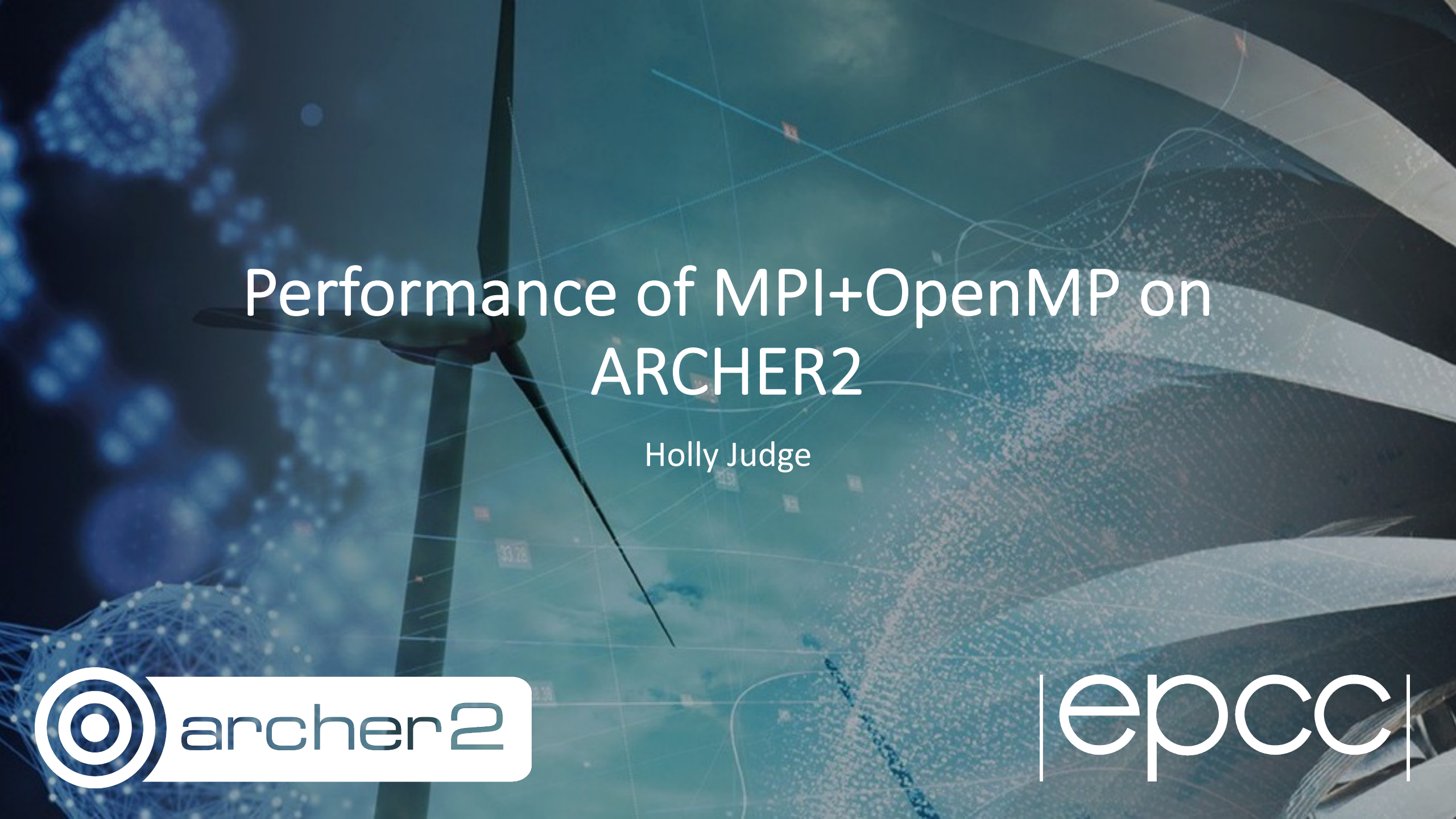


Performance of MPI+OpenMP on ARCHER2

Holly Judge

archer2

epcc

Reusing this material

Partners


UKRI
Engineering and Physical Sciences Research Council
Natural Environment Research Council


epcc


THE UNIVERSITY of EDINBURGH


Hewlett Packard
Enterprise

Introduction

- HPC applications are typically run with MPI
  - For some applications MPI+OpenMP can be used as an alternative to MPI
  - This may achieve better performance than MPI – but this depends
- Using MPI+OpenMP may be good for taking advantage of the resources on ARCHER2 nodes
  - 128 cores per node
  - Different to ARCHER – 24 cores per node
- When can we get good performance with MPI+OpenMP?
  - Which applications?
  - What sort of systems/test cases?
  - How many threads?

Outline

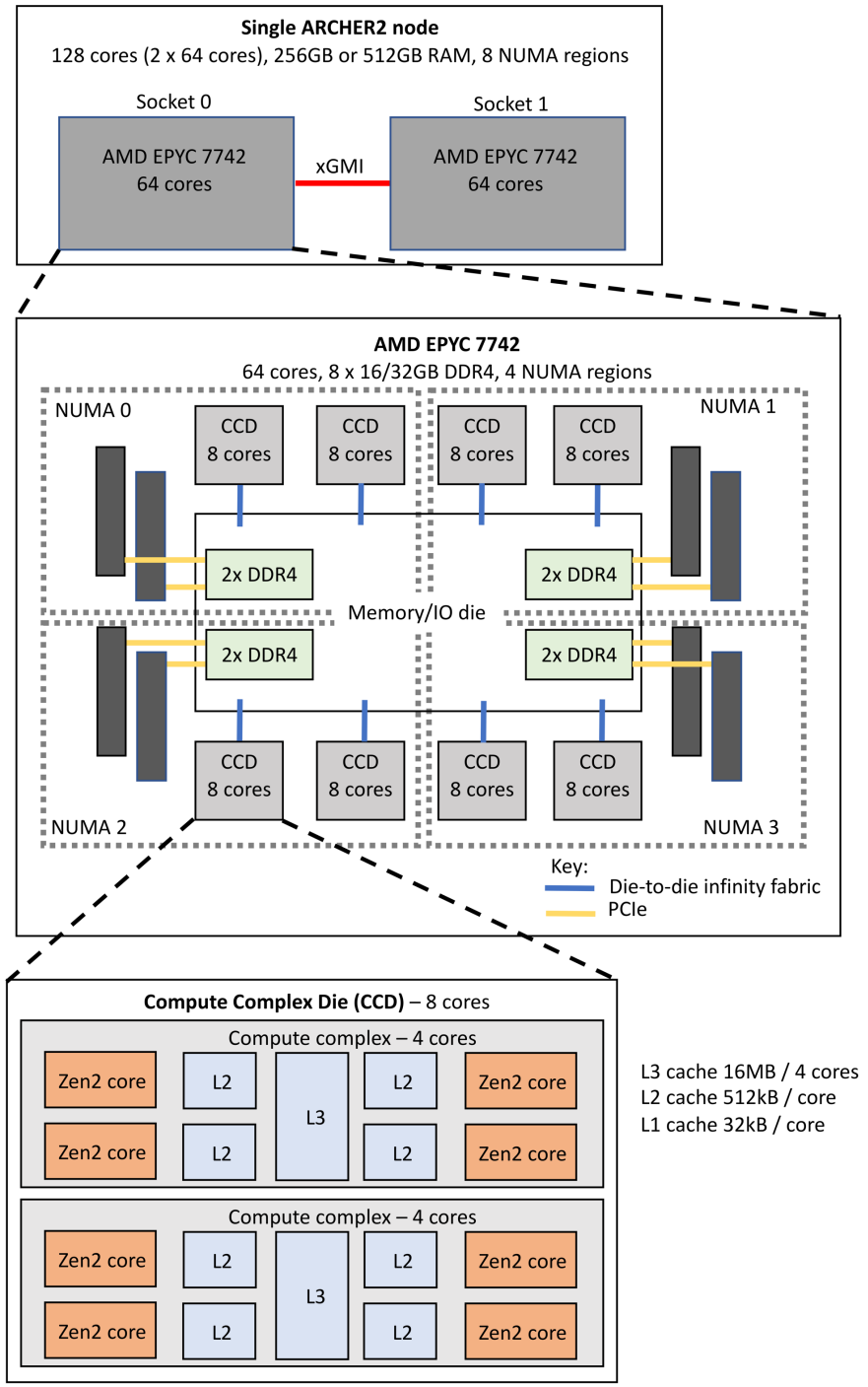
- ARCHER2
- MPI+OpenMP overview
  - How is it different to MPI?
  - How can it be used on ARCHER2?
- MPI+OpenMP performance investigation on ARCHER2
  - Applications investigated
  - Test cases
  - Results
- How can users benefit from MPI+OpenMP on ARCHER2?

ARCHER2

- UK national service used for scientific research
- HPE EX system
- 5,860 nodes (750,080 cores)
- AMD EPYC processors
- HPE Slingshot interconnect

Node-level architecture

- Two 64 core AMD EPYC 7742 processors – 128 cores
- 256 GB of memory per standard node
- 8 non-uniform memory access (NUMA) regions of 16 cores
- Groups of 4 cores which share L3 cache



ARCHER2 node usage

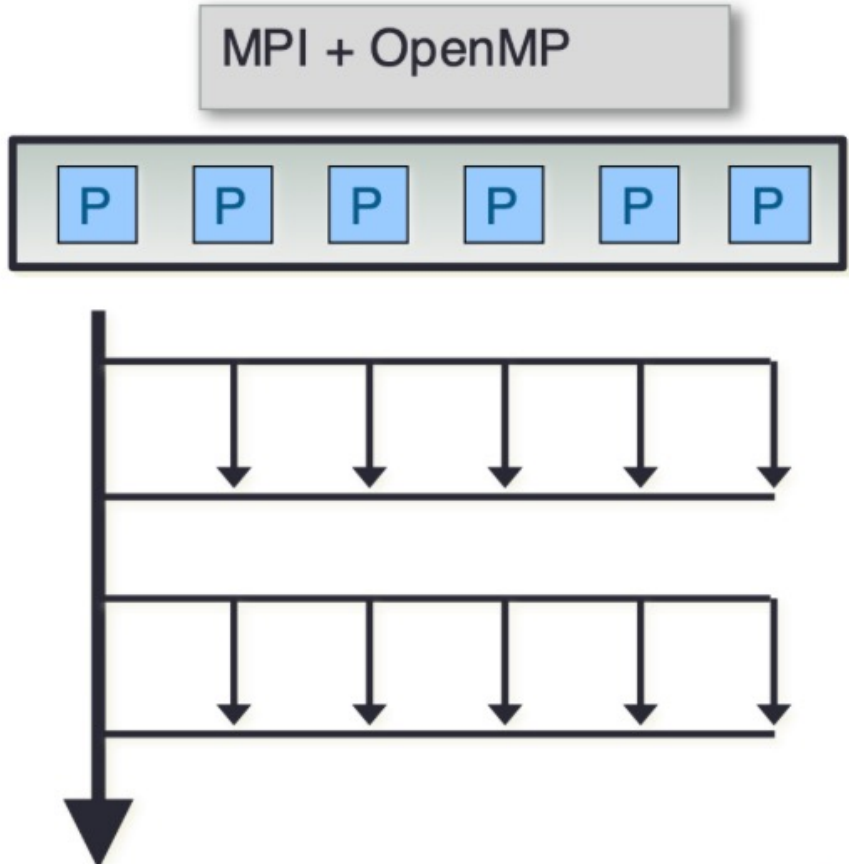
- ARCHER2 nodes have 256 GB of memory and 128 cores
- Using one MPI process per core gives 2 GB of memory per process
  - Less than ARCHER - roughly 2.6 GB per process
- This can be not enough for memory intensive applications – OOM errors – underpopulation sometimes necessary
- Using one process per core can affect the MPI communication performance
- Using MPI+OpenMP (less processes with multiple threads per process) can maybe help here

MPI and MPI+OpenMP in practice

- MPI
  - each core has one process (task) spawned on it
    - Own distinct memory space for each process
  - Communication between processes takes place via messages
  - 128 cores per node on ARCHER2 --tasks-per-node=128
    - OMP_NUM_THREADS = 1
    - 2 GB of memory per task
- MPI+OpenMP
  - Using a mixture of threads and processes spawned onto the cores
    - Groups of multiple threads within a memory region
    - At least one process per memory region
    - Messages between processes
  - Less processes mean more memory per process (shared between threads)
    - Less copies of data on a node
  - Less processes can mean less internode messages

MPI+OpenMP Usage

- Implemented in many HPC applications
- However not always more performant than using a single thread (e.g. pure MPI)
- Good OpenMP coverage within the code required
- Number of threads per process to use needs to be tuned to the use case and the machine
- Need to be aware of the underlying node structure when running e.g. regions when memory is shared



MPI+OpenMP ARCHER2 usage

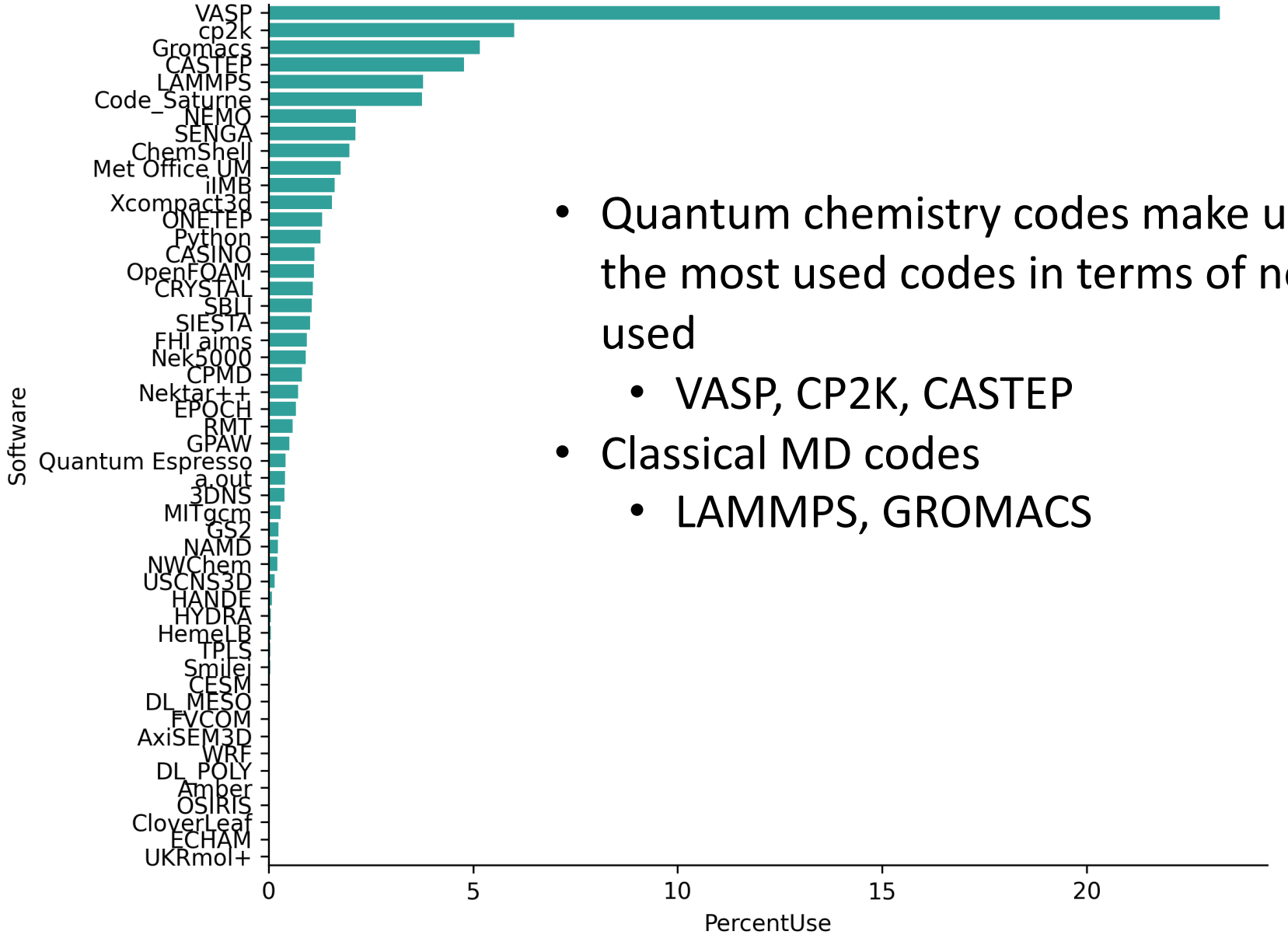
- Using multiple OpenMP threads per process
  - Less processes (or tasks) than 128
- Need to ensure `tasks-per-process` x threads = 128
  - Fully occupied nodes
  - Threads not greater than 16 (size of NUMA region on ARCHER2)
- `cpus-per-task` should equal `OMP_NUM_THREADS`
- 1, 2, 4, 8, 16 good values for threads on ARCHER2

```
#!/bin/bash

# Slurm job options (job-name, compute nodes, job time)
#SBATCH --job-name=Example_MPI_Job
#SBATCH --time=0:20:0
#SBATCH --nodes=4
#SBATCH --tasks-per-node=8
#SBATCH --cpus-per-task=16

# Replace [budget code] below with your project code (e.g. t01)
#SBATCH --account=[budget code]
#SBATCH --partition=standard
#SBATCH --qos=standard



# Set the number of threads to 16 and specify placement
#   There are 16 OpenMP threads per MPI process
#   We want one thread per physical core
export OMP_NUM_THREADS=16
export OMP_PLACES=cores

# Launch the parallel job
#   Using 32 MPI processes
#   8 MPI processes per node
#   16 OpenMP threads per MPI process
#   Additional srun options to pin one thread per physical core
srun --hint=nomultithread --distribution=block:block ./my_mixed_executable.x
```

ARCHER2 PERFORMANCE INVESTIGATION

MPI+OpenMP performance of ARCHER2 applications

ARCHER2 applications usage – June 2022



- Quantum chemistry codes make up the bulk of the most used codes in terms of node hours used
  - VASP, CP2K, CASTEP
- Classical MD codes
  - LAMMPS, GROMACS

ARCHER2 applications investigated

- Well used applications
- Centrally supported by the ARCHER2 team
- MPI+OpenMP enabled
- Different test cases for each application for different scales
- Full results: https://github.com/holly-t/ARCHER2_hybrid_benchmarking
- Compare performance of using 1, 2, 4, 8 and 16 threads per MPI process

Applications

- CASTEP - density functional theory software package for electronic structure calculations using plane waves
  - Version 20.11 - GCC version 10.2, Intel MKL 19, Cray-mpich 8.1.4, Cray-fftw 3.3.8.11
  - OpenMP usage – FFTW, linear algebra libraries
- CP2K - quantum chemistry and solid state physics package – mixed plane wave/Gaussian
  - Version 8.1 - GCC version 11.2, Intel MKL 19, Cray-mpich 8.1.9, Cray-fftw 3.3.8.11
  - OpenMP usage - realspace to planewave transfer, collocate and integrate, FFTW, linear algebra libraries, + more
- GROMACS – classical MD of biological systems
  - Version 2021.3 - GCC version 11.2, Cray-mpich 8.1.9
  - OpenMP usage – PME calculations
- LAMMPS – classical MD for materials modelling
  - Jan 2022 version - GCC version 10.2, Cray-mpich 8.1.4, Cray-fftw 3.3.8.11
  - OpenMP usage – pair interactions, FFTW
- Quantum ESPRESSO - electronic-structure calculations with plane waves
  - Version 6.8 - GCC version 11.2, Cray-libsci 21.08.1.2, Cray-mpich 8.1.9, Cray-fftw 3.3.8.11
  - OpenMP usage - space integrals, point function evaluations, 3D FFTW, linear algebra libraries

Application test cases

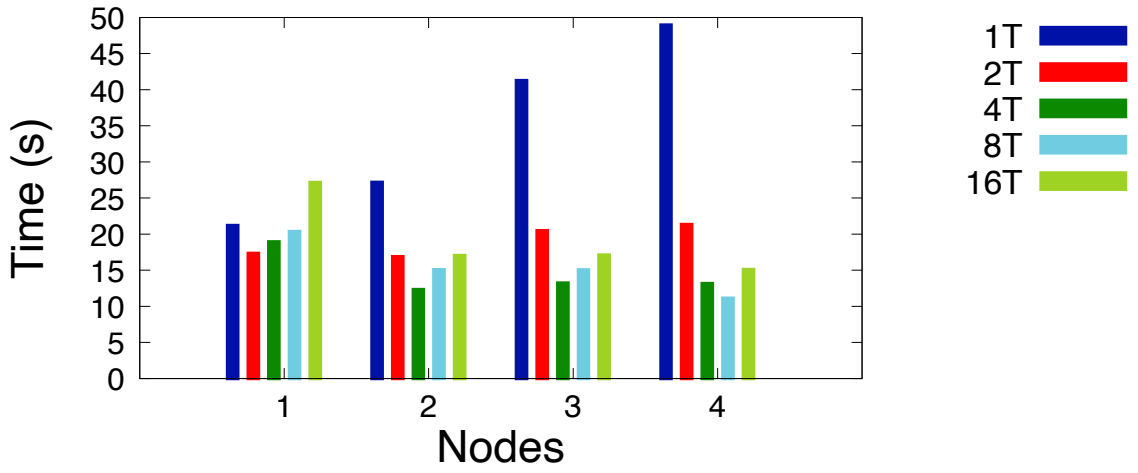
- CASTEP
  - DNA (large memory intensive system)
  - Al3x3 (smaller system)
- CP2K
  - H2O-64 (small water system, LDA)
  - H2O-512 (larger water)
  - LiH-HFX (Hartree Fock exchange, memory intensive)
- GROMACS
  - 1400k and 3000k atom MD simulations
  - BenchPEP – 12m atom MD simulation
- LAMMPS
  - 3000k atom MD simulation
- Quantum ESPRESSO
  - GRIR (scf calculation, 4 k-points)
  - CNT (single k-point, large and memory intensive system)

RESULTS

Application MPI+OpenMP results

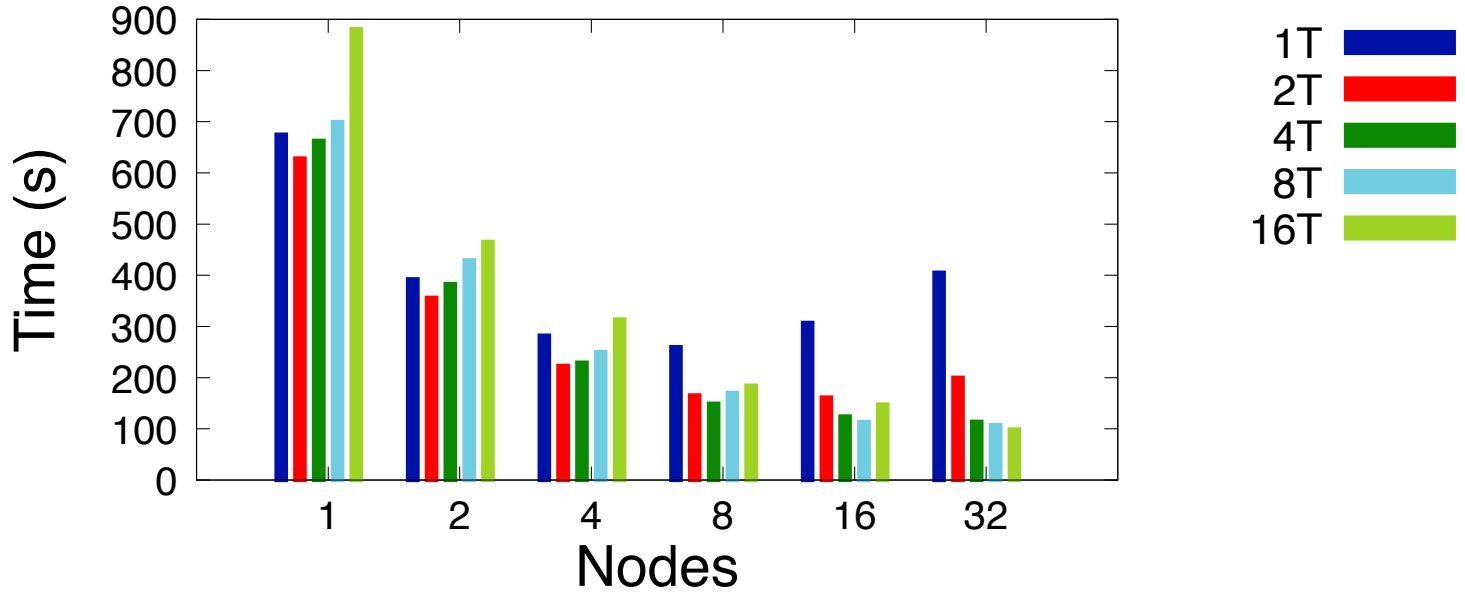
CP2K - H2O-64 benchmark

- Small system that does not scale beyond a couple of nodes
- Clear benefit from using multiple threads per process
- Run time dominated by MPI_Alltoall, which contributes more on multiple nodes
- Using MPI+OpenMP reduces the run time of these communications – message aggregation



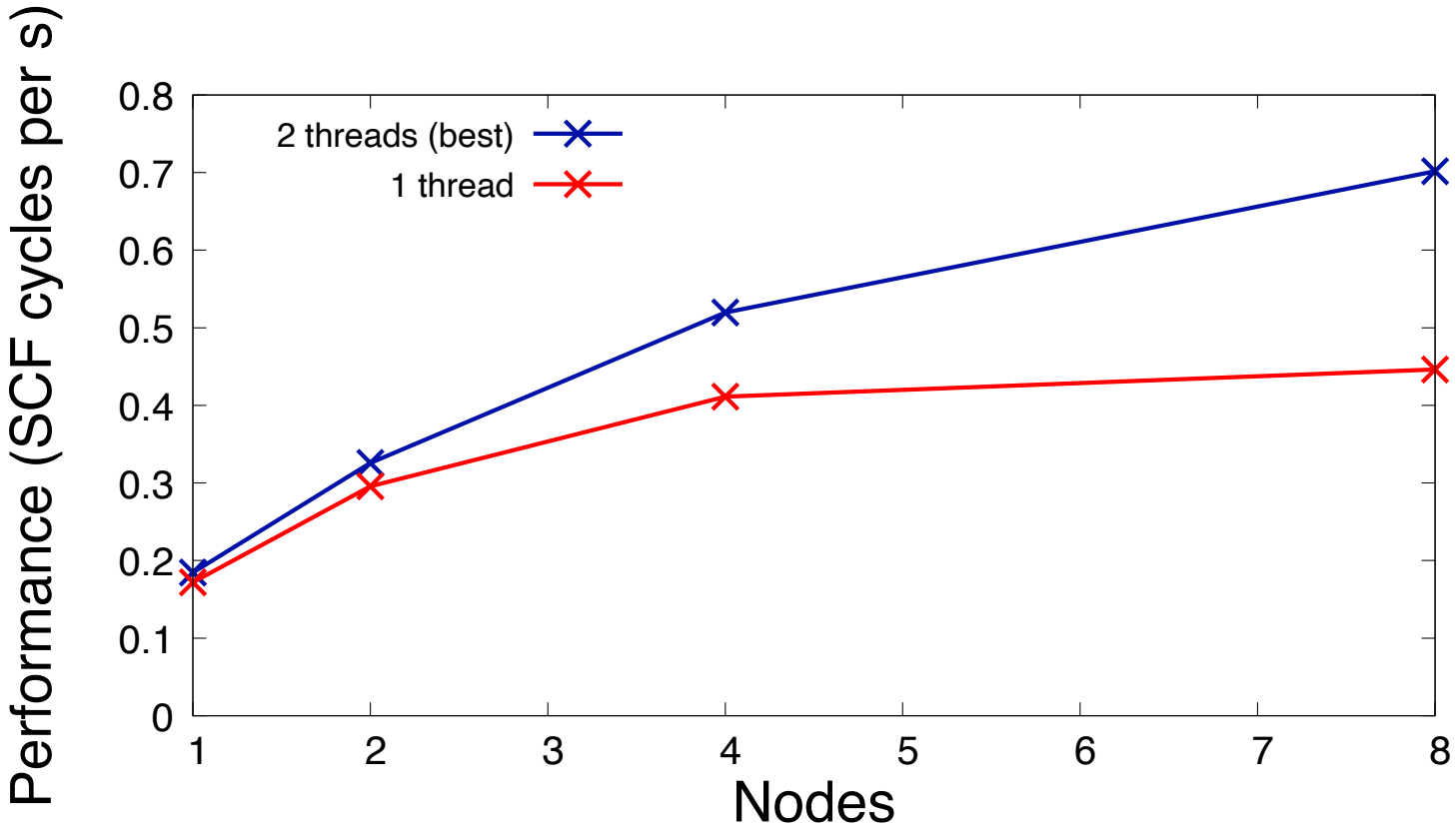
Nodes	mp_alltoall time (s)	
	1 thread	4 threads
1	2.12	1.22
2	10.844	1.599
3	18.918	2.138
4	23.612	3.273

CP2K - H2O-512 benchmark



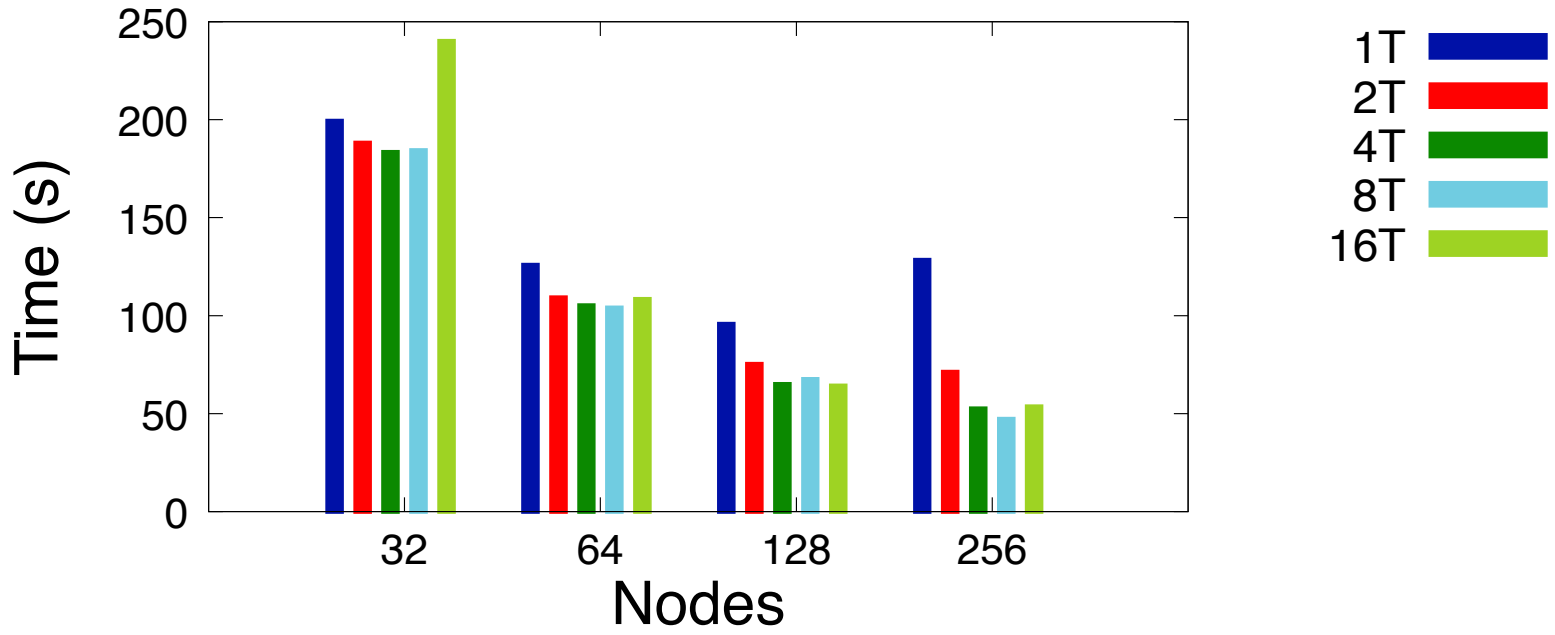
- Larger version of H2O-64 benchmark
- Using multiple threads per process allows for further scaling beyond the single threaded version
- Using 2 or 4 threads per process gives best performance due to reduction in communications overhead

CP2K - H2O-512 benchmark



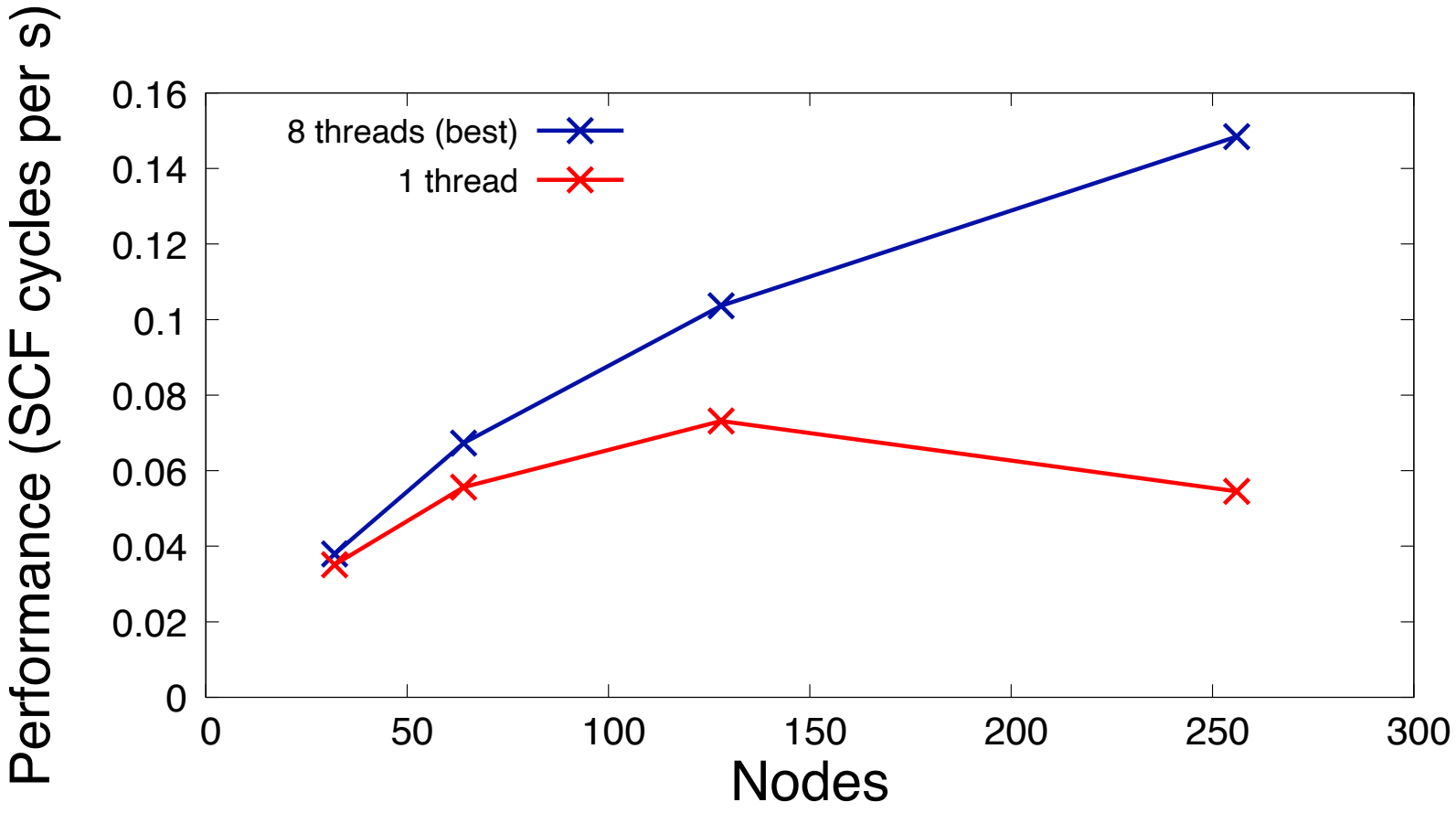
Parallel efficiency	4 nodes
1 thread per process	60%
2 threads per process	70%

CP2K – LiH-HFX benchmark



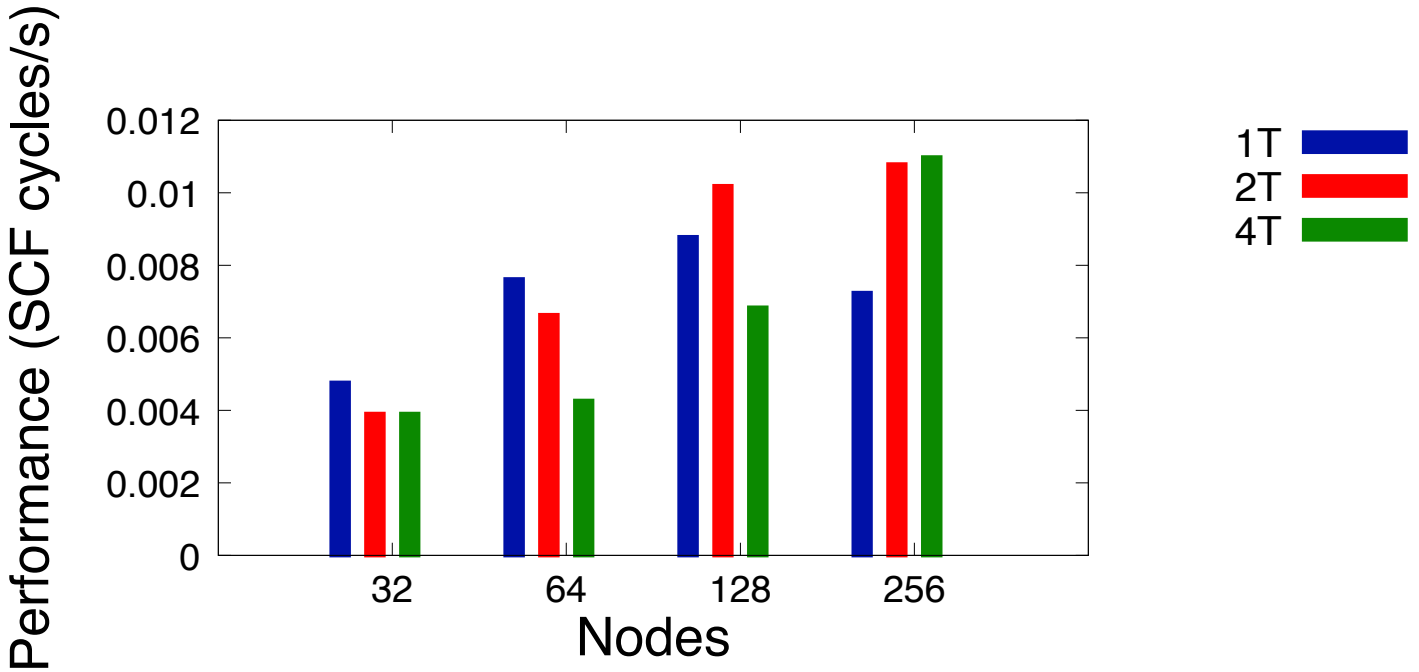
- Hybrid Hartree-Fock exchange calculation which is memory and compute intensive
- 4 or 8 threads per process gives the best performance
- The increase in performance with multiple threads is less significant as this calculation is dominated more by the computation of the integrals rather than comms.

CP2K – LiH-HFX benchmark



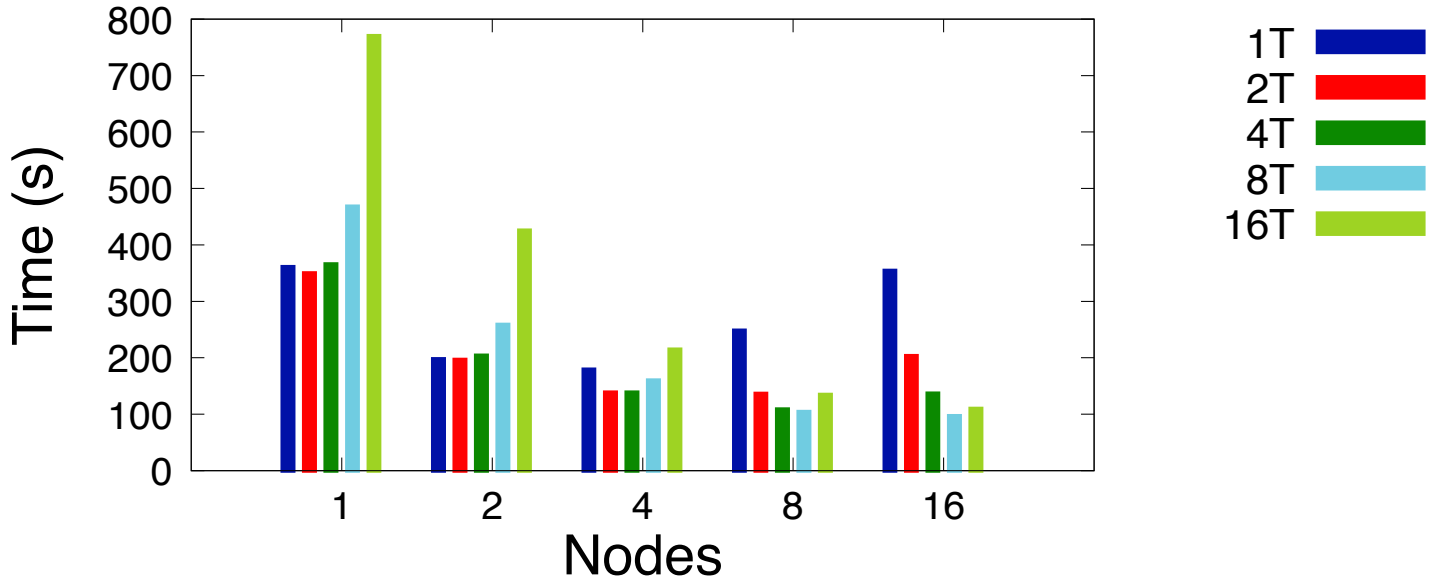
Parallel efficiency	128 nodes	256 nodes
1 thread per process	52%	19%
8 threads per process	68%	49%

CASTEP – DNA benchmark



- Large, memory intensive calculation
- On 32 and 64 nodes using a single thread per process gives the better performance than using multiple threads
- However on 128 and 256 nodes using 2 and 4 threads respectively yields the best performance
- Again MPI collectives dominate in CASTEP

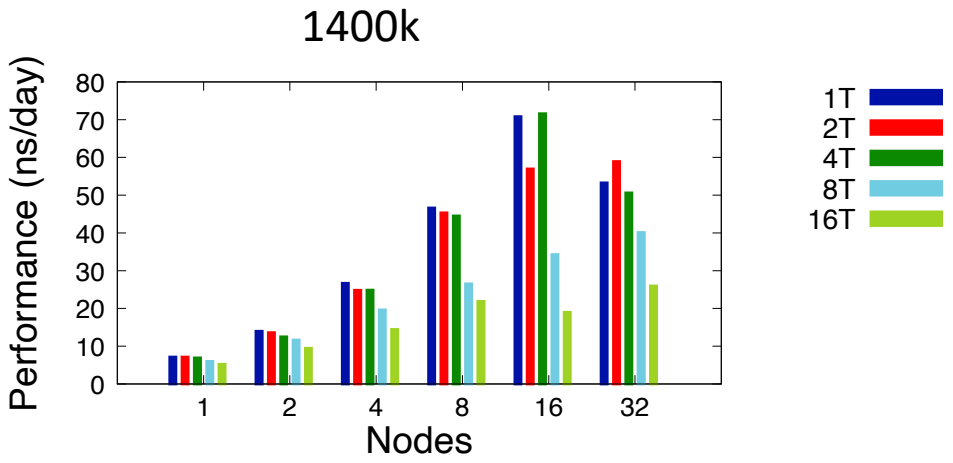
CASTEP – Al3x3 benchmark

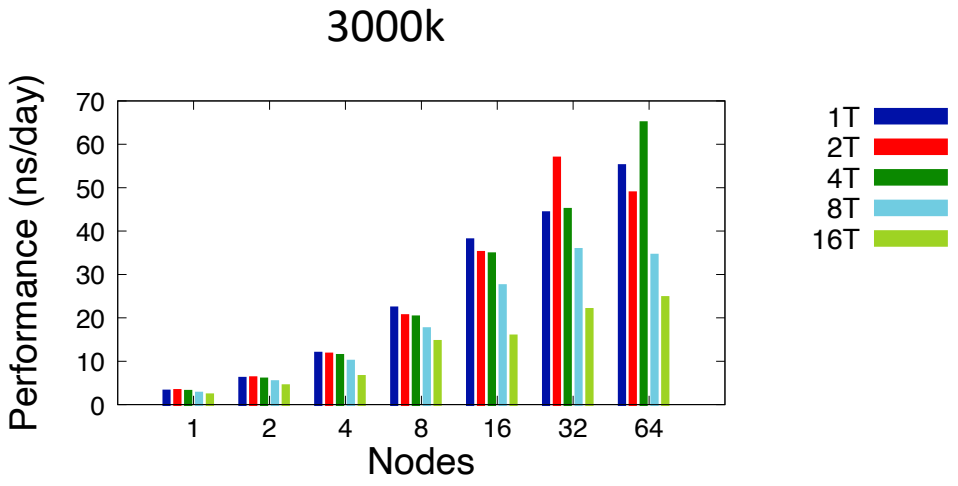


- Smaller system - single point energy calculation
- Using 2 or 4 threads per process gives a performance similar to the single threaded version
- On 4 or more nodes using MPI+OpenMP improves the performance

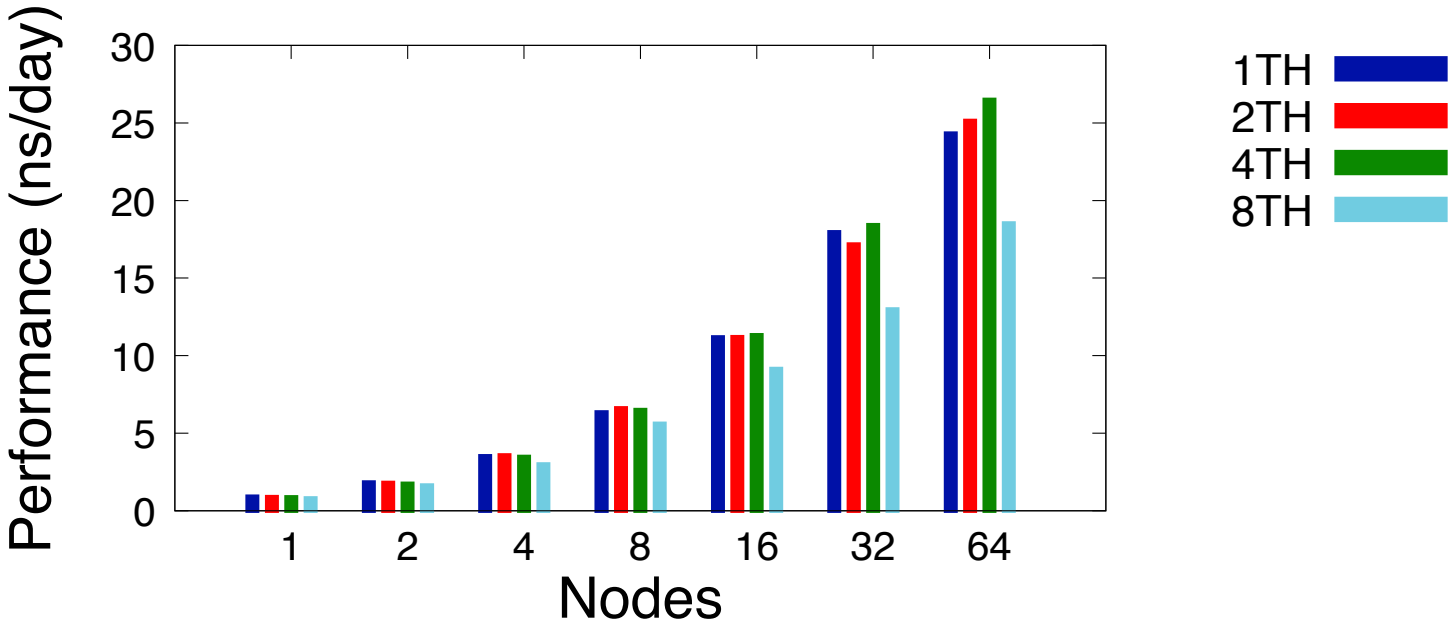
GROMACS – 1400k and 3000k benchmarks

- MD simulations of 1400k and 3000k atoms
- Timing of PME communications become significant at scale
- Using MPI+OpenMP can help reduce the communications cost at the scaling limit
- Performance not great for more than 4 threads
  - Sharing memory beyond the L3 cache



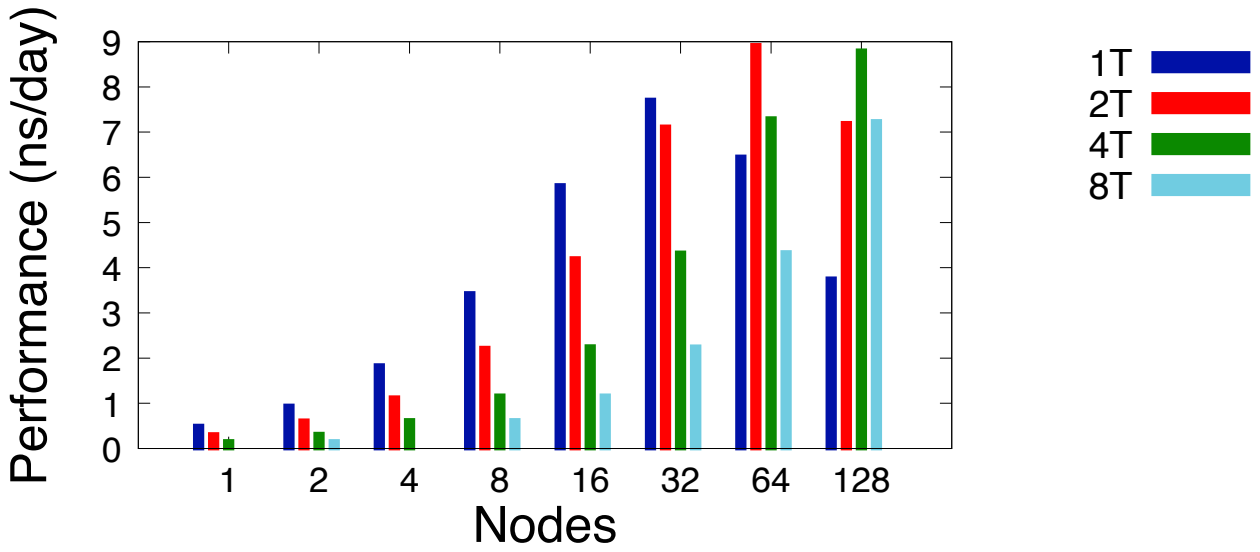


GROMACS – benchPEP benchmark



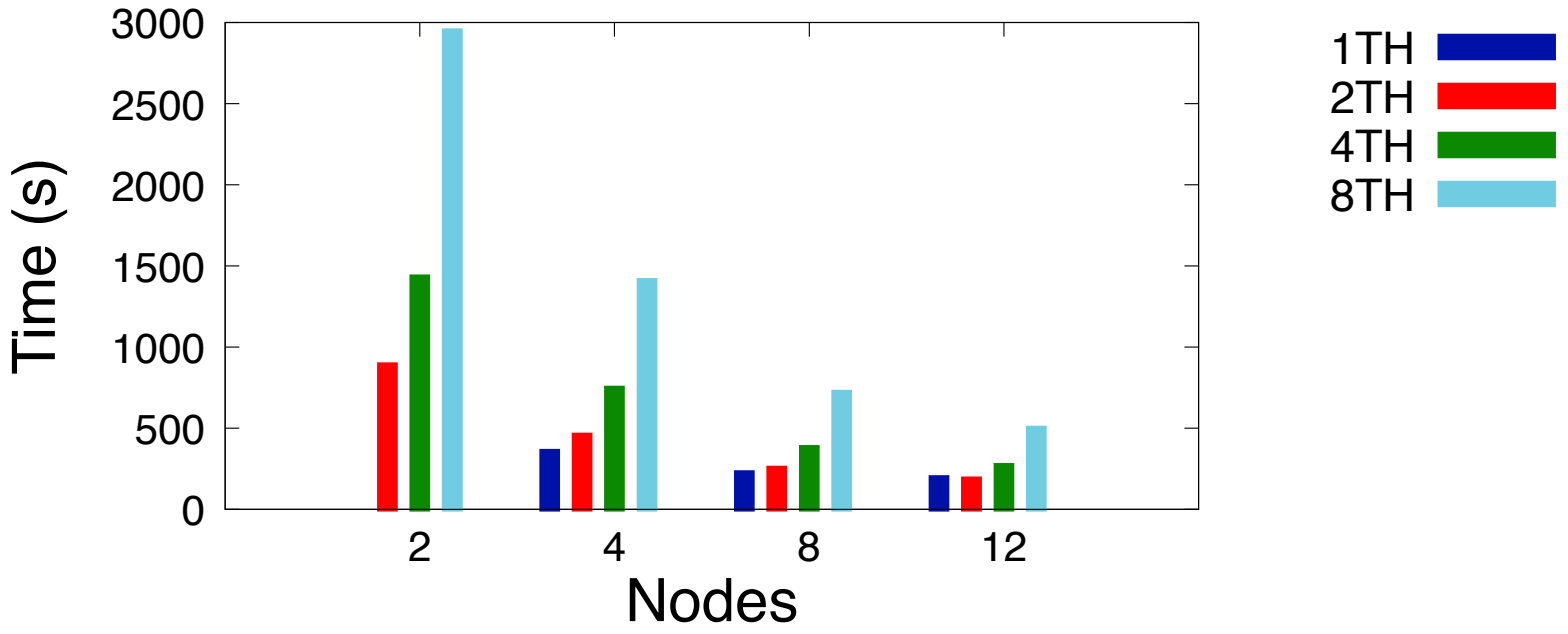
- 12 million atom MD calculation
- For 1 – 16 nodes there is not much performance difference for 1-4 threads
- On 32 and 64 nodes 4 threads per process yields best performance
  - Again with more than 4 threads the performance suffers due to cache effects

LAMMPS – 3000k atom benchmark



- Using a single thread gives the best performance up to 32 nodes.
- Only on 64 and 128 nodes does using MPI+OpenMP improve the performance
- Main overhead on many nodes is writing to file (for checkpointing in LAMMPS) which is poor on many processes – a known issue under investigation
- The performance of this improves with MPI+OpenMP as there are less processes writing to disk

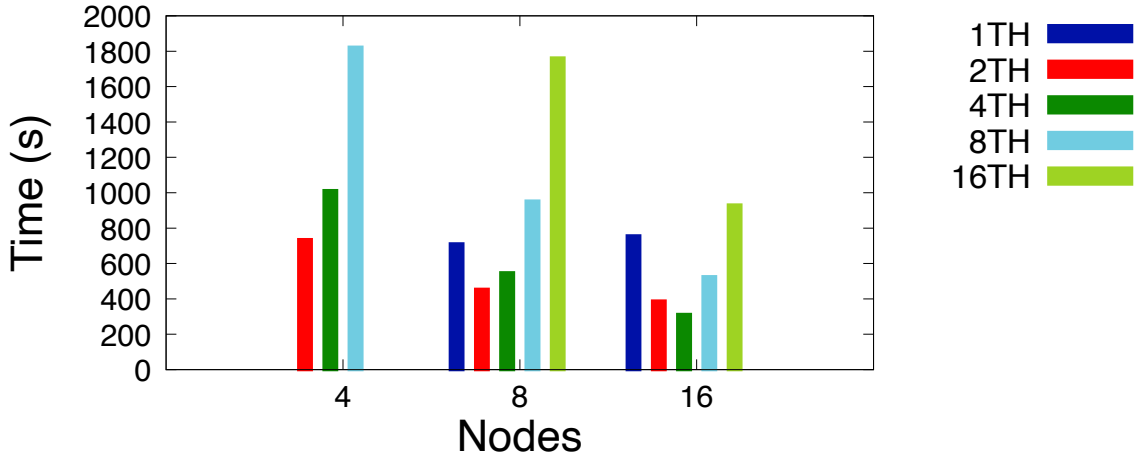
Quantum ESPRESSO – GRIR benchmark



- On 1 node this calculation fails with an out of memory error
- On 2 nodes this fails with OOM on a single thread – not enough memory per process
- Using MPI+OpenMP does not benefit the performance of this calculation – even at the limit of scaling

Quantum ESPRESSO – CNT benchmark

- This test case has very high memory requirements
- Using MPI+OpenMP can prevent OOM errors
- MPI+OpenMP also increases the memory per process, which improves the performance for this system on 8 and 16 nodes
- Underpopulation alone also improves the performance



Threads × PPN	Run time (s)
1 × 128	708
1 × 64	467
2 × 64	454

Test case run time on 8 nodes

Summary

- Quantum chemistry codes CP2K and CASTEP are able to benefit from using MPI+OpenMP in general
  - The run time of these codes are dominated by MPI collective calls
  - MPI+OpenMP reduces the communication overhead
- MPI+OpenMP is useful for Quantum ESPRESSO as it allows more memory per process
  - Calculations can be memory intensive and may require underpopulation to run
- The classical MD codes GROMACS and LAMMPS generally do not benefit much from MPI+OpenMP
  - It can improve the performance, but only at the scaling limit

BENEFITING FROM MPI+OpenMP

How can users benefit?

- Check if your application is MPI+OpenMP enabled
- Performance
  - Applications and test cases where MPI collective communications dominate the run time may benefit from MPI+OpenMP
  - The overhead of MPI communications is reduced as there are less messages between processes
  - At the scaling limit performance can be improved
- Out of memory errors
  - If you have a memory intensive calculation which runs out of memory
  - Requires underpopulation of processes on the node (task-per-node=64)
  - MPI+OpenMP allows more memory per process similar to underpopulation
  - But, using threads as well may give a performance benefit

Using MPI+OpenMP for your application

- Test different values of threads per process for your particular test case to see the effect on the performance
  - 1, 2, 4, 8, 16 threads per process on ARCHER2
  - Usually, 2 or 4 threads
  - Make sure the placement is correct and the processes and threads fully populate the node
- Value for best performance likely to vary depending on:
  - Application
  - Particular problem
  - Number of nodes/cores used
- If underpopulating for memory reasons, then try using multiple threads
  - E.g. 2 threads for half population (64 tasks)

Conclusions

- On ARCHER2 applications can benefit from using MPI+OpenMP
  - Performance benefits – applications which are communications heavy benefit from MPI+OpenMP, particularly on more nodes
  - Aggregation of messages
  - Using 2 or 4 threads per process
- MPI+OpenMP can also help with memory requirements
  - Particularly important in memory limited applications
- MPI+OpenMP might be less useful for classical MD codes
- Overall using MPI+OpenMP on ARCHER2 can be of significant benefit to users – but this is dependent on many factors