Introduction to LONI QB-4 Cluster

Le Yan
Assistant Director
LONI HPC
Objectives

• Understand how QB-4’s architecture and user environment are different from those of QB-3
• Understand how various factors affect application performance on QB-4
Outline

• QB-4 hardware architecture
• QB-4 user environment
• Application performance benchmarks and tuning on QB-4
QB-4 architecture and software environment
QB-4 Specs

547 Compute Nodes (35,008 CPU cores, 144 GPUs, 161 TB RAM)

480 regular nodes
2 Intel 32-core CPUs
256 GB RAM

52 2-GPU nodes
2 Intel 32-core CPUs
512 GB RAM
2 NVIDIA A100 GPUs

10 4-GPU nodes
2 Intel 32-core CPUs
512 GB RAM
4 NVIDIA A100 GPUs

5 big memory nodes
2 Intel 32-core CPUs
2 TB RAM

Infiniband NDR200 200Gb/s Fabric

6.5 PetaBytes Lustre Storage
QB-4 vs QB-3 vs QB-2

<table>
<thead>
<tr>
<th></th>
<th>In production since</th>
<th>Theoretical peak performance (PFLOPS)</th>
<th>CPU cores</th>
<th>Total RAM (TB)</th>
<th>SUs per year (million)</th>
</tr>
</thead>
<tbody>
<tr>
<td>QB-2*</td>
<td>2014</td>
<td>1.5</td>
<td>10,192</td>
<td>38</td>
<td>89</td>
</tr>
<tr>
<td>QB-3</td>
<td>2020</td>
<td>0.9</td>
<td>9,696</td>
<td>41</td>
<td>85</td>
</tr>
<tr>
<td>QB-4</td>
<td>2024</td>
<td>4.3</td>
<td>35,008</td>
<td>161</td>
<td>307</td>
</tr>
</tbody>
</table>

* This is the original QB-2 before some nodes were decommissioned.

8/7/2024
# QB-4 Node Specification

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Ice Lake (Xeon Platinum 8358) (2 sockets *32 cores/socket)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU frequency</td>
<td>2.6 G Hz</td>
</tr>
<tr>
<td>Floating operation per clock cycle (double precision)</td>
<td>32</td>
</tr>
<tr>
<td>Memory</td>
<td>256GB/512GB/2TB DDR4</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA A100 80GB PCIe</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Mellanox 200 Gbps Infiniband</td>
</tr>
</tbody>
</table>
## QB-4 vs QB-3 (Node over Node)

<table>
<thead>
<tr>
<th></th>
<th>QB-4</th>
<th>QB-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU frequency</td>
<td>$2.6 \times 10^9$</td>
<td>$2.4 \times 10^9$</td>
</tr>
<tr>
<td>CPU cores</td>
<td>64</td>
<td>48</td>
</tr>
<tr>
<td>Operation per cycle</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>~400 GB/s</td>
<td>~280 GB/s</td>
</tr>
<tr>
<td>Interconnect</td>
<td>200 Gbps</td>
<td>100 Gbps</td>
</tr>
</tbody>
</table>

### Node peak performance

- **QB-4**: $64 \text{ cores/node} \times 2.6 \times 10^9 \text{ cycles/second} \times 32 \text{ flop/cycle} = 5.32 \times 10^{12} \text{ flops}$
- **QB-3**: $48 \text{ cores/node} \times 2.4 \times 10^9 \text{ cycles/second} \times 32 \text{ flop/cycle} = 3.69 \times 10^{12} \text{ flops}$

**Theoretical speedup = 1.4**
## Software Environment (1)

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>RHEL 8.8</td>
</tr>
<tr>
<td>Workload Manager</td>
<td>Slurm</td>
</tr>
<tr>
<td>Software Environment Manager</td>
<td>Environment Module</td>
</tr>
<tr>
<td>Container Engine</td>
<td>Singularity</td>
</tr>
<tr>
<td>Default Toolchain (Compiler + MPI)</td>
<td>Intel 2021.5.0, Intel MPI 2021.5.1</td>
</tr>
</tbody>
</table>

The QB-4 Open OnDemand portal will be available soon!
$ module av

-- /project/containers/modulekeys --
agat/1.4.0  blast/2.14.1  busco/5.7.1
jellyfish/2.3.0  octopus/14.0

-- /usr/local/packages/Modules/default/modulefiles/icelake --
amber/22/intel-2021.5.1-intel-mpi-2021.5.1
intel/2.7.1/intel-2021.5.0
parallel-netcdf/1.12.3/intel-2021.5.0-intel-mpi-2021.5.1
hdf5/1.12.2/intel-2021.5.0-intel-mpi-2021.5.1
parallel/20220522/intel-2021.5.0
boost/1.83.0/intel-2021.5.0

Use the “module av” command to list all installed packages.
Software packages are installed either by compilation or as container images. The difference should be minimal from the users’ point of view.
If a package is missing, please submit a ticket to sys-help@loni.org,
## Workload Management (1)

- **List of job partitions/queues**

<table>
<thead>
<tr>
<th>Single</th>
<th>Jobs that will only execute on a single node. Default queue.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkpt</td>
<td>Jobs that use multiple nodes and are preemptable.</td>
</tr>
<tr>
<td>Workq</td>
<td>Jobs that use multiple nodes.</td>
</tr>
<tr>
<td>Bigmem</td>
<td>Jobs that use the big memory nodes (2TB per node).</td>
</tr>
<tr>
<td>Gpu2</td>
<td>Jobs that use GPUs on the 2-GPU nodes.</td>
</tr>
<tr>
<td>Gpu4</td>
<td>Jobs that use GPUs on the 4-GPU nodes.</td>
</tr>
</tbody>
</table>
Workload Management (2)

• GPU jobs can request less than a whole node

On **QB-3** a GPU job must request at least a GPU node, i.e. two GPU devices:

```bash
$ srun -p gpu -N1 -A loni_my_allocation my_gpu_executable
```

On **QB-4** a GPU job can request one GPU device on a GPU node with 2 or 4 GPUs:

```bash
$ srun -p gpu2 -N1 --gres=gpu:1 -A loni_my_allocation my_gpu_executable
```

On **QB-4** a GPU job can request three GPU devices on a GPU node with 4 GPUs:

```bash
$ srun -p gpu4 -N1 --gres=gpu:3 -A loni_my_allocation my_gpu_executable
```
AI/DL Frameworks

- All AI/DL frameworks are installed via container images and/or Conda VE’s
  - Command line
    - jax/0.4.26 pytorch/2.2.2 tensorflow/2.16.1
  - Open OnDemand portal (coming soon)
Policy Update (1)

- Max SUs for a single allocation: 6M -> 8M
- Total max active SUs per PI: 12M -> 16M
- Startup allocation threshold: 50K -> 150K
- Instructional allocations
  - A new type of allocations for courses and/or training activities
  - Limit: 150K SUs
  - Can be submitted and reviewed at any time
  - PI still needs to provide a justification in the form of a proposal
Policy Update (2)

• Higher charge rate for the A100 GPUs on QB-4
  – Each A100 GPU is treated as a regular compute node (64 CPU cores)

A job runs 2 hours on a QB-3 GPU node (2 V100 GPU devices):

Charge = 2 hours * 48 CPU cores = 96 SUs

A job runs 2 hours on a QB-4 2-GPU node using both A100 GPUs:

Charge = 2 hours * 2 GPU devices * 64 CPU cores/GPU device = 256 SUs

A job runs 2 hours on a QB-4 4-GPU node using all four A100 GPUs:

Charge = 2 hours * 4 GPU devices * 64 CPU cores/GPU device = 512 SUs
Application Performance Benchmarks and Tuning
Disclaimer

• Target audience: users who run (and sometimes compile) applications developed by others
  – Not an in-depth guide for programmers and developers
QB-4 Architecture – Node Level

Node level view

Memory

Memory
• The 64 cores on a QB-4 node are grouped into 4 sets.
• The data exchange cost is not homogeneous.
• Depending on the data exchange pattern, how the threads are arranged could affect performance significantly.
QB-4 Architecture – Cluster Level

Cluster level view

Switch

Switch

Switch
Why This Matters

- Keys for good performance
  - Distribute the workload (well) among the CPU cores
  - Keep the CPU cores busy by keeping them well fed (with data)
- Data supply efficiency depends on their relative positions within the hierarchy
- Applications are
  - **CPU-bound** if CPU cores are well fed
  - **Memory-bound** (or I/O bound) if CPU cores are hungry most of the time
## Parallel Paradigms

<table>
<thead>
<tr>
<th></th>
<th><strong>Pro</strong></th>
<th><strong>Con</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Intranode</strong></td>
<td>- Low latency, high bandwidth</td>
<td>- Shared memory system only (Limited to one node)</td>
</tr>
<tr>
<td>(data exchange</td>
<td>- Implicit communication</td>
<td></td>
</tr>
<tr>
<td>through <strong>memory</strong></td>
<td>- Fine granularity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- (Relatively) Easy to balance the load</td>
<td></td>
</tr>
<tr>
<td><strong>Internode</strong></td>
<td>- Scalability beyond 1 node</td>
<td>- High latency, low bandwidth</td>
</tr>
<tr>
<td>(data exchange</td>
<td></td>
<td>- Explicit communication</td>
</tr>
<tr>
<td>through <strong>network</strong></td>
<td></td>
<td>- Hard to balance the load</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Parallel Paradigms

## Intranode (data exchange through memory)
- **Pro**
  - Low latency, high bandwidth
  - Implicit communication
  - Fine granularity
  - (Relatively) Easy to balance the load
- **Con**
  - Shared memory system only (Limited to one node)

## Internode (data exchange through network)
- **Pro**
  - Scalability beyond 1 node
- **Con**
  - High latency, low bandwidth
  - Explicit communication
  - Hard to balance the load

### OpenMP (Multi-threaded)
- Multi-threaded approach

### MPI (Multi-process)
- Multi-process approach
What About MPI+OpenMP Hybrid?

- Getting the benefits from both worlds?
- In theory, yes
- But adding OpenMP to (well-written) MPI programs might hurt the performance
- Hybrid helps to
  - Reduce memory footprint
  - Extend scalability
What About MPI+OpenMP Hybrid?

- Getting the benefits from both worlds?
- In theory, yes
- But adding OpenMP to (well-written) MPI programs might hurt the performance
- Hybrid helps to
  - Reduce memory footprint
  - Extend scalability

Your mileage may vary!
Single Node Performance
QB-4 vs QB-3 (Node over Node)

<table>
<thead>
<tr>
<th></th>
<th>QB-4</th>
<th>QB-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU frequency</td>
<td>$2.6 \times 10^9$</td>
<td>$2.4 \times 10^9$</td>
</tr>
<tr>
<td>CPU cores</td>
<td>64</td>
<td>48</td>
</tr>
<tr>
<td>Operation per cycle</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>~115 GB/s</td>
<td>~48 GB/s</td>
</tr>
<tr>
<td>Interconnect</td>
<td>200 Gbps</td>
<td>100 Gbps</td>
</tr>
</tbody>
</table>

**Node peak performance**

QB-4: 64 cores/node $\times 2.6 \times 10^9$ cycles/second $\times$ 32 flop/cycle $= 5.32 \times 10^{12}$ flops

QB-3: 48 cores/node $\times 2.4 \times 10^9$ cycles/second $\times$ 32 flop/cycle $= 3.69 \times 10^{12}$ flops

**Theoretical speedup** $= 1.4$
STREAM Benchmark

• The de facto industry standard benchmark in HPC domain for the measurement of sustainable memory bandwidth (in GB/s).
STREAM Benchmark - Results

Memory bandwidth 95% saturated when only half of the cores are used.

When all cores are used, memory bandwidth per core drops to 1/4 of the maximum.

QB-3: 225.0 GB/s with 48 threads.

STREAM 5.10
Array size = 160 MB
Intel 2021.5.0 with "-O3 -xcORE-AVX512"
OMP_PROC_BIND=spread
Thread Affinity

- The 64 cores on a SM-3 node are grouped into 4 sets.
- The data exchange cost is not homogeneous.
- Depending on the data exchange pattern, how the threads are arranged could affect performance significantly.
Thread Affinity

- For programs compiled with Intel compilers, use the KMP_AFFINITY environment variable to control thread placement/affinity.
- The options are “none” (default), “disabled”, “balanced”, “compact”, and “scatter”.
- Use OMP_PROC_BIND=spread in place of “balanced”.
STREAM – Thread Affinity

STREAM 5.10
Array size = 160 MB
Intel 2021.5.0 with "-O3 -xCORE-AVX512 -qopt-zmm-usage=high"

Higher is better
• High Performance Linpack
  – Standard benchmark for CPU-bound HPC applications

• Results
  – QB-4: 3847 GFLOPS per node
  – QB-3: 2452 GFLOPS per node
  – Speedup = 1.57 (compared to 1.4 theoretical)
HPCG Benchmark

• High Performance Conjugate Gradient
  – Standard benchmark for memory-bound HPC applications

• Results
  – QB-4: 55.0 GFLOPS per node
  – QB-3: 33.1 GFLOPS per node
  – Speedup: 1.66 (compared to 1.4 theoretical)
NPB Benchmark Suite

- NAS Parallel Benchmarks
  - a small set of programs derived from computational fluid dynamics applications
- Five kernels and three pseudo-applications
  - IS - Integer Sort, random memory access
  - EP - Embarrassingly Parallel
  - CG - Conjugate Gradient, irregular memory access and communication
  - MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
  - FT - discrete 3D fast Fourier Transform, all-to-all communication
  - BT - Block Tri-diagonal solver
  - SP - Scalar Penta-diagonal solver
  - LU - Lower-Upper Gauss-Seidel solver
NPB Benchmarks – Node over Node

Higher is better

NPB 3.4.2, Class D
Intel 2021.5.0 with “-O3 -xCORE-AVX512”
KMP_AFFINITY=none
NPB Benchmarks – Core over Core

Higher is better

NPB 3.4.2, Class D
Intel 2021.5.0 with "-O3 -xCORE-AVX512"
KMP_AFFINITY=none
NPB Benchmarks – Core over Core

**Takeway**

- Performance gain on QB-4 varies from application to application.
- Core-over-core performance gain could be very limited.

Higher is better

NPB 3.4.2, Class D
Intel 2021.5.0 with 

```
-O3 -xCORE-AVX512
KMP_AFFINITY=none
```

8/7/2024
NPB Results – Thread Affinity

Higher is better

NPB 3.4.2, Class E
Intel 2021.5.0 with “-O3 –xCORE-AVX512”
NPB Results – Thread Affinity

Higher is better

Benchmark Names

Default
Spread
Disabled
Compact
Scatter

Takeway

• “OMP_PROC_BIND=spread” is a good starting point.
Compiler Flags (Intel)

• -O2, -O3  
  – Generic, aggregated optimization flags
• -xCORE-AVX512  
  – Turns on optimization for the Ice Lake (and SkyLake/Cascade Lake) processor
• -march=icelake-server  
  – Turns on optimization specifically for Ice Lake
• -qopt-zmm-usage=high  
  – Improves performance for some codes
NPB Results – Compiler Flags

Higher is better

NPB 3.4.2, Class E
Intel 2021.5.0
KMP_AFFINITY=default

8/7/2024
NPB Results – Compiler Flags

**Takeway**

- “-O3 –xCORE-AVX512” gives reasonable performance in most cases and should be used as the baseline.
Amber 22

 Explicit solvent

Software: Amber 22 built with Intel 2021.5.1
MPI: Intel-MPI 2021.5.1 running under RHEL 8
Model name: Intel(R) 8358 CPU 2.60GHz
Single compute node
HPC Cluster: QueenBee at LONI

Higher is better

Explicit solvent

Software: Amber 22 built with gcc 9.2.0
MPI: OpenMPI 4.1.3 running under Rocky 8
Model name: AMD(R) 7543 CPU 2.80GHz
Single compute node
HPC Cluster: BiOWULF at NIH

ns/day

Number of CPU cores (n)

JAC-NVE-4fs
JAC-NPT-4fs
FactorX-NVE-4fs
FactorX-NPT-4fs
Cellulose-NVE-4fs
Cellulose-NPT-4fs
STMV-NVE-4fs
STMV-NPT-4fs

25 50 75 100 125 150

2 4 8 16 32 64

JAC: 23558 atoms
FactorX: 90906 atoms
Cellulose: 408609 atoms
STMV: 1067095 atoms

JAC-NVE-4fs
JAC-NPT-4fs
FactorX-NVE-4fs
FactorX-NPT-4fs
Cellulose-NVE-4fs
Cellulose-NPT-4fs
STMV-NVE-4fs
STMV-NPT-4fs

25 50 75 100 125 150

2 4 8 16 32 64

JAC: 23558 atoms
FactorX: 90906 atoms
Cellulose: 408609 atoms
STMV: 1067095 atoms

8/7/2024
## GPU Performance

<table>
<thead>
<tr>
<th></th>
<th>QB-3</th>
<th>QB-4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td>V100 PCIe</td>
<td>A100 PCIe</td>
</tr>
<tr>
<td><strong>On-board memory</strong></td>
<td>32GB</td>
<td>80GB</td>
</tr>
<tr>
<td><strong>Processing power (DP)</strong></td>
<td>7 TFLOPS</td>
<td>9.7 TFLOPS</td>
</tr>
<tr>
<td><strong>NVLink</strong></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>InterGPU bandwidth</strong></td>
<td>~20 GB/s</td>
<td>~520 GB/s*</td>
</tr>
</tbody>
</table>

*On nodes with 4 GPU’s, this is the bandwidth between device 0 and 1, and device 2 and 3.

- Higher is better
- 4.2x speed up

---

**RESNET50**

- GPU QB-3
- GPU QB-4
- 4x speed up

---

8/7/2024
Multi-node Performance
Pure MPI - NPB LU Benchmark

Higher is better

QB-4, Class D

1.5-2x speed up (node over node)

QB-3, Class D

Strong scaling (fixed problem size)

NPB 3.4.2, Class D
Intel MPI 2021.5.1
Intel 2021.5.0 with “-O3 -xCORE-AVX512”
Pure MPI - NPB LU Benchmark

Higher is better

Ideal

QB-4, Class D

Strong scaling (fixed problem size)
Pure MPI - NPB LU Benchmark

**NPB 3.4.2, Class D and E**
**Intel MPI 2021.5.1**
**Intel 2021.5.0 with “-O3 –xCORE-AVX512”**
Pure MPI – Amber 22

Explicit solvent

Higher is better

Software: Amber 22 built with Intel 2021.5.1
MPI: Intel-MPI 2021.5.1 running under RHEL 8
Model name: Intel(R) 8358 CPU 2.60GHz
Single compute node
HPC Cluster: QueenBee at LONI

Strong scaling (fixed problem size)
Pure MPI - GROMACS

Higher is better

Strong scaling (fixed problem size)
Pure MPI – LAMMPS

Higher is better

Strong scaling
(fixed problem size)
Hybrid – HPL

Weak scaling (problem size increases with core count)
1 MPI process per node * 64 threads per MPI process

Ideal speedup
93.5% efficiency with 540 nodes

HPL 2.3 (Intel MKL version)
Problem size = 90% installed memory
Intel MPI 2021.5.1
Intel 2021.5.0 with "-O3 -xCORE-AVX512 -qopt-zmm-usage=high"
Hybrid - HPCG

Weak scaling (problem size increases with core count)
1 MPI process per node * 64 threads per MPI process

- HPCG 3.0 (Intel MKL version)
- Problem size = 336 336 336
- Intel MPI 2021.5.1
- Intel 2021.5.0 with “-O3 -xCORE-AVX512 -qopt-zmm-usage=high”
Hybrid - GROMACS

Benchmark: HecBioSim 3M atoms
GROMACS
Intel MPI 2021.5.1
Intel 2021.5.0
KMP_AFFINITY=default

Higher is better
GPU Scaling – GPT-2 Training

GPT-2 (124M) model
Python 3.10.12
Pytorch 2.3.0
Transformer 4.41.2
CUDA 12.4.1
Configuration: https://github.com/karpathy/llm.c/discussions/481
I/O Consideration

• You want to avoid letting your program accesses to disk excessively
  – Reading/writing hundreds of GBs to checkpoint or output files frequently
  – Running with thousands of MPI tasks all reading/writing individual files

• What you should and should not do
  – Avoid writing intermediate/checkpoint files unless necessary
  – Look for and use options that allow one or a few big files instead of files per process
  – Reduce the frequency of writing output files
  – Do not use /home for productive jobs – use /work instead
Takeaways (1)

• In most cases, your application will run faster and scale better on QB-4 (compared to QB-3)
• That being said, how much faster depends on a lot of factors
• You need to run your own experiments before making a (educated) decision whether or not switch to QB-4
Takeaways (2)

• Baseline
  – Use “-O3 -xCORE-AVX512” to compile
  – The default settings work reasonably well in most cases

• Serial (single core) programs
  – The performance gain can be limited

• OpenMP programs
  – Use KMP_AFFINITY=balanced as baseline and try different settings

• MPI programs
  – Find the optimal number of nodes by running scaling tests
  – Remember the scaling behavior depends on the problem size

• Hybrid programs
  – They do not always perform and scale better than MPI programs, especially for small problems and low node count
  – The optimal number of threads (and affinity) could be tricky to find and varies from application to application
Takeaways (2)

• Baseline
  – Use “-O3 -xCORE-AVX512” to compile
  – The default settings work reasonably well in most cases

• Serial (single core) programs
  – The performance gain can be limited

• OpenMP programs
  – Use KMP_AFFINITY=balanced as baseline and try different settings

• MPI programs
  – Find the optimal number of nodes by running scaling tests
  – Remember the scaling behavior depends on the problem size

• Hybrid programs
  – They do not always perform and scale better than MPI programs, especially for small problems and low node count
  – The optimal number of threads (and affinity) could be tricky to find and varies from application to application

Avoid excessive I/O!!!
Takeaways (2)

• Baseline
  – Use “-O3 –xCORE-AVX512” to compile
  – The default settings work reasonably well in most cases

• Serial (single core) programs
  – The performance gain can be limited

• OpenMP programs
  – Use KMP_AFFINITY=balanced as baseline and try different settings

• MPI programs
  – Find the optimal number of nodes by running scaling tests
  – Remember the scaling behavior depends on the problem size

• Hybrid programs
  – They do not always perform and scale better than MPI programs, especially for small problems and low node count
  – The optimal number of threads (and affinity) could be tricky to find and varies from application to application

If you need help, let us know!