JUQUEEN
Best Practices

4. Februar 2013 | Florian Janetzko
Outline

Production Environment
  - Module Environment
  - Job Execution

Basic Porting
  - Compilers and Wrappers
  - Compiler Flags

Tuning Applications
  - Advanced Compiler Flags
  - Runtime Environment
  - MPI Extensions
  - QPX
  - Thread-Level Speculation, Transactional Memory
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
Module Environment

Module concept

- Provides overview over available software packages
- Eases use of software packages
  - Access to software packages, libraries
  - Supply of different versions of applications
  - Supply of application-specific information
- Enables dynamic modification of users’ environment
  - Environment variables (PATH, LD_LIBRARY_PATH, MANPATH, …) are set appropriately
  - Detection of conflicts between applications
# Module Environment

```
$ module <options> <module>
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;no option&gt;</td>
<td>Lists available options of the module command</td>
</tr>
<tr>
<td>avail</td>
<td>Lists all available modules</td>
</tr>
<tr>
<td>list</td>
<td>Lists modules currently loaded</td>
</tr>
<tr>
<td>load</td>
<td>Loads a module</td>
</tr>
<tr>
<td>unload</td>
<td>Unloads a module</td>
</tr>
<tr>
<td>help</td>
<td>Lists information about a module</td>
</tr>
<tr>
<td>show</td>
<td>Information about settings done by the module</td>
</tr>
<tr>
<td>purge</td>
<td>Unloads all modules</td>
</tr>
</tbody>
</table>
Module Environment

Six module categories

- **COMPILER**
  - Different compilers and versions of compilers
- **IO**
  - I/O libraries and tools
- **MATH**
  - Mathematical libraries and software packages
- **MISC**
  - Software not fitting into another category
- **SCIENTIFIC**
  - Software packages from different scientific fields
- **TOOLS**
  - Performance analysis, debugger, etc.

Software for
Compute Nodes: `/bgsys/local`
Front-end Nodes: `/usr/local`
Module Environment – Applications & Libraries

Mathematical applications and libraries

- arpack (2.1)
- gsl (1.15)
- mumps (4.10.0)
- scalapack (2.0.1)
- fftw (2.1.5,3.3.2)
- hypre (2.8.0)
- parmetis (3.2.0,4.0.2)
- sprng (1.0, 2.0)
- gmp (5.0.5)
- lapack (3.3.0)
- petsc (3.3)
- sundials (2.5.0)

Scientific applications

- CPMD (3.15.1)
- Gromacs (4.5.5)
- OpenFOAM*
- CP2K (2.2.12394)
- Lammmps (5May12,30Aug12)
- QuantumEspresso*
- GPAW*
- Namd (2.8, 2.9)
- VASP**

* In preparation
** Software not installed but makefiles are available
Module Environment – Applications & Libraries

I/O libraries
- HDF5***
- netCDF***
- SIONlib***

Tools
- Cmake (2.8.8)
- hpctoolkit (5.2.1)
- Tau (2.21.2, 2.21.3)
- DDT*
- PAPI (4.4.0)
- Totalview (8.11.0)
- Extrae (2.2.1)
- Scalasca (1.4.2)
- Darshan (2.2.4)

* In preparation
*** Software installed, module files to be installed

Talk:
- Wolfgang Frings, JSC Parallel I/O
  5. February 10:15

Talk:
- Markus Geimer, JSC Performance tools and debuggers
  4. February 11:00
Outline

Production Environment

- Module Environment
- Job Execution

Basic Porting

- Compilers and Wrappers
- Compiler Flags

Tuning Applications

- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
LoadLeveler Batch System – Commands

Execution of applications managed by LoadLeveler
http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/UserInfo/LoadLeveler.html

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

ASCII file containing two major parts

1. LoadLeveler job keywords block at the beginning of a file
   - LoadLeveler keywords have the form
     \#@<keyword>
   - # and @ can be separated by any number of blanks

2. One or more application script blocks
   - Regular shell script
   - Can contain any shell command
### LoadLeveler – Standard 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</td>
</tr>
<tr>
<td><code>  end</code></td>
<td>if the job is finished</td>
</tr>
<tr>
<td><code>  error</code></td>
<td>if the job returned an error code ≠ 0</td>
</tr>
<tr>
<td><code>  never</code></td>
<td>never</td>
</tr>
<tr>
<td><code>  start</code></td>
<td>upon the start of the job</td>
</tr>
<tr>
<td><code>  always</code></td>
<td>combination of <code>start, end, error</code></td>
</tr>
<tr>
<td><code>#@notify_use=&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>Queue job</td>
</tr>
</tbody>
</table>
## LoadLeveler – Blue Gene/Q Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@job_type=[serial, bluegene]</code></td>
<td>Specifies the type of job step to process. Must be set to <code>bluegene</code> for parallel applications.</td>
</tr>
<tr>
<td><code>@bg_size=&lt;number of nodes&gt;</code></td>
<td>Size of the Blue Gene job, keywords <code>bg_size</code> and <code>bg_shape</code> are mutually exclusive.</td>
</tr>
<tr>
<td><code>@bg_shape=&lt;A&gt;x&lt;B&gt;x&lt;C&gt;x&lt;D&gt;</code></td>
<td>Specifies the requested shape of a job. The max. shape on JUQUEEN is 2x2x2x2.</td>
</tr>
<tr>
<td><code>@bg_rotate=[True,False]</code></td>
<td>whether the scheduler should consider all possible rotations of the given shape</td>
</tr>
<tr>
<td><code>@bg_connectivity=[TORUS,MESH,EITHER] Xa Xb Xc Xd</code></td>
<td>Type of wiring requested for the block (can be specified for each dimension separately)</td>
</tr>
</tbody>
</table>
### LoadLeveler – 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>m048*</td>
<td>16385 – 24576</td>
<td>24:00:00</td>
<td>06:00:00</td>
</tr>
<tr>
<td>m056*</td>
<td>24577 – 28672</td>
<td>24:00:00</td>
<td>06:00:00</td>
</tr>
</tbody>
</table>

*On demand only

You will be charged for the **full partition** (e.g. if you request 513 nodes you will be charged for **1024** nodes!) ⇒ Always use full partitions!
LoadLeveler – Job Scheduling

Backfill scheduler

- The biggest job has the highest priority (*Top Dog*)
- LoadLeveler fills gaps with smaller, short-running jobs while freeing the system for the Top Dog

Tip: Specify the wall time for your jobs as exact as possible, because jobs requesting a shorter wall time have a better chance to be executed.

Big jobs

- Jobs requesting >16 racks are collected and run in dedicated time slots (e.g. after a maintenance) at least once a week
Running Simulations – runjob Command

Launch command for parallel applications

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--args &lt;prg_arg&gt;</td>
<td>Passes &quot;prg_arg&quot; to the launched application on the compute node.</td>
</tr>
<tr>
<td>--exe &lt;executable&gt;</td>
<td>Specifies the full path to the executable</td>
</tr>
<tr>
<td>--exp &lt;ENV_Var=Value&gt;</td>
<td>Sets the environment variable ENV_Var=Value</td>
</tr>
<tr>
<td>--exp-env &lt;ENV_Var&gt;</td>
<td>Sets the environment variable ENV_Var</td>
</tr>
<tr>
<td>--np &lt;number&gt;</td>
<td>Total number of (MPI) tasks</td>
</tr>
<tr>
<td>--ranks-per-node &lt;number&gt;</td>
<td>Number of (MPI) tasks per compute node</td>
</tr>
</tbody>
</table>
Running Simulations – MPI/OpenMP Codes

• On Blue Gene/P
  – Three modes were available
    1. VN mode (4 MPI tasks, no thread per task)
    2. DUAL mode (2 MPI tasks with 2 OpenMP threads each)
    3. SMP mode (1 MPI task with 4 OpenMP threads)

• On Blue Gene/Q
  – One node has 16 cores with 4-way SMT each
  – Several configurations possible
    • $\text{ntasks} \times \text{nthreads} = 64$
    • $\text{ntasks} = 2^n, 0 \leq n \leq 6$

Test carefully, which configuration gives the best performance for your application and setup!
LoadLeveler – Example Job Command File

```bash
#@job_name = hybrid_code
#@comment = "16x4 configuration"
#@output = test_$(_jobid)_$(stepid).out
#@error = test_$(_jobid)_$(stepid).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\n   --env OMP_NUM_THREADS=4 --exe app.x
```
Filesystems

$HOME (group limits: 6 TB, 2 million files)
- for source code, binaries, libraries and applications
- Automatic backup

$WORK (group limits: 20 TB, 4 million files)
- temporary storage for applications
- No automatic backup, files older than 90 days are deleted automatically

$ARCH (group limits: 2 million files)
- Storage of data during the project’s lifetime
- Use file archives (tar), store files of size 500 – 1000 GB
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
Compilers

- Different compilers for front-end and compute nodes
- GNU and IBM XL family of compilers available

Tip: It is recommended to use the XL suite of compilers for the CN since they produce in general better optimized code.

<table>
<thead>
<tr>
<th>Language</th>
<th>XL compiler</th>
<th>GNU compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>xlc, xlc_r</td>
<td>gcc</td>
</tr>
<tr>
<td>C++</td>
<td>xlc++, xlc++_r, xlC, xlC_r</td>
<td>g++</td>
</tr>
<tr>
<td>Fortran</td>
<td>xlf, xlf90, xlf95, xlf2003</td>
<td>gfortran</td>
</tr>
<tr>
<td></td>
<td>xlf_r, xlf90_r, xlf95_r, xlf2003_r</td>
<td></td>
</tr>
</tbody>
</table>
# Compilers for CN

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler invocation (thread-safe: *_r)</th>
<th>MPI wrapper (thread-safe: *_r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>powerpc64-bgq-linux-gcc</td>
<td>mpiexec</td>
</tr>
<tr>
<td>C++</td>
<td>powerpc64-bgq-linux-g++</td>
<td>mpiexec++</td>
</tr>
<tr>
<td>Fortran</td>
<td>powerpc64-bgq-linux-gfortran</td>
<td>mpiexecfortran</td>
</tr>
</tbody>
</table>

## GNU (GCC)

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler invocation</th>
<th>MPI wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>powerpc64-bgq-linux-gcc</td>
<td>mpiexec</td>
</tr>
<tr>
<td>C++</td>
<td>powerpc64-bgq-linux-g++</td>
<td>mpiexec++</td>
</tr>
<tr>
<td>Fortran</td>
<td>powerpc64-bgq-linux-gfortran</td>
<td>mpiexecfortran</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler invocation</th>
<th>MPI wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>powerpc64-bgq-linux-gcc</td>
<td>mpiexec</td>
</tr>
<tr>
<td>C++</td>
<td>powerpc64-bgq-linux-g++</td>
<td>mpiexec++</td>
</tr>
<tr>
<td>Fortran</td>
<td>powerpc64-bgq-linux-gfortran</td>
<td>mpiexecfortran</td>
</tr>
</tbody>
</table>

## XL Compilers

<table>
<thead>
<tr>
<th>Language</th>
<th>Compiler invocation</th>
<th>MPI wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>bgxlc, bgc89, bgc99</td>
<td>mpiexecl</td>
</tr>
<tr>
<td>C++</td>
<td>bgxlc++, bgxlc</td>
<td>mpiexeclxx</td>
</tr>
<tr>
<td>Fortran</td>
<td>bgxlf, bgxlf90, bgxlf95, bgxlf2003</td>
<td>mpiexeclf77, mpiexeclf90, mpiexeclf95, mpiexeclf2003</td>
</tr>
</tbody>
</table>
## Basic Compiler Options – XL Compilers I

Flags in order of increasing optimization potential

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

#### Additional compiler flags

<table>
<thead>
<tr>
<th>Compiler Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-qsmp=omp</code> <code>-qthreaded</code></td>
<td>Switch on OpenMP support</td>
</tr>
<tr>
<td><code>-qreport</code> <code>-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</code> <code>-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>
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
Diagnostic Compiler Flags (XL Compilers)

Diagnostic messages are given on the terminal and/or in a separate file

- **-qreport**: compilers generate a file `name.lst` for each source file
- **-qlist**: compiler listing including an object listing
- **-qlistopt**: options in effect during compilation included in listing

**Listen to the compiler!**

- **-qflag**=`<listing-severety>:<terminal-severety>`
  - `i`: informal messages, `w`: warning messages, `s`: severe errors
  - *Use* `-qflag=i:i` *to get all information*
- **-qlistfmt**=`(xml|html)`=`<option>`
Example: Compilers Diagnostics

```fortran
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).

```

4. Februar 2013
Single-Core Optimization – Compiler Flags

Take advantage of vector instructions
-qsimd=auto

Function inlining
-qinline=auto:level=5
-qinline+procedure1[:procedure2[:...]]

Aggressive loop analysis and transformations
-qhot=level=[0-2]

Loop unrolling
-qunroll

Intra-/inter-procedural optimization (compiling and linking)
-qipa
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
Tuning Runtime Environment

Network
- Topology on BG/Q: 5D Torus
  \[ A \times B \times C \times D \times E(\times T) \]

Shape
- Extension of a partition in A, B, C, and D direction in terms of midplanes

Mapping
- Assignment of processes to nodes and cores
- Best performance for nearest-neighbor communication
- Processes should be mapped accordingly
  - *Optimal mapping depends on application / communication pattern*
  - *Might be performance critical for jobs sizes > 1 midplane*
Choosing Shape and Mapping

Shape

```bash
#@bg_shape = <AxBxCxD> #JUQUEEN: 2x7x2x2 maximum
#@bg_rotate = False|True
```

Mapping

1. Specified as a permutation of ABCDET (rightmost fastest)
2. Specified via a map file

```bash
1. runjob --mapping ACBDET
2. Runjob --mapping <mapfile>
```

- Default mapping: ABCDET
- Good for 1D communication patterns (communication with task ± 1)
Guidance and Map File

Example for Mapping considerations:
   Job size of 1 midplane with 16 tasks/node
   Default mapping: ABCDET = 4x4x4x4x2x16
   → Good for simulations with a 2D decomposition 256×32 or 64x128
   → For simulations with a 2D decomposition 128x64 chose TEDCBA

A map file is a plane ASCII file
   The $n^{th}$ line contains the coordinate of the $n^{th}$ task
   0 0 0 0 0 0 0 # task 0; coordinates ( 0, 0, 0, 0, 0, 0, 0)
   1 0 0 0 0 0 0 # task 1; coordinates ( 1, 0, 0, 0, 0, 0, 0)
   2 0 0 0 0 0 0 # task 2; coordinates ( 2, 0, 0, 0, 0, 0, 0)
   [...]
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
MPI Tuning – BG/Q Extensions

Blue Gene/Q specific MPI extensions (MPIX)

- Only C/C++ interfaces available, Fortran interface requested
- Include header: `#include <mpix.h>`

Examples

```c
int MPIX_Torus_ndims(int *numdim)
```
Determines the number of physical hardware dimensions

```c
int MPIX_Rank2torus(int rank, int *coords)
```
Returns the physical coordinates of an MPI rank

```c
int MPIX_Torus2rank(int *coords, int *rank)
```
Returns the MPI rank with the physical coordinates specified

```c
int MPIX_Hardware(MPIX_Hardware_t *hw)
```
Returns information about the hardware the application is running on
Example: 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;
    unsigned sizeOfPset;
    unsigned idOfPset;
} MPIX_Hardware_t;
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
Quad Floating Point Extension Unit (QPX)

4 double precision pipelines, usable as:
- scalar FPU
- 4-wide FPU SIMD (Single Instruction Multiple Data)
- 2-wide complex arithmetic SIMD

8 concurrent floating point ops (FMA) + load + store
IBM XL Compiler Support for QPX

Usage of QPX
- Compiler flag \texttt{-qsimd=auto}
- \textbf{Check} that simd vectorization is actually done!
  - \texttt{-qreport}
  - \texttt{-qlist}

>>> LOOP TRANSFORMATION SECTION <<<
[...]
------------------------------------
0 9 1  Loop with nest-level 1 and iteration count 1000 was SIMD vectorized
[...]
------------------------------------
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.
QPX Usage – Hints for the Compiler

Compiler needs hints
- Hint compiler to likely iteration counts
- Instruct compiler to align fields
- Tell that FORTRAN assumed-shape arrays are contiguous

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];
}
QPX Example using Compiler Intrinsics

```c
typedef vector4double qv;
qv dx,dy,dz,dx2,dy2,dz2
for (i=0;i<4;i++)
{
    xd[i] = xdip1[j];
    yd[i] = ydip1[j];
    zd[i] = zdip1[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
Outline

Production Environment
- Module Environment
- Job Execution

Basic Porting
- Compilers and Wrappers
- Compiler Flags

Tuning Applications
- Advanced Compiler Flags
- Runtime Environment
- MPI Extensions
- QPX
- Thread-Level Speculation, Transactional Memory
Thread Level Speculation (TLS)

Parallelize potentially dependent serial fragments
- runtime creates threads for each speculative section
- threads run parallel and commit \textit{in order} if no conflict
- on conflict, all threads except current master is rolled back

Performance governed by tradeoff of overhead and conflict probability

Number of times to try rollback before non-speculative execution can be set

Hardware Limitation: maximum of 16 domains
Thread Level Speculation

Enabling of TLS by compiler flag and pragmas

-qsm=speculative

**Fortran**

```fortran
!SEP$ SPECULATIVE DO
do i = 1, N
    call code_to_be_spec(i)
enddo
!SEP$ END SPECULATIVE DO
!SEP$ SPECULATIVE SECTIONS
call some_code()
!SEP$ SPECULATIVE SECTION
call other_code()
!SEP$ END SPECULATIVE SECTIONS
```

**C/C++**

```c
#pragma speculative for
for (int i=0;i<N;i++) {
    code_to_be_spec(i);
}
#pragma speculative sections
{
    #pragma speculative section
    { some_code(); }
    #pragma speculative section
    { other_code(); }
}
```
Thread Level Speculation

Loop:

Thread 0:

Thread 1:

Thread 2:

Loop finished:

export SE_MAX_NUM_ROLLBACK=N

Talk:
Thilo Maurer, IBM
Memory hierarchy, transactional memory, speculative execution
5. February 09:30
Transactional Memory

- Mechanism to enable atomic operations on arbitrary set of memory locations
- Application needs to allow that transactions commit *out-of order*
- May be used to parallelize workload into collaborative but independent tasks on shared data
- Hardware detects write/read conflicts
- Runtime rolls back on failure
Transactional Memory

Enabling by compiler flag and pragmas

- `qtm`

Identification of atomic code blocks:

### Fortran

```fortran
!$omp parallel
!$omp do private(i)
do i = 1, N
!TM$ TM_ATOMIC SAFE_MODE
    call code_to_be_atomic(i)
!TM$ END TM_ATOMIC
enddo
!$omp end do
!$omp end parallel
```

### C/C++

```c
#pragma omp parallel
{
#pragma omp for
for (int i=0; i<N; i++) {
#pragma TM_ATOMIC
    code_to_be_atomic(i);
}
}
```

```bash
export TM_MAX_NUM_ROLLBACK=N
export TM_REPORT=...
```
User Information and Support

Information about JUQUEEN

- JSC websites at http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/JUQUEEN_node.html

 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

→Talk:
Paul Gibbon, JSC
Support structure at JSC
4. February 12:00