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

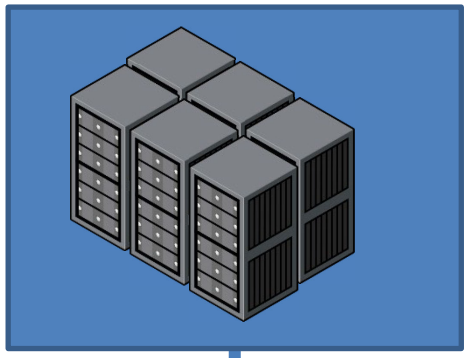
2 Intel 32-core CPUs
512 GB RAM
2 NVIDIA A100 GPUs

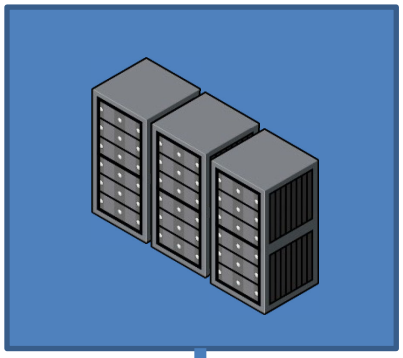
10 4-GPU nodes

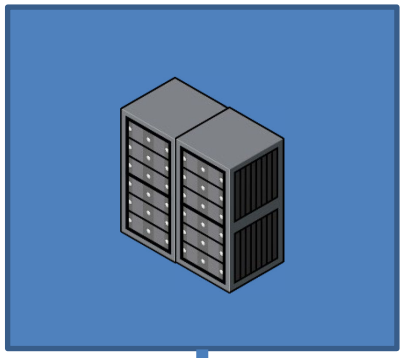
2 Intel 32-core CPUs
512 GB RAM
4 NVIDIA A100 GPUs

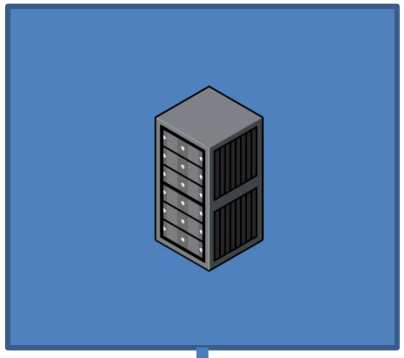
5 big memory nodes

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

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

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

QB-4 Node Specification

CPU	Intel Ice Lake (Xeon Platinum 8358) (2 sockets *32 cores/socket)
CPU frequency	2.6 G Hz
Floating operation per clock cycle (double precision)	32
Memory	256GB/512GB/2TB DDR4
GPU	NVIDIA A100 80GB PCIe
Interconnect	Mellanox 200 Gbps Infiniband

QB-4 vs QB-3 (Node over Node)

	QB-4	QB-3
CPU frequency	2.6×10^9	2.4×10^9
CPU cores	64	48
Operation per cycle	32	32
Memory bandwidth	~400 GB/s	~280 GB/s
Interconnect	200 Gbps	100 Gbps

<u>Node peak performance</u>

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

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

Theoretical speedup = 1.4

Software Environment (1)

Operating System	RHEL 8.8
Workload Manager	Slurm
Software Environment Manager	Environment Module
Container Engine	Singularity
Default Toolchain (Compiler + MPI)	Intel 2021.5.0 Intel MPI 2021.5.1

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

Software Environment (2)

```
$ 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/modu
icelake --

amber/22/intel-2021.5.1-intel-mpi-2021.5.1
gsl/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

single	Jobs that will only execute on a single node. Default queue.
checkpt	Jobs that use multiple nodes and are preemptable.
workq	Jobs that use multiple nodes.
bigmem	Jobs that use the big memory nodes (2TB per node).
gpu2	Jobs that use GPUs on the 2-GPU nodes.
gpu4	Jobs that use GPUs on the 4-GPU nodes.

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:

$ 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:

$ 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:

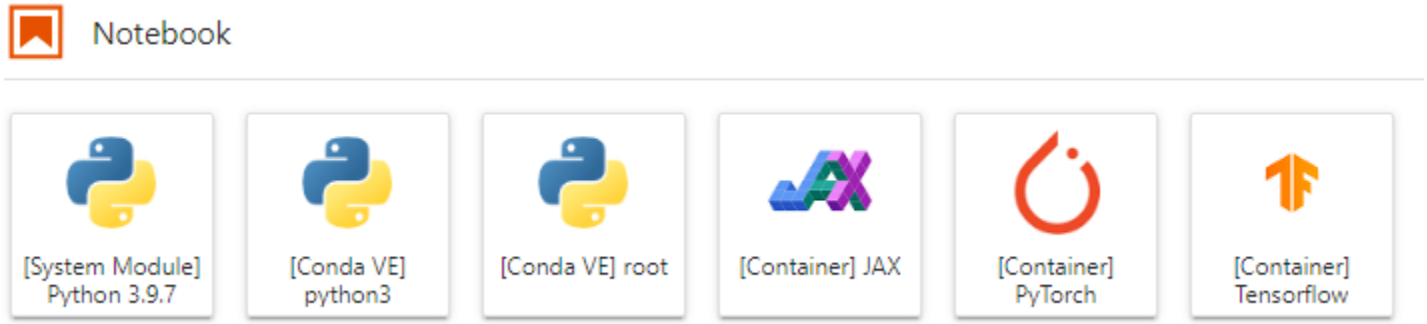
$ srun –p gpu4 –N1 **--gres=gpu:3** –A loni_my_allocation my_gpu_executable

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

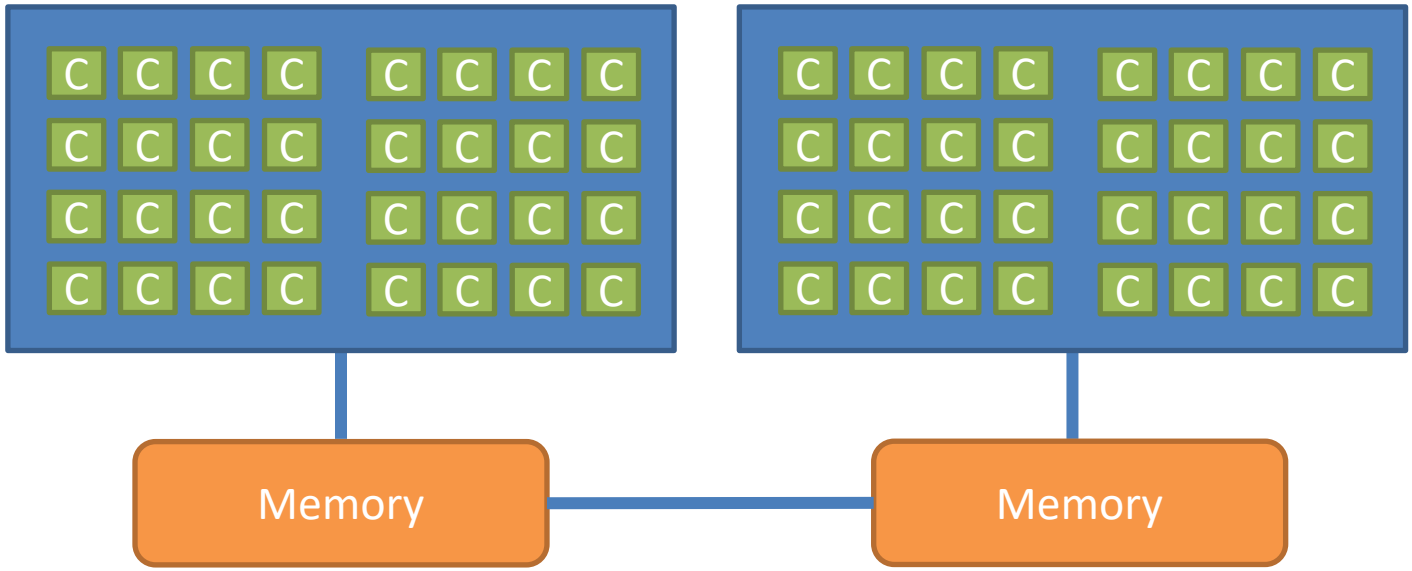
Application Performance Benchmarks and Tuning

Disclaimer

- Target audience: users who run (and sometimes compile) applications developed by others
  - Not an in-depth guide for programmers and developers

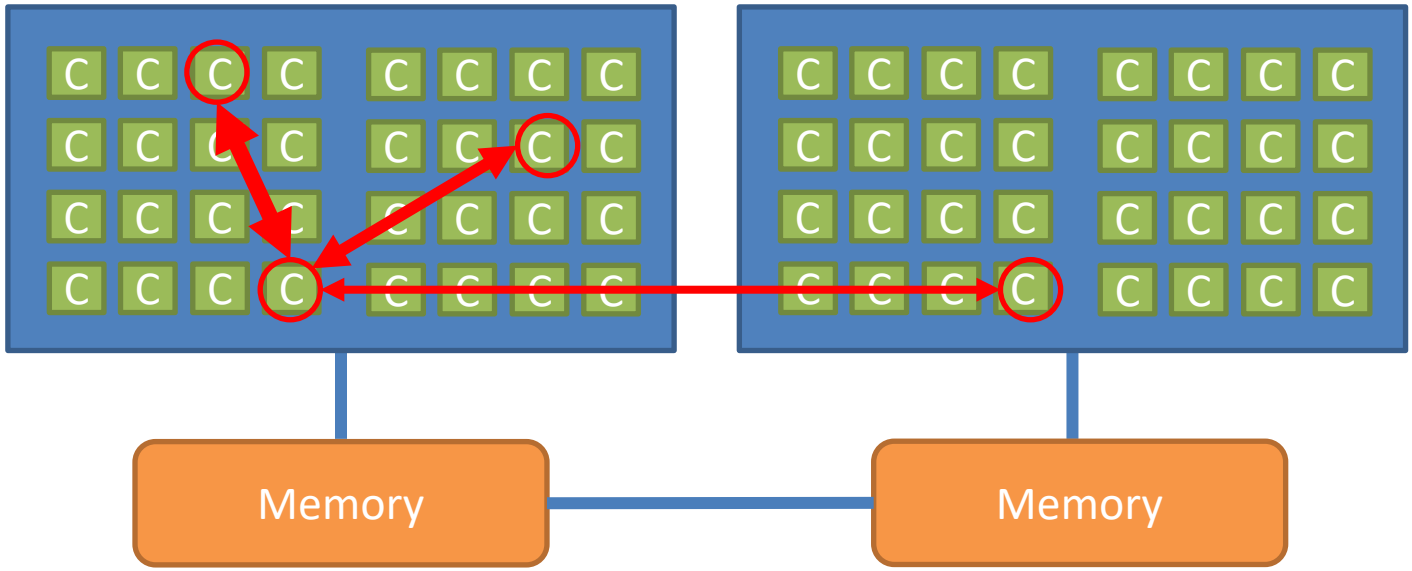
QB-4 Architecture – Node Level

Node level view



QB-4 Architecture – Node Level

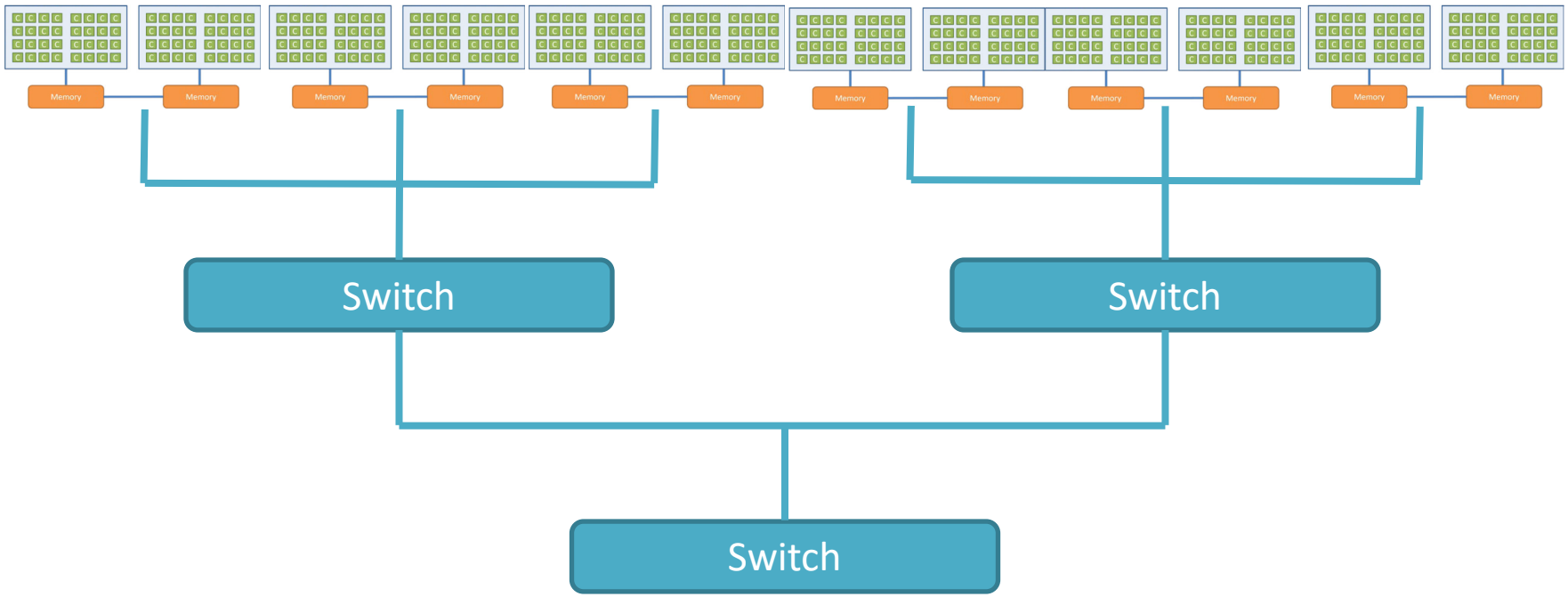
Node level view



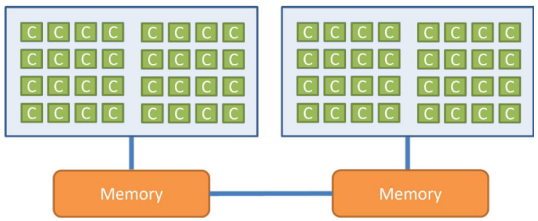
- The 64 cores on a QB-4 node are grouped into 4 sets.
- The data exchange cost is not homogeneous.
- Depending on the data exchange pattern, how the threads are arranged could affect performance significantly.

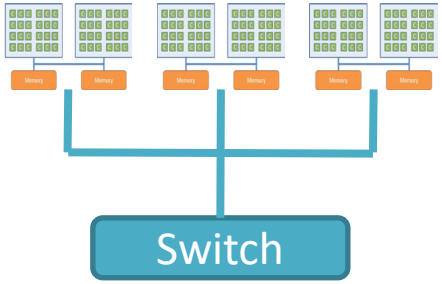
QB-4 Architecture – Cluster Level

Cluster level view



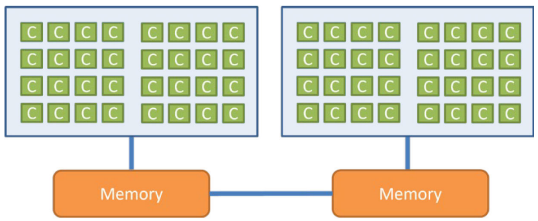
Why This Matters

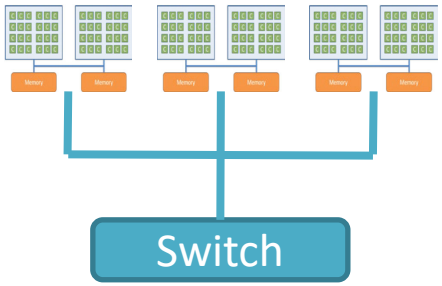




- Keys for good performance
  - Distribute the workload (well) among the CPU cores
  - Keep the CPU cores busy by keeping them well fed (with data)
- Data supply efficiency depends on their relative positions within the hierarchy
- Applications are
  - **CPU-bound** if CPU cores are well fed
  - **Memory-bound**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

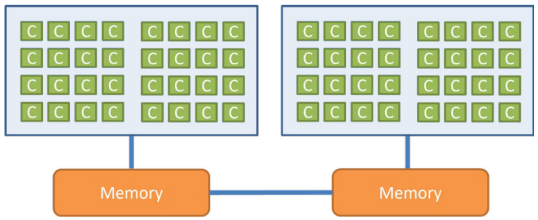
Parallel Paradigms

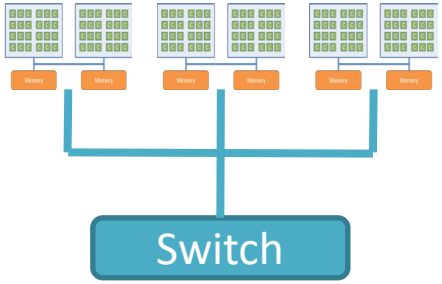




	Pro	Con
Intranode (data exchange through **memory**)	- Low latency, high bandwidth - Implicit communication - Fine granularity - (Relatively) Easy to balance the load	- Shared memory system only (Limited to one node)
Internode (data exchange through **network**)	- Scalability beyond 1 node	- High latency, low bandwidth - Explicit communication - Hard to balance the load

Parallel Paradigms



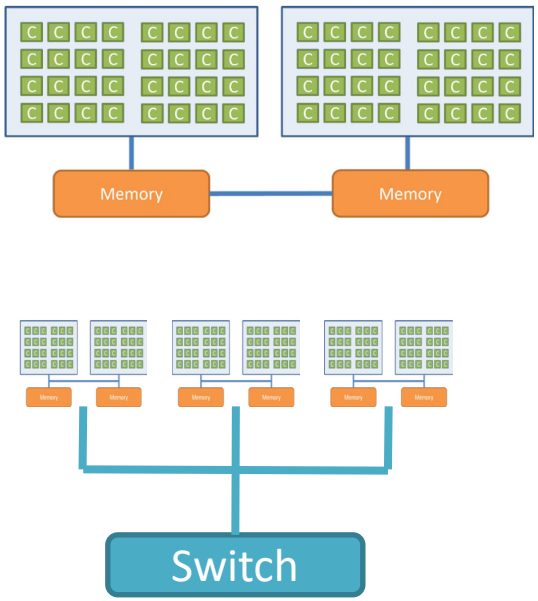


	Pro	Con
Intranode (data exchange through **memory**)	- Low latency, high bandwidth - Implicit - Fine granularity - (Relatively) Easy to balance the load	- Shared memory system only - Limited to one node
Internode (data exchange through **network**)	- Scalability beyond 1	- High latency, low bandwidth - Explicit communication - Hard to balance the load

OpenMP (Multi-threaded)

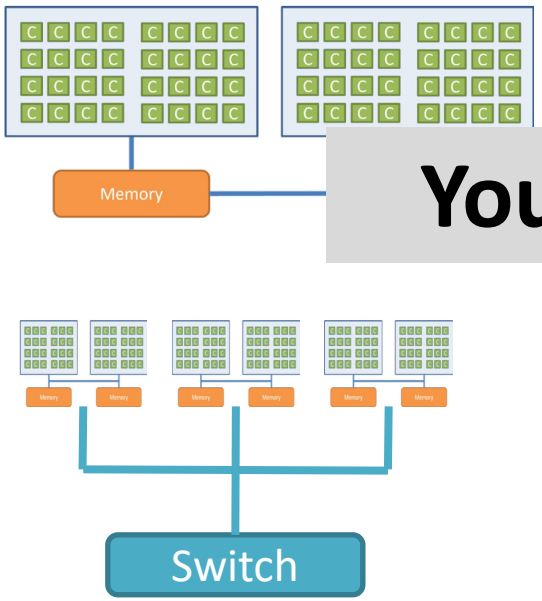
MPI (Multi-process)

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?

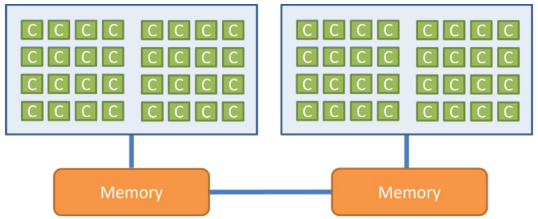


- Getting the benefits from both worlds?
- In theory, yes

Your mileage may vary!

(well-written) MPI programs might hurt the performance
- Hybrid helps to
  - Reduce memory footprint
  - Extend scalability

Single Node Performance



QB-4 vs QB-3 (Node over Node)

	QB-4	QB-3
CPU frequency	2.6×10^9	2.4×10^9
CPU cores	64	48
Operation per cycle	32	32
Memory bandwidth	~115 GB/s	~48 GB/s
Interconnect	200 Gbps	100 Gbps

<u>Node peak performance</u>

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

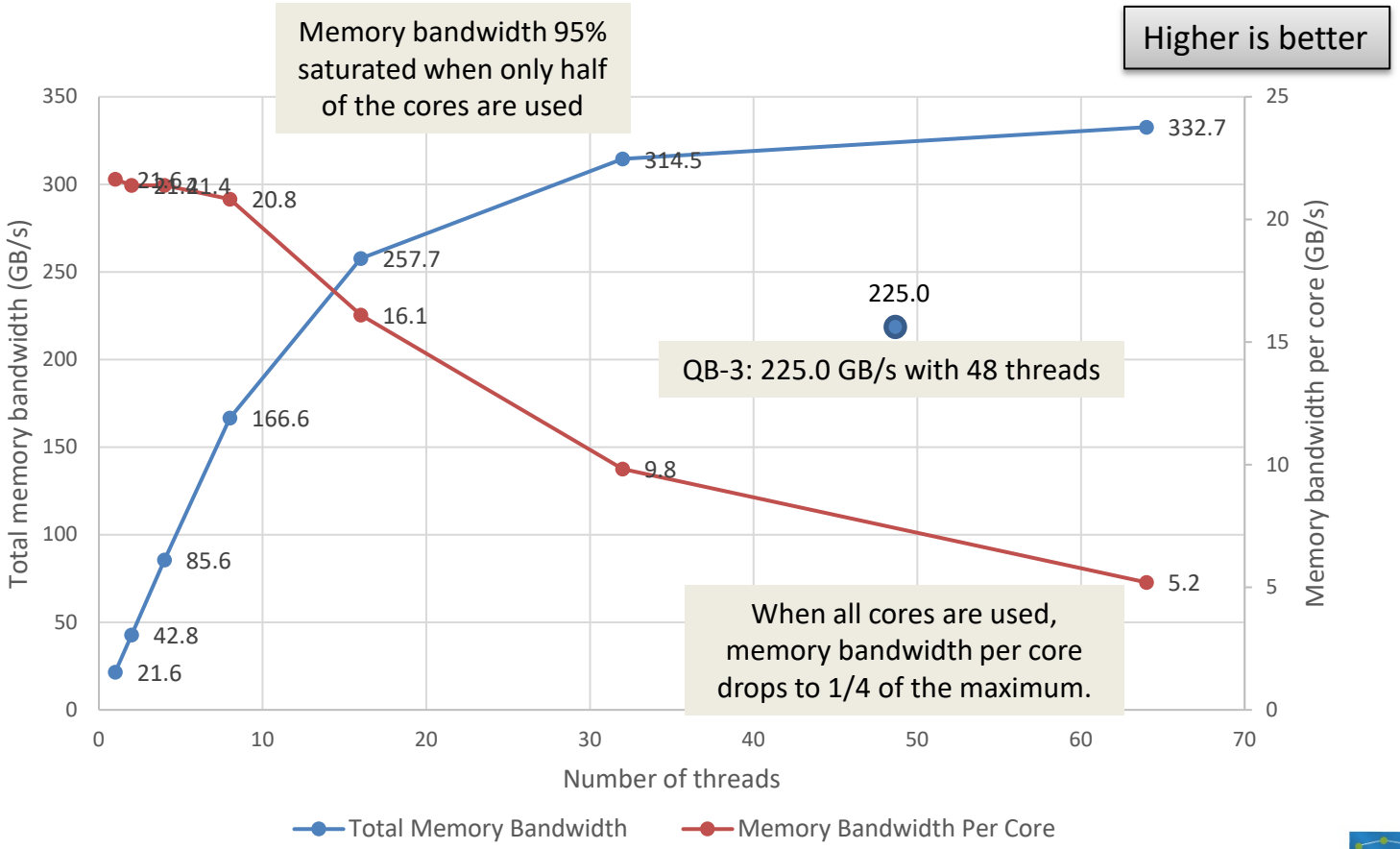
QB-3: 48 cores/node * 2.4×10^9 cycles/second * 32 flop/cycle = 3.69×10^{12} flops

Theoretical speedup = 1.4

STREAM Benchmark

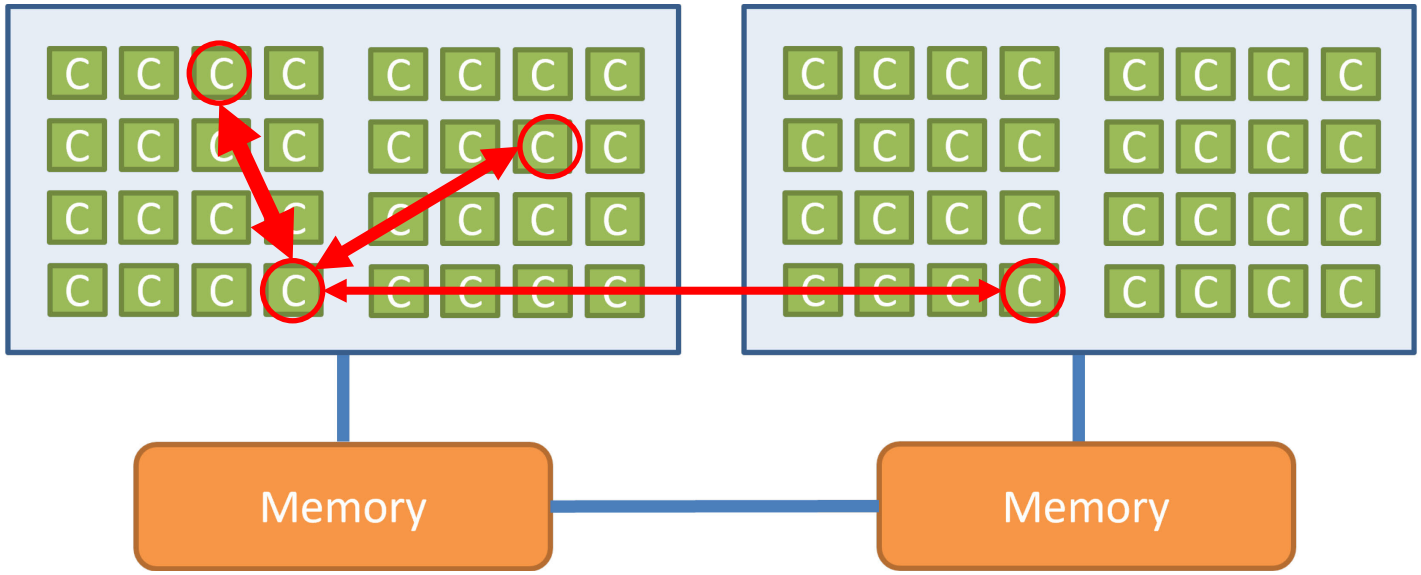
- The de facto industry standard benchmark in HPC domain for the measurement of sustainable memory bandwidth (in GB/s).

STREAM Benchmark - Results



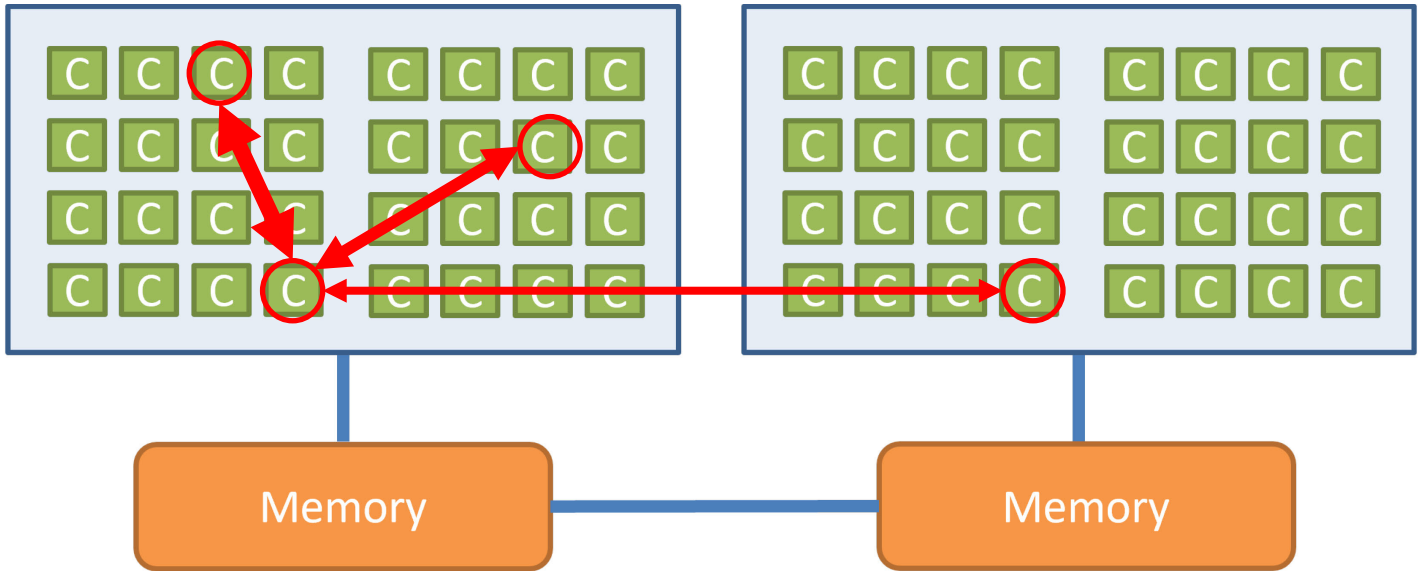
STREAM 5.10
Array size = 160 MB
Intel 2021.5.0 with "-O3 –xCORE-AVX512"
OMP_PROC_BIND=spread

Thread Affinity



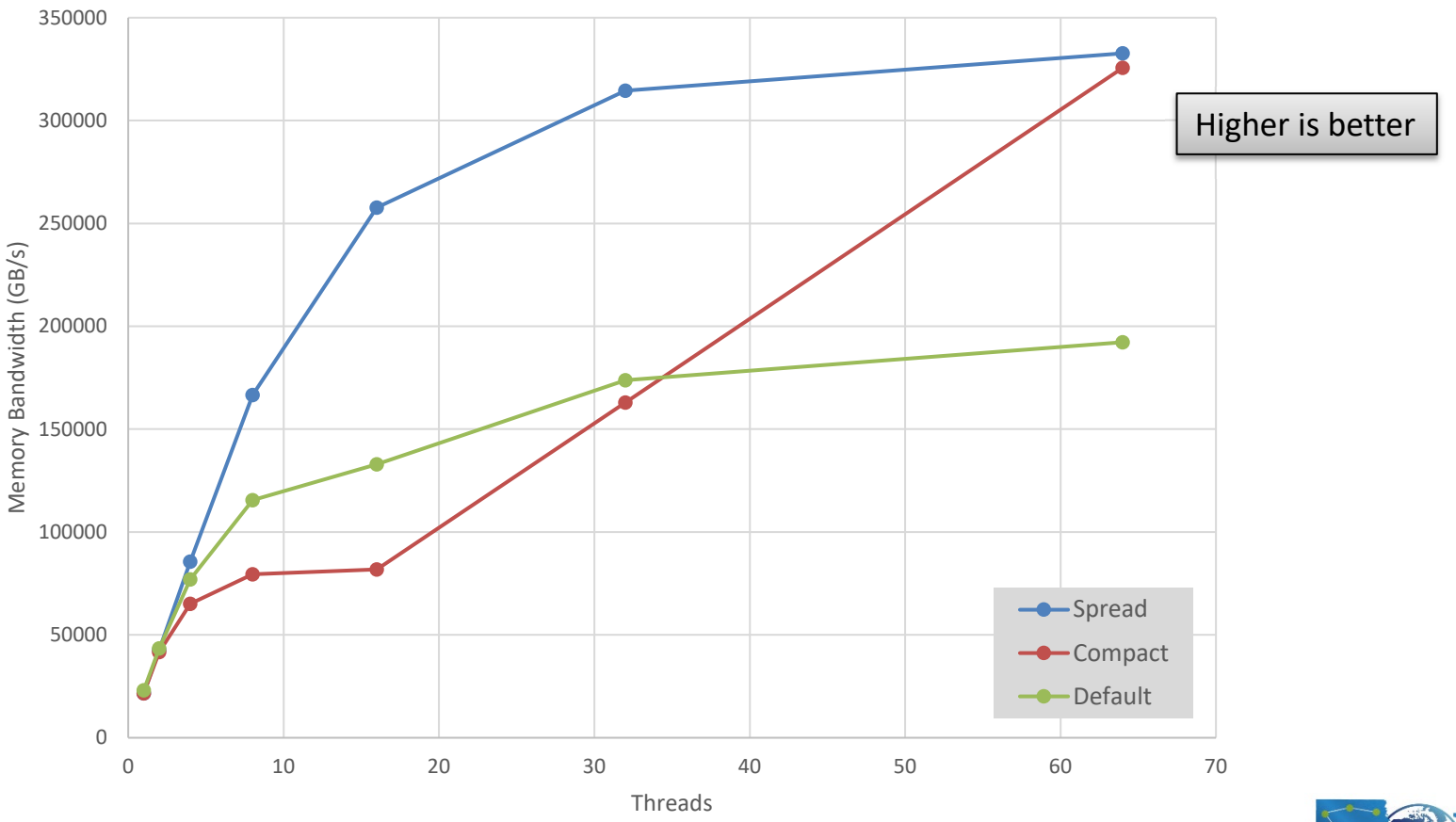
- The 64 cores on a SM-3 node are grouped into 4 sets.
- The data exchange cost is not homogeneous.
- Depending on the data exchange pattern, how the threads are arranged could affect performance significantly.

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"

HPL Benchmark

- High Performance Linpack
  - Standard benchmark for **CPU-bound** HPC applications

    CPU is well fed
- Results
  - QB-4: 3847 GFLOPS per node
  - QB-3: 2452 GFLOPS per node
  - Speedup = 1.57 (compared to 1.4 theoretical)

HPL 2.3 (Intel MKL version)
Problem size = 90% installed memory
Intel 2021.5.0 with "-O3 –xCORE-AVX512 "

HPCG Benchmark

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

CPU is always hungary

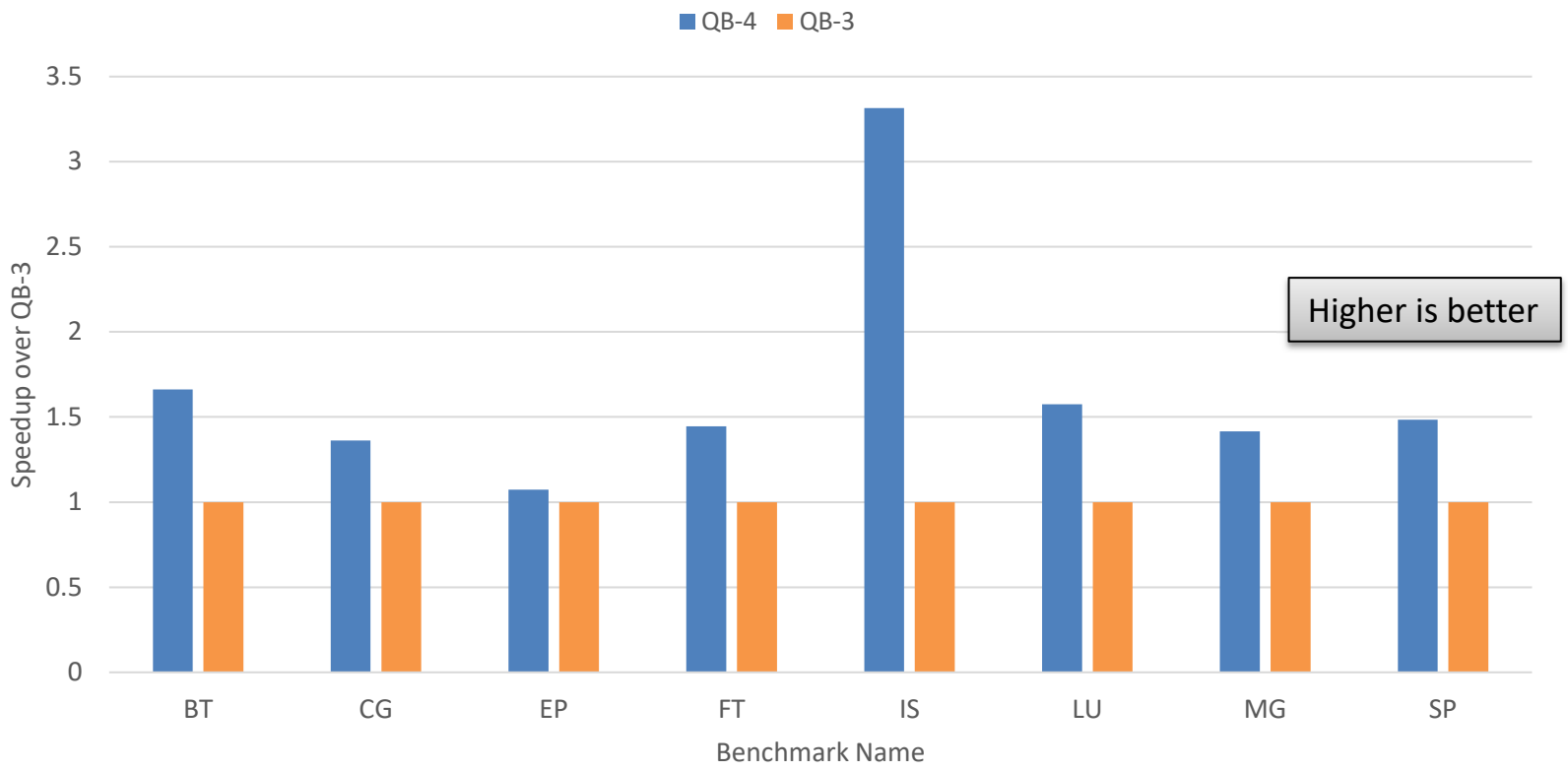
- Results
  - QB-4: 55.0 GFLOPS per node
  - QB-3: 33.1 GFLOPS per node
  - Speedup: 1.66 (compared to 1.4 theoretical)

HPCG 3.0 (Intel MKL version)
Problem size = 336 336 336
Intel 2021.5.0 with "-O3 –xCORE-AVX512 "

NPB Benchmark Suite

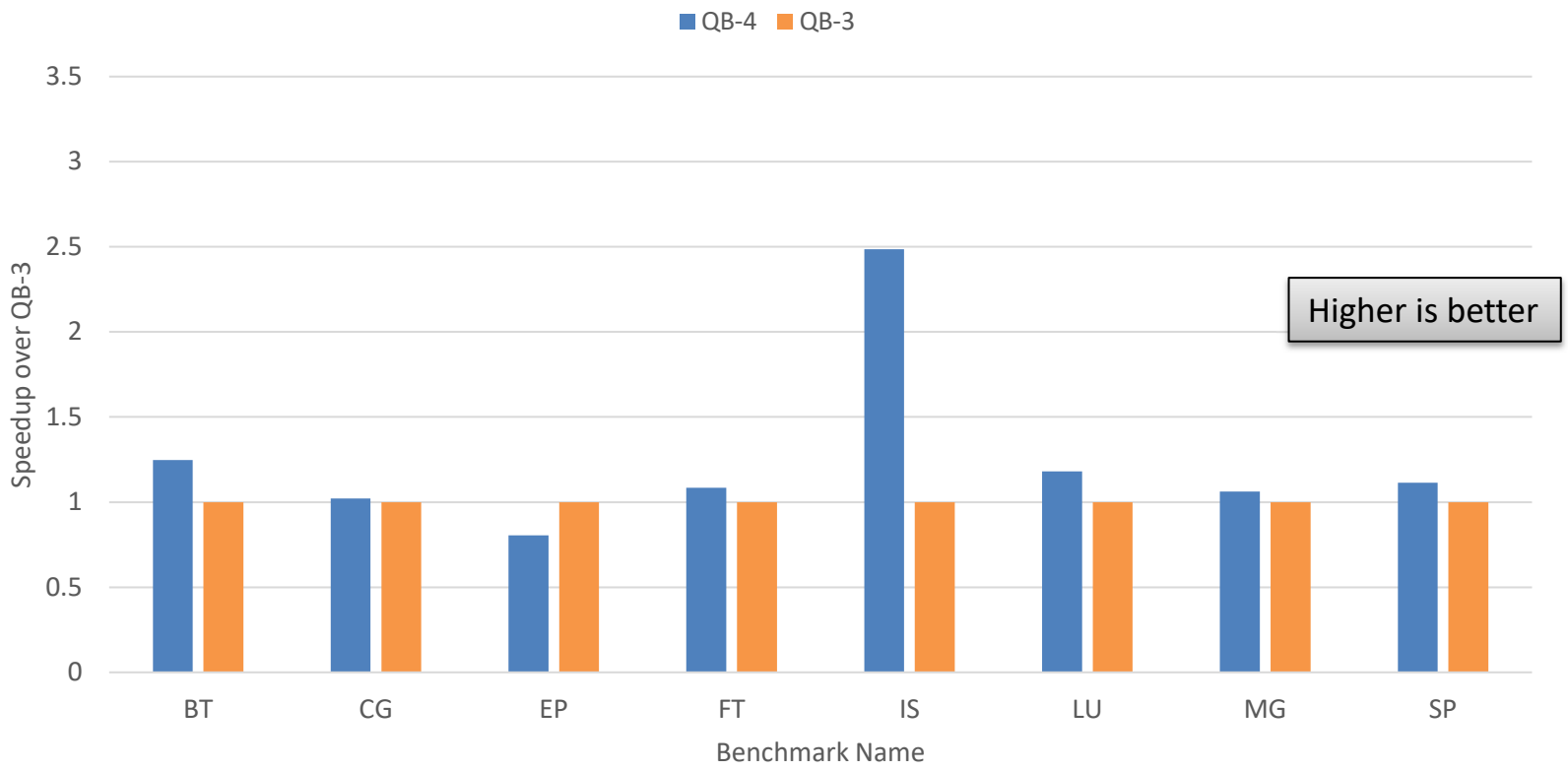
- NAS Parallel Benchmarks
  - a small set of programs derived from computational fluid dynamics applications
- Five kernels and three pseudo-applications
  - IS - Integer Sort, random memory access
  - EP - Embarrassingly Parallel
  - CG - Conjugate Gradient, irregular memory access and communication
  - 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



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

NPB Benchmarks – Core over Core

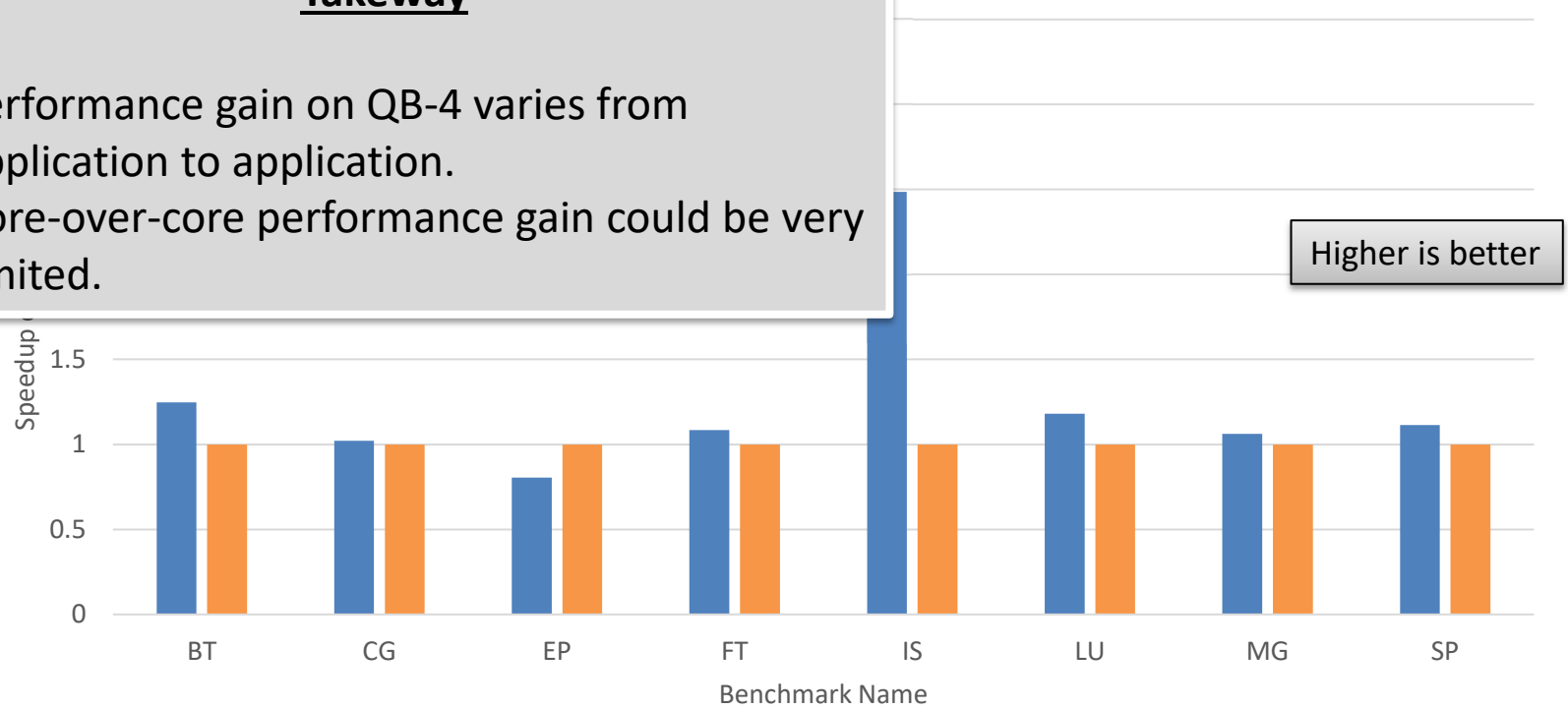


NPB 3.4.2, Class D
Intel 2021.5.0 with "-O3 –xCORE-AVX512"
KMP_AFFINITY=none

NPB Benchmarks – Core over Core

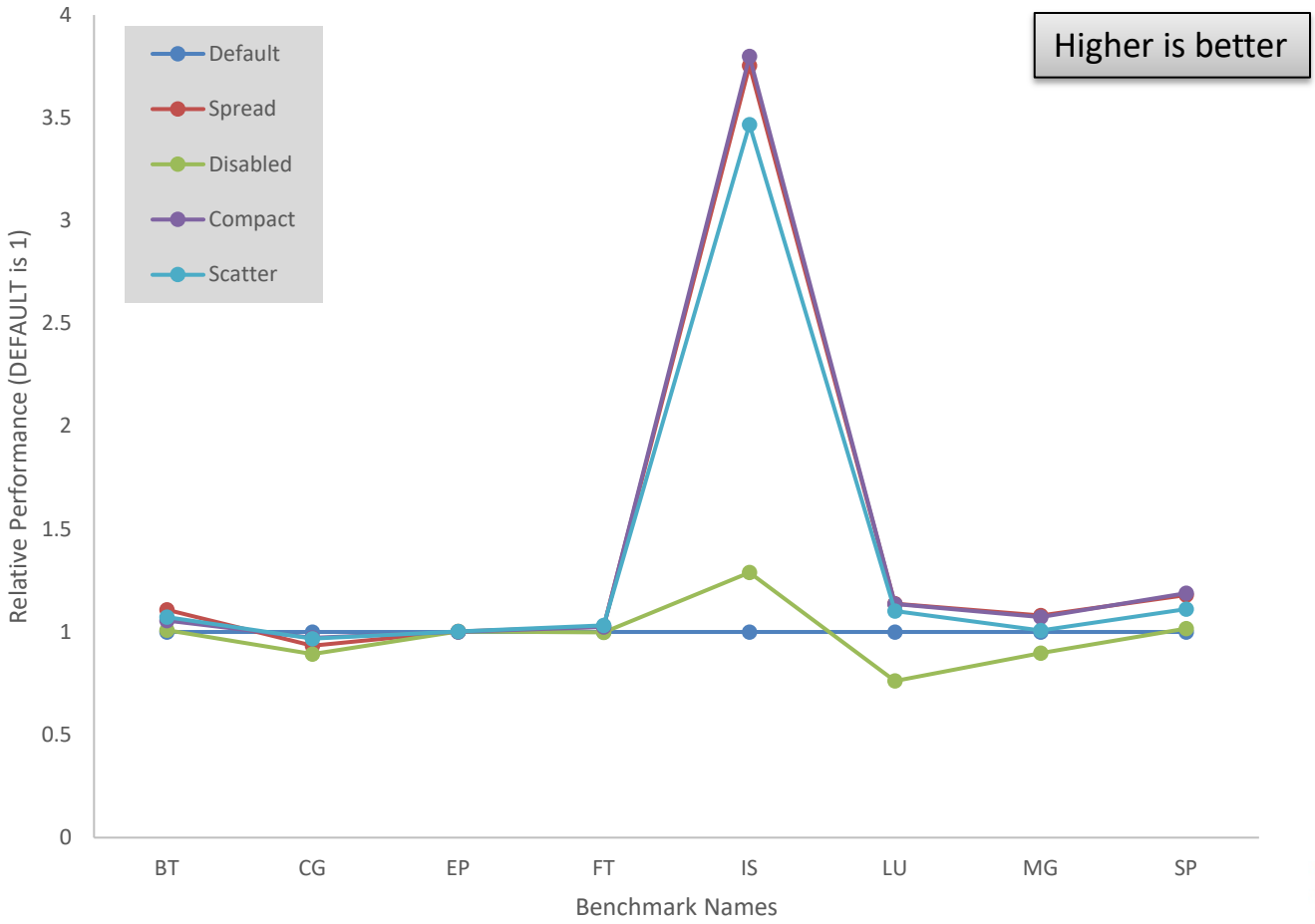
Takeway

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



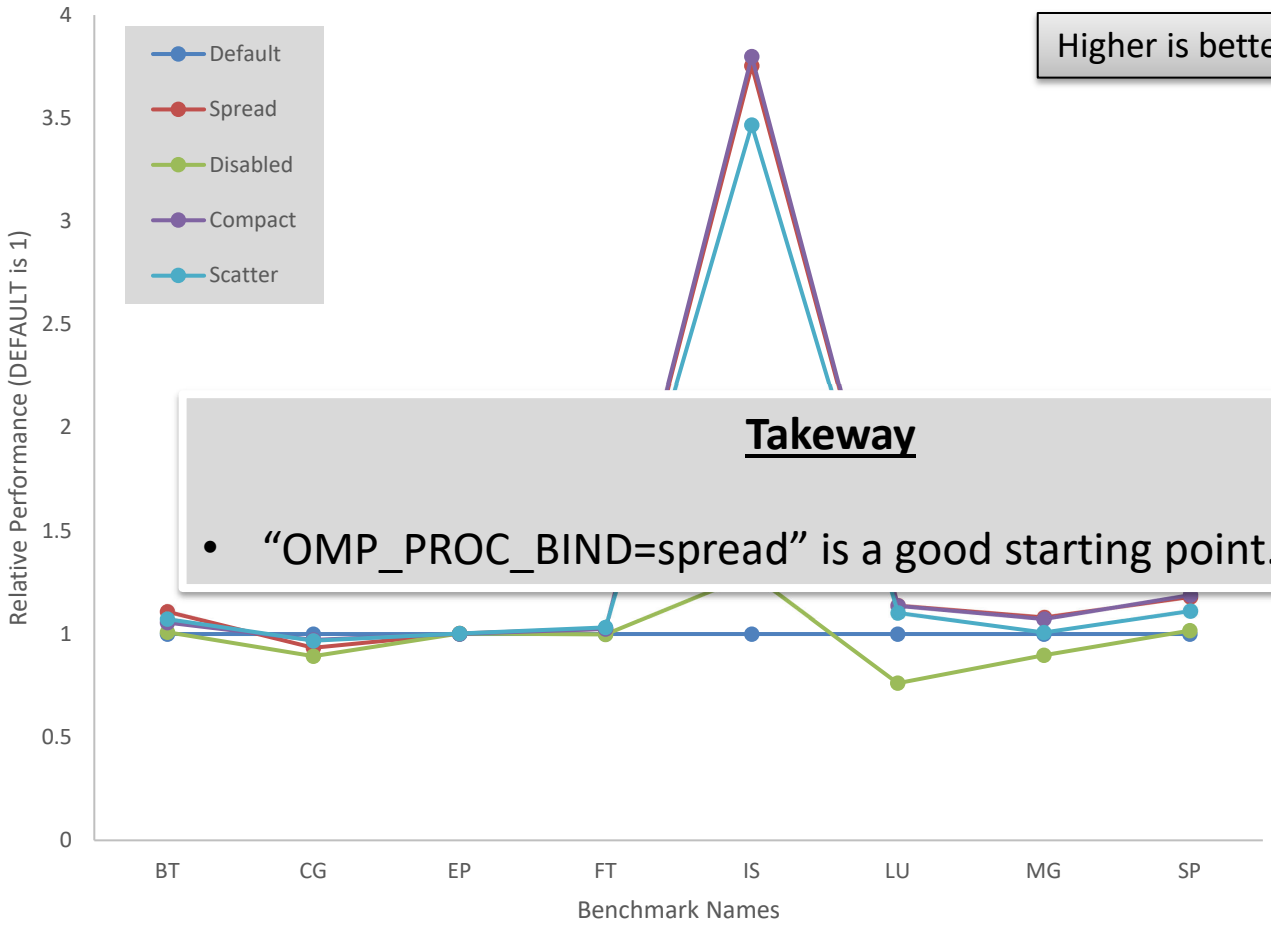
NPB 3.4.2, Class D
Intel 2021.5.0 with "-O3 –xCORE-AVX512"
KMP_AFFINITY=none

NPB Results – Thread Affinity



NPB 3.4.2, Class E
Intel 2021.5.0 with "-O3 –xCORE-AVX512"

NPB Results – Thread Affinity



Takeway

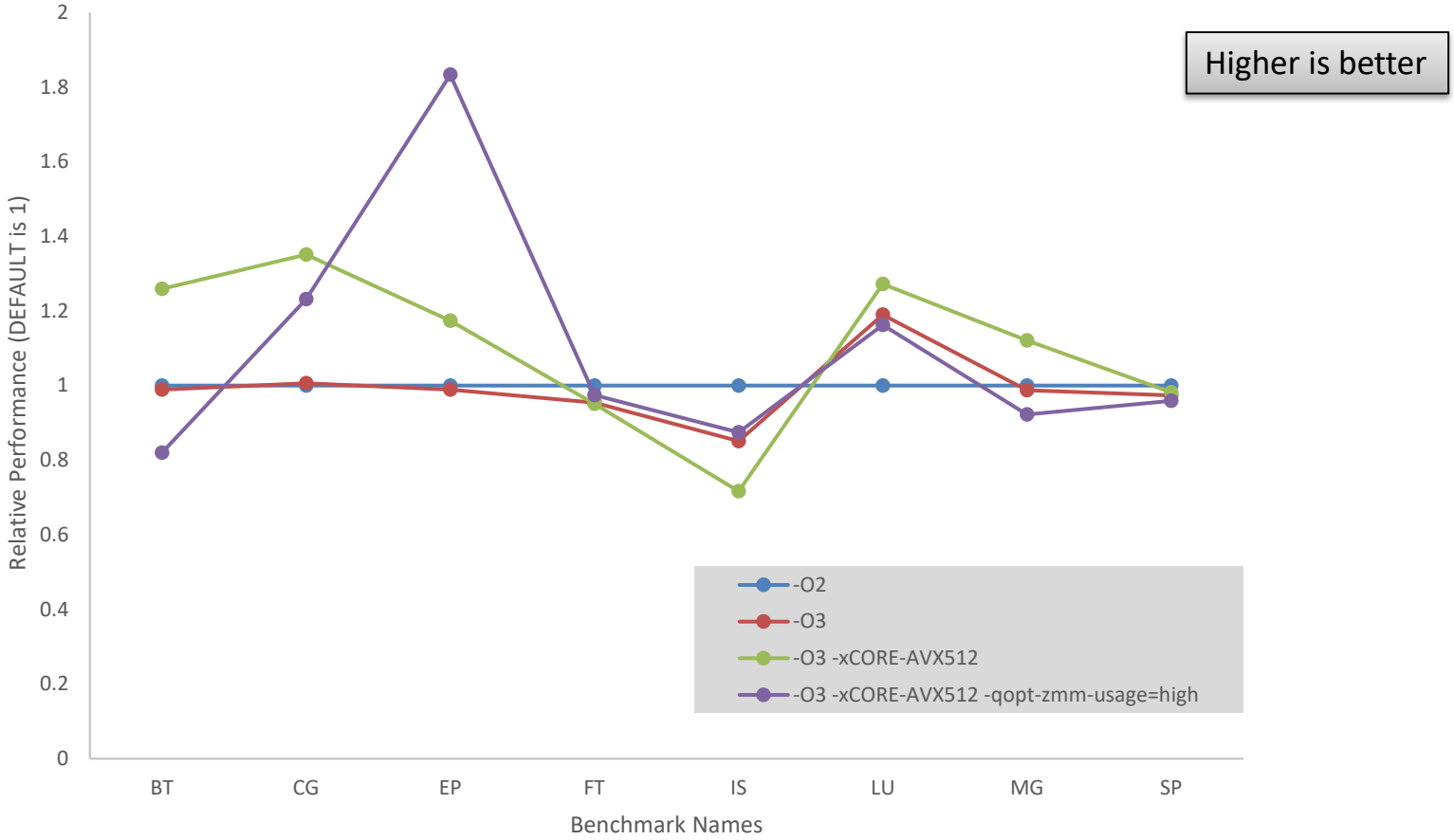
- "OMP_PROC_BIND=spread" is a good starting point.

NPB 3.4.2, Class E
Intel 2021.5.0 with "-O3 –xCORE-AVX512"

Compiler Flags (Intel)

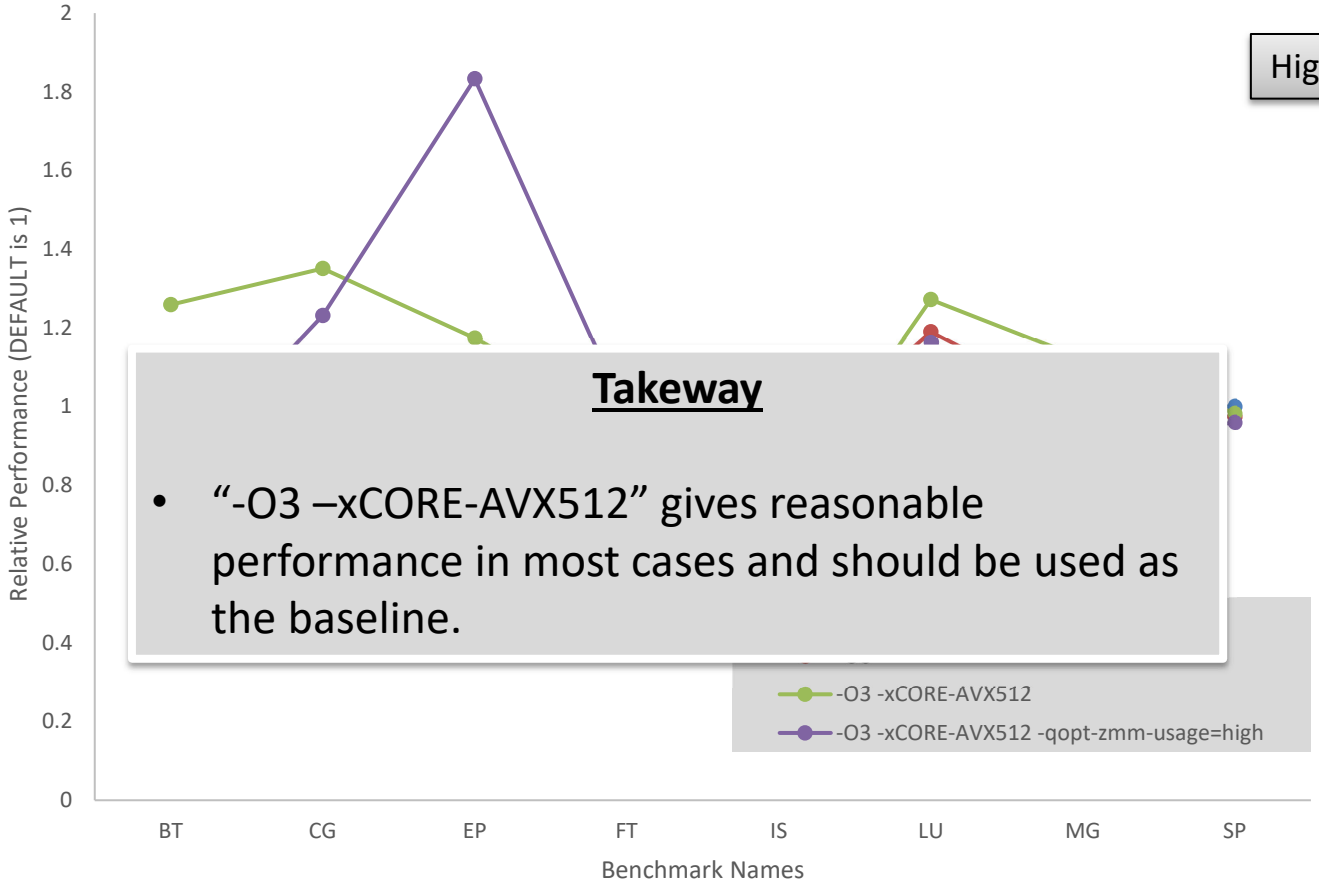
- -O2, -O3
  - Generic, aggregated optimization flags
- -xCORE-AVX512
  - Turns on optimization for the Ice Lake (and SkyLake/Cascade Lake) processor
- -march=icelake-server
  - Turns on optimization specifically for Ice Lake
- -qopt-zmm-usage=high
  - Improves performance for some codes

NPB Results – Compiler Flags



NPB 3.4.2, Class E
Intel 2021.5.0
KMP_AFFINITY=default

NPB Results – Compiler Flags



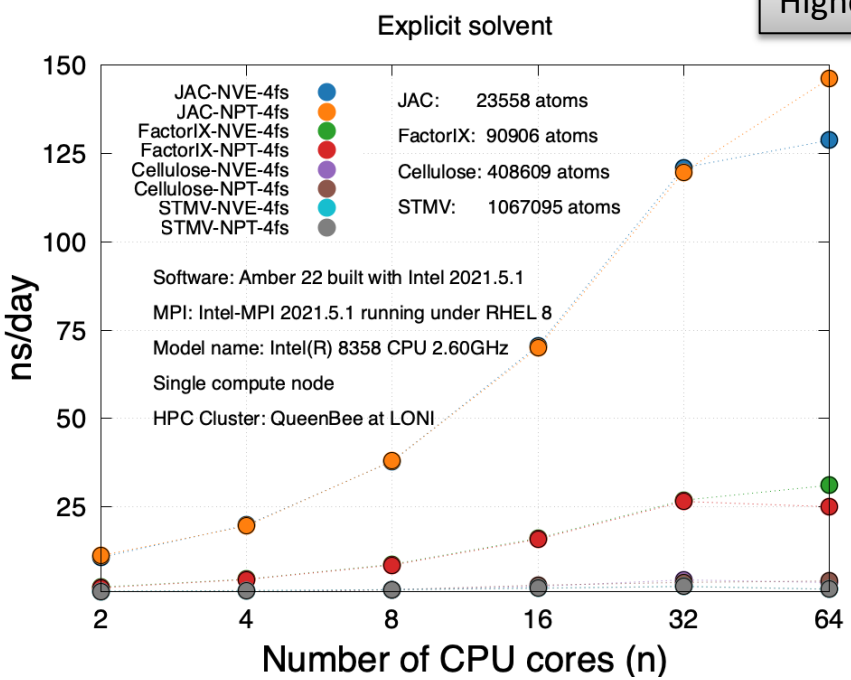
Takeway

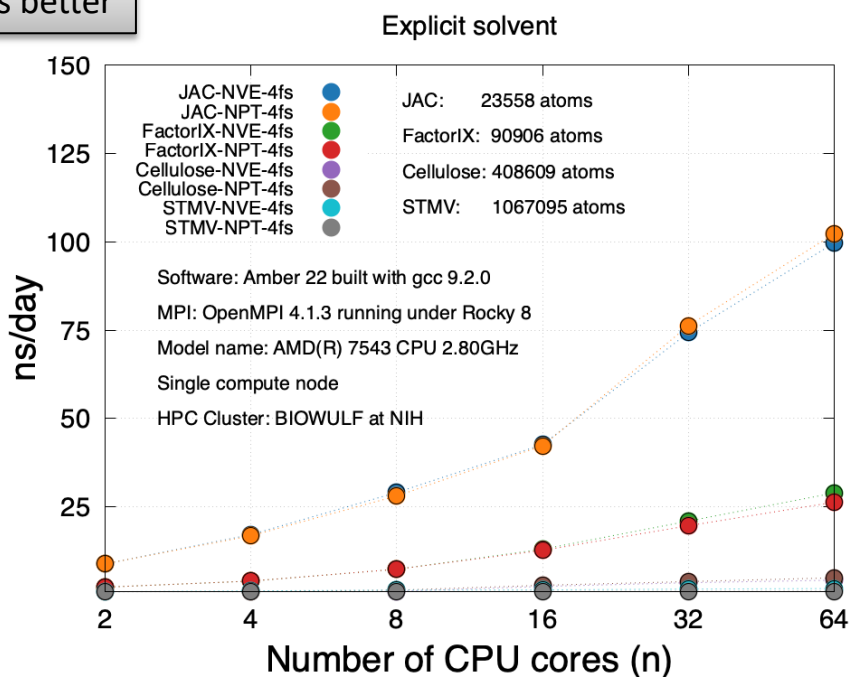
- "-O3 –xCORE-AVX512" gives reasonable performance in most cases and should be used as the baseline.

NPB 3.4.2, Class E
Intel 2021.5.0
KMP_AFFINITY=default

Amber 22

Higher is better

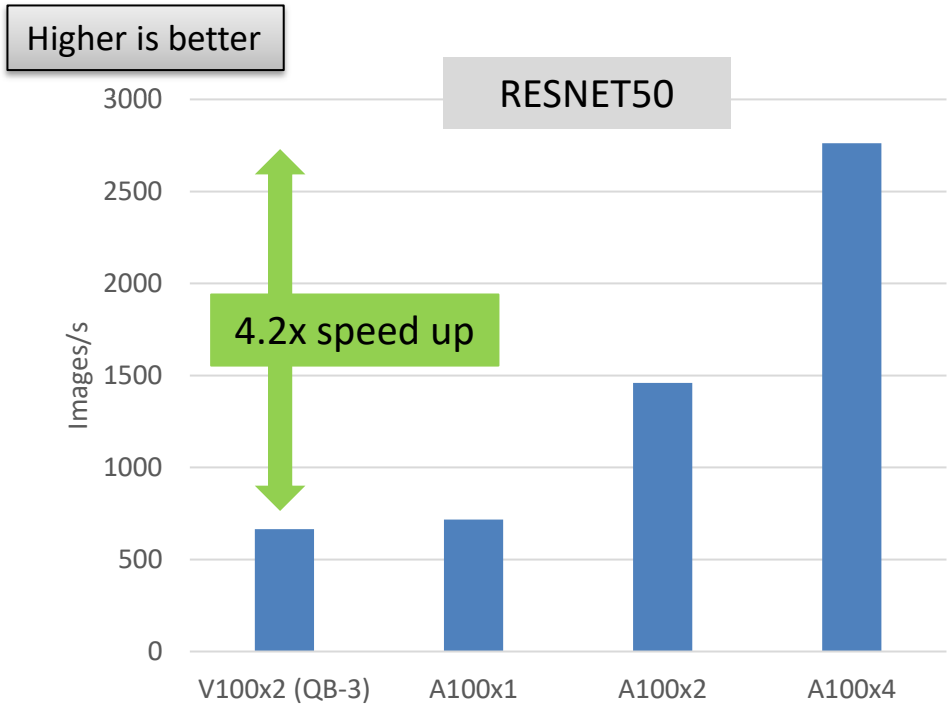




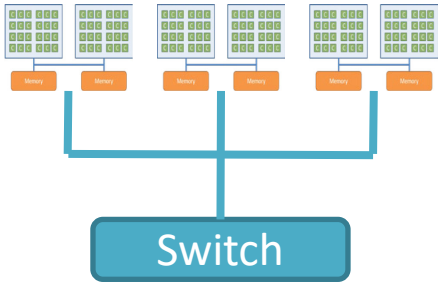
GPU Performance

GPU	QB-3	QB-4
Model	V100 PCIe	A100 PCIe
On-board memory	32GB	80GB
Processing power (DP)	7 TFLOPS	9.7 TFLOPS
NVLink	No	Yes
InterGPU bandwidth	~20 GB/s	~520 GB/s*

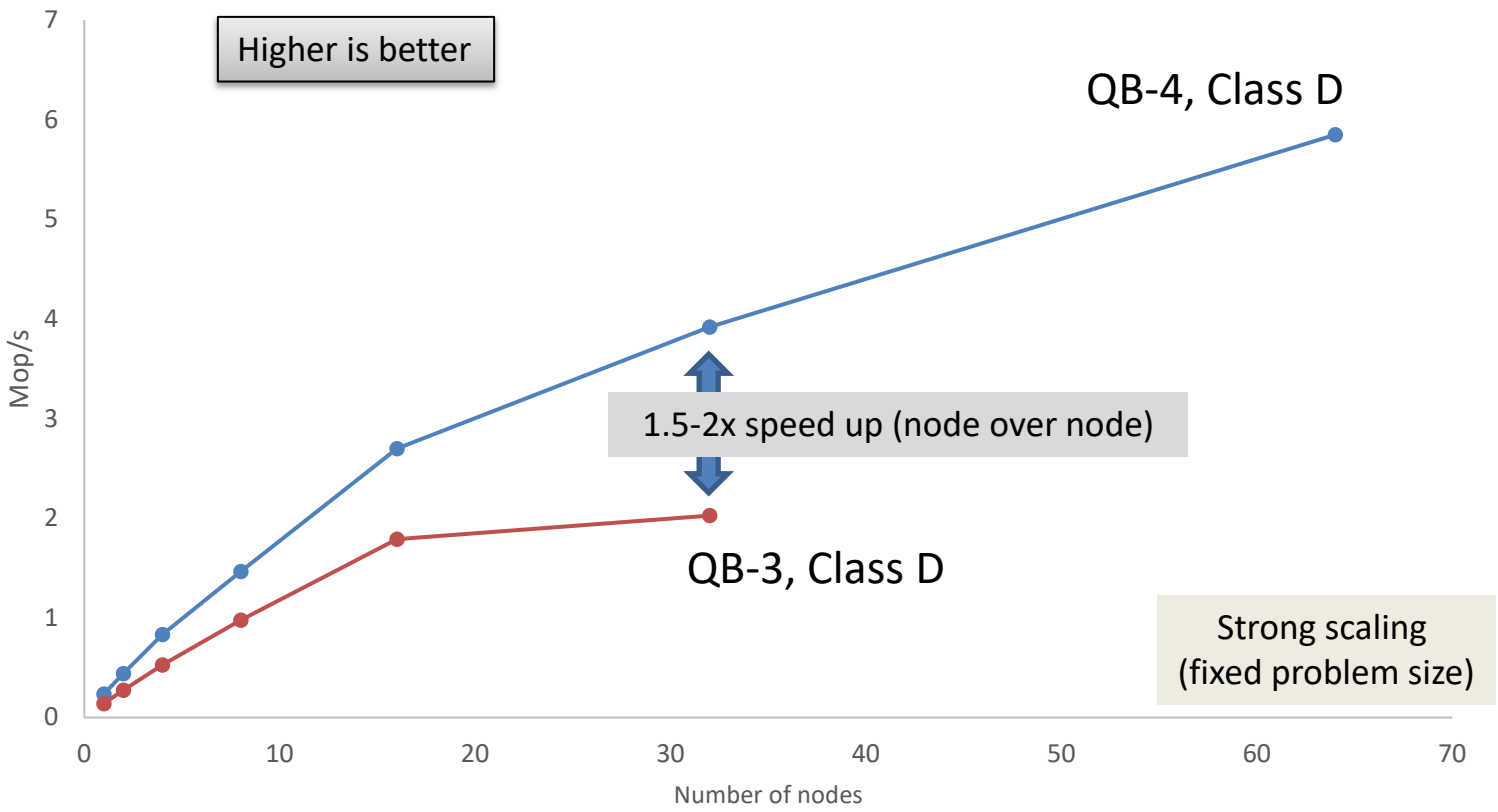
** On nodes with 4 GPU's, this is the bandwidth between device 0 and 1, and device 2 and 3*



Multi-node Performance

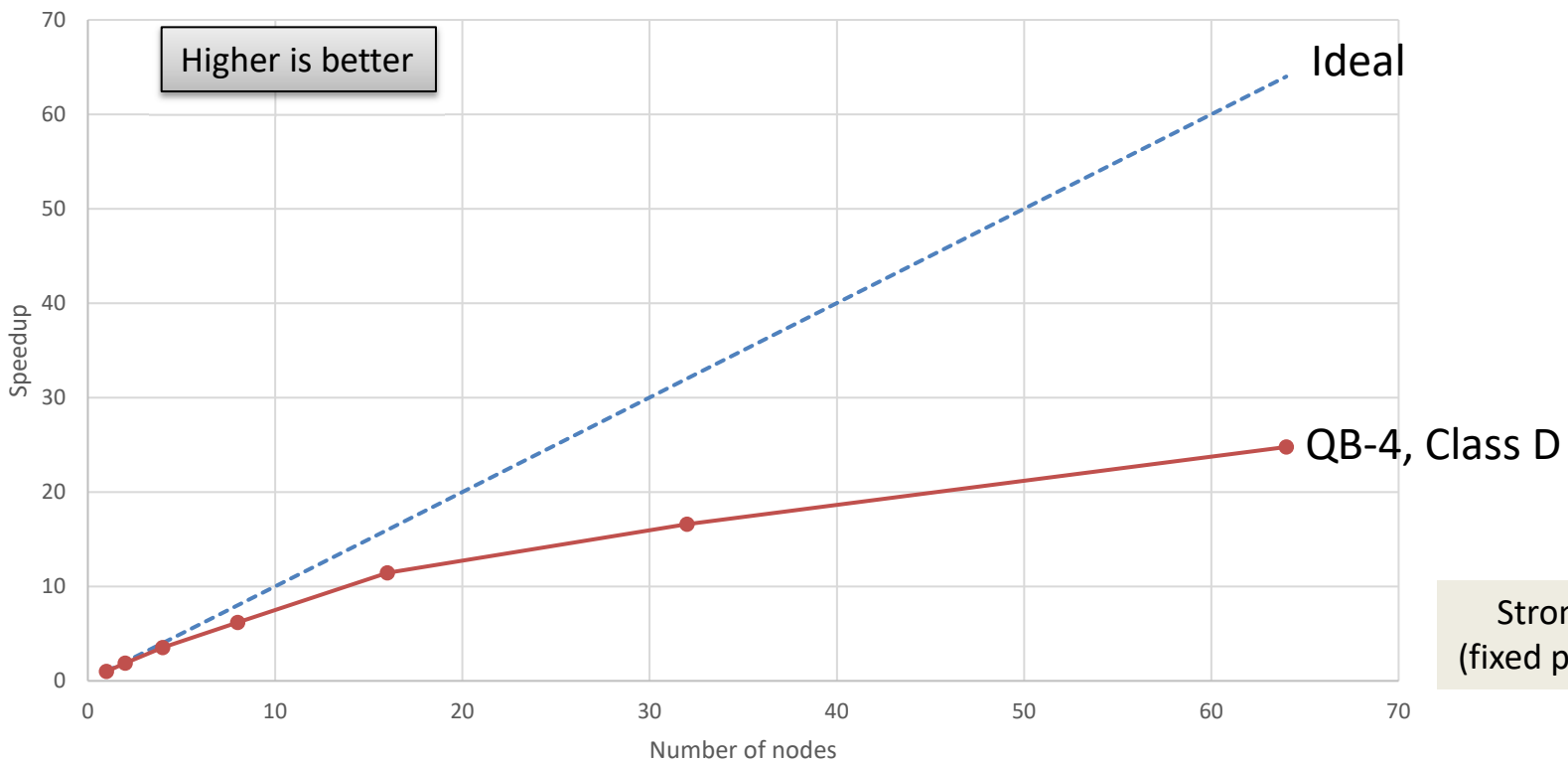


Pure MPI - NPB LU Benchmark



NPB 3.4.2, Class D
Intel MPI 2021.5.1
Intel 2021.5.0 with "-O3 –xCORE-AVX512"

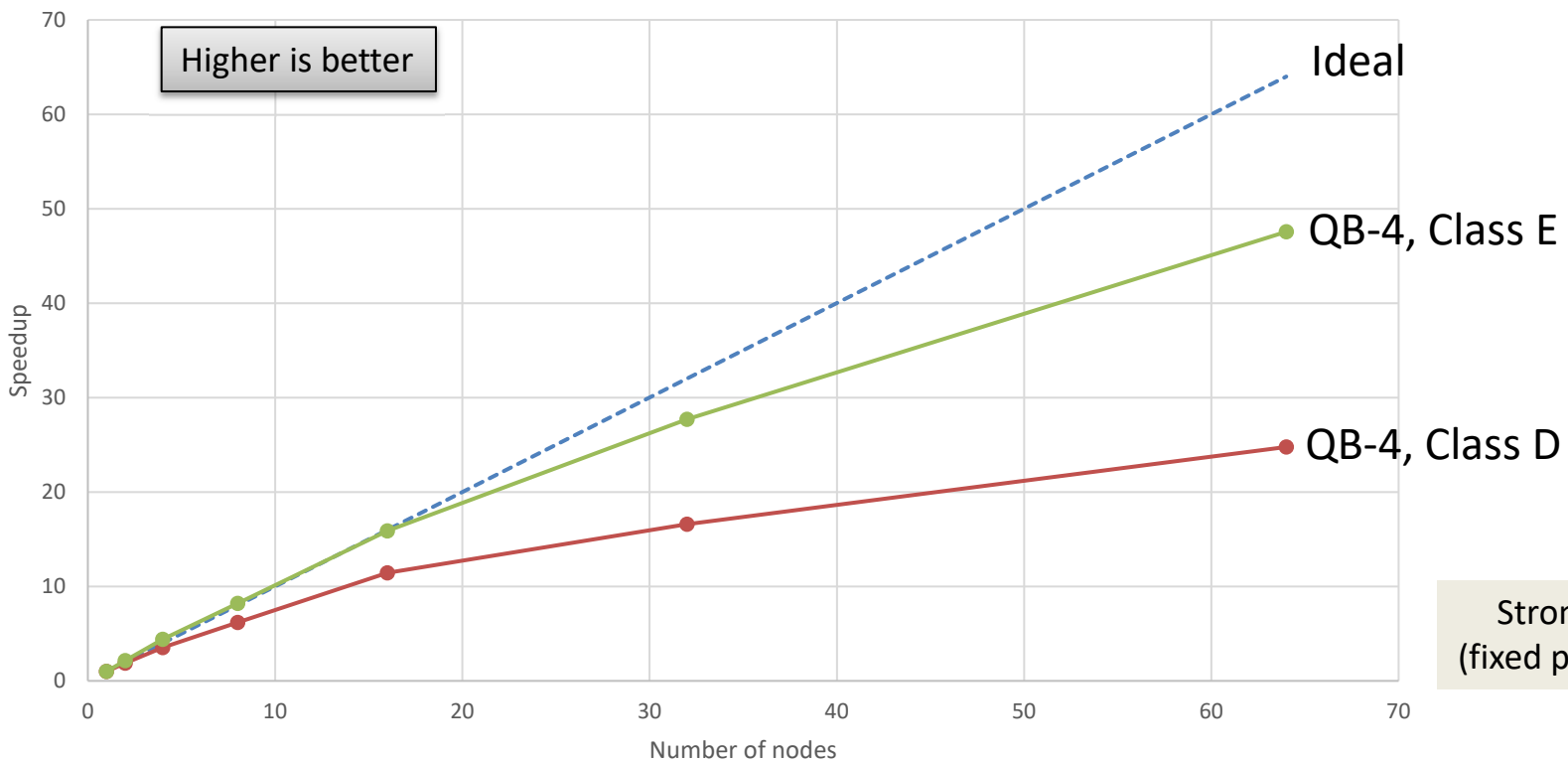
Pure MPI - NPB LU Benchmark



Strong scaling
(fixed problem size)

NPB 3.4.2, Class D and E
Intel MPI 2021.5.1
Intel 2021.5.0 with "-O3 –xCORE-AVX512

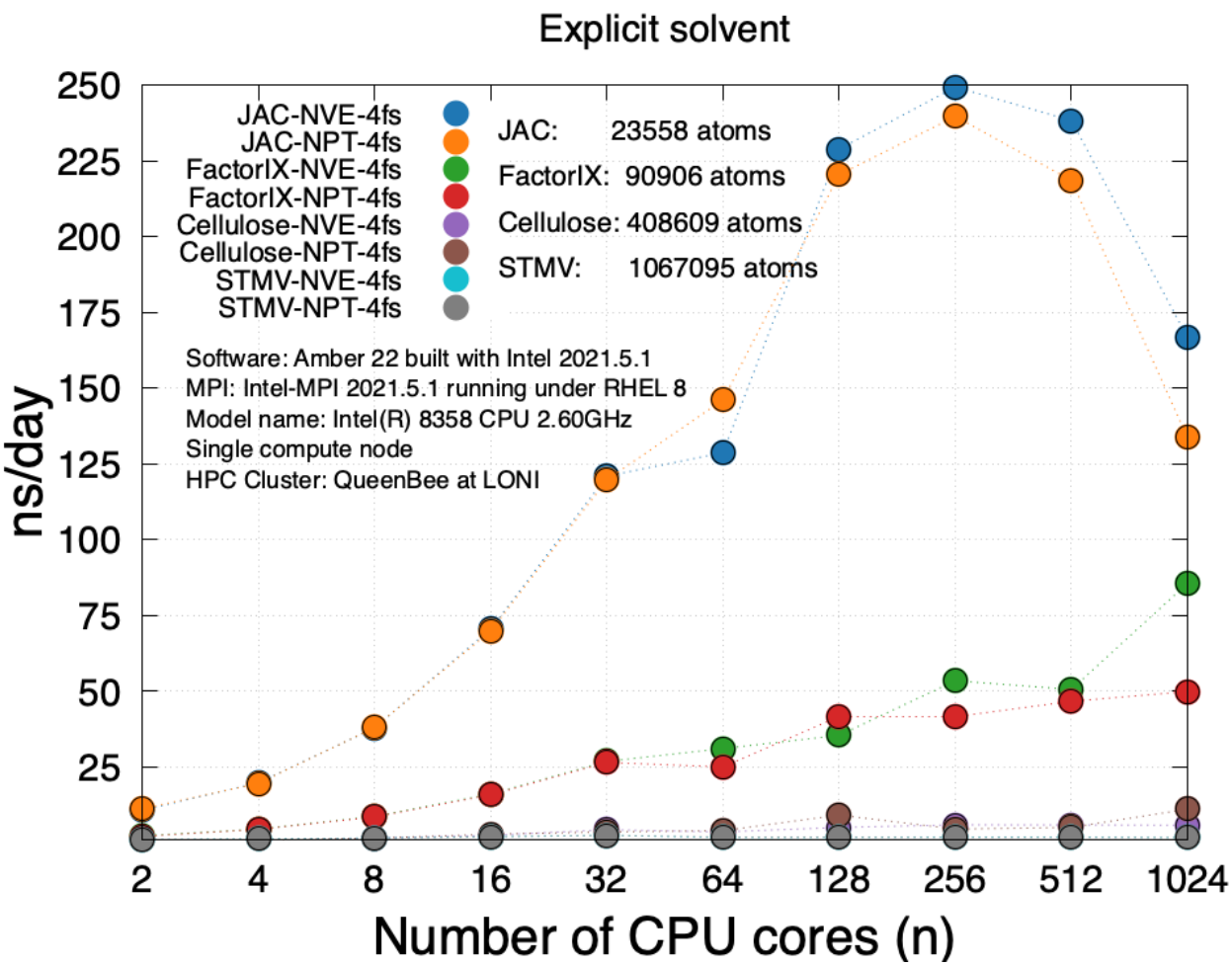
Pure MPI - NPB LU Benchmark



Strong scaling
(fixed problem size)

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

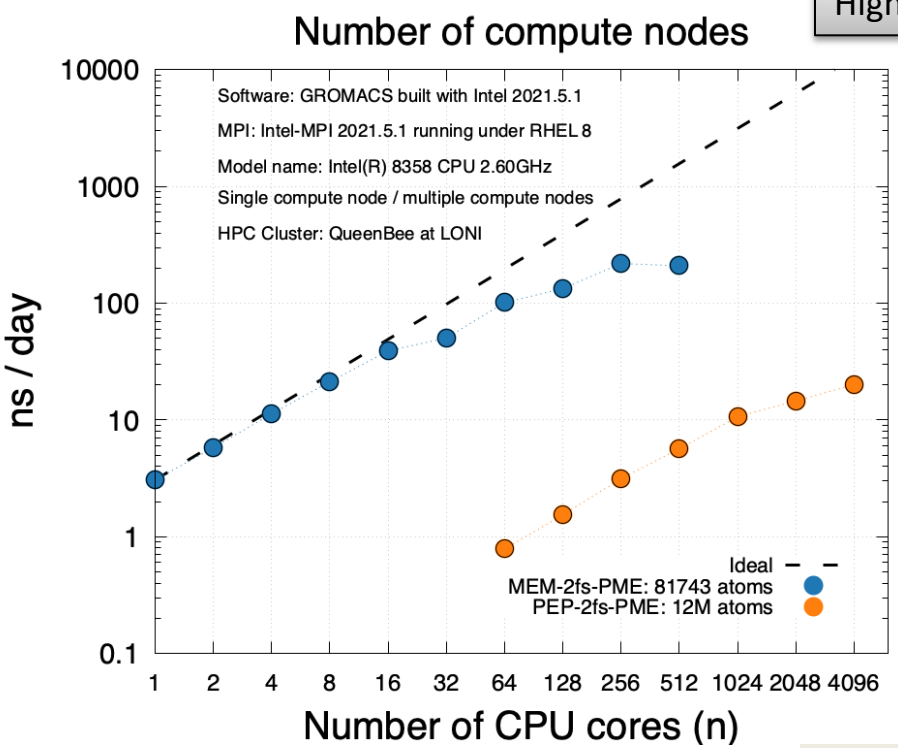


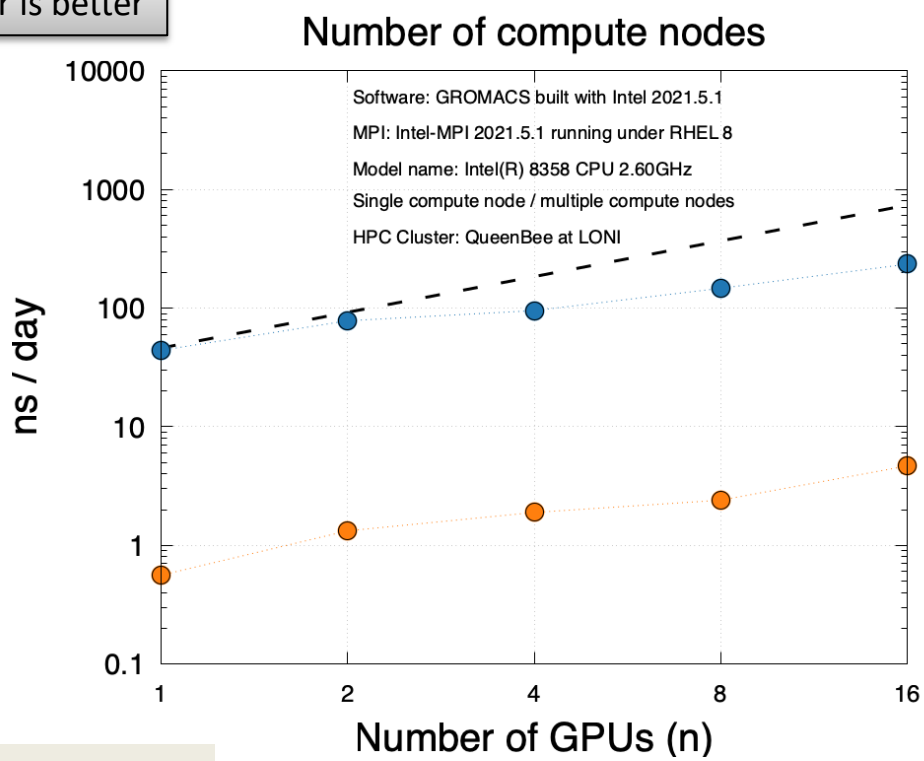
Higher is better

Strong scaling
(fixed problem size)

Pure MPI - GROMACS

Higher is better

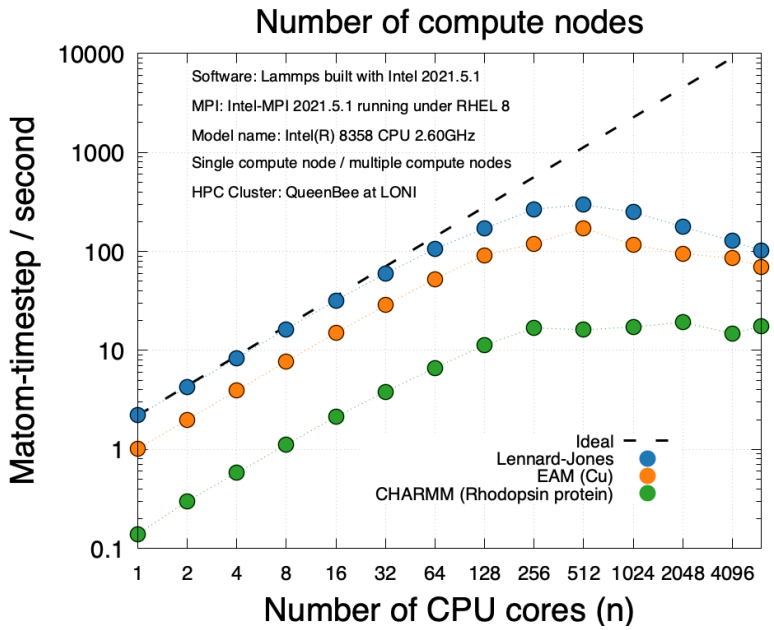


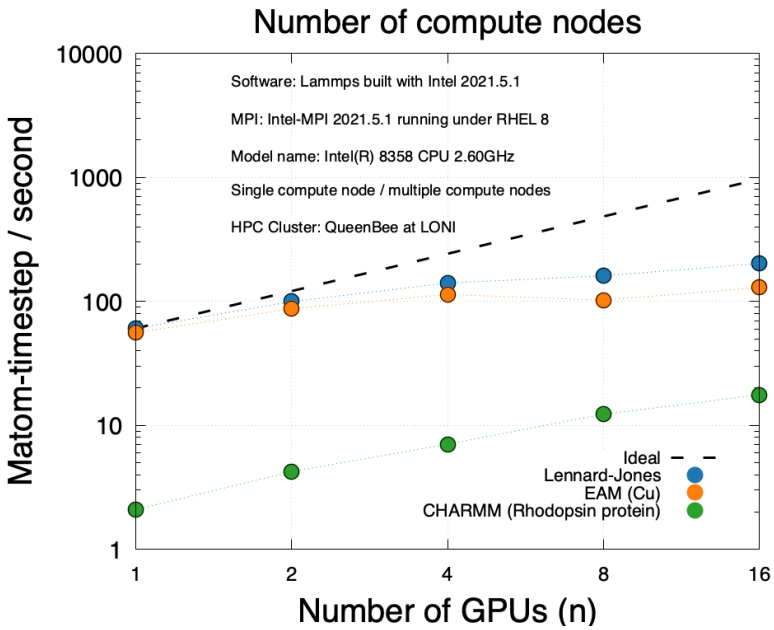


Strong scaling
(fixed problem size)

Pure MPI – LAMMPS

Higher is better

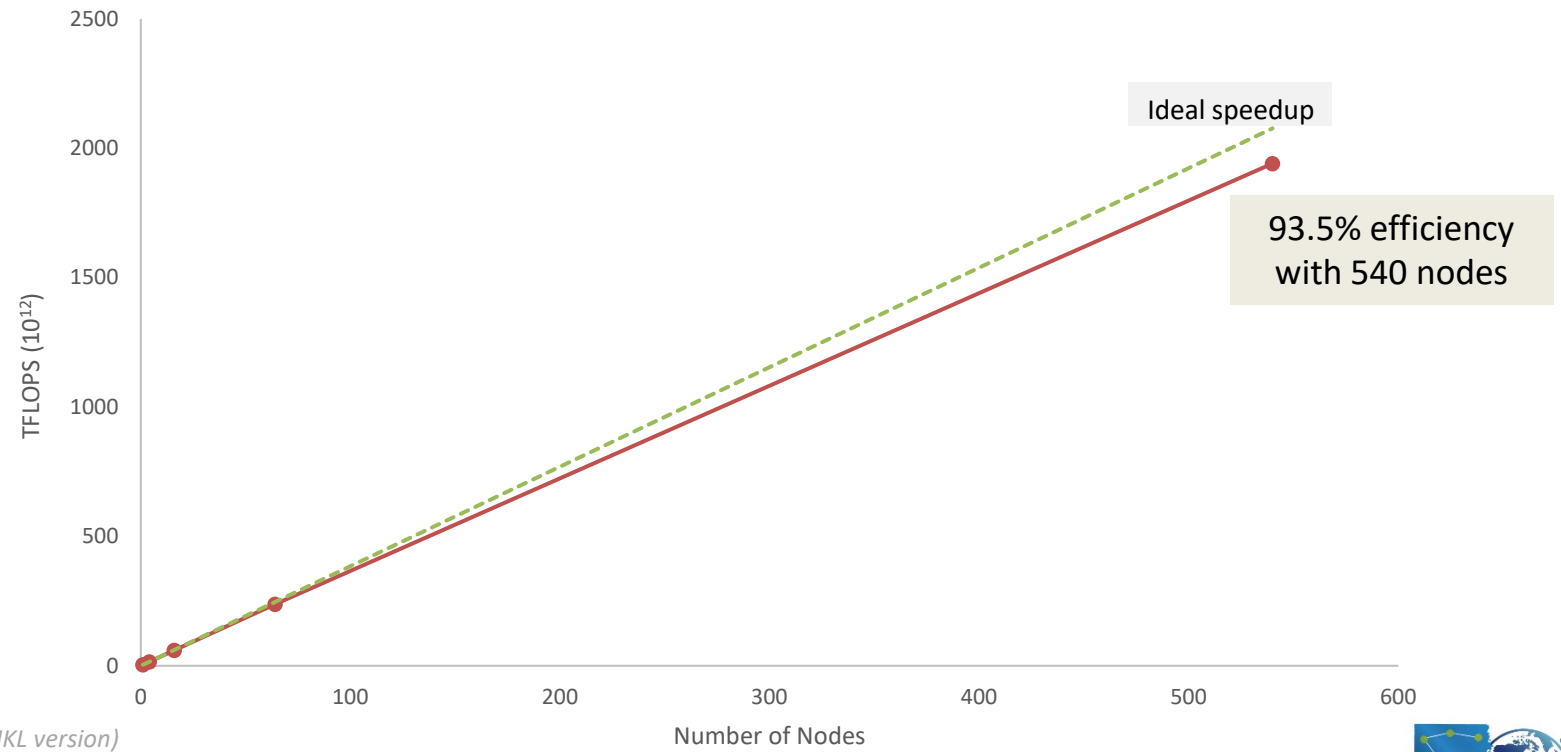




Strong scaling
(fixed problem size)

Hybrid – HPL

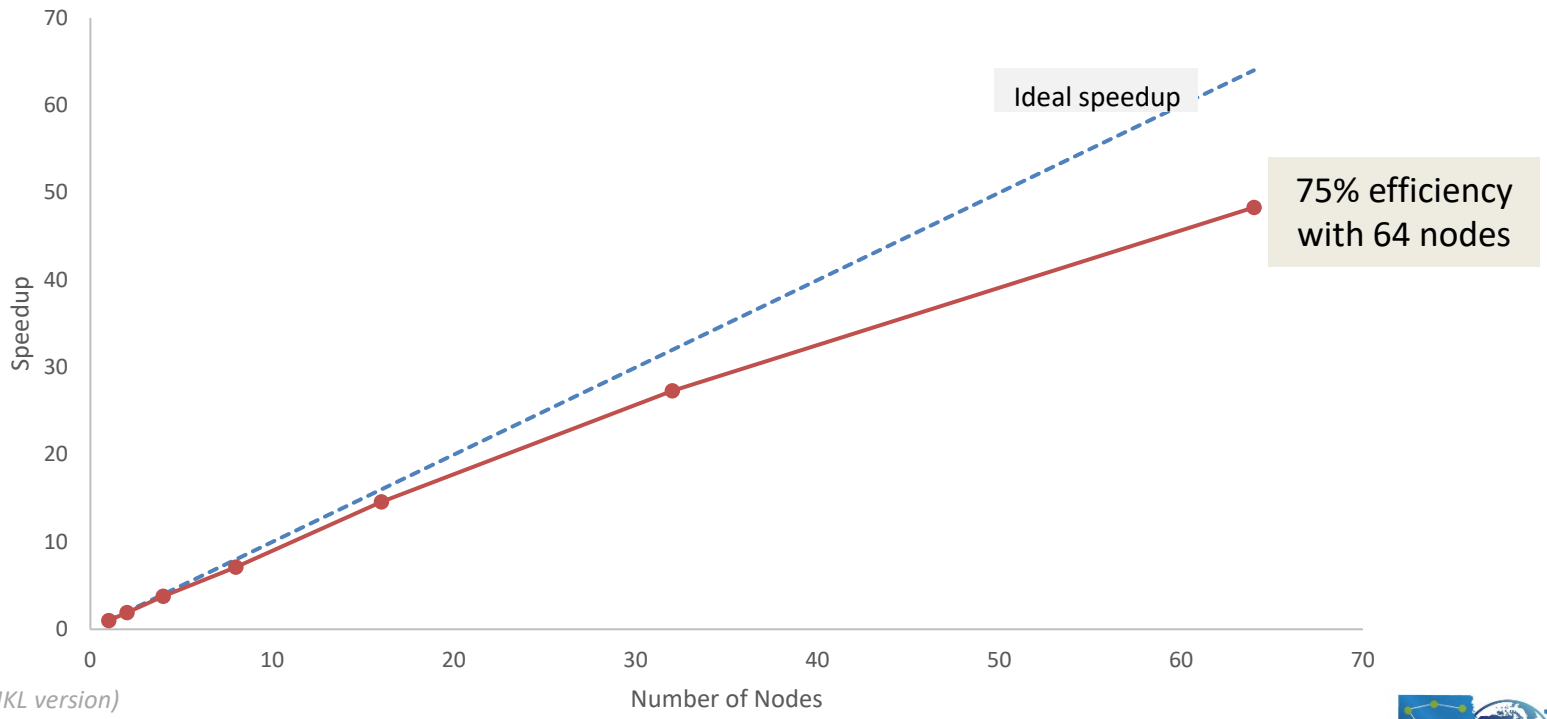
Weak scaling (problem size increases with core count)
1 MPI process per node * 64 threads per MPI process



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"

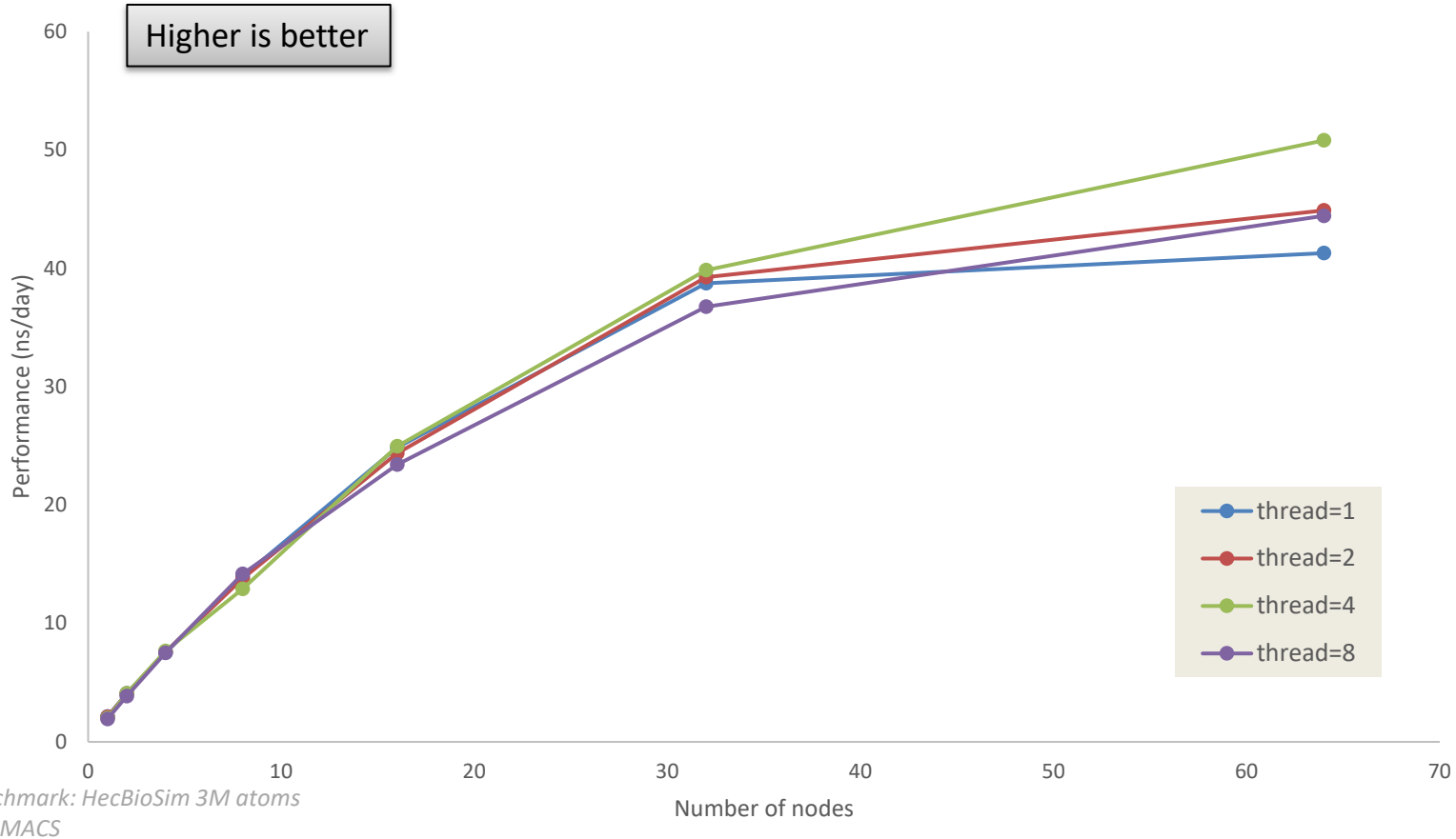
Hybrid - HPCG

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



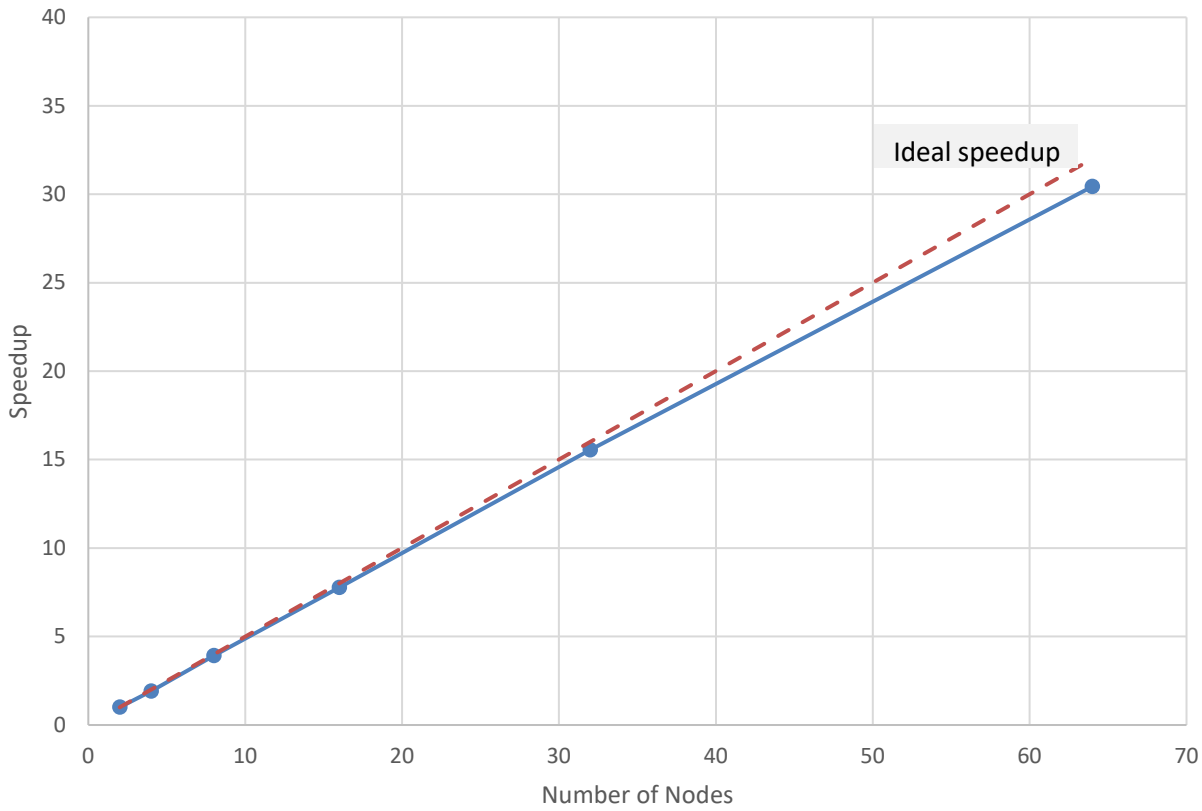
HPCG 3.0 (Intel MKL version)
Problem size = 336 336 336
Intel MPI 2021.5.1
Intel 2021.5.0 with "-O3 –xCORE-AVX512 -qopt-zmm-usage=high"

Hybrid - GROMACS



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

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
  - Use K

Avoid excessive I/O!!!

- 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
  - Use k
- 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!