JÜLICH
FORSCHUNGSZENTRUM

JUQUEEN

Best Practices

19 May 2015 | Florian Janetzko

Outline

- Production environment
  - Module environment
  - Job execution
- General porting of applications
  - Compilers and wrappers
  - Compiler/linker flags
- Porting and tuning – selected topics
  - Static and shared libraries
  - Mapping
  - MPMD
  - MPI extensions
  - QPX

Module environment

Modules – Features

Overview of available products

- Provides fast overview over all available products
- Provides fast overview over all available versions of a selected product

Use of software and libraries

- Enables access to software packages and libraries
- Supports different versions of one product side by side
- Provides application-specific information

Dynamic modification of user environment

- Necessary environment variables are set appropriately, no need to set variables manually
- Detection of conflicts between applications

Module environment

Command	`module [options] [module]`

Selected options	
	`<no option>`
	Lists all available options of the module command
	`avail`
	Lists all available modules and versions
	`list`
	Lists modules currently loaded
	`load/unload`
	Loads/unloads a module
	`help`
	Lists information about a module
	`show`
	Displays information about environment settings applied by the module
	`purge`
	Unloads all currently loaded modules

Module environment

Module categories on JUQUEEN

```
COMPILER
IO
MATH
MISC
SCIENTIFIC
TOOLS
```

Software for compute nodes

`/bgsys/local`

Software for front-end nodes

`/usr/local`

Example

```
------------ /bgsys/local/modulefiles/TOOLS ------------
------------ /bgsys/local/modulefiles/SCIENTIFIC ------------
OpenFOAM/2.1.1-gcc(default)    cpmd/3.15.3
QuantumEspresso/5.0.1(default) lammps/30Aug12
[...]
------------ /usr/local/modulefiles/COMPILER ------------
cmake/2.8.11(default)  python3/3.4.1(default)
------------ /usr/local/modulefiles/MATH ------------
[...]
```

Module environment – available software

Selected mathematical software

- Elemental (0.81)
- FFTW (2.1.5/3.3.3)
- GSL (1.15)
- hypre (2.9.0)
- LAPACK (3.4.2)
- MUMPS (4.10.0)
- P3DFFT (2.5.2)
- ParMETIS (4.0.2)
- PETSc (3.5.1)
- ScaLAPACK (2.0.2)
- SPRNG (2.0)
- sundials (2.5.0)

Science

- cp2k (2.4.0)
- cpmd (3.15.3)
- lammps (30AUG12)
- NAMD (2.9)
- OpenFOAM (2.1.1)
- QE (5.0.1)

I/O libraries and tools

- cdo (1.6.1)
- darshan (2.2.8)
- HDF5 (1.8.9)
- Nco (4.3.4)
- netCDF (4.3)
- p-netCDF (1.3.1)
- SIONlib (1.5.2)

Tools

- extrae (2.4.1)
- MUST (1.2.1)
- Scalasca (2.1)
- Scorep (1.3)
- Tau (2.23.1)
- Totalview (8.14.0)
- Vampir (8.4)

Compiler (tools)

- cmake (2.8.11)
- clang (3.4)
- gcc (4.9.1)
- Not via modules
- bgxlc (12.01)
- bgxlf (14.01)

Job execution – Batch system

Execution of applications managed by LoadLeveler

- Users submit jobs using a job command file
- LoadLeveler allocates computing resources to run jobs
- The scheduling of jobs depends on
  - Availability of resources
  - Job priority (jobs with larger core counts are privileged)
- Jobs run in queues (job classes)
  - Chosen by LoadLeveler according to core count

More information about LoadLeveler on JUQUEEN

http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/LoadLeveler.html

Job execution – Job command file

ASCII file containing two major parts

Job command file

Keyword block

- LoadLeveler keywords have the form
  `#@<keyword>`
- # and @ can be separated by any number of blanks

Shell script block

- Regular shell script
- Can contain any shell command

Job execution – LoadLeveler keywords

Keyword	Description
`#@job_name=<name>`	Name of the job

`#@notification=`
`  end`
`  error`
`  never`
`  start`
`  always`
	Send notification
	if the job is finished
	if the job returned an error code ≠ 0
	never
	upon the start of the job
	combination of `start,end,error`

`#@notify_user=<mailaddr>`	Mail address to send messages to
`#@wall_clock_limit=hh:mm:ss`	Requested wall time for the job

`#@input=<input file name>`
`#@output=<file name for stdout>`
`#@error=<file name for stderr>`
	Specifies corresponding file names

`#@environment=[<variable>,`COPY_ALL]	Environment variable to be exported to job
`#@queue`	End of keyword section

Job execution – LoadLeveler keywords

Keyword	Description
`#@job_type=[serial, bluegene]`	Specifies the type of job step to process. Must be set to **`bluegene`** for parallel applications.
`#@bg_size=<number of nodes>`	Size of the Blue Gene job, keywords **`bg_size`** and **`bg_shape`** are mutually exclusive.
`#@bg_shape=<A>x<B>x<C>x<D>`	Specifies the requested shape of a job (in midplanes).
`#@bg_rotate=[True,False]`	whether the scheduler should consider all possible rotations of the given shape
`#@bg_connectivity=[TORUS,MESH,EITHER] Xa Xb Xc Xd`	Type of wiring requested for the block (can be specified for each dimension separately)

Job execution – Shell script section

- Can contain any shell scripts commands
- Launching parallel applications:

Command	`runjob [options]: <executable> [arguments]`

Selected options	
	`--envs <ENV_Var=Value>`
	Sets the environment variable **`ENV_Var=Value`**
	`--exp-env <ENV_Var>`
	Sets the environment variable **`ENV_Var`**
	`--np <number>`
	Total number of (MPI) tasks
	`--ranks-per-node <number>`
	Number of (MPI) tasks per compute node

Job execution – Example Job command file

```
#@job_name          = MPI_code
#@comment           = "32 ranks per node"
#@output            = test_$(jobid).out
#@error             = test_$(jobid).err
#@environment       = COPY_ALL
#@job_type          = bluegene
#@notification      = never
#@bg_size           = 512
#@bg_connectivity   = torus
#@wall_clock_limit = 14:00:00
#@queue

runjob --np 16384 --ranks-per-node 32 : app.x
```

! **Pure MPI applications need to use 32 tasks per node in order to use the architecture efficiently, i.e., to fill all pipe lines and registers and to avoid idle times!** **!**

Job execution – MPI/OpenMP hybrid jobs

```
#@job_name          = hybrid_code
#@comment           = "16x4 configuration"
#@output            = test_$(jobid).out
#@error             = test_$(jobid).err
#@environment       = COPY_ALL
#@job_type          = bluegene
#@notification      = never
#@bg_size           = 512
#@bg_connectivity   = torus
#@wall_clock_limit = 14:00:00
#@queue

runjob --np 8192 --ranks-per-node 16\
       --env OMP_NUM_THREADS=4 : app.x –i input
```

!

Several process/thread configurations are possible:

- ntasks × nthreads = 64
- ntasks = 2^n, $0 \leq n \leq 6$

Test which configuration gives the best performance for your application and setup!

!

Job execution – Job classes

Class name	#Nodes	Max. run time	Default run time
n001	1 – 32	00:30:00	00:30:00
n002	33 – 64	00:30:00	00:30:00
n004	65 – 128	12:00:00	06:00:00
n008	129 – 256	12:00:00	06:00:00
m001	257 – 512	24:00:00	06:00:00
m002	513 – 1024	24:00:00	06:00:00
m004	1025 – 2048	24:00:00	06:00:00
m008	2049 – 4096	24:00:00	06:00:00
m016	4097 – 8192	24:00:00	06:00:00
m032	8193 – 16384	24:00:00	06:00:00
m056*	24577 – 57344	24:00:00	06:00:00

* On demand only

! **You will be charged for full partitions (e.g. when requesting 513 nodes you will be charged for 1024 nodes!) → Always use full partitions!** **!**

Job execution – Loadleveler commands

Command	Description
`llsubmit <jobfile>`	Sends job to the queuing system

`llq`
`llq -l <job ID>`
`llq -s <job ID>`
 	
`llq -u <user>`
	Lists all queued and running jobs
	detailed information about the specified job
	detailed information about a specific queued job, e.g. expected start time
	lists all jobs of the specified user

`llcancel <job ID>`	Kills the specified job
`llstatus`	Displays the status of LoadLeveler
`llclass`	Lists existing classes and their properties
`llqx`	Shows detailed information about all jobs

Job execution – Monitoring of jobs

LoadLeveler

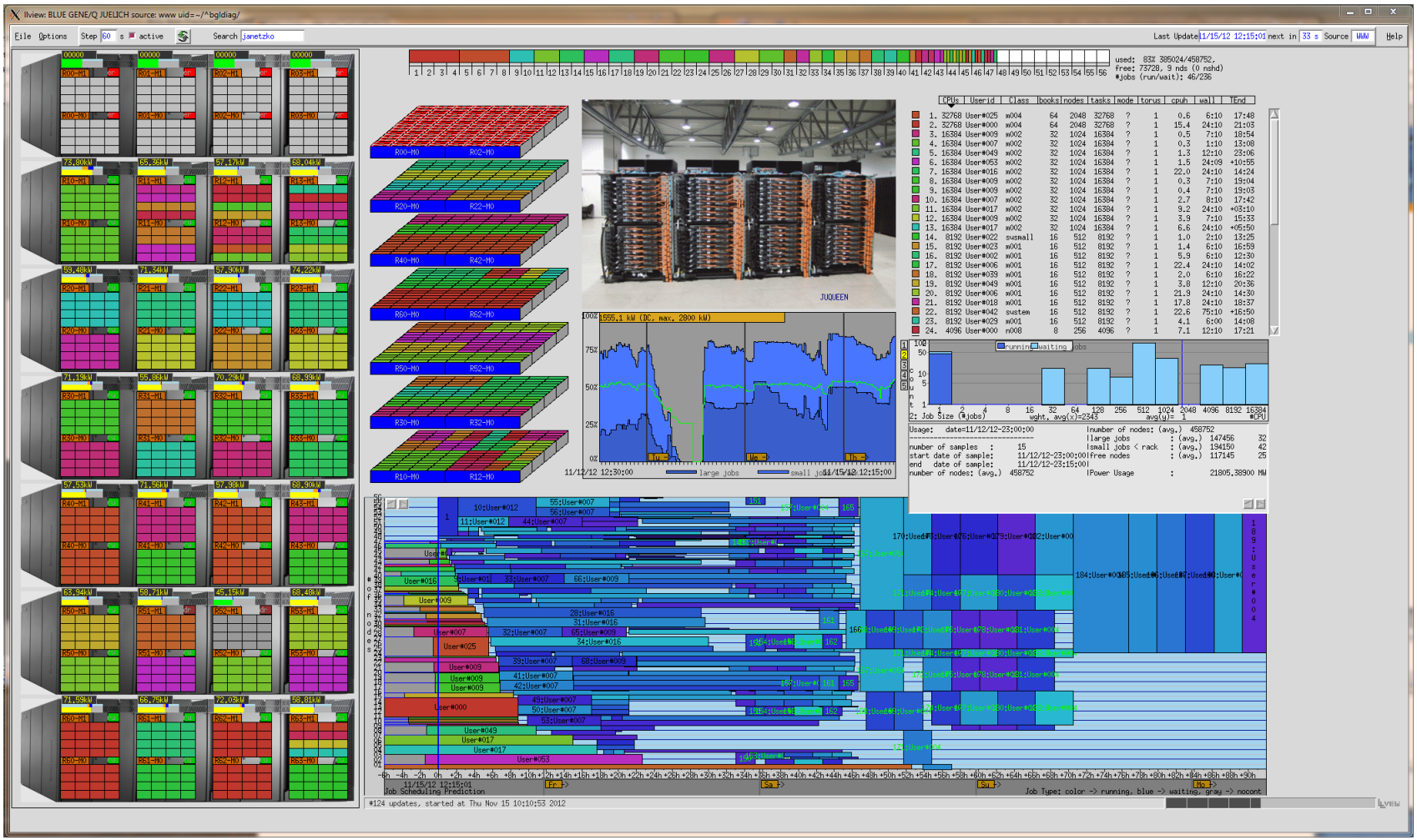
Command	`llq [options]`

llview

- Client-server based application
- compact summary of different information (e.g. current usage of system, job prediction, expected and average waiting times, …)
- Customizable
- Developed by W. Frings (JSC)

Command	`llview`

Job execution – llview



Outline

- Production environment
  - Module environment
  - Job execution
- General porting of applications
  - Compilers and wrappers
  - Compiler/linker flags
- Porting and tuning – selected topics
  - Static and shared libraries
  - Mapping
  - MPMD
  - MPI extensions
  - QPX

Compilers and wrappers

- Different compilers for front-end and compute nodes
- GNU and IBM XL family of compilers available
- For C/C++ also Clang compiler for compute nodes available
  - Full support of C++ 11 standard and special BGQ hardware features like QPX and vector intrinsics

Front – end nodes	Language	XL (thread-safe: add _r)	GNU	Clang
	C	`xlc`	`gcc`	`---`
	C++	`xlc++, xlC`	`g++`	`---`
	Fortran	`xlf, xlf90, xlf95, xlf2003`	`gfortran`	`---`

Compilers and wrappers

		Language	Compiler	MPI wrapper
Compute nodes	XL (thread-safe: add _r)	C	`bgxlc, bgc89, bgc99`	`mpixlc`
		C++	`bgxlc++, bgxlC`	`mpixlcxx`
		Fortran	`bgxlf, bgxlf90, bgxlf95, bgxlf2003`	`mpixlf77, mpixlf90, mpixlf95, mpixlf2003`
	GNU	C	`powerpc64-bgq-linux-gcc`	`mpigcc`
		C++	`powerpc64-bgq-linux-g++`	`mpig++`
		Fortran	`powerpc64-bgq-linux-gfortran`	`mpigfortran`
	Clang	C	bgclang	`mpiclang`
		C++	bgclang++	`mpiclang++`
		Fortran	`---`	`---`

! If working with the XL suite of compilers, always use the thread-safe MPI wrappers when compiling for the compute nodes! !

! If you use OpenMP tasks, the gcc suite might lead to executables with a noticeable better performance than the XL suite of compilers! !

MPI setups

Six different MPI setups available on JUQUEEN

Location of wrappers	`/bgsys/drivers/ppcfloor/comm/<name>/bin`

<name>	Description	Usage
gcc	MPI library compiled with gcc, fine-grained locking, error checking and assertions enabled	Initial porting
gcc.legacy	MPI library compiled with gcc, coarse-grained locking, error checking and assertions enabled → slightly better latency for single-threaded codes	
xl	MPI library compiled with XL compilers, PAMI compiled with gcc, error checking and assertions enabled ***(default wrappers)***	
xl.legacy	MPI library compiled with XL compilers, PAMI compiled with gcc, coarse-grained locking, error checking and assertions enabled	
xl.ndebug	MPI library compiled with XL compilers, PAMI compiled with gcc, fine-grained locking, error checking and assertions disabled	Correctly running production code
xl.legacy.ndebug	MPI library compiled with XL compilers, PAMI compiled with gcc, coarse-grained locking, error checking and assertions disabled	

XL: basic compiler and linker options

Optimization Level	Description
`-O2 -qarch=qp -qtune=qp`	Basic optimization
`-O3 -qstrict -qarch=qp -qtune=qp`	More aggressive, not impact on acc.
`-O3 -qhot -qarch=qp -qtune=qp`	More aggressive, may influence acc. (high-order transformations of loops)
`-O4 -qarch=qp -qtune=qp`	Interprocedural optimization at compile time
`-O5 -qarch=qp -qtune=qp`	Interprocedural optimization at link time, whole program analysis

! **Some flags need to be used for compilation AND linking – Check the compiler manual. If you are not sure, include all flags used in the compile step in the linking step as well!** **!**

XL: basic compiler and linker options

Compiler/linker flag	Description
`-qsmp=omp -qthreaded`	Switch on OpenMP support
`-qreport -qlist`	Generates for each source file `<name>` a file `<name>.lst` with pseudo code and a description of the kind of code optimizations which were performed
`-qessl –lessl[smp]bg`	Compiler attempts to replace some intrinsic FORTRAN 90 procedures by essl routines where it is safe to do so

More information about using ESSL

http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/SystemDependentLibraries/ESSL_ESSLSMPJuqueen.html

Example: compiler diagnostics

```
subroutine mult(c,a,ndim)

implicit none
integer :: ndim,i,j
double precision ::
a(ndim),c(ndim,ndim)

! Loop
do i=1,1000
  do j=1,1000
    c(i,j) = a(i)
  enddo
enddo

end subroutine mult
```

```
>>>>> LOOP TRANSFORMATION SECTION <<<<<
1| SUBROUTINE mult (c, a, ndim)

[...]
 Id=1   DO $$CIV2 = $$CIV2,124
   10|    IF (.FALSE.) GOTO lab_11
            $$LoopIV1 = 0
 Id=2   DO $$LoopIV1 = $$LoopIV1,999
[...]
-----------------------------------
0 9 1    Loop interchanging applied
         to loop nest.
0 9 1    Outer loop has been
         unrolled 8 time(s).
```

!

!

XL: further compiler and linker options

Compiler/linker flag	Description
`-qinline=auto:level=<number>` `-qinline+procedure1[:procedure2[:…]]`	Function inlining `0 <= <number> <= 10`
`-qunroll[={auto|yes}]`	Unrolling of loops, default is `-qunroll=auto`
`-qipa[=<suboptions_list>]`	Intra/inter-procedural optimization, see manual
`-qsimd[=auto|noauto]`	Switches on/off SIMD vectorization (check with `-qreport -qlist` !!)

! **Intra-procedural optimization (-O5 or –qipa=2) can cause very long compilation times, especially for C++ codes!** !

Outline

- Production environment
  - Module environment
  - Job execution
- General porting of applications
  - Compilers and wrappers
  - Compiler/linker flags
- Porting and tuning – selected topics
  - Static and shared libraries
  - Mapping
  - MPMD
  - MPI extensions
  - QPX

User Information and Support

Information about JUQUEEN

- JSC websites at
  http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html
- IBM Blue Gene/Q Application Development Redbook
  http://www.redbooks.ibm.com/redpieces/pdfs/sg247948.pdf

Dispatch and User Support

- Applications for accounts (for approved projects)

Forschunszentrum Jülich GmbH,JSC, Dispatch, 52425 Jülich
Tel: +49 2461 61 5642, Fax: +49 2461 61 2810
email: dispatch.jsc@fz-juelich.de

- User Support

Tel: +49 2461 61 2828
email: sc@fz-juelich.de

Creating static libraries

Preferred kind of library on BG/Q

```
  bgxlc -c pi.c
  bgxlc -c main.c
#
### Create the library
  ar rcs libpi.a pi.o
#
### Create the executable program
  bgxlc -o pi main.o -L. -lpi
```

Creating shared libraries

```
  bgxlc -c pi.c
  bgxlc -c main.c
### Create the dynamic library
  bgxlc -qmkshrobj -Wl,-soname, libpi.so.0 \
        -o libpi.so.0.0 libpi.o
### Set up the soname
  ln -s libpi.so.0.0 libpi.so.0
### Create a linker name
  ln -s libpi.so.0 libpi.so
### Create the executable program
  bgxlc -o pi main.o -L. -lpi -qnostaticlink \
        -qnostaticlink=libgcc
```

! **Shared libraries might become a bottleneck when using large core counts on Blue Gene systems! Try to avoid them!** !

Mapping

Mapping

The placement of MPI ranks on the cores of a partition, i.e., the physical layout of the MPI ranks, is called mapping.

Network topology of JUQUEEN

- 5 dimensional hypercube or torus
- Each node is connected to 10 nearest neighbors in directions ±A, ±B, ±C, ±D, ±E
- A 6th dimension T is added to specify the hardware thread ID within one node (0 < T <= *N*-1, where *N* is the number of MPI ranks per node)

Node location in the torus	<A, B, C, D, E>
Process or thread location in the torus	<A, B, C, D, E, T>

Mapping – Receiving partition information

Command `runjob : /bgsys/local/samples/personality/personality.elf`

```
Partition Information:
block shape : <2,2,2,2,2>
torus links enabled : <0,0,0,0,1>

IO Bridge Information:
rank 0000017 of 0000032 location <01,00,00,00,01> core 00 hwthread 0 procid 00 ioBridge
<01,00,00,00,01> cn 'R63-M0-N06-J11' ion 'R63-ID-J03'
rank 0000023 of 0000032 location <01,00,01,01,01> core 00 hwthread 0 procid 00 ioBridge
<01,00,01,01,01> cn 'R63-M0-N06-J06' ion 'R63-ID-J01'

Compute Node Information:
rank 0000023 location <01,00,01,01,01> core 00 hwthread 0 procid = 00 ( Task 23 of 32 (1,0,1,1,1,0)  R63-M0-N06-J06 )
rank 0000017 location <01,00,00,00,01> core 00 hwthread 0 procid = 00 ( Task 17 of 32 (1,0,0,0,1,0)  R63-M0-N06-J11 )
rank 0000022 location <01,00,01,01,00> core 00 hwthread 0 procid = 00 ( Task 22 of 32 (1,0,1,1,0,0)  R63-M0-N06-J01 )
rank 0000000 location <00,00,00,00,00> core 00 hwthread 0 procid = 00 ( Task 0 of 32 (0,0,0,0,0,0)  R63-M0-N06-J17 )
rank 0000016 location <01,00,00,00,00> core 00 hwthread 0 procid = 00 ( Task 16 of 32 (1,0,0,0,0,0)  R63-M0-N06-J12 )
rank 0000007 location <00,00,01,01,01> core 00 hwthread 0 procid = 00 ( Task 7 of 32 (0,0,1,1,1,0)  R63-M0-N06-J27 )
rank 0000021 location <01,00,01,00,01> core 00 hwthread 0 procid = 00 ( Task 21 of 32 (1,0,1,0,1,0)  R63-M0-N06-J10 )
rank 0000013 location <00,01,01,00,01> core 00 hwthread 0 procid = 00 ( Task 13 of 32 (0,1,1,0,1,0)  R63-M0-N06-J20 )
rank 0000025 location <01,01,00,00,01> core 00 hwthread 0 procid = 00 ( Task 25 of 32 (1,1,0,0,1,0)  R63-M0-N06-J08 )
rank 0000001 location <00,00,00,00,01> core 00 hwthread 0 procid = 00 ( Task 1 of 32 (0,0,0,0,1,0)  R63-M0-N06-J22 )
rank 0000027 location <01,01,00,01,01> core 00 hwthread 0 procid = 00 ( Task 27 of 32 (1,1,0,1,1,0)  R63-M0-N06-J04 )
rank 0000012 location <00,01,01,00,00> core 00 hwthread 0 procid = 00 ( Task 12 of 32 (0,1,1,0,0,0)  R63-M0-N06-J19 )
rank 0000019 location <01,00,00,01,01> core 00 hwthread 0 procid = 00 ( Task 19 of 32 (1,0,0,1,1,0)  R63-M0-N06-J07 )
rank 0000005 location <00,00,01,00,01> core 00 hwthread 0 procid = 00 ( Task 5 of 32 (0,0,1,0,1,0)  R63-M0-N06-J23 )
rank 0000029 location <01,01,01,00,01> core 00 hwthread 0 procid = 00 ( Task 29 of 32 (1,1,1,0,1,0)  R63-M0-N06-J09 )
rank 0000010 location <00,01,00,01,00> core 00 hwthread 0 procid = 00 ( Task 10 of 32 (0,1,0,1,0,0)  R63-M0-N06-J30 )
rank 0000031 location <01,01,01,01,01> core 00 hwthread 0 procid = 00 ( Task 31 of 32 (1,1,1,1,1,0)  R63-M0-N06-J05 )
...
```

shape (A,B,C,D,E)

TORUS enabled (A,B,C,D,E)

Setup: 32 nodes, -np 32 -p 1

Coordinates of I/O bridge nodes A,B,C,D,E

MPI ranks

Coordinates A,B,C,D,E / A,B,C,D,E,T

Mapping

Changing the Mapping

Pre-defined mappings

Command `runjob --mapping <permutation of ABCDET>`

The rightmost coordinate is varied fastest. If no mapping is specified, the default is `ABCDET`, i.e., the processes are mapped first within one node before occupying the next node, etc.

User-defined mappings

Command `runjob --mapping <mapfile>`

`<mapfile>` is an ASCII file, where the n^{th} line specifies the 6 coordinates of the MPI rank *n*:

```
4 0 0 0 0 0 # task 0
0 0 0 2 0 0 # task 1
0 0 1 0 0 0 # task 2
[...]
```

Mapping

Hints for choosing an optimal mapping

- Can increase performance considerably
- Depends on the communication pattern of applications
- Ranks communicating with each other should be close
- Folding of dimensions according to decomposition

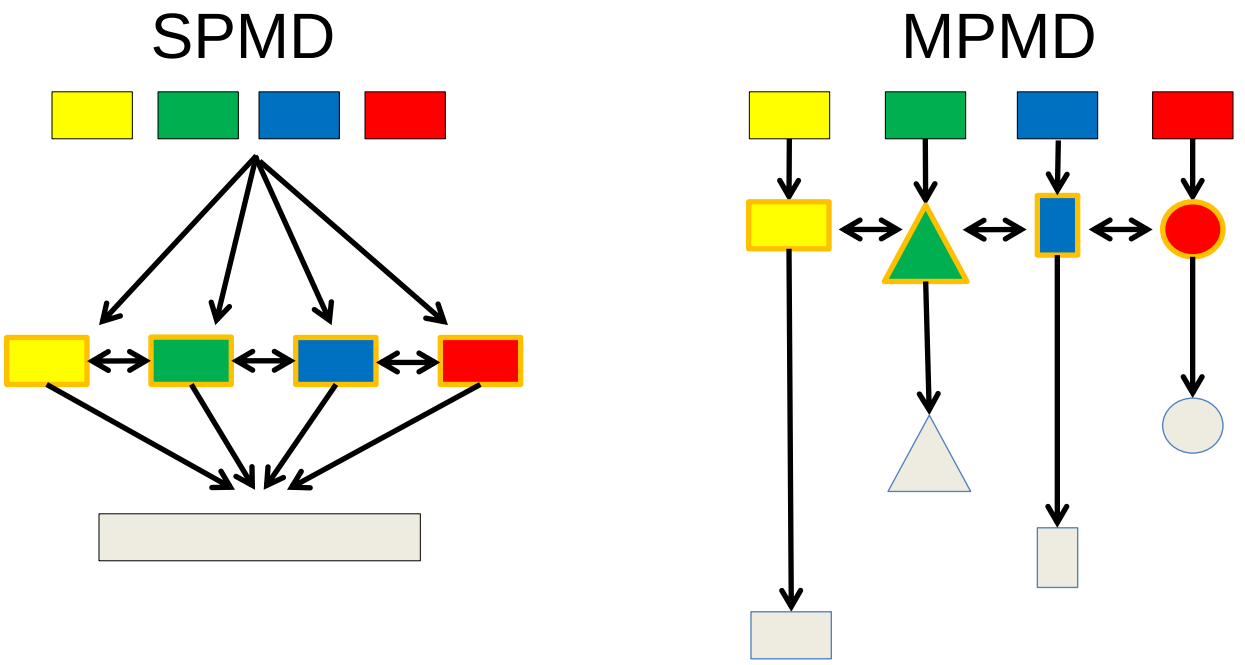
Example

Job size: 1 midplane with 16 MPI ranks/node	
Mapping `ABCDET`	Mapping `TEDCBA`
Dimensions: `4×4×4×4×2×16`	Dimensions: `16×2×4×4×4×4`

• Good for simulations with, e.g., following 2D decompositions:
`256×32 = (4⁴)×(2×16)`
`64×128 = (4³)×(4×2×16)`
	• Good for simulations with a 2D decomposition of
	`128×64 = (16×2×4)×4³`

MPMD – Multiple Program Multiple Data

Instead of using only one executables several different executables can be used at the same time sharing one JUQUEEN partition



MPMD job execution – Job command file

Example

MPMD setup

app1.x: 4 MPI processes

app2.x: 4 MPI processes

app3.x: 8 MPI processes

Job command file

```
#@job_name          = MPMD
#@comment           = "MPMD Example"
#@environment       = COPY_ALL
#@job_type          = bluegene
#@notification      = never
#@bg_size           = 32
#@bg_connectivity   = torus
#@wall_clock_limit  = 00:30:00
#@queue

runjob -n 16 -p 4 –mapping mpmd_mapfile : dummy.x
```

Mapfile for MPMD – Section I

Example

MPMD setup

app1.x: 4 MPI processes

app2.x: 4 MPI processes

app3.x: 8 MPI processes

mpmd_mapfile (ASCII file) – Section I

```
#mpmdbegin 0-3
#mpmdcmd app1.x
#mpmdend
```

app1.x with rank 0 to 3

```
#mpmdbegin 4-7
#mpmdcmd app2.x
#mpmdend
```

app2.x with rank 4 to 7

```
#mpmdbegin 8-15
#mpmdcmd app3.x
#mpmdend
```

app3.x with rank 8 to 15

Mapfile for MPMD – Section II

Example

MPMD setup

app1.x: 4 MPI processes

app2.x: 4 MPI processes

app3.x: 8 MPI processes

mpmd_mapfile (ASCII file) – Section II positions in the 6D torus (ABCDET)

```
0 0 0 0 0 0
0 0 0 0 0 1
0 0 0 0 0 2
0 0 0 0 0 3

0 0 0 0 1 0
0 0 0 0 1 1
0 0 0 0 1 2
0 0 0 0 1 3

0 0 0 1 0 0
0 0 0 1 0 1
0 0 0 1 0 2
0 0 0 1 0 3
0 0 1 0 0 0
0 0 1 0 0 1
0 0 1 0 0 2
0 0 1 0 0 3
```

`app1.x` with rank 0 to 3,
first node, core 0 to 3

`app2.x` with rank 4 to 7,
second node in E, core 0 to 3

`app3.x` with rank 8 to 15,
first node in D and C,
cores 0 to 3 on each node

MPMD – important hints

- On one node only one executable can run.
- The number of ranks specified in the `runjob` command (`-p` or `--ranks-per-node`) must be the largest number of ranks per node needed.
- The memory available to an MPI task depends on the number of ranks per node specified in the `runjob` command (`-p 8` → 2 GB per rank, regardless how many ranks per node are actually started).
- The executable specified in the `runjob` command is a dummy argument

Obtaining information about the allocated partition and the distribution of processes

http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/ShapeMapping.html#doc1413730bodyText4

MPI extensions for Blue Gene/Q

IBM offers extensions to the MPI standard for Blue Gene/Q

- *Not* part of the official MPI standard!
- C (and Fortran77 interface for most functions)
- Functions start with `MPIX_` instead of `MPI_`

Overview over all available extensions

http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/MPIextensions.html

C	`#include <mpix.h>`
Fortran	`include 'mpif.h'`

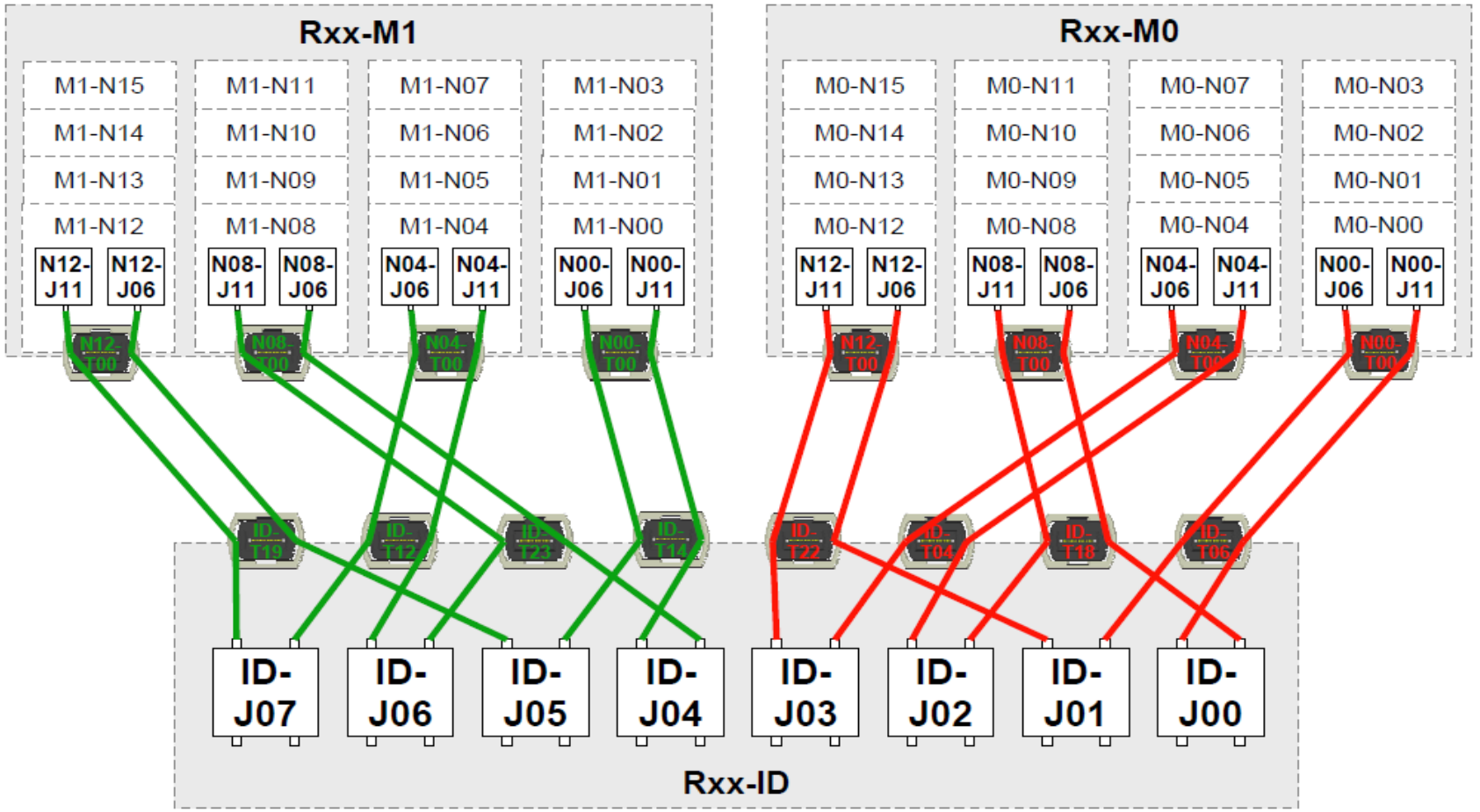
MPI extensions for Blue Gene/Q

C

```
MPIX_Hardware(MPIX_Hardware_t *hw)

typedef struct
{
  unsigned prank;                          // Physical rank of node
  unsigned psize;                          // Size of partition
  unsigned ppn;                            // Processes per node
  unsigned coreID;                         // Process ID
  unsigned clockMHz;                       // Frequency in MHz
  unsigned memSize;                        // Memory in MB
  unsigned torus_dimension;                // Actual torus dimension
  unsigned Size[MPIX_TORUS_MAX_DIMS];      // Max. torus dimensions
  unsigned Coords[MPIX_TORUS_MAX_DIMS];    // Node's coordinated
  unsigned isTorus[MPIX_TORUS_MAX_DIMS]; // Wrap-around dims?
  unsigned rankInPset;                     // Zero on Blue Gene/Q
  unsigned sizeOfPset;                     // Zero on Blue Gene/Q
  unsigned idOfPset;                       // Zero on Blue Gene/Q
} MPIX_Hardware_t;
```

Blue Gene/Q: I/O Node Cabling (8 ION/Rack)



MPI BGQ Extensions – I/O related functions

C

```
int MPIX_Pset_diff_comm_create(MPI_Comm *pset_comm_diff)
int MPIX_Pset_same_comm_create(MPI_Comm *pset_comm_same)
```

Fortran

```
MPIX_PSET_DIFF_COMM_CREATE(PSET_COMM_DIFF, IERROR)
MPIX_PSET_SAME_COMM_CREATE(PSET_COMM_SAME, IERROR)

  INTEGER PSET_COMM_DIFF, PSET_COMM_SAME, IERROR
```

- Collective call on `MPI_COMM_WORLD`
- Returns a communicator which contains only MPI ranks which run on nodes belonging to different/same I/O Bridge Nodes
- The name of this function is chosen for backwards compatibility, since there are no psets on the Blue Gene/Q anymore

MPI BGQ Extensions – I/O related functions

C

```
int MPIX_IO_node_id()
```

Fortran

```
MPIX_IO_NODE_ID(IO_NODE_ID)
  INTEGER    :: IO_NODE_ID
```

- Returns the ID of the associated I/O node

! **NO** `IERROR` parameter for Fortran !

MPI BGQ Extensions – I/O related functions

C

```
int MPIX_IO_distance()
```

Fortran

```
MPIX_IO_DISTANCE(IO_DISTANCE)
  INTEGER    :: IO_DISTANCE
```

- Returns the distance to the associated I/O node in number of hops

! **NO** `IERROR` parameter for Fortran !

MPI BGQ Extensions – I/O related functions

C

```
int MPIX_IO_link_id()
```

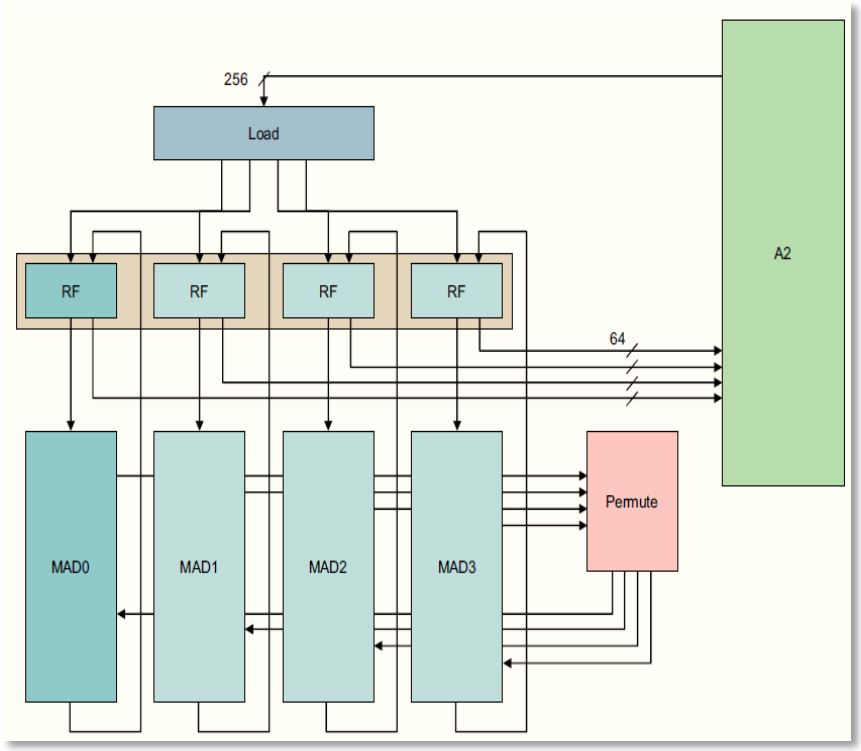
Fortran

```
MPIX_IO_LINK_ID(IO_LINK_ID)
  INTEGER    :: IO_LINK_ID
```

- Returns the ID of the link to the associated I/O node

! **NO** `IERROR` parameter for Fortran !

Quad–processing extension (QPX)



- Allows to perform 8 concurrent floating point operations (fused multipy-add, FMA) + load + store
- Four double precision pipelines, usable as
  - Scalar FPU
  - 4-wide double SIMD
  - 2-wide complex SIMD

IBM XL compiler support for QPX

Usage of QPX

- Compiler flag `-qsimd=auto`
- Check that SIMD vectorization is actually done!
  - `-qreport`
  - `-qlist`

```
>>>> LOOP TRANSFORMATION SECTION <<<<
[...]
-----------------------------------
0 9 1     Loop with nest-level 1 and
          iteration count 1000 was
          SIMD vectorized
[...]
```

```
>>>> LOOP TRANSFORMATION SECTION <<<<
[...]
-----------------------------------
0 9 1     Loop was not SIMD vectorized
          because the loop is not the
          innermost loop.
0 10 1    Loop was not SIMD vectorized
          because it contains memory
          references with non-
          vectorizable alignment.
```

IBM XL compiler support for QPX

Provide hints for the compiler

- Point to compiler likely iteration counts
- Instruct compiler to align fields
- Tell that FORTRAN assumed-shape arrays are contiguous
  `-qassert=contig`

Fortran

```
real*8 :: x(:),y(:),a

!ibm* align(32, x, y)
!ibm* assert(itercnt(100))

do i=m, n
  z(i) = x(i) + a*y(i)
enddo
```

C/C++

```
double __align(32) *x, *y;
double a;
#pragma disjoint(*x, *y)
#pragma disjoint(*x, a)
#pragma ibm iterations(100)
for (int i=m;i<n;i++)
  z[i] = x[i] + a*y[i]
void foo(double* restrict a1,
         double* restrict a2){
for (int i=0; i<n; i++)
a1[i]=a2[i];}
```

IBM XL compiler QPX intrinsics

XL intrinsics		
	New intrinsic variable type	
	C/C++	`vector4double`
	Fortran	`vector(real(8))`
	Wide set of elemental functions available	
	LOAD, STORE, MULT, MULT-ADD, ROUND, CEILING, SQRT, …	

Strengths:

- User may layout calculation by hand, if compiler not smart enough

Easy to use:

- Leave stack, register layout, load/store scheduling to compiler

QPX example using compiler intrinsics

```
…
typedef vector4double qv;
qv dx,dy,dz,dx2,dy2,dz2
for (i=0;i<4;i++)
{
…
  xd[i] = xdipl[j];
  yd[i] = ydipl[j];
  zd[i] = zdipl[j];
}
dx2 = vec_mul(dx,dx);
dy2 = vec_mul(dy,dy);
dz2 = vec_mul(dz,dz);

d = vec_swsqrt(dx2+dy2+dz2);
…
```

Source: IBM Corporation