlf_r xlf90_r xlf95_r xlf2003_r xlf2008_r
LLVM	clang	clang++	xlflang
PGI	pgcc	pgCC	pgf90

Note : The commands specified above constitute the « thread-safe » variants of the IBM XL compilers. It is recommended to exclusively use these variants, even in the case of non-multithreaded applications.

The table below provides the main equivalences between the options of the different compilers available on the platform:

Purpose	GCC	IBM XL	LLVM	PGI
Architecture	-mcpu=power8	-qarch=pwr8	-target powerpc64le-unknown-linux-gnu -mcpu=pwr8	-m64
Disabled Optimization	-O0	-O0 -qnoopt	-O0	-O0
Optimization Levels	-O / -O1 -O2 -O3 -Ofast	-O -O2 -O3 -O4 -O5	-O0 -O2 -O3 -Os	-O1 -O -O2 -O3 -O4 -fast
Recommended Optimization	-O3 -mcpu=power8 -funroll-loops	-O3 [-qipa] [-qhot]		-fast -Mipa=fast,inline [-Msmartalloc]
Profile-Guided Optimization (PGO)	-fprofile-generate -fprofile-use	-qpdf1 -qpdf2	-fprofile-instr-generate -fprofile-instr-use	-Mpfi -Mpfo
Inter-Procedural Optimization	-flto	-qipa		-Mipa
Vectorization	-ftree-vectorize	-qsimd=auto		-Mvect
OpenMP Support	-fopenmp	-qsmp=omp	-fopenmp	-mp
OpenMP/OpenACC Target		-qoffload -qtgtarch=sm_60	-fopenmptargets=nvptx64-nvidia-cuda	-acc -Minfo=accel -ta:tesla:cc60
Automatic Parallelization	-floop-parallelize-all	-qsmp=auto		-Mconcur
Loop Optimization	-fpeel-loops -funroll-loops	-qhot	-funroll-loops	
Debugging Symbols	-g	-g	-g	-g

4.1.1. IBM XL

The IBM XL compilers allow the following optimization levels:

Options	Purpose
-O0 -qnohot -qnosmp -qstrict	Disable any optimization
-O2	Limited optimization (safe)
-O3 -qstrict	Medium optimization without modification of the code semantics
-O3	Medium optimization with potential modifications of the code semantics
-O4	Agressive optimization + Light Inter-Procedural Analysis (IPA)
-O5	Equivalent to -O4 level with deeper IPA

Note : Increasing the optimization level tends to lead to a potentially significant increase in compilation time. It is therefore recommended to keep limited optimization levels while porting an application onto the platform.

For a given optimization level, the IBM XL compilers automatically introduce additional options by default. These options can be in turn complemented with other additional options.

The table below details the main possible combinations:

Optimization Level	Additional Options Introduced By Default	Additional Options	Other Options with Potential Benefits
-O0	-qsimd=auto	-qarch=pwr8	
-O2	-qmaxmem=8192 -qsimd=auto -qstrict=all	-qarch=pwr8 -qtune=pwr8	-qmaxmem=-1 -qhot=level=0
-O3	-qhot=noarraypad:level=0 :novector:fastmath -qnostrict -qmaxmem=-1 -qsimd=auto	-qarch=pwr8 -qtune=pwr8	
-O4	-qarch=auto -qcache=auto -qhot=noarraypad:level=1 :vector:fastmath -qipa -qmaxmem=-1 -qnostrict -qsimd=auto -qtune=auto	-qarch=pwr8 -qcache -qtune=pwr8	-qsmp=auto
-O5	[Idem -O4] -qipa=level=2	-qarch=pwr8 -qcache -qtune=pwr8	-qsmp=auto

4.1.2. PGI ACCELERATOR

It is highly recommended to specify the GPU target explicitely, through the following options:

Component	Option
Compute Capability	-ta={ cc35 | cc60 }
CUDA	-ta=cuda8.0

4.2. MPI LIBRARY

4.2.1. MPI WRAPPERS

The invocation commands of the MPI wrappers are the following:

C	C++	Fortran
mpicc	mpicxx	mpifort

Note : The MPI wrappers have strictly the same name independently from the compiler used. The selection of a given compiler is performed through the definition of the following environment variables : OMPI_CC / OMPI_CXX / OMPI_F77 / OMPI_F90 / OMPI_FC. When using environment modules, this definition is performed automatically in the MPI library module.

Note : The '-show' option allows to check which is the compiler used when calling the MPI wrapper. This option is therefore very useful to confirm that the expected compiler is the one actually invoked.

Example : Checking the Compiler called by the MPI Wrapper

```
[login@ouessant ~]$ mpicc -show
xlc_r -I/opt/ibm/spectrum_mpi/include -pthread -L/opt/ibm/spectrum_mpi/lib -lmpi_ibm
```

4.2.2. CUDA-AWARENESS

The IBM Spectrum MPI library is actually CUDA-aware, as indicated in the official product documentation.

However:

- GPU support is disabled by default, and must be explicitely enabled through the '-gpu' option.
- GPU support has a certain number of limitations at this time. The official product documentation states:
    - Support for GPU-accelerated applications is provided only if you are using the IBM Spectrum MPI PAMI backend and the IBM collective library (libcollectives). These are the default options for IBM Spectrum MPI, but if you choose to use a different messaging protocol or collective component, note that it will not support GPUs.
    - The libcollectives component is the only collective component that is able to support CUDA buffers. As a result, when -gpu is specified with the mpirun command, libcollectives must be the preferred collective component. Therefore, you cannot specify mpirun -gpu with any of the following options:
        - -mxm/-MXM
        - -mxmc/-MXMC
        - -tcp/-TCP
        - -ibv/-IBV
        - -ib/-IB
        - -openib/-OPENIB
        - -fca/-FCA
        - -hcoll/-HCOLL
    - The following MPI functions are not CUDA-aware:
        - MPI_Alltoallw
        - MPI_Ialltoallw
        - MPI_Ineighbor_allgather
        - MPI_Ineighbor_allgatherv
        - MPI_Ineighbor_alltoall
        - MPI_Ineighbor_alltoallv
        - MPI_Ineighbor_alltoallw
        - MPI_Neighbor_allgather
        - MPI_Neighbor_allgatherv

        - MPI_Neighbor_alltoall
        - MPI_Neighbor_alltoallv
        - MPI_Neighbor_alltoallw
    - IBM Spectrum MPI behavior is undefined when an MPI application passes a GPU device address to an MPI API that is not CUDA aware; the application might trigger a segmentation fault and dump core, or it might give incorrect results.
    - Most MPI APIs are CUDA-aware, however, IBM Spectrum MPI does not supply header files or Fortran module files that identify which MPI APIs accept GPU device addresses as arguments. Refer to the preceding restriction for a list of MPI APIs that may not be used with GPU device addresses.

5. JOB SUBMISSION

The Workload Scheduler is responsible for:

- Allocating computing resources to user jobs dynamically.
- Executing the user jobs on the allocated computing resources.

5.1. IBM SPECTRUM LSF COMMANDS

The main commands to interact with the workload scheduler are the following:

Command	Argument	Purpose
bjobs		Display active user jobs (waiting or executing)
	-u { <User ID> | all }	Specify user
bkill	<Job ID>	
bpeek	<Job ID>	Display user job stdout/stderr
	-f	Refresh display in realtime
bqueues		Display submission queues
bstatus	<Job ID>	Display user job status
bsub	-app <Application Profile>	Call a specific application profile
	< <Submission Script>	Submit user job into a submission queue

Note : One specificity of BM Spectrum LSF is related to the buffering of the stdout / stderr streams during job execution, before final writing into the location specified by the user. Therefore, during execution, target files remain empty, and the following command allows visualizing the progress of the execution : 'bpeek -f'.

5.2. SUBMISSION SCRIPT

A submission script allows submitting user jobs through the workload scheduler.

A submission script is made of two distinct sections:

- Section #1: Directives
  The header consists in a set of IBM Spectrum LSF directives.
  These directives are introduced by the following prefix: '#BSUB'.
  These directives are specified through the following syntax : <Option> <Value>.
- Section #2: Shell Script
  The script in itself is made of standard Bash commands.
  For parallel executions, the script involves a call to the parallel submission command - which depends on the MPI library used.

The main IBM Spectrum LSF directive keywords are the following:

Option	Value	Purpose
-cwd	<Path>	Execution directory
-e	<File>	stderr file
-J	<Job Name>	User job name
-n	<# MPI Tasks>	Total number of MPI tasks
-o	<File>	stdout file
-q	<Queue>	Target queue
-R	"span[ptile=<ppn>]"	Number of MPI tasks per node
-W	HH:MM	Runlimit
-x		Exclusive job

Note: IBM Spectrum LSF uses an automatically-created directory inside the user Home Directory in order to temporarily store the elements related to a submitted job. The location of this directory is the following:

${HOME}/.lsbatch

5.3. ENVIRONMENT VARIABLES

When a user job starts, IBM Spectrum LSF initializes a set of environment variables. These variables can be used in the submission script if necessary.

The main environment variables are the following:

Variable	Value	Origin
LS_SUBCWD	Current Working Directory	LSF directive
LSB_DJOB_HOSTFILE	Absolute path to hostfile	Defined by sbatchd
LSB_DJOB_NUMPROC	Number of allocated slots	Defined by sbatchd
LSB_DJOB_RANKFILE	Absolute path to rankfile	Defined by sbatchd
LSB_ERRORFILE	stderr file	LSF directive
LSB_EXECHOSTS	List of allocated nodes	Defined by sbatchd
LSB_JOBFILENAME	Submission script	Defined by sbatchd
LSB_JOBID	User job ID	Defined by sbatchd
LSB_HOSTS	List of allocated nodes	Defined by sbatchd
LSB_JOBNAME	User job name	LSF directive
LSB_MCPU_HOSTS	List of allocated nodes including number of slots	Defined by sbatchd
LSB_OUTPUTFILE	stdout file	LSF directive
LSB_USER_BIND_CPU_LIST	CPU list for processor affinity	Defined by sbatchd

5.4. QUEUES

The queues constitute the waiting lists that are available to receive the user job submissions.

Each queue has a set of properties which defines, in particular,

- The target compute nodes.
- The maximum number of nodes that can be allocated.
- The maximum number of tasks that can be run on a node.
- The maximum runlimit.

5.5. EXECUTION CONFIGURATION

Execution configuration encompasses several different aspects related to the job execution. It is therefore essential to distinguish the following notions:

- Task placement defines the mechanism for distributing each MPI task onto a given compute node.
- Processor affinity defines the mechanism for pinning each MPI task (and, potentially, its associated threads) onto one or several specific CPU cores inside a given compute node.
- GPU affinity defines the mechanism for assigning to each MPI task one or several GPU devices inside a given compute node.

5.5.1. TASK PLACEMENT

By default, task placement is performed according to the "group round-robin" policy: each compute node gets as many MPI tasks as specified as number of tasks per node, one compute node after the other.

> Note: The actual task placement can be checked through reading the rankfile of the job. The absolute path of this rankfile is provided by the LSB_DJOB_RANKFILE environment variable.

An alternate task placement can be requested by using the following environment variable: LSB_TASK_GEOMETRY.

This environment variable allows the user to specify the group of MPI tasks that will coexist on the same compute node.

Example : Specifying an Alternate Task Placement through LSB_TASK_GEOMETRY Environment Variable

```
export LSB_TASK_GEOMETRY="{(0,3)(1,4)(2,5)}"
```

The result of such a geometry would be the following:

- MPI Tasks #0 and #3 on node #1
- MPI Tasks #1 and #4 on node #2
- MPI Tasks #2 and #5 on node #3

> Note: The geometry does not define the identify for nodes #1, #2 and #3. This assignment remains under the responsibility of the workload manager.

> Note: It is also possible to specify an alternate task placement by using a mechanism which is specific to the MPI library used for the parallel execution.

5.5.2. PROCESSOR AFFINITY

The processor affinity can be defined in two distinct ways:

- Through an LSF-internal application (easy way).
  This mechanism is invoked through the following syntax inside the submission script:

```
#BSUB -a p8aff(<num_threads_per_task>,<smt>,<cpus_per_core>,<distribution_policy>)
```

The ‘p8aff’ application takes the following arguments:

Argument	Value	Purpose
<num_threads_per_task>	1-160	Number of threads per task (OMP_NUM_THREADS environment variable will be initialized with the given value)
<smt>	1, 2, 4, 8	SMT mode to apply to allocated compute nodes
<cpus_per_core>	1-8	Number of logical cores (CPUs) used inside every physical core
<distribution_policy>	balance pack	Task distribution inside a compute node

Note: Except for the number of threads per task, all arguments are considered as optional, and can therefore remain unspecified. However, for clarity purpose, it is recommended to specify all four arguments.

Note : The list of arguments is comma-separated, with no space between separators and values.

The following table shows the ‘p8aff’ applications parameters for a given set of very classical execution configurations:

Execution Configuration	‘p8aff’ esub Options
2 MPI Tasks x 10 Threads Per Task	#BSUB -a “p8aff(10,1,1,balance)”
4 MPI Tasks x 5 Threads Per Task	#BSUB -a “p8aff(5,1,1,balance)”
10 MPI Tasks x 2 Threads Per Task	#BSUB -a “p8aff(2,1,1,balance)”
20 MPI Tasks x 1 Thread Per Task	#BSUB -a “p8aff(1,1,1,balance)”
20 MPI Tasks x 2 Threads Per Task	#BSUB -a “p8aff(2,2,2,balance)”
20 MPI Tasks x 4 Threads Per Task	#BSUB -a “p8aff(4,4,4,balance)”
20 MPI Tasks x 8 Threads Per Task	#BSUB -a “p8aff(8,8,8,balance)”

- Through an explicit resource requirement specification (manual way).
  This resource requirement is expressed through the following syntax inside the submission script:

```
#BSUB -R “affinity[<pu_type>(#):cpubind=<level>:distribute=<policy>]”
```

Sub-Option	Value	Purpose
<pu_type>(#)	numa socket core thread	Type and number of Processor Units assigned to each task
cpubind=<level>	numa socket core thread	Task binding policy
distribute=<policy>	balance pack	Task distribution onto the Processor Units inside a compute node

Distribution Policy

Distribution policy is a two-option strategy that affects the way MPI tasks are spread onto the two sockets of the compute nodes:

- 'Pack' Strategy:
  No balancing is performed between the two sockets. MPI tasks placement starts with socket #0, and moves to socket #1 whenever socket #0 is full.
- 'Distribute' Strategy:
  A balancing is performed between the two sockets, so that they host approximately the same number of MPI tasks at the end.

The following examples show the difference between the two strategies in a SMT=1 and 4 MPI Task configuration:

- 'Pack' Strategy:

Socket #0										Socket #1									
0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152
X	X	X	X																

- 'Distribute' Strategy:

Socket #0										Socket #1									
0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152
X	X									X	X								

5.5.3. GPU SELECTION

GPU selection is managed at two distinct levels:

- Resource Requirement & Resource Sharing.
  At the Spectrum LSF level, the characteristics of the GPU devices to be allocated to the job must be defined:
  - Number of GPU devices
  - Compute Mode of the GPU devices: Shared or Exclusive Process
  - NVIDIA Multi-Process Service (MPS) activation
    - MPS is recommended whenever GPU devices are shared between multiple processes
  - Sharing of GPU devices between multiple jobs

This resource requirement is achieved through the following syntax inside the submission script:

```
#BSUB -gpu "num=<num_gpus>:mode={shared|exclusive_process}:mps={no|yes}:j_exclusive=yes"
```

Note: A GPU device can either be dedicated to one single process or shared between multiple processes. In the latter case, two distinct configurations are valid:

1) GPU device set in 'Shared' Compute Mode

2) GPU device in 'Exclusive Process' Compute Mode with MPS enabled

For performance reasons, configuration #2 is recommended over configuration #1.

- GPU Device Visibility Management.
  The set of GPU devices that are visible to a given process is managed through the environment variable: CUDA_VISIBLE_DEVICES.
  If a proper GPU resource requirement has been expressed in the job submission script, Spectrum LSF assigns a default value to this variable. This value corresponds to the GPU devices that have been allocated to the job.
  However, it might be necessary to manually redefine the value of the variable, so that each MPI task can only address one single GPU device.

Important: CUDA_VISIBLE_DEVICES='' (null value) means that no GPU are assigned to the task. This would trigger error messages whenever the task would try accessing the GPU resource for the first time.

5.5.4. CUSTOMIZED HELPER SCRIPT

In order to make it easier for the user to select each process target GPU device, a customized helper script has been made available:

```
/pwrlocal/ibmccmpl/bin/task_prolog.sh
```

This script has the following usage:

```
$ /pwrlocal/ibmccmpl/bin/task_prolog.sh -help
Usage: task_prolog.sh <options>

Options:

   -nvprof-enable             Start nvprof
   -nvprof-rank rank          Restrict Profiling To Specified MPI Rank
   -nvprof-options "opts"     Additional nvprof Options

   -perf-enable               Start Perf
   -perf-rank rank            Restrict Profiling To Specified MPI Rank
   -perf-options "opts"       Additional Perf Options

   -devices dev               Set Visible Devices: { auto | <Semi-Colon Separated List of GPU Devices> }

   -env-display               Display User Environment

   -verbosity level           Set Verbosity Level: { 0 | 1* | 3 }

Examples:

   Set Visible GPU Devices Automatically
      $ mpirun /pwrlocal/ibmccmpl/bin/task_prolog.sh -devices auto poisson.exe

   Set Visible GPU Devices Explicitely (GPU0 For Rank #0, GPU1 For Rank #1)
      $ mpirun /pwrlocal/ibmccmpl/bin/task_prolog.sh -devices 0:1 poisson.exe

   Start nvprof Profiling
      $ mpirun /pwrlocal/ibmccmpl/bin/task_prolog.sh -devices auto -nvprof-enable poisson.exe
```

The automatic GPU device selection should be the preferred mode by default. In this mode, the GPU device selection will be performed based on the processor affinity of the process, ensuring this selection is optimal from a Host-To-Device / Device-To-Host perspective.

6. DEBUGGING

6.1. GNU DEBUGGER

The GNU Debugger is installed on the login nodes and on the compute nodes.

The GNU Debugger is useful for:

- Performing interactive debugging - for a single process execution (or on a limited number of processes).
- Analyzing 'core' files generated by an execution that terminated with a memory dump.

Memory Dump 'core' Files Analysis

The following command allows opening a memory dump 'core' file with GDB :

```
gdb <binary> <coredump>
```

In the GDB session, the 'backtrace' command makes it possible to inspect the backtrace of routine calls.

Example : Analysis of a Memory Dump 'core' File (EMMA Application)

```
[login@ouessant ~]$ gdb ~/emma/1.2/bin/emmacpu coredir.0/core.93667
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-80.el7
[…]
Core was generated by `~/emma/1.2/bin/emmacpu param.'.
Program terminated with signal 11, Segmentation fault.
#0  0x00001000279e1b78 in ?? ()
Missing separate debuginfos, use: debuginfo-install glibc-2.17-105.el7.ppc64le libgcc-4.8.5-4.el7.ppc64le
libibverbs-1.1.8mlnx1-OFED.3.1.1.0.0.ppc64le libmlx4-1.0.6mlnx1-OFED.3.1.1.0.0.ppc64le libmlx5-1.0.2mlnx1-
OFED.3.1.1.0.5.ppc64le libnl-1.1.4-3.el7.ppc64le libstdc++-4.8.5-4.el7.ppc64le numactl-libs-2.0.9-
5.el7_1.ppc64le xorg-x11-drv-nvidia-devel-352.59-1.el7.ppc64le
 (gdb) backtrace
#0  0x00001000279e1b78 in ?? ()
#1  0x0000000010033030 in PoissonMgrid (level=-952000048, param=0x10000022b5b0 <MPIDI_SendDoneCB>,
firstoct=0x1000000f250c <MPIR_Barrier_intra+684>, cpu=0x10000032c770, stencil=0x14, stride=-952000048,
tsim=6.72623263e-44)
    at /pwrwork/login/emma/1.2/000008/000000_build/work/EMMA-develop/src/poisson_utils.c:1346
#2  0x0000000010033898 in PoissonSolver (level=-951999664, param=0x10028044442, firstoct=0x3fffc741a2b0,
cpu=0x10000032c770, stencil=0x1000000f21fc <MPIR_Barrier_impl+172>, stride=-951999664, aexp=0)
    at /pwrwork/login/emma/1.2/000008/000000_build/work/EMMA-develop/src/poisson_utils.c:1546
#3  0x0000000010023bac in Advance_level (level=-951998944, adt=0x0, cpu=0x3f8a502100000000,
param=0x405335436a8046bf, firstoct=0xc01eb0a6e0000000, lastoct=0xbff6dc31c0000000, stencil=0x1000006d1370
<_pthread_cleanup_pop_restore>,
    gstencil=0x0, rstencil=0x10009c180010, ndt=0x1000598dbc0, nsteps=-951998944, tloc=0) at
/pwrwork/login/emma/1.2/000008/000000_build/work/EMMA-develop/src/advanceamr.c:442
#4  0x0000000001000350c in main (argc=2, argv=0x3fffc7437688) at
/pwrwork/login/emma/1.2/000008/000000_build/work/EMMA-develop/src/emma.c:1949
(gdb)
```

7. PERFORMANCE ANALYSIS & OPTIMIZATION

7.1. PERF

The Linux standard Perf tool makes it possible to perform performance profiling of a specific process.

Perf mainly relies on the following commands:

Command	Option	Purpose
perf annotate		Annotate source code with events
perf list		List supported hardware counters
perf record		Record events
	-e rNNN	Include specified hardware counter
	-g	Include Call Graph
perf report		Display event by process / routine / …
	-g	Include Call Graph

Performance Data Collection

The performance data collection is triggered by the 'perf record' command.

The command can apply to a binary execution :

```
perf record <binary>
```

The command can also apply to a given process designated by its PID:

```
perf record --pid=<pid>
```

Data collection produces an output file named 'perf.data'.

Performance Reporting

The 'perf report' command produces a by-routine performance report:

```
perf report --input perf.data > perf.report
```

The 'perf annotate' command generates line-by-line annotations inside the routines:

```
perf annotate --input perf.data > perf.annotate
```

7.2. OPROFILE

OProfile is an Open-Source project including a statistical tool for performance profiling. OProfile is available inside the Linux distribution itself, or through the 'IBM SDK for LoP' (up-to-date version with full POWER8 support).

OProfile mainly relies on the following commands:

Command	Purpose
ophelp	List available hardware counters
operf	Record events
opreport	Display performance report
opannotate	Annotate source code with events

Performance Data Collection

Performance data collection is achieved through the 'operf' command:

```
[login@ouessant ~]$ operf <binary>
```

Performance Data Reporting

Performance data reporting is achieved through the 'opreport' command:

```
[login@ouessant ~]$ opreport
CPU_CLK_UNHALT...|
  samples|      %|
------------------
    25202 100.000 cg.W.x
CPU_CLK_UNHALT...|
  samples|      %|
------------------
    18509 73.4426 cg.W.x
     5190 20.5936 libiomp5.so
     1443  5.7257 no-vmlinux
       37  0.1468 ld-2.19.so
       12  0.0476 libifcoremt.so.5
        7  0.0278 libc-2.19.so
        2  0.0079 libpthread-2.19.so
```

7.3. MPI TRACE LIBRARY

The MPI Trace Library is an IBM-internal profiling tool.

Initial Setup

Profiling through the MPI Trace Library is enabled by linking with one of the three available libraries:

Library File	Profiling
libhpmprof.so	CPU Profile
libmpihpm.so	Hardware Counters
libmpitrace.so	MPI Trace

```
[login@ouessant ~]$ mpicc -g -o <Binary> -L/pwrlocal/ibmccmpl/opt/mpitrace/17.02.07/lib -lhpmprof <Object
Files>
```

Performance Data Collection

Performance data collection is automatically performed at runtime.

The following environmental variables give user some control over the performance data collection process:

Variable	Value	Purpose
SAVE_ALL_TASKS	yes	Save trace data for all MPI ranks
SAVE_LIST	<List of Ranks>	Save trace data for given MPI ranks
TRACE_ALL_EVENTS	yes	Turn tracing of all MPI events
TRACE_DIR	<Path>	Target directory for trace data

Depending on the selected profiling, the following output files can be present at the end of the execution:

Output File	File Type	Purpose
events.trc	Binary	MPI events binary trace
hpm_summary.<Job #>	Text	Hardware counters
hpm_histogram.<Job #>.<Rank>	Binary	CPU Profile data
mpi_profile.<Job #>.<Rank>	Text	MPI Profile Per Rank

Flat Profile Generation

Using the profiling output data files, the flat profile of the execution can be generated through the following command:

```
[login@ouessant ~]$ bfdprof <binary> hpm_histogram.<jobid>.<rank> > bfdprof.out.<rank>
```

MPI Events Trace View

MPI events can be visualized through the 'traceview' utility.

7.4. NVPROF

[To Be Documented]

7.5. IBM PARALLEL PERFORMANCE TOOLKIT FOR POWER

The IBM Parallel Performance Toolkit is the standard performance analysis tool for OpenPOWER platforml.

IBM Parallel Performance Toolkit is a new, rebranded release of Parallel Environment Developer Edition (PE DE).

Binary Instrumentation

Instrumentation of the executable is performed through the following command:

```
hpctInst <options> <binary>
```

The main options for selecting the type of instrumentation are the following:

Option	Purpose
-dhpm	Hardware Counters
-dmio	I/O Profiling
-dmpi	MPI Profiling
-dpomp	OpenMP Profiling

Hardware Counters

The following command displays the list of supported Hardware Counter groups (267 in total):

```
[login@ouessant ~]$ hpccount -l

 Linux_Power8: number of CPU groups supported: 267
 Group 0:
 PM_CYC ----------- Cycles
 PM_RUN_CYC ------- Run cycles
 PM_INST_DISP -----   PPC Dispatched
 PM_INST_CMPL ----- Number of PowerPC Instructions that completed. PPC Instructions Finished (completed).
 PM_RUN_INST_CMPL - Run instructions
 PM_RUN_CYC ------- Run cycles

[…]

 Group 266:
 PM_MRK_DTLB_MISS_16G - Marked Data TLB Miss page size 16G
 PM_MRK_DTLB_MISS_4K -- Marked Data TLB Miss page size 4k
 PM_DTLB_MISS --------- Data PTEG reload
 PM_INST_CMPL --------- Number of PowerPC Instructions that completed. PPC Instructions Finished
(completed).
 PM_RUN_INST_CMPL ----- Run instructions
 PM_RUN_CYC ----------- Run cycles
```

8. MACHINE LEARNING / DEEP LEARNING WORKLOADS

This section is specifically targeted to the Machine Learning / Deep Learning (ML/DL) workloads that are expected to take place on the platform.

8.1. SOFTWARE STACK

8.1.1. POWERAI

The ML/DL software stack is made available on the platform mainly through the PowerAI package collection.

More specifically, PowerAI 4.0 brings the following software packages:

Software Package	Version
Caffe [Berkeley]	1.0.0
Caffe [NVIDIA]	0.15.14
Caffe [IBM]	1.0.0
Chainer	1.23.0
DIGITS	5.0.0
Google TensorFlow	1.1.0
Theano	0.9.0
Torch	7

8.1.2. ADDITIONAL SOFTWARE PACKAGES

The following software packages are also part of the ML/DL image:

Software Package	Version
NVIDIA cuDNN	6.0.21-1

8.1.3. ADDITIONAL PYTHON PACKAGES

Additional Python packages have also been installed on the shared filesystem upon users request.

These packages have been made available as part of the 'Anaconda' Python distribution. Both versions of Anaconda have been installed:

- Anaconda2

```
/pwrlocal/pub/anaconda2/4.4.0/arch/ubuntu-16.04/
```

- Anaconda3

```
/pwrlocal/pub/anaconda3/4.4.0/arch/ubuntu-16.04/
```

At the time of writing, the following additional Python packages have been made available (non-comprehensive list):

- filelock
- future

- markdown
- mock
- keras
- pydot-ng

8.2. JOB SUBMISSION

8.2.1. DOCKER CONTAINER

For compatibility reasons, the execution of ML/DL workloads on the allocated compute nodes must be performed inside a Docker container.

The container provides an Ubuntu 16.04 runtime environment running on top of the installed Linux kernel.

Note: For more information regarding the Docker container technology, please refer to: https://www.docker.com/what-container

8.2.2. SUBMISSION PROCESS

Job submission is performed according to the general principles described in §5 'Job Submission'.

However, because of the container specificity, the following two actions are required from the user:

- User environment needs to be properly configured for running inside a container, which is achieved through loading the following module prior to submission:

```
$ ml powerai
```

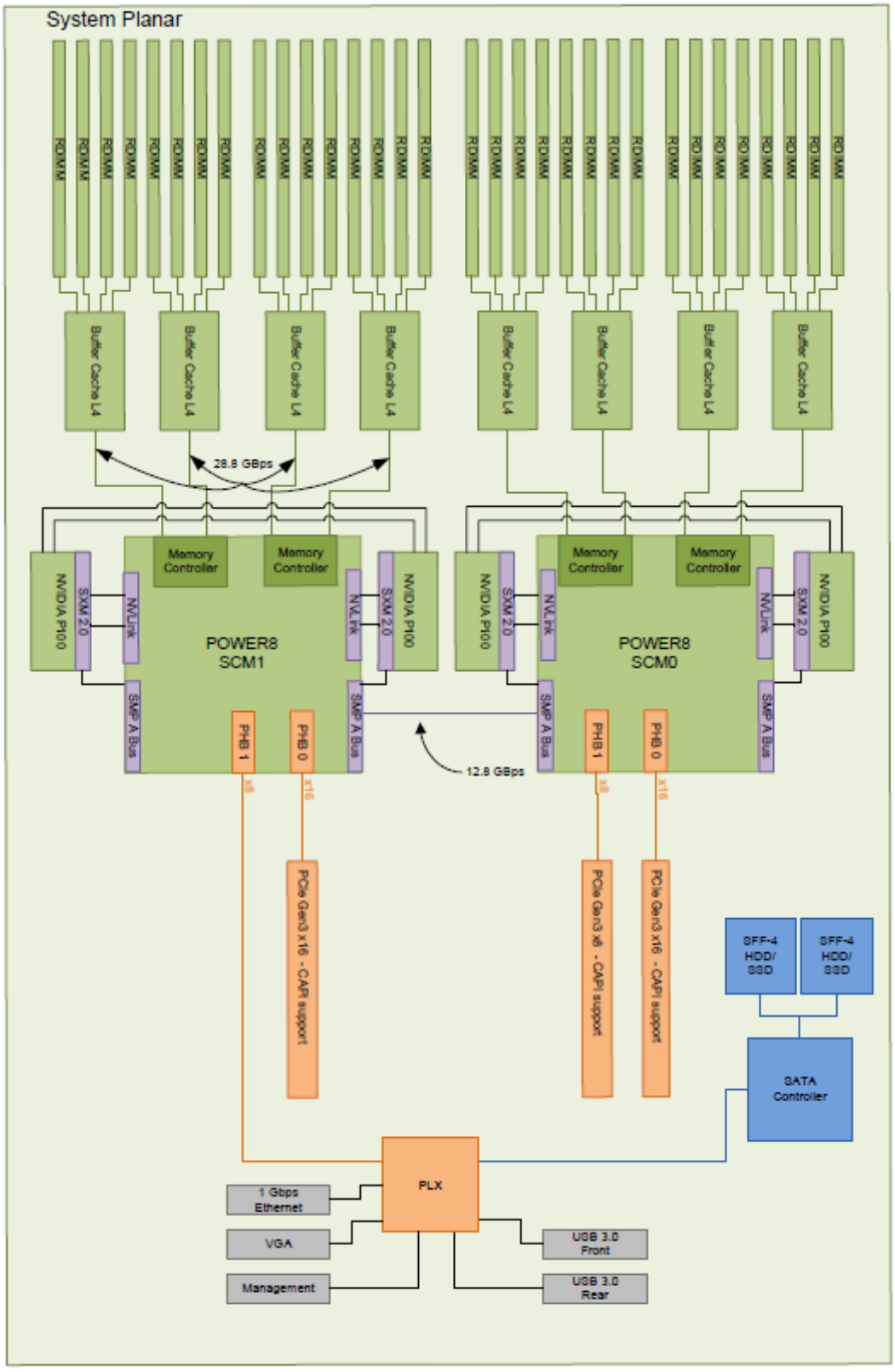
- The container needs to be explicitely requested through the following Spectrum LSF directive:

```
#BSUB -app docker
```

Note: A template file for a typical PowerAI job submission is available on the platform.

9. REFERENCE

9.1. IBM POWER SYSTEM S822LC 'MINSKY' LOGICAL ARCHITECTURE



9.2. IBM SPECTRUM LSF SAMPLE SUBMISSION SCRIPT

Parallel Execution: 2 Compute Nodes, 8 Tasks Total, 4 Tasks / Node, 1 GPU / Task

```
#!/bin/bash

#BSUB –a p8aff(1,1,1,balance)
#BSUB -cwd /pwrhome/login/myjob
#BSUB -e myjob.%J.log
#BSUB -gpu "num=4:mode=exclusive_process:mps=yes:j_exclusive=yes"
#BSUB -J MyJob
#BSUB -n 8
#BSUB –o myjob.%J.log
#BSUB -R "span[ptile=4]"
#BSUB -W 01:00
#BSUB -x

ml xlc xlf smpi

mpirun -display-allocation -prot -report-bindings /pwrlocal/ibmccmpl/bin/task_prolog.sh -devices auto
/pwrhome/login/myjob/myjob.bin
```

10. FREQUENTLY ASKED QUESTIONS (FAQ)

How To Display the Map of Physical Cores / Logical Cores?

The 'ppc64_cpu' command displays the relationship between physical cores and logical cores (hardware threads) of the system:

```
[login@ouessant ~]$ ppc64_cpu --info
Core   0:    0*    1*    2*    3*    4*    5*    6*    7*
Core   1:    8*    9*   10*   11*   12*   13*   14*   15*
Core   2:   16*   17*   18*   19*   20*   21*   22*   23*
Core   3:   24*   25*   26*   27*   28*   29*   30*   31*
Core   4:   32*   33*   34*   35*   36*   37*   38*   39*
Core   5:   40*   41*   42*   43*   44*   45*   46*   47*
Core   6:   48*   49*   50*   51*   52*   53*   54*   55*
Core   7:   56*   57*   58*   59*   60*   61*   62*   63*
Core   8:   64*   65*   66*   67*   68*   69*   70*   71*
Core   9:   72*   73*   74*   75*   76*   77*   78*   79*
Core  10:   80*   81*   82*   83*   84*   85*   86*   87*
Core  11:   88*   89*   90*   91*   92*   93*   94*   95*
Core  12:   96*   97*   98*   99*  100*  101*  102*  103*
Core  13:  104*  105*  106*  107*  108*  109*  110*  111*
Core  14:  112*  113*  114*  115*  116*  117*  118*  119*
Core  15:  120*  121*  122*  123*  124*  125*  126*  127*
Core  16:  128*  129*  130*  131*  132*  133*  134*  135*
Core  17:  136*  137*  138*  139*  140*  141*  142*  143*
Core  18:  144*  145*  146*  147*  148*  149*  150*  151*
Core  19:  152*  153*  154*  155*  156*  157*  158*  159*
```

In the above example, the system is configured with SMT8. Each of the 20 physical cores ('Core 0' to 'Core 19') is made of 8 logical cores (hardware threads).

How To Specify Pre-Processing Option (Macro) to IBM XL Fortran Compiler?

The IBM XL Fortran does not support the standard syntax related to pre-processing options.

With this compiler, it is required to add the '-WF' option to introduce macros, as shown in the example below:

```
xlf90_r -qfree=f90 '-WF,-Dmacro1=1,-Dmacro2' a.F
```

How To Increase The IBM XL SMP Library OpenMP Stack Size?

By default, the IBM XL SMP library has a very limited OpenMP stack size value (4 MB).

It is therefore frequent to alter this default value through the following environment variable setting:

```
export XLSMPOPTS="stack=16777216"
```

Note: Stack size must be specified in Bytes.

How To Avoid Some PGI Compilation Errors?

By default, PGI Accelerator uses its own preprocessor.

It is often the case that the preprocessor must be set to the default system preprocessor to avoid compilation errors. Setting the CPP environment variable is normally enough:

```
export CPP=/usr/bin/cpp
```

How To Execute A Parallel Job With Open MPI Through IBM Spectrum LSF?

The proper integration between IBM Spectrum LSF et Open MPI requires:

- Usage of '-rf <rankfile' argument in the Open MPI execution command:

```
mpiexec -rf ${LSB_RANK_HOSTFILE} <Binary>
```

- Reference to a standard application profile when submitting job into Spectrum LSF:

```
bsub -app ompi < job.sh
```

How To Open An Interactive Session Through IBM Spectrum LSF?

IBM Spectrum LSF allows opening an interactive session onto the compute nodes through the following command:

```
bsub -Ip -n <cpus> -q <queue> /bin/bash
```

Obviously, the interactive session will wait for the requested resource allocation to be possible before the session can actually open.

An additional, specific interconnect option is required by the 'mpirun' command when launching an MPI execution in interactive mode:

```
-pami_noib
```

How To Solve The 'Accelerator Does Not Match' Error Message?

The error message has the following syntax:

```
Current region was compiled for:
NVIDIA Tesla GPU sm30 sm35
Available accelerators:
device[1]: Native X86 (CURRENT DEVICE)
The accelerator does not match the profile for which this program was compiled
```

This message is linked to the fact that no visible GPU device has been defined for the given task.

The solution involves properly setting the environment variable CUDA_VISIBLE_DEVICES.

How To Monitor GPU Activity on Interactive Nodes?

Two options exist for monitoring the GPU activity on one of the interactive nodes:

- Option #1: NVIDIA System Management Interface Command

```
[login@ouessant ~]$ nvidia-smi pmon
# gpu     pid  type    sm   mem   enc   dec   command
# Idx       #   C/G     %     %     %     %   name
    0  145349     C    64     0     0     0   matrixMul
    1       -     -     -     -     -     -   -
    2  145351     C    65     0     0     0   matrixMul
    3       -     -     -     -     -     -   -
```

- Option #2: 'gpustat' Utility

```
ouessant  Mon Feb 13 19:08:43 2017
[0] Tesla K80        | 51'C,  23 % |    64 / 11441 MB | login:matrixMul/40281(62M)
[1] Tesla K80        | 33'C,   0 % |     2 / 11441 MB |
[2] Tesla K80        | 37'C,   0 % |     2 / 11441 MB |
[3] Tesla K80        | 31'C,   0 % |     2 / 11441 MB |
```

11. ADDITIONAL RESOURCES

IBM POWER8

Implementing an IBM High-Performance Computing Solution on IBM POWER8

http://www.redbooks.ibm.com/redbooks/pdfs/sg248263.pdf

Performance Optimization and Tuning Techniques for IBM Processors, including IBM POWER8

http://www.redbooks.ibm.com/redbooks/pdfs/sg248171.pdf

HPC Software Stack

IBM Spectrum LSF

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

http://www.redbooks.ibm.com/abstracts/redp5048.html?Open

http://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/lsf_admin/affinity_power8_lsf_submit.html

IBM Spectrum MPI

https://www.ibm.com/support/knowledgecenter/SSZTET/

IBM XL C/C++ for Linux

http://www-01.ibm.com/support/knowledgecenter/SSXVZZ/

IBM XL Fortran for Linux

http://www-01.ibm.com/support/knowledgecenter/SSAT4T/

Parallel Performance Toolkit For POWER

http://www.ibm.com/support/knowledgecenter/SSFK5S/

PGI Accelerator

https://www.pgroup.com/resources/accel.htm

Performance Analysis

OProfile

http://oprofile.sourceforge.net/

perf

https://perf.wiki.kernel.org/

http://www.brendangregg.com/perf.html

PowerAI

https://public.dhe.ibm.com/software/server/POWER/Linux/mldl/ubuntu/