JUQUEEN: Application Stack and Best Practices

Florian Janetzko, Jülich Supercomputing Centre, Institute for Advanced Simulation, Forschungszentrum Jülich
Outline

Supercomputing@JSC
  Jülich Supercomputing Centre
  Supercomputing Systems and User Groups
  HPC Resources and User Support

JUQUEEN – Best Practices
  System Architecture and Configuration
  Production Environment and Application Stack
  Basic Porting
  Tuning Applications
  Performance Analysis
  Debugging
Outline

Supercomputing@JSC
- Jülich Supercomputing Centre
- Supercomputing Systems and User Groups
- HPC Resources and User Support

JUQUEEN – Best Practices
- System Architecture and Configuration
- Production Environment and Application Stack
- Basic Porting
- Tuning Applications
- Performance Analysis
- Debugging
Forschungszentrum Jülich
Jülich Supercomputing Centre (JSC)
Tasks of the JSC

• Operation of the supercomputers for local, national and European scientists.
• User support in programming and optimization of applications
• Support of research groups from different research domains by means of simulation laboratories
• Research & Development work on new computer architectures, algorithms, performance analysis and tools, GRID computing, etc.
• Education and training of users, education of students (bachelor and master courses, PhD programs)
Integration of JSC in Networks and Alliances

Gauss Centre for Supercomputing (GCS)
- Alliance of 3 German centres
  - Höchstleistungsrechenzentrum Stuttgart (HLRS)
  - Jülich Supercomputing Centre (JSC)
  - Leibnitz-Rechenzentrum (LRZ)
- German representative in PRACE

Jülich Aachen Research Alliance (JARA)
- Initiative of RWTH Aachen and Forschungszentrum Jülich

John von Neumann Institute for Computing (NIC)
- Joint foundation of 3 Helmholtz Centres
  - Deutsches Elektronensynchrotron (DESY)
  - Gesellschaft f. Schwerionenforschung (GSI)
  - Jülich Supercomputing Centre (JSC)
National and European Users @ JSC
Supercomputer Systems@JSC: Dual Concept

2004
- IBM Power 4+
- JUMP, 9 TFlop/s

2006-8
- IBM Power 6
- JUMP, 9 TFlop/s

2009
- JUROPA
- 200 TFlop/s
- HPC-FF
- 100 TFlop/s

2012
- JUROPA++
- Cluster, 1-2 PFlop/s
- Booster

2014
- General-Purpose
- File Server
- GPFS, Lustre
- Highly-Scalable
- IBM Blue Gene/P
- JUGENE, 1 PFlop/s
- IBM Blue Gene/L
- JUBL, 45 TFlop/s
- IBM Blue Gene/Q
- JUQUEEN
- 5.7 PFlop/s
Supercomputing Resources

Accessible for European users via PRACE

- Computational time
- Enabling work
  - Porting of codes
  - Performance optimization
  - Workflow setup/optimization

JUROPA
- Calls twice a year
- Combined work (enabling + prod.) possible

JUQUEEN
- Regular calls twice a year
- Preparatory access (continuous call)

http://www.prace-project.eu/Call-Announcements
User Support @ JSC - Overview

Cross-Sectional Teams
- Methods & Algorithms
- Application Optimization
- Parallel Performance

Simulation Laboratories
- Biology
- Climate Science
- Molecular Systems
- Plasma Physics
- Solid & Fluid Engineering
User Information and Support

Information about HPC system hosted by JSC

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
Outline

Supercomputing@JSC
  Jülich Supercomputing Centre
  Supercomputing Systems and User Groups
  HPC Resources and User Support

JUQUEEN – Best Practices
  System Architecture – Challenges for Applications
  Production Environment and Application Stack
  Basic Porting
  Tuning Applications
  Performance Analysis
  Debugging
Literature


• Using the IBM XL Compilers for Blue Gene
  – C/C++: [http://pic.dhe.ibm.com/infocenter/compg/v121v141/nav/1](http://pic.dhe.ibm.com/infocenter/compg/v121v141/nav/1)
JUQUEEN – System Architecture

IBM Blue Gene/Q JUQUEEN

- IBM PowerPC A2 1.6 GHz, 16 cores/node, 4-way SMT, 64-bit
- 4-wide (dbl) SIMD (FMA)
- 16 GB RAM per node
- Torus network
- 8 racks, 131,072 cores
- 1.6 Petaflop/s peak
  1.37 Petaflop/s Linpack
- Connected to a Global Parallel File System (GPFS) with 6 PByte online disk and up to 25 PByte offline tape capacity
JUQUEEN – Challenges

Chip
- 4-way SMT with 1 integer + 1 FPU instruction per cycle → filling pipes
- 4-wide SIMD → efficient vectorization

Memory
- 1 GB/core and 0.5 GB/core for pure MPI codes → memory consumption
- HW support for transactional memory → efficient usage

Network
- Torus network → Mapping of tasks, communicators (communication pattern)

I/O
- Processing large amounts of data → Efficient I/O strategy and management

Parallelism
- MPP system → Scalability
Outline

Supercomputing@JSC
  Jülich Supercomputing Centre
  Supercomputing Systems and User Groups
  HPC Resources and User Support

JUQUEEN – Best Practices
  System Architecture – Challenges for Applications
  Production Environment and Application Stack
  Basic Porting
  Tuning Applications
  Performance Analysis
  Debugging
Environment – Operating System

Front-end Node (FEN)
- RedHat Linux V6.3 (64 Bit)
- Address: juqueen.fz-juelich.de
- Default shell: bash

I/O Node
- Fully Featured RedHat V6.3 based Linux
- 64 Bit

Compute Nodes (CN)
- Lightweight proprietary 64 bit kernel (Compute Node Kernel, CNK)
- Not all system calls are supported, check the RedBook!
Accounting

Billing
- Jobs charged for wall clock time

\[ \text{\#nodes} \times \text{twall clock time} \]

Quota
- Granted time (core hours, coreh) divided into monthly quota
- Quota of previous, current, and next month accessible
- Lower priority and max. wall time 6 hours if quota is used up
Quota Information

$ q_cpuquota <options>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-?</td>
<td>usage information and all options</td>
</tr>
<tr>
<td>-j &lt;jobid&gt;</td>
<td>for a single job with specified id</td>
</tr>
<tr>
<td>-t &lt;time&gt;</td>
<td>for all jobs in the specified time period, for example: -t 01.05.2011-15.05.2011</td>
</tr>
<tr>
<td>-d &lt;number&gt;</td>
<td>for the last &lt;number&gt; days (positive integer)</td>
</tr>
</tbody>
</table>

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.
Module Environment

$ module avail
------------ /bgsys/local/modulefiles/SCIENTIFIC  
          cpmd/3.15.1(default)     cpmd/3.15.1_b      namd/2.8  
          cpmd/3.15.1_a          lammmps/5May12(default) namd/2.9(default)  
------------ /bgsys/local/modulefiles/MISC  
------------ /bgsys/local/modulefiles/MATH  
          arpack/2.1_g
          fftw2/2.1.5
          fftw3/3.3.2

------------ /usr/local/modulefiles/COMPILER  
          cmake/2.8.8(default)
------------ /usr/local/modulefiles/MATH  
------------ /usr/local/modulefiles/SCIENTIFIC  
          gromacs/4.5.5
------------ /usr/local/modulefiles/IO
$ module help arpack/2.1_g
-----------------------------------------------
Module Specific Help for /bgsys/local/modulefiles/MATH/arpack/2.1_g:

ARPACK/PARPACK  2.1_g
 (P)ARPACK (Parallel) ARnoldi PACKage
http://www.caam.rice.edu/software/arpack/
http://www.caam.rice.edu/~kristyn/parpack_home.html/

Usage:
This version is compiled with -O3 -g -qsimd=noauto
Compiling and linking a Fortran program main.f
is done by
  mpixlf77_r -O3 -g -qsimd=noauto main.f -L$(ARPACK_LIB) -lparpack -larpack -L$(LAPACK_LIB) -llapack -L/bgsys/local/lib -lesslbg

See also: $ARPACK_DIR/README
-----------------------------------------------
Module Environment

$ module show arpack/2.1_g
-------------------------------------------------------------------
/bgys/local/modulefiles/MATH/arpack/2.1_g:

module-whatis ARPACK/PARPACK (Parallel) ARnoldi PACKage ($version)
setenv ARPACK_DIR /bgys/local/arpack
setenv PARPACK_DIR /bgys/local/arpack
setenv ARPACK_LIB /bgys/local/arpack/lib_g
setenv PARPACK_LIB /bgys/local/arpack/lib_g
setenv ARPACK_INCLUDE /bgys/local/arpack/include
setenv LAPACK_LIB /bgys/local/lapack/3.3.0_g/lib
-------------------------------------------------------------------

$ module load arpack/2.1_g
$ module list
Currently Loaded Modulefiles:
  1) arpack/2.1_g
JUQUEEN – Application Stack

Mathematical applications and libraries

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

Scientific applications

- CPMD (3.15.1)
- Gromacs (4.5.5)
- OpenFOAM
- CP2K
- Lammps (5May12)
- QuantumEspresso
- GPAW
- Namd (2.8, 2.9)
- VASP
JUQUEEN – Application Stack

I/O libraries

- HDF5
- SIONlib
- MPI I/O
- netCDF

Tools

- DDT
- hpctoolkit (5.2.1)
- PAPI (4.4.0)
- Scalasca (1.4.2)
- Score-P
- Totalview
- Vampir
- Tau (2.21.2)
Running Simulations – 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)
  - Classes chosen by LoadLeveler according to core count of job
  - Jobs requesting the full system (currently 8 racks) run on dedicated days
# 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>
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 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_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]</code></td>
<td>Type of wiring requested for the block (can be specified for each dimension separately)</td>
</tr>
<tr>
<td>Xa Xb Xc Xd</td>
<td></td>
</tr>
</tbody>
</table>


## LoadLevele – 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>
</tbody>
</table>

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!
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>
LoadLeveler – Example Job Command File I

```bash
# @job_name         = MPI_code
# @comment          = “32 ranks per node”
# @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 16384 --ranks-per-node 32 --exe app.x
```

Pure MPI applications need to use 32 tasks per node in order use the architecture efficiently!
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
    • \( n_{\text{tasks}} \times n_{\text{threads}} = 64 \)
    • \( n_{\text{tasks}} = 2^n, 0 \leq n \leq 6 \)

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

```plaintext
#@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 16384 --ranks-per-node 16\n    --env OMP_NUM_THREADS=4 --exe app.x
```
Monitoring of Jobs

• **LoadLeveler**
  – `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)
Llview

JUQUEEN
Outline

Supercomputing@JSC
  Jülich Supercomputing Centre
  Supercomputing Systems and User Groups
  HPC Resources and User Support

JUQUEEN – Best Practices
  System Architecture – Challenges for Applications
  Production Environment and Application Stack
  Basic Porting
  Tuning Applications
  Performance Analysis
  Debugging
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 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</th>
<th>MPI wrapper</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GNU (GCC)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td><code>powerpc64-bgq-linux-gcc</code></td>
<td><code>mpigcc</code></td>
</tr>
<tr>
<td>C++</td>
<td><code>powerpc64-bgq-linux-g++</code></td>
<td><code>mpig++</code></td>
</tr>
<tr>
<td>Fortran</td>
<td><code>powerpc64-bgq-linux-gfortran</code></td>
<td><code>mpigfortran</code></td>
</tr>
</tbody>
</table>

<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><strong>XL COMPILERS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td><code>bgx1c</code>, <code>bgc89</code>, <code>bgc99</code></td>
<td><code>mpixlc</code></td>
</tr>
<tr>
<td>C++</td>
<td><code>bgx1c++</code>, <code>bgx1C</code></td>
<td><code>mpixlcxx</code></td>
</tr>
<tr>
<td>Fortran</td>
<td><code>bgxlf</code>, <code>bgxlf90</code>, <code>bgxlf95</code>, <code>bgxlf2003</code></td>
<td><code>mpixlf77</code>, <code>mpixlf90</code>, <code>mpixlf95</code>, <code>mpixlf2003</code></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 -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>
**ESSL**

Engineering and Scientific Subroutine Library (IBM)
- High-performance mathematical libraries
- Tuned for Blue Gene/Q
- Can be called from C, C++, and Fortran
- Multi-threaded version (SMP) available
- Highly recommended if you use BLAS or LAPACK routines

**Fortran**

```bash
mpixlf90_r -o prog.x prog.f -L/bgsys/local/lib -qessl -lesslbg
mpixlf90_r -o prog.x prog.f -L$(LAPACK_LIB) -llapack \ 
    -L/bgsys/local/lib -lesslsmplib
```

Pay attention to the order of libraries when linking. The last symbol found by the linker will be used.
MPI

Message-Passing Interface

- Based on MPICH2 implementation
- MPI 2.1 Standard, BUT
  - Spawning of tasks is not supported
  - Asynchronous (non-blocking) I/O (e.g. MPI_File_iwrite, MPI_File_iread) is not supported

- Compiling applications: MPI wrappers → see compilers
- Running applications:
  - Using runjob in batch jobs (→ LoadLeveler, runjob)
OpenMP

Open Multi-Processing

- XL compiler family
- OpenMP 3.1 standard is supported
- Compiler flag: `qsmp=omp`
- Use thread-safe versions (e.g. `mpif90_r`, `mpixlc_r`, etc.)

GNU compiler family

- Default GNU compiler (4.4.6) supports OpenMP 3.0 standard
- Compiler flag: `-fopenmp`
Creating static Libraries

Preferred kind of library on BG/Q

```c
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

```bash
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 system! Try to avoid them!
Outline

Supercomputing@JSC
  Jülich Supercomputing Centre
  Supercomputing Systems and User Groups
  HPC Resources and User Support

JUQUEEN – Best Practices
  System Architecture – Challenges for Applications
  Production Environment and Application Stack
  Basic Porting
  Tuning Applications
  Performance Analysis
  Debugging
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>`
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).
Single-Core Optimization – Compiler Flags

Insertion of prefetch instructions
- `qprefetch=(no)aggressive`

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
- `qipa`
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
### QPX Expected Performance Improvements

<table>
<thead>
<tr>
<th>Type of Code</th>
<th>Expected performance improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear algebra dominated code</td>
<td>Up to 4x performance improvement over scalar code convenient to leverage “IBM ESSL for BG/Q”</td>
</tr>
<tr>
<td>Runtime-dependent alignment dominated code</td>
<td>1-4x performance improvement over scalar code (depending on runtime alignment values)</td>
</tr>
<tr>
<td>Memory-latency dominated code</td>
<td>Similar performance to BG/L and BG/P</td>
</tr>
</tbody>
</table>
IBM XL Compiler Support for QPX

Usage of QPX

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

```plaintext
>>> 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.
```
QPX Usage – Hints for the Compiler

Compiler needs hints
- Hint compiler to likely iteration counts
- Instruct compiler to align fields
- Tell that FOTRAN 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 QPX 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 (e.g. where no loop)

• Easy to use:
  – Leave stack, register layout, load/store scheduling to compiler
QPX Example using Compiler Intrinsics

```c
void dipole_force_qpx(
    int *i1, int *i2,
    double* x1, double* y1, double* z1,
    double* x2, double* y2, double* z2,
    double* xdp, double* ydp, double* zdp, double* charge, double* sumphi)
{
    typedef vector4double qv;
    qv dx,dy,dz,xd,yd,zd,rd,rd2,rd3,dx2,dy2,dz2,phi,one=vec_splats(1.0);
    int i,j,k;
    for (i=0;i<4;i++)
    {
        k=i1[i];
        j=i2[i];
        dx[i]=x1[k] - x2[j];
        dy[i]=y1[k] - y2[j];
        dz[i]=z1[k] - z2[j];
        c[i]=charge[j];
        xd[i]=xdp[i];
        yd[i]=ydp[i];
        zd[i]=zdp[i];
    }
    dx2 = dx*dx;
    dy2 = dy*dy;
    dz2 = dz*dz;
    d = vec_swsqrt(dx2+dy2+dz2);
    rd = one/d;
    rd2 = rd*rd;
    rd3 = rd2*rd;
    phi = c*rd;
    phi += (dx*xd + dy*yd + dz*zd)*rd3;
    *(qv*)sumphi+=phi;
}
```

\[ \phi' = \phi + \sum_{i=1}^{4} \frac{q_{i2}}{|\vec{r}_{i1} - \vec{r}_{i2}|} + \frac{\vec{p}_{i2} \cdot (\vec{r}_{i1} - \vec{r}_{i2})}{|\vec{r}_{i1} - \vec{r}_{i2}|^3} \]
Thread Level Speculation (TLS)

• Parallelize potentially dependent serial fragments
  – runtime creates threads for each speculative section
  – threads run parallel and commit *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

-qsmp=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: save thread
Thread 1: save thread
Thread 2: save thread

commit
commit
commit

Loop finished:

export SE_MAX_NUM_ROLLBACK=N
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 SAFE_MODE
    {
      code_to_be_atomic(i);
    }
  }
}
```

```c
export TM_MAX_NUM_ROLLBACK=N
export TM_REPORT_...=...
```
Tuning Runtime Environment

• Network
  – Topology on BG/Q: 5D Torus
    \( \text{A} \times \text{B} \times \text{C} \times \text{D} \times \text{E}(\times \text{T}) \)

• 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**
  
  ```
  #@bg_shape = <AxBxCxD> #JUQUEEN: 2x2x2x2 maximum
  #@bg_rotate = False|True
  ```

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

  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 # task 0; coordinates ( 0, 0, 0, 0, 0, 0)
1 0 0 0 0 0 # task 1; coordinates ( 1, 0, 0, 0, 0, 0)
2 0 0 0 0 0 # task 2; coordinates ( 2, 0, 0, 0, 0, 0)
[...]
**MPI Tuning – BG/Q Extensions**

Blue Gene/Q specific MPI extensions (MPIX)
- Only C/C++ interfaces available
- Include header: `#include <mpiX.h>`

**Examples**

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

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

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

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;
File I/O – Task-local Files

Example: Creating files in parallel in the same directory

Jugene, IBM Blue Gene/P, GPFS, filesystem /work using fopen()

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>0</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Files</td>
<td>1k</td>
<td>2k</td>
<td>4k</td>
<td>8k</td>
<td>16k</td>
</tr>
<tr>
<td></td>
<td>32k</td>
<td>64k</td>
<td>128k</td>
<td>256k</td>
<td></td>
</tr>
</tbody>
</table>

> 3 minutes
> 11 minutes
> 33 minutes

> 0
> 1,7
> 3,2
> 4,0
> 5,5
> 18,2
> 76,2
> 182,3
> 676,5
> 2004,6

Caution: The creation of 256k files costs 142.500 core hours! Task-local files are not an option on Blue Gene!
File I/O – Guidelines

Use the correct file system
  - $\texttt{WORK}$ has the best performance – use it!

Pay attention to block alignment
  - Best performance if the I/O is done in chunks with block size of the file system
    - $\texttt{WORK}$: 4 MB block size
    - \texttt{stat()} or \texttt{fstat()} can be used to get block size

Use parallel I/O (libraries)
  - MPI I/O, ADIOS, HDF5, netCDF, SIONlib
SIONlib – An Alternative to Task-Local Files

SIONlib
(Scalable I/O library for parallel access to task-local files)

- Collective I/O to binary shared files
- Logical task-local view to data
- Collective open/close, independent write/read
- Support for MPI, OpenMP, MPI+OpenMP
- C, C++, and Fortran-wrappers
- Tools for manipulation of files available (extraction of data, splitting of files, dumping meta data, etc)
SIONlib – Basic Principle

- Idea: Mapping many logical files onto one or a few physical file(s)
- Task-local view to local data not changed
SIONlib – An Alternative to Task-Local Files

/* Open */
sprintf(tmpfn,
"%s.%06d",filename,my_nr);
fileptr=fopen(tmpfn, "bw", ...

 [...] /* Write */
fwrite(bindata,1,nbytes,fileptr);
 [...] /* Close */
fclose(fileptr);

/* Collective Open */
nfiles=1;chunksize=nbytes;
sid=sion_paropen_mpi(
filename,"bw", &nfiles,
&chunksize,
MPI_COMM_WORLD, &lcomm,
&fileptr,...);

 [...] /* Write */
sion_fwrite(bindata,1,nbytes,sid);
 [...] /* Collective Close */
sion_parclose_mpi(sid);
Outline

Supercomputing@JSC
- Jülich Supercomputing Centre
- Supercomputing Systems and User Groups
- HPC Resources and User Support

JUQUEEN – Best Practices
- System Architecture – Challenges for Applications
- Production Environment and Application Stack
- Basic Porting
- Tuning Applications
- Performance Analysis
- Debugging
Performance Analysis

Goal: Detection and elimination of performance bottlenecks

- Prepare application, insert extra code (probes/hooks)
- Collection of data relevant to performance analysis
- Calculation of metrics, identification of performance metrics
- Visualization of results in an intuitive/understandable form
- Elimination of performance problems
Performance Analysis Tools

Accessible via module command:

```
module load UNITE <tool>
```

Available tools (selection):

- gprof (GNU profiler) → no module
- hpctoolkit → hpctoolkit/5.2.1
- marmot (MPI checker)
- papi (hardware counter) → papi/5.0.0
- Scalasca → scalasca/1.4.2
- tau → tau/2.21.3
- vampir

Basic help, first steps and references:

```
module help <tool>
```
Scalasca

Toolset for callpath and communication profiles

- Analysis of MPI and hybrid OpenMP/MPI applications
- Results can be visualized

Scalasca workflow

Instrument your code:

Prepend compile/link command with skin

Run the analysis:

Prepend the runjob command with scan
This will create an experiment archive epik_<title>

Examine results

square epik_<title>
Scalasca
Outline

Supercomputing@JSC
  Jülich Supercomputing Centre
  Supercomputing Systems and User Groups
  HPC Resources and User Support

JUQUEEN – Best Practices
  System Architecture – Challenges for Applications
  Production Environment and Application Stack
  Basic Porting
  Tuning Applications
  Performance Analysis
  Debugging
Debugging – Compiler Options

Compilers give hints

- Turn optimization off
  - `00 -g`

- Switch on compiler compile-time and runtime checks (floating point exceptions, boundary checks, pointer checks, etc.)
  - `-qcheck[=<suboptions>]` (C/C++)
  - `-qformat[=<suboptions>]` (C/C++)
  - `-qinfo[=[<suboptions>],[,<goups_list>]]]` (C/C++)
  - `-C` (Fortran)
  - `-qsigtrap[=<trap_handler>]` (Fortran)
  - `-qinit=f90ptr` (Fortran)
  - `-qflttrap[=<suboptions>]`
Using Debuggers

Prepare executable:
- `O0 -g -qfullpath`

Available debuggers
- `gdb`
- `DDT (porting on the way)`
- `Totalview (ported, currently testing)`
Debugging – Analyzing Core Dump Files

Activate generation of lightweight core dump files

```
runjob --env BG_COREDUMPDISABLED=0 […]
```

– Core dump files are plain ASCII files including information in hexadecimal
– Generate your executable with the -g flag

Analyzing lightweight core dump files

1. **addr2line**
   ```
   addr2line -e application.x <hexaddr>
   ```

2. **Totalview**
3. **DDT**
Thank You!

JSC user and application support:

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