Efficient use of MPI+OpenMP on a Cray EX supercomputer

Holly Judge

Reusing this material

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

Partners

UKRI

Engineering and Physical Sciences Research Council

Natural Environment Research Council

epcc

THE UNIVERSITY of EDINBURGH

Hewlett Packard Enterprise

Introduction

- ARCHER2 is a Cray EX machine based at the University of Edinburgh, UK
  - UK national supercomputing service
  - 5,860 nodes (750,080 cores)
- Consider use of MPI+OpenMP to take 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?
- What information can we get to users for using MPI+OpenMP?

Outline

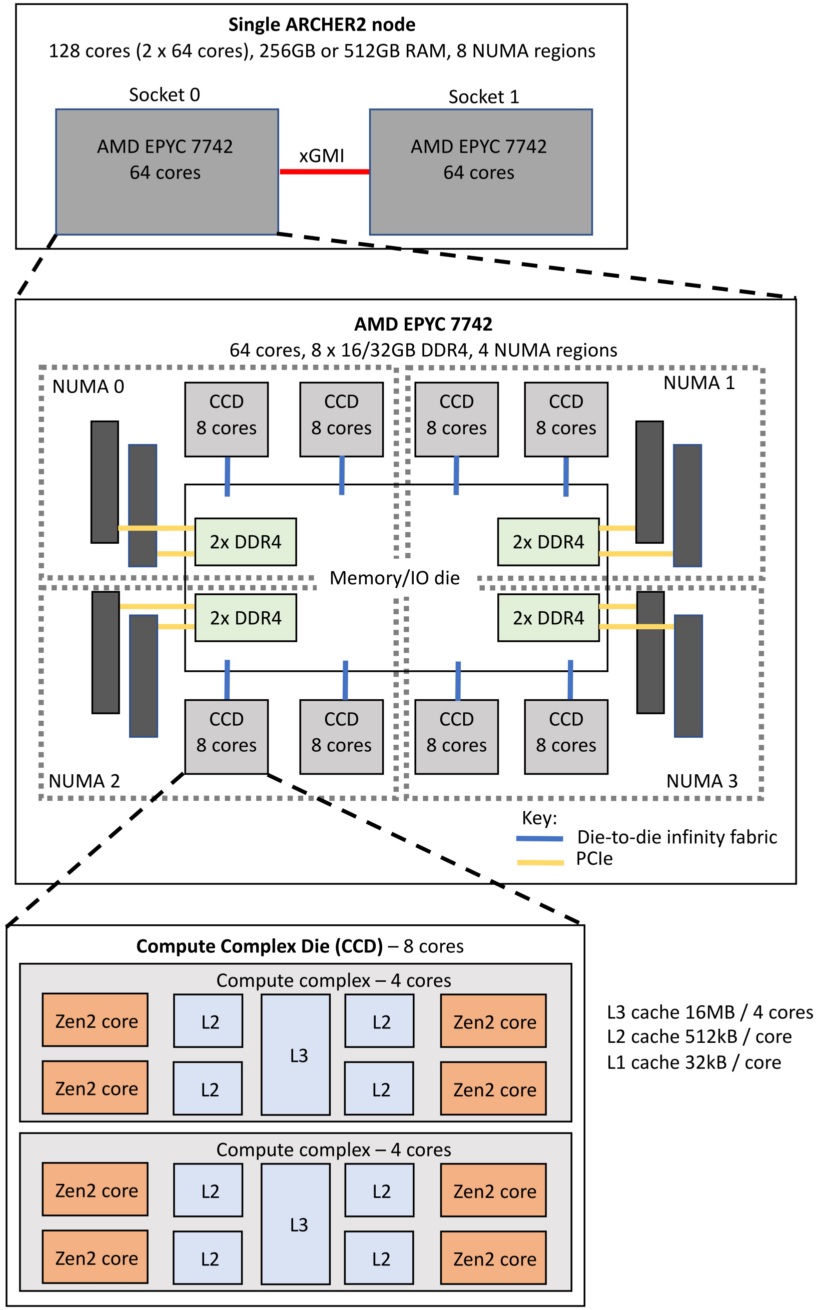
- ARCHER2 overview
- Applications
    - Usage on ARCHER2
    - Applications investigated
    - Application test cases
- Results
- User system benefiting from MPI+OpenMP

ARCHER2

- UK national service used for scientific research
- Supported by the ARCHER2 CSE team
- Runs a range of core applications for different research areas
- 5,860 nodes (750,080 cores)
- 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
- Sensible thread choices of 1, 2, 4, 8, 16



Single ARCHER2 node
128 cores (2 x 64 cores), 256GB or 512GB RAM, 8 NUMA regions

Socket 0: AMD EPYC 7742, 64 cores

xGMI

Socket 1: AMD EPYC 7742, 64 cores

AMD EPYC 7742
64 cores, 8 x 16/32GB DDR4, 4 NUMA regions

Key:
Die-to-die infinity fabric
PCIe

Compute Complex Die (CCD) – 8 cores

Compute complex – 4 cores

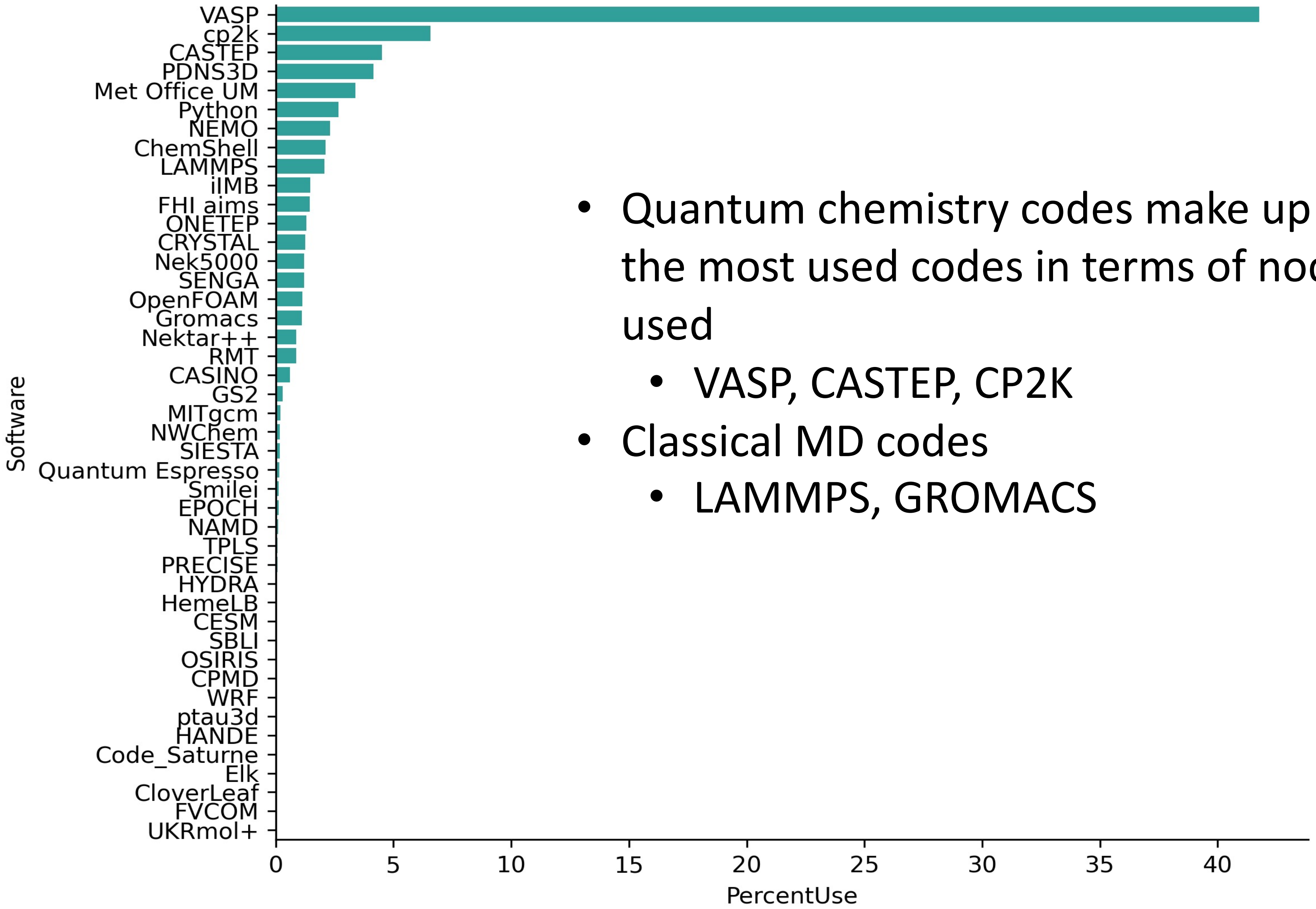
Compute complex – 4 cores

L3 cache 16MB / 4 cores
L2 cache 512kB / core
L1 cache 32kB / core

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 also affect the MPI communication performance
- Using less processes with multiple threads per process (MPI+OpenMP) can maybe help here

ARCHER2 applications usage



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

CP2K

LAMMPS

CASTEP

QUANTUMESPRESSO

GROMACS
FAST. FLEXIBLE. FREE.

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 ESPRESSSO - 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

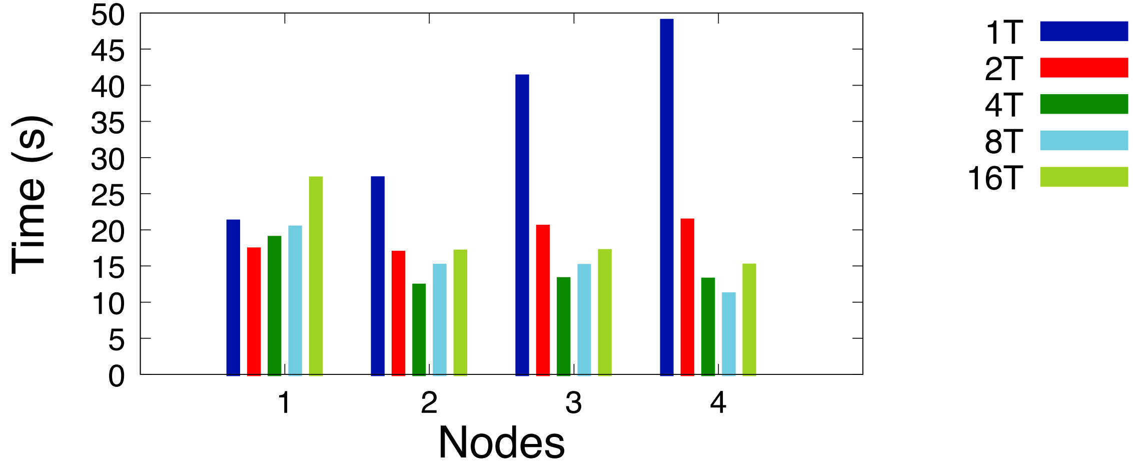
- CASTEP
  - DNA (large memory intensive system)
  - Al3x3 (smaller system)
- CP2K
  - H2O-64 (small water system, LDA)
  - H2O-512 (larger water)
  - LiH-HFX (Hartree Fock exchange, memory intensive)
- GROMACS
  - 1400k and 3000k atom MD simulations
- LAMMPS
  - 3000k atom MD simulation
- Quantum ESPRESSSO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

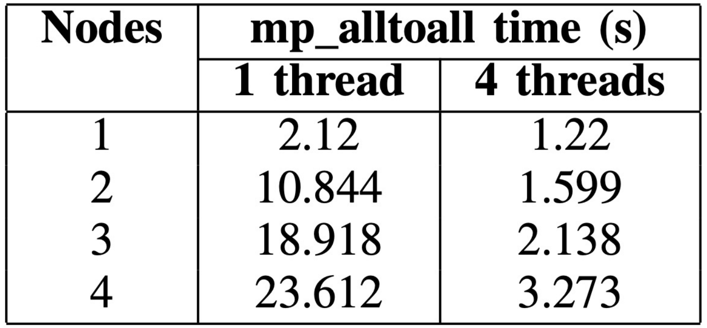
RESULTS

Application benchmark results

CP2K - H2O-64 benchmark

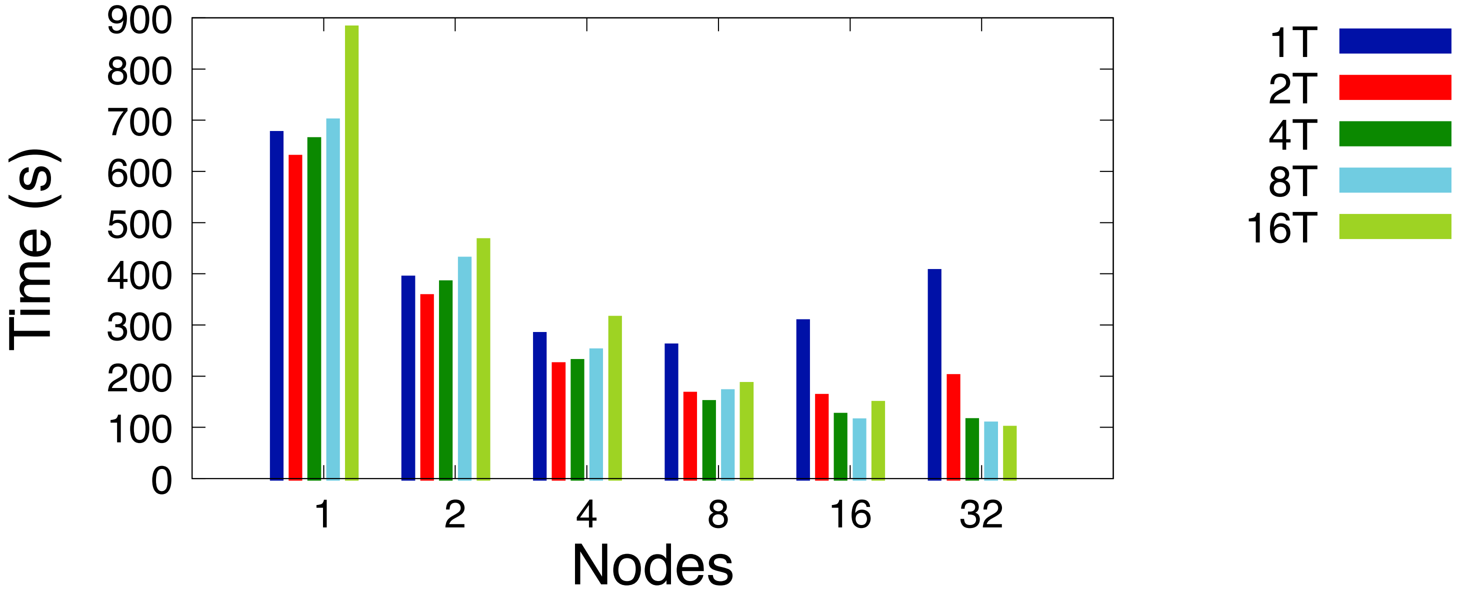
- Small system that does not scale beyond a couple of nodes
- Clear benefit from using multiple threads per process
- Run time dominated by MPI_Alltoall, which contributes more on multiple nodes
- Using MPI+OpenMP reduces the run time of these communications – message aggregation





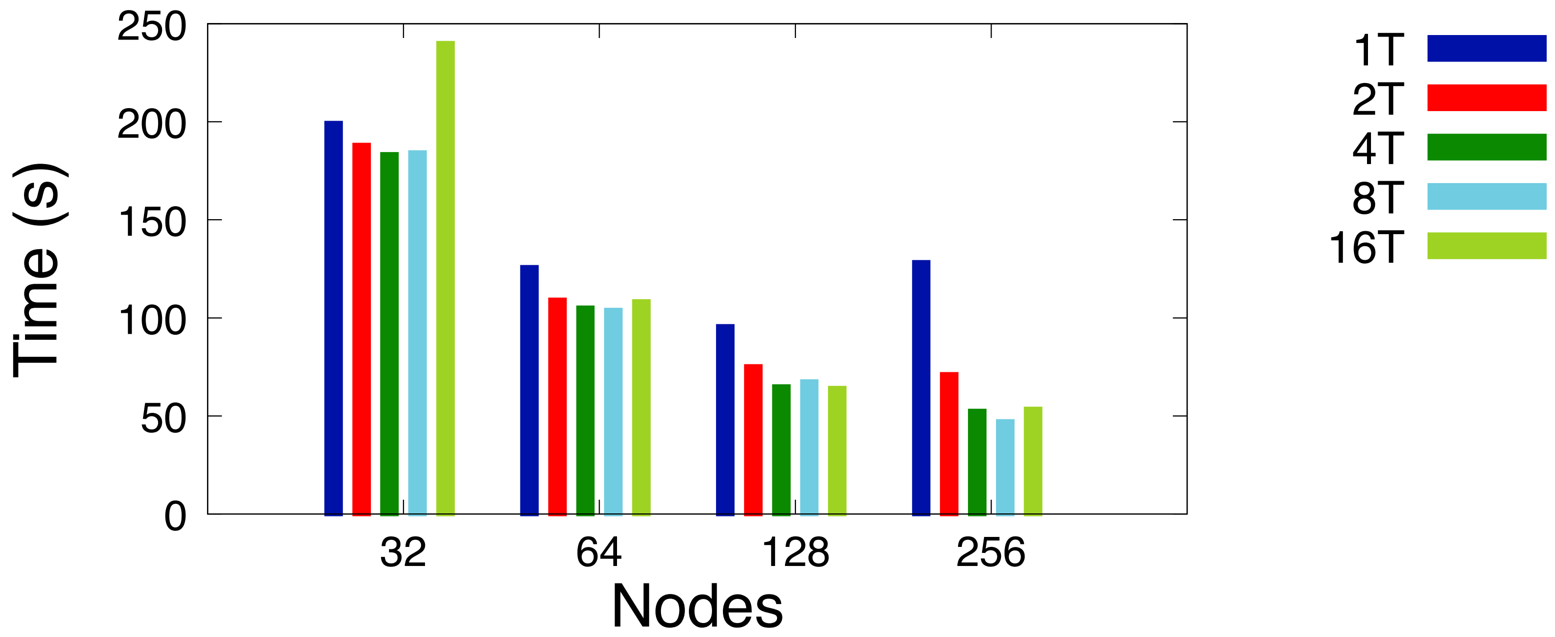
Nodes	mp_alltoall time (s)	
	1 thread	4 threads
1	2.12	1.22
2	10.844	1.599
3	18.918	2.138
4	23.612	3.273

CP2K - H2O-512 benchmark



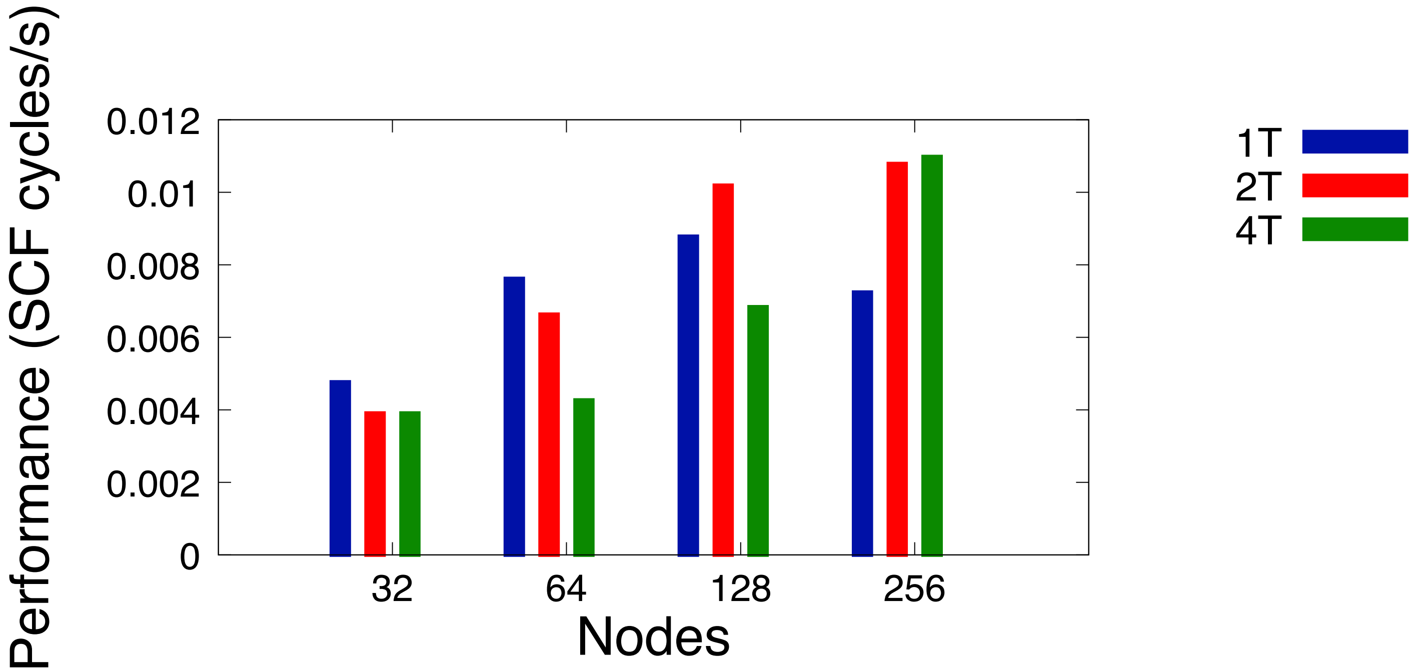
- Larger version of H2O-64 benchmark
- Using multiple threads per process allows for further scaling beyond the single threaded version
- Using 2 or 4 threads per process gives best performance due to reduction in communications overhead

CP2K – LiH-HFX benchmark



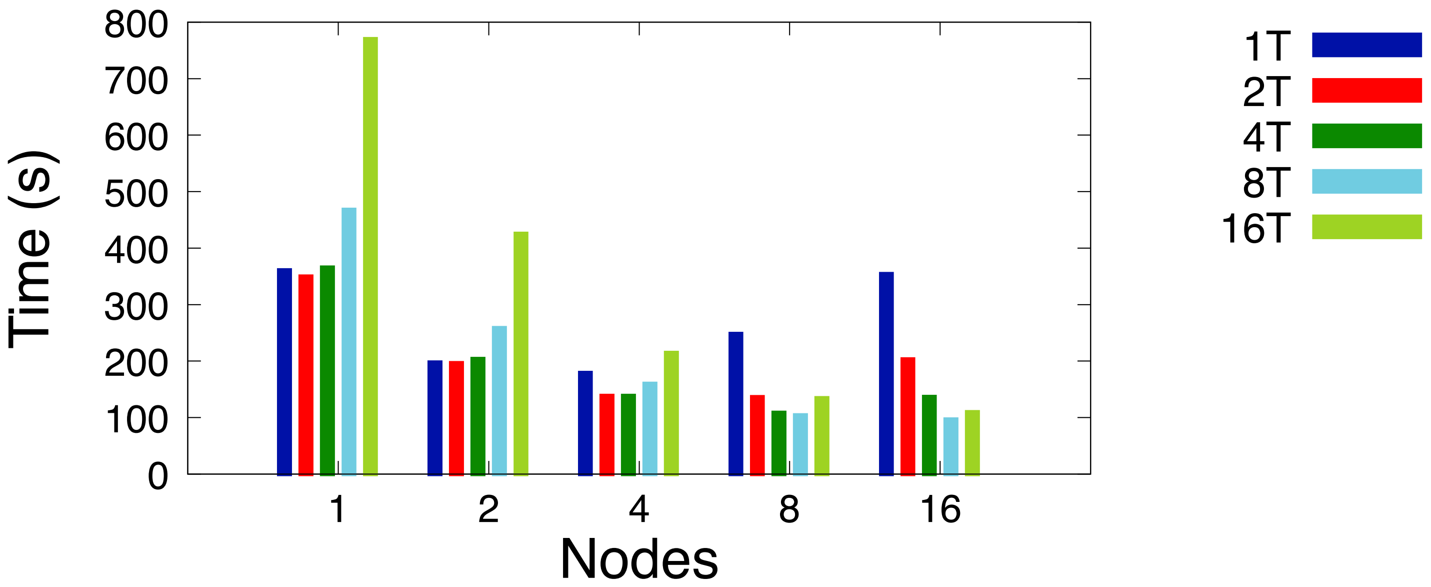
- Hybrid Hartree-Fock exchange calculation which is memory and compute intensive
- 4 or 8 threads per process gives the best performance
- The increase in performance with multiple threads is less significant as this calculation is dominated more by the computation of the integrals rather than comms.

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

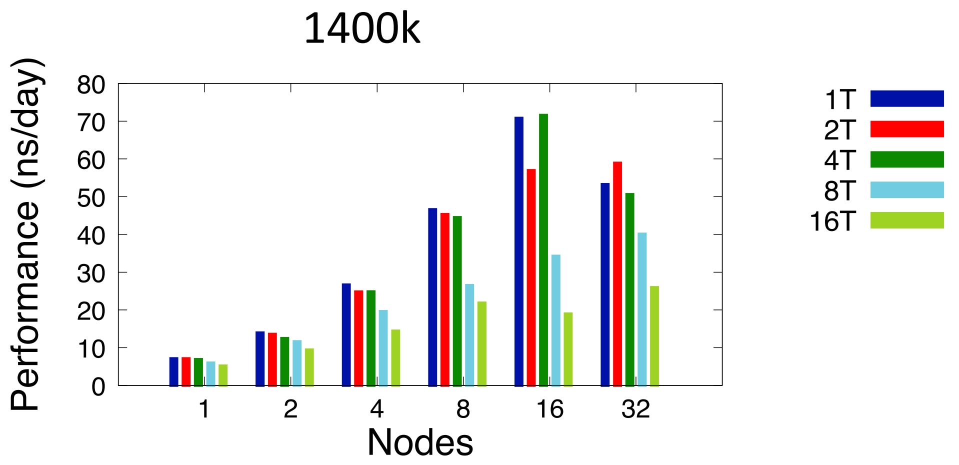
CASTEP – Al3x3 benchmark

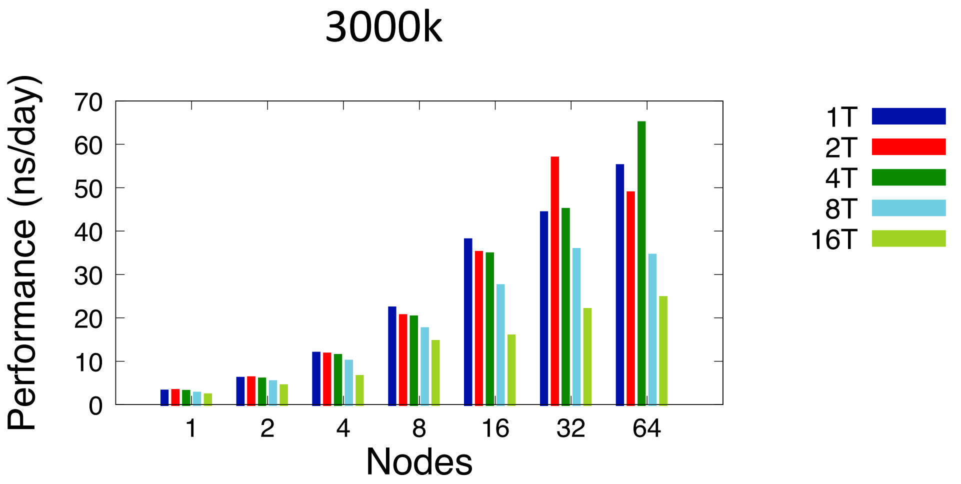


- Smaller system - single point energy calculation
- Using 2 or 4 threads per process gives a performance similar to the single threaded version
- At scale using MPI+OpenMP improves the performance

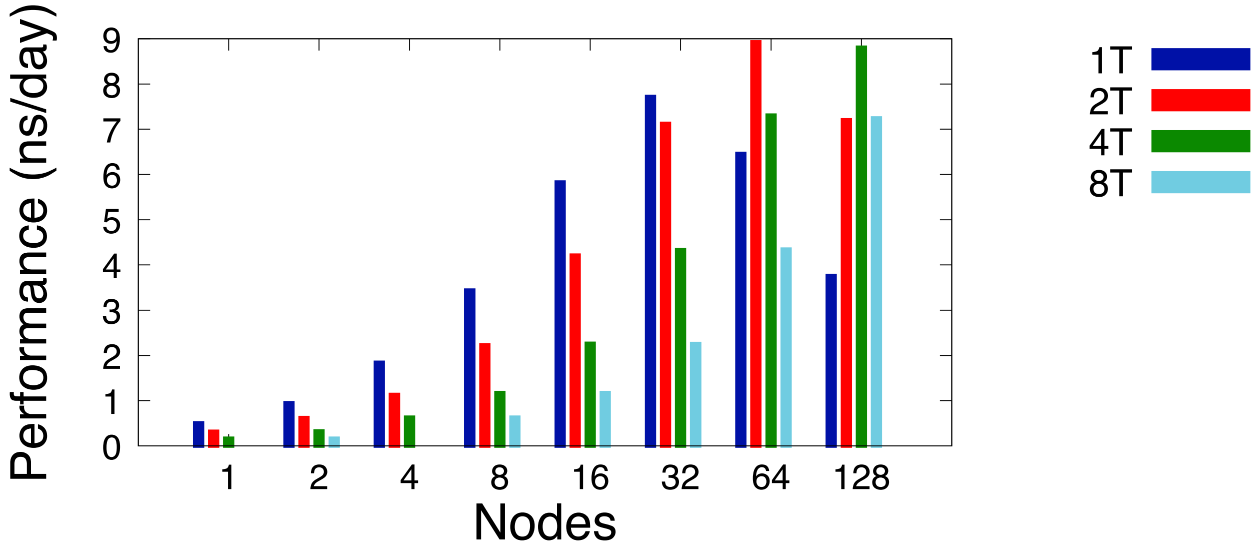
GROMACS – 1400k and 3000k benchmarks

- 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



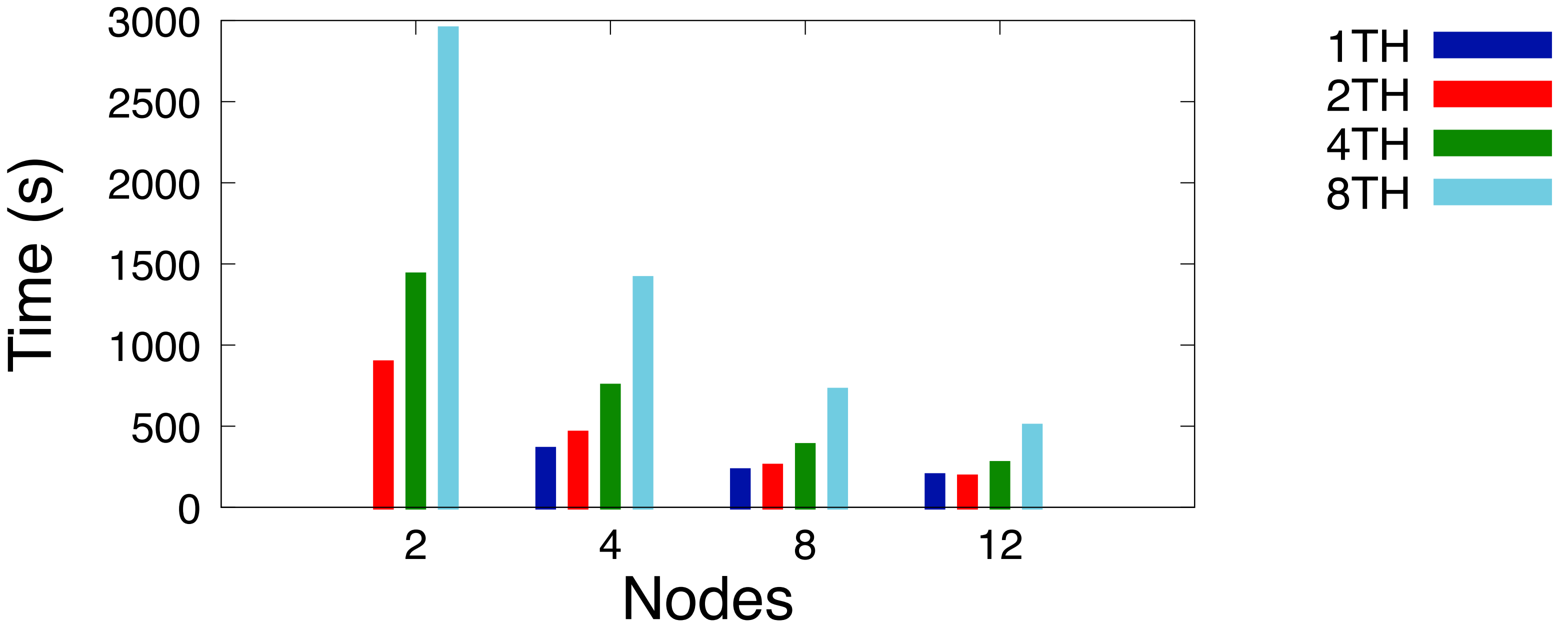


LAMMPS – 3000k atom benchmark



- 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

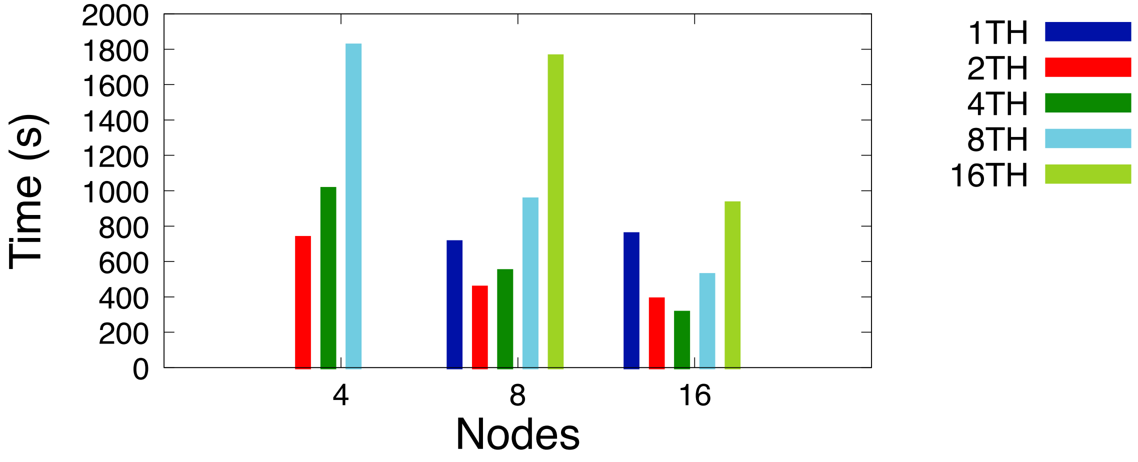
Quantum ESPRESSO – GRIR benchmark

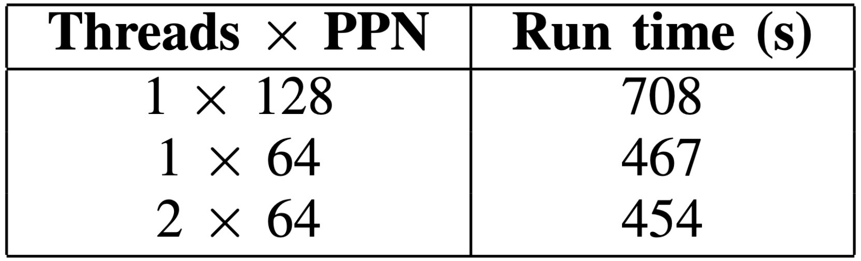


- On 1 node this calculation fails with an out of memory error
- On 2 nodes this fails with OOM on a single thread – not enough memory per process
- Using MPI+OpenMP does not benefit the performance of this calculation – even at the limit of scaling

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

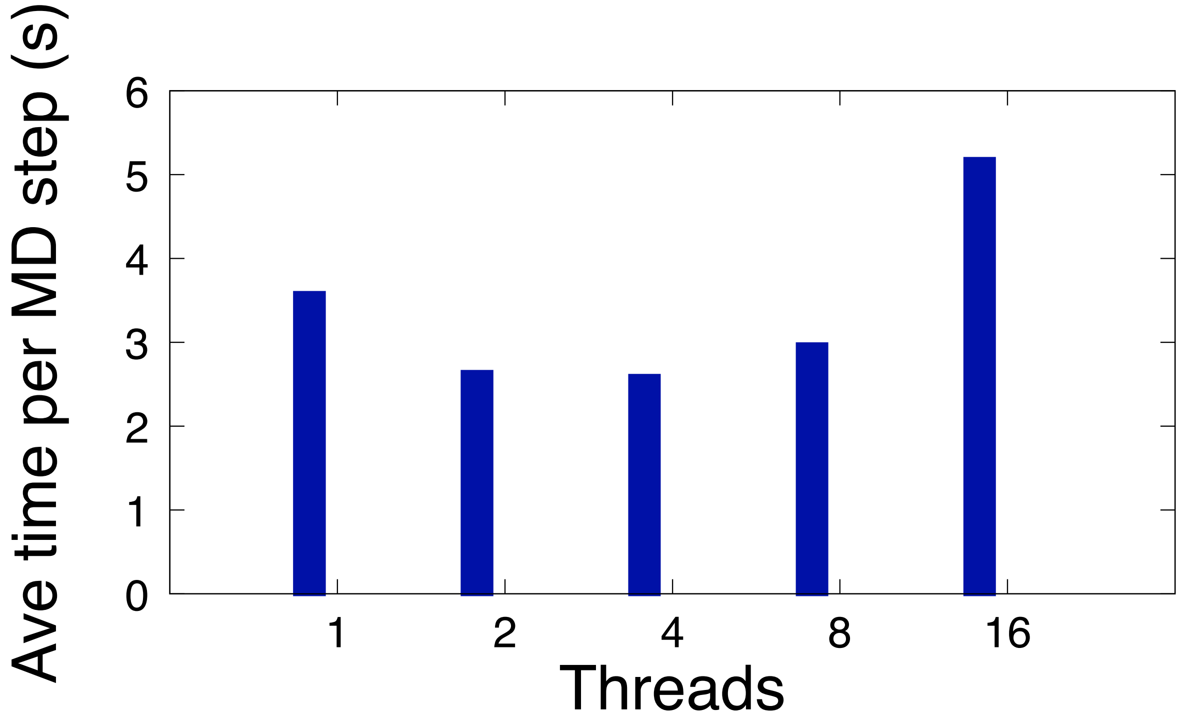
USER EXAMPLE

CP2K calculation

User example system

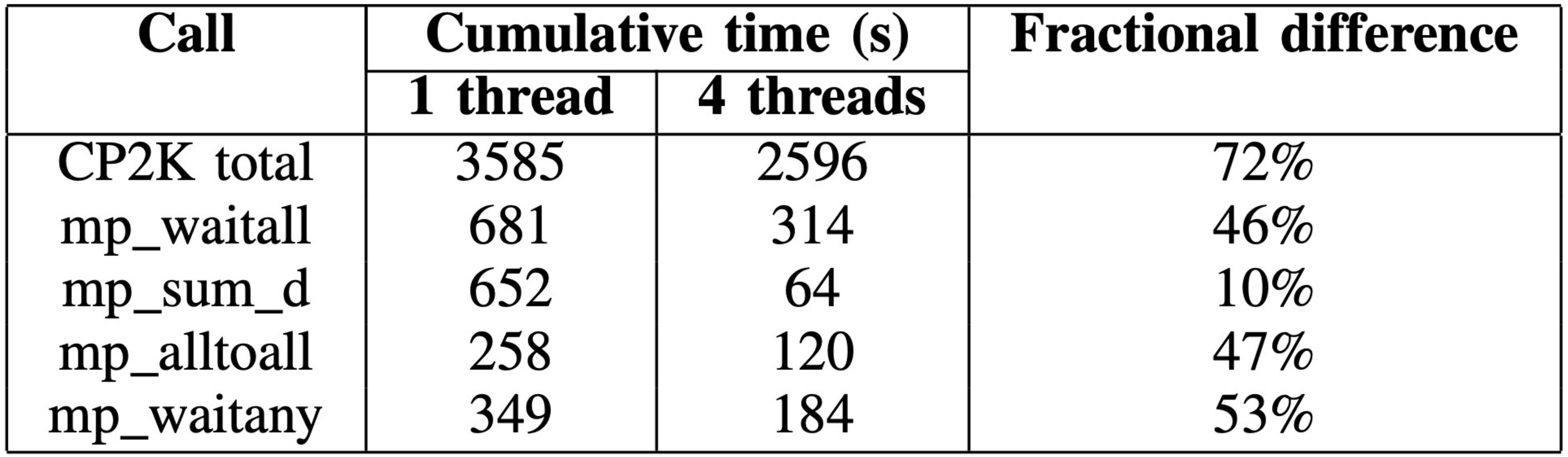
- For CP2K using MPI+OpenMP has been shown to improve the performance of the test cases – but how does this relate to real user simulations?
- This is a real user’s system; a long running molecular dynamics simulation of around 100 water molecules
- Using 4 threads per process was found to greatly increase the performance

User example – average time per MD step on 4 nodes



- Average time per step collected for a 1,000 step MD run
- The time per step on 4 threads is 70% of that of the pure MPI case
- Makes a big difference to run time and resources consumed

User example – MPI communications



Call	Cumulative time (s)		Fractional difference
	1 thread	4 threads	
CP2K total	3585	2596	72%
mp_waitall	681	314	46%
mp_sum_d	652	64	10%
mp_alltoall	258	120	47%
mp_waitany	349	184	53%

- Large reductions in the cumulative time taken for MPI calls
- Adds up to a significant reduction in the total run time

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 is less useful for classical MD codes
- Overall using MPI+OpenMP on ARCHER2 can be of significant benefit to users – but this is dependent on the application