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

<table>
<thead>
<tr>
<th>Modules – Features</th>
<th>Overview of available products</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Provides fast overview over all available products</td>
</tr>
<tr>
<td></td>
<td>• Provides fast overview over all available versions of a selected product</td>
</tr>
<tr>
<td>Use of software and libraries</td>
<td>• Enables access to software packages and libraries</td>
</tr>
<tr>
<td></td>
<td>• Supports different versions of one product side by side</td>
</tr>
<tr>
<td></td>
<td>• Provides application-specific information</td>
</tr>
<tr>
<td>Dynamic modification of user environment</td>
<td>• Necessary environment variables are set appropriately, no need to set variables manually</td>
</tr>
<tr>
<td></td>
<td>• Detection of conflicts between applications</td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>COMPILER</th>
<th>Software for compute nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO</td>
<td>/bgsys/local</td>
</tr>
<tr>
<td>MATH</td>
<td></td>
</tr>
<tr>
<td>MISC</td>
<td></td>
</tr>
<tr>
<td>SCIENTIFIC</td>
<td></td>
</tr>
<tr>
<td>TOOLS</td>
<td>/usr/local</td>
</tr>
</tbody>
</table>

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 (0.85)
- FFTW (2.1.5/3.3.3)
- GSL (1.15)
- hypre (2.9.0)
- LAPACK (3.6.0)
- MUMPS (5.0.0)
- P3DFFT (2.5.1)
- ParMETIS (4.0.2)
- PETSc (3.6.0)
- ScaLAPACK (2.0.2)
- SPRNG (2.0)
- sundials (2.6.2)

### Science
- cp2k (3.0.0)
- cpmd (3.15.3)
- lammmps (15MAY15)
- NAMD (2.9)
- OpenFOAM (2.1.1)
- QE (5.0.1)

### I/O libraries and tools
- cdo (1.6.1)
- darshan (2.3.1)
- HDF5 (1.8.15)
- Nco (4.3.4)
- netCDF (4.3.3)
- p-netCDF (1.6.0)
- SIONlib (1.6.1)

### Tools
- extrae (3.3.0)
- MUST (1.2.1)
- Scalasca (2.3)
- Scorep (2.0.1)
- Tau (2.25)
- Totalview (8.15.4)
- Vampir (8.4.1)

### Compiler (tools)
- cmake (2.8.11)
- clang (3.7)
- gcc (4.9.3)
- bgxlc (12.01)
- bgxlf (14.01)

- Not via modules
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

<table>
<thead>
<tr>
<th>Job command file</th>
<th>Keyword block</th>
<th>Shell script block</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• LoadLeveler keywords have the form</td>
<td></td>
</tr>
<tr>
<td></td>
<td>#@&lt;keyword&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• # and @ can be separated by any number of blanks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Regular shell script</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Can contain any shell command</td>
<td></td>
</tr>
</tbody>
</table>
# Job execution – LoadLeveler keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#@job_name=&lt;name&gt;</code></td>
<td>Name of the job</td>
</tr>
<tr>
<td><code>#@notification=</code></td>
<td>Send notification if the job is finished</td>
</tr>
<tr>
<td><code>end</code></td>
<td>if the job returned an error code ≠ 0</td>
</tr>
<tr>
<td><code>error</code></td>
<td>never</td>
</tr>
<tr>
<td><code>never</code></td>
<td>upon the start of the job combination of <code>start,end,error</code></td>
</tr>
<tr>
<td><code>start</code></td>
<td></td>
</tr>
<tr>
<td><code>always</code></td>
<td></td>
</tr>
<tr>
<td><code>#@notify_user=&lt;mailaddr&gt;</code></td>
<td>Mail address to send messages to</td>
</tr>
<tr>
<td><code>#@wall_clock_limit=hh:mm:ss</code></td>
<td>Requested wall time for the job</td>
</tr>
<tr>
<td><code>#@input=&lt;input file name&gt;</code></td>
<td>Specifies corresponding file names</td>
</tr>
<tr>
<td><code>#@output=&lt;file name for stdout&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>#@error=&lt;file name for stderr&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>#@environment=[&lt;variable&gt;,COPY_ALL]</code></td>
<td>Environment variable to be exported to job</td>
</tr>
<tr>
<td><code>#@queue</code></td>
<td>End of keyword section</td>
</tr>
</tbody>
</table>
## Job execution – LoadLeveler keywords

| Keyword                                      | Description                                                                                                                                 |
|----------------------------------------------|---------------------------------------------------------------- Adam's lib                                                                                                                                 |
| `@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]`       | Type of wiring requested for the block (can be specified for each dimension separately)                                           |

Xa Xb Xc Xd
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

```bash
@@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 pipeline 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 ≤ n ≤ 6

Test which configuration gives the best performance for your application and setup!
Job execution – Job classes

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

* 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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>llsubmit &lt;jobfile&gt;</code></td>
<td>Sends job to the queuing system</td>
</tr>
<tr>
<td><code>llq</code></td>
<td>Lists all queued and running jobs</td>
</tr>
<tr>
<td><code>llq -l &lt;job ID&gt;</code></td>
<td>detailed information about the specified job</td>
</tr>
<tr>
<td><code>llq -s &lt;job ID&gt;</code></td>
<td>detailed information about a specific queued job, e.g. expected start time</td>
</tr>
<tr>
<td><code>llq -u &lt;user&gt;</code></td>
<td>lists all jobs of the specified user</td>
</tr>
<tr>
<td><code>llcancel &lt;job ID&gt;</code></td>
<td>Kills the specified job</td>
</tr>
<tr>
<td><code>llstatus</code></td>
<td>Displays the status of LoadLeveler</td>
</tr>
<tr>
<td><code>llclass</code></td>
<td>Lists existing classes and their properties</td>
</tr>
<tr>
<td><code>llqx</code></td>
<td>Shows detailed information about all jobs</td>
</tr>
</tbody>
</table>
Job execution – Monitoring of jobs

LoadLeveler

- Command `llq [options]`

Ilview

- 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 – Ilview
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

<table>
<thead>
<tr>
<th>Language</th>
<th>XL</th>
<th>GNU</th>
<th>Clang</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>xlc</td>
<td>gcc</td>
<td>---</td>
</tr>
<tr>
<td>C++</td>
<td>xlc++, xlC</td>
<td>g++</td>
<td>---</td>
</tr>
<tr>
<td>Fortran</td>
<td>xlf, xlf90, xlf95, xlf2003</td>
<td>gfortran</td>
<td>---</td>
</tr>
</tbody>
</table>
## Compilers and wrappers

<table>
<thead>
<tr>
<th>Compute nodes</th>
<th>Language</th>
<th>Compiler</th>
<th>MPI wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>XL (thread-safe: add -r)</td>
<td>C</td>
<td>bgxlc, bgc89, bgc99</td>
<td>mpixlc</td>
</tr>
<tr>
<td></td>
<td>C++</td>
<td>bgxlc++, bgxLC</td>
<td>mpixlcxx</td>
</tr>
<tr>
<td></td>
<td>Fortran</td>
<td>bgxlf, bgxlf90, bgxlf95, bgxlf2003</td>
<td>mpixlf77, mpixlf90, mpixlf95, mpixlf2003</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>powerpc64-bgq-linux-gcc</td>
<td>mpiGCC</td>
</tr>
<tr>
<td></td>
<td>C++</td>
<td>powerpc64-bgq-linux-g++</td>
<td>mpiGCC++</td>
</tr>
<tr>
<td></td>
<td>Fortran</td>
<td>powerpc64-bgq-linux-gfortran</td>
<td>mpiGFortran</td>
</tr>
<tr>
<td>GNU</td>
<td>C</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>C++</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>Fortran</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Clang</td>
<td>C</td>
<td>bgclang</td>
<td>mpiclang</td>
</tr>
<tr>
<td></td>
<td>C++</td>
<td>bgclang++</td>
<td>mpiclang++</td>
</tr>
<tr>
<td></td>
<td>Fortran</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

- **!** 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

<table>
<thead>
<tr>
<th>&lt;name&gt;</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>MPI library compiled with gcc, fine-grained locking, error checking and assertions enabled</td>
<td></td>
</tr>
<tr>
<td>gcc.legacy</td>
<td>MPI library compiled with gcc, coarse-grained locking, error checking and assertions enabled $\rightarrow$ slightly better latency for single-threaded codes</td>
<td>Initial porting</td>
</tr>
<tr>
<td>xl</td>
<td>MPI library compiled with XL compilers, PAMI compiled with gcc, error checking and assertions enabled (default wrappers)</td>
<td></td>
</tr>
<tr>
<td>xl.legacy</td>
<td>MPI library compiled with XL compilers, PAMI compiled with gcc, coarse-grained locking, error checking and assertions enabled</td>
<td>Correctly running production code</td>
</tr>
<tr>
<td>xl.ndebug</td>
<td>MPI library compiled with XL compilers, PAMI compiled with gcc, fine-grained locking, error checking and assertions disabled</td>
<td></td>
</tr>
<tr>
<td>xl.legacy.ndebug</td>
<td>MPI library compiled with XL compilers, PAMI compiled with gcc, coarse-grained locking, error checking and assertions disabled</td>
<td></td>
</tr>
</tbody>
</table>

Location of wrappers: `/bgsys/drivers/ppcfloor/comm/<name>/bin`
### XL: basic compiler and linker options

<table>
<thead>
<tr>
<th>Optimization Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-O2 -qarch=qp -qtune=qp</code></td>
<td>Basic optimization</td>
</tr>
<tr>
<td><code>-O3 -qstrict -qarch=qp -qtune=qp</code></td>
<td>More aggressive, not impact on acc.</td>
</tr>
<tr>
<td><code>-O3 -qhot -qarch=qp -qtune=qp</code></td>
<td>More aggressive, may influence acc. (high-order transformations of loops)</td>
</tr>
<tr>
<td><code>-O4 -qarch=qp -qtune=qp</code></td>
<td>Interprocedural optimization at compile time</td>
</tr>
<tr>
<td><code>-O5 -qarch=qp -qtune=qp</code></td>
<td>Interprocedural optimization at link time, whole program analysis</td>
</tr>
</tbody>
</table>

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

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

**More information about using ESSL**

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

<table>
<thead>
<tr>
<th>Compiler/linker flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qinline=auto:level=&lt;number&gt;</td>
<td>Function inlining 0 &lt;= &lt;number&gt; &lt;= 10</td>
</tr>
<tr>
<td>-qinline+procedure1[:procedure2[:...]]</td>
<td></td>
</tr>
<tr>
<td>-qunroll[={auto</td>
<td>yes}]</td>
</tr>
<tr>
<td>-qipa[=&lt;suboptions_list&gt;]</td>
<td>Intra/inter-procedural optimization, see manual</td>
</tr>
<tr>
<td>-qsimd[=auto</td>
<td>noauto]</td>
</tr>
</tbody>
</table>

*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

Dispatch and User Support

- Applications for accounts (for approved projects)

  Forschungszentrum 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

```bash
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

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)

<table>
<thead>
<tr>
<th>Node location in the torus</th>
<th>&lt;A, B, C, D, E&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process or thread location in the torus</td>
<td>&lt;A, B, C, D, E, T&gt;</td>
</tr>
</tbody>
</table>
Mapping – Receiving partition information

Command

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

Partition Information:
block shape : \textless 2,2,2,2,2\textgreater

TORUS links enabled : \textless 0,0,0,0,1\textgreater

IO Bridge Information:
rank 0000017 of 0000032 location \textless 01,00,00,00,01\textgreater core 00 hthread 0 procid 00 ioBridge
\textless 01,00,00,00,01\textgreater cn ‘R63-M0-N06-J11’ ion ‘R63-ID-J03’
rank 0000023 of 0000032 location \textless 01,00,01,01,01\textgreater core 00 hthread 0 procid 00 ioBridge
\textless 01,00,01,01,01\textgreater cn ‘R63-M0-N06-J06’ ion ‘R63-ID-J01’

Compute Node Information:
rank 0000023 location \textless 01,00,01,01,01\textgreater core 00 hthread 0 procid = 00 ( Task 23 of 32 (1,0,1,1,0) R63-M0-N06-J06 )
rank 0000000 location \textless 00,00,00,00,00\textgreater core 00 hthread 0 procid = 00 ( Task 0 of 32 (0,0,0,0,0,0) R63-M0-N06-J17 )
rank 0000016 location \textless 01,00,00,00,00\textgreater core 00 hthread 0 procid = 00 ( Task 16 of 32 (1,0,0,0,0,0) R63-M0-N06-J12 )

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 2 0 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

<table>
<thead>
<tr>
<th>Example</th>
<th>Job size: 1 midplane with 16 MPI ranks/node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td>ABCDE</td>
</tr>
<tr>
<td>Dimensions: 4×4×4×4×2×16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Good for simulations with, e.g., following 2D decompositions: 256×32 = (4⁴)×(2×16)</td>
</tr>
<tr>
<td>64×128 = (4³)×(4×2×16)</td>
<td>256×32 = (4⁴)×(2×16)</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Example</th>
<th>MPMD setup</th>
<th>mpmd_mapfile (ASCII file) – Section I</th>
</tr>
</thead>
</table>
|         | app1.x: 4 MPI processes | #mpmdbegin 0-3
#mpmdcmd app1.x
#mpmdend |
|         | app2.x: 4 MPI processes | #mpmdbegin 4-7
#mpmdcmd app2.x
#mpmdend |
|         | app3.x: 8 MPI processes | #mpmdbegin 8-15
#mpmdcmd app3.x
#mpmdend |
|         | app1.x with rank 0 to 3 | app1.x with rank 0 to 3 |
|         | app2.x with rank 4 to 7 | app2.x with rank 4 to 7 |
|         | app3.x with rank 8 to 15 | app3.x with rank 8 to 15 |
# Mapfile for MPMD – Section II

## 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)

<table>
<thead>
<tr>
<th>Example</th>
<th>app1.x with rank 0 to 3, first node, core 0 to 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>app2.x with rank 4 to 7, second node in E, core 0 to 3</td>
</tr>
<tr>
<td></td>
<td>app3.x with rank 8 to 15, first node in D and C, cores 0 to 3 on each node</td>
</tr>
</tbody>
</table>

| 0 0 0 0 0 0 | 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 0 1 0 |
| 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 |
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

<table>
<thead>
<tr>
<th>C</th>
<th><code>#include &lt;mpix.h&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td><code>include 'mpif.h'</code></td>
</tr>
</tbody>
</table>
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;
```
MPI BGQ Extensions – I/O related functions

- 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong> int MPIX_IO_node_id()</td>
<td>Returns the ID of the associated I/O node</td>
</tr>
<tr>
<td><strong>Fortran</strong></td>
<td><code>MPIX_IO_NODE_ID(IO_NODE_ID)</code></td>
</tr>
<tr>
<td></td>
<td><code>INTEGER :: IO_NODE_ID</code></td>
</tr>
</tbody>
</table>

- Returns the ID of the associated I/O node

**Note:** No `IERROR` parameter for Fortran
# MPI BGQ Extensions – I/O related functions

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
</table>
| 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

**Note:** No `IERROR` parameter for Fortran
MPI BGQ Extensions – I/O related functions

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

C

```c
int MPIX_IO_link_id()
```

Fortran

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

- **NO IERROR parameter for Fortran**
Quad–processing extension (QPX)

- Allows to perform 8 concurrent floating point operations (fused multiply-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`

```
0 9 1 Loop with nest-level 1 and iteration count 1000 was SIMD vectorized

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

\texttt{\textbf{-qassert=contig}}

\begin{verbatim}
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
\end{verbatim}

\begin{verbatim}
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];}
\end{verbatim}
IBM XL compiler QPX intrinsics

<table>
<thead>
<tr>
<th>XL intrinsics</th>
<th>New intrinsic variable type</th>
<th>Easy to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++</td>
<td>vector4double</td>
<td>Leave stack, register layout, load/store scheduling to compiler</td>
</tr>
<tr>
<td>Fortran</td>
<td>vector(real(8))</td>
<td></td>
</tr>
</tbody>
</table>

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

```c
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