Effective use of MPI+OpenMP on a Cray EX supercomputer

Dr Holly Judge
EPCC
University of Edinburgh
Edinburgh, UK
h.judge@epcc.ed.ac.uk

***Abstract*—ARCHER2 is a HPE Cray EX supercomputer based at the University of Edinburgh. It is the UK national supercomputing service for scientific research in the UK, containing a total of 5,860 nodes (750,080 cores). The Cray EX consists of dual AMD 7742 EPYC ™ processors and has 128 cores per node. Commonly, scientific applications are run using MPI, where one process is spawned on each computer core. For this machine this results in a large number of processes being used, particularly when running on many nodes. Using this many processes has been shown to have a negative impact on the performance of MPI communications for some use cases. As an alternative, MPI+OpenMP may be used. This approach naturally results in fewer processes and this can reduce the overhead of communications, and have other positive affects, such as lowering the memory requirements.**

This paper will present a study of the performance of MPI+OpenMP in a variety of scientific applications on ARCHER2. In doing this different configurations of threads and processes will be explored. This will aim to develop an understanding of which applications can see a performance benefit from using MPI+OpenMP, and why this is the case for the Cray EX in particular. This will also aim to act as guidance for users, who in some circumstances are able to benefit considerably from using MPI+OpenMP in their calculations.

***Index Terms*—Hybrid MPI+OpenMP, Cray EX, Performance, CASTEP, CP2K, GROMACS, LAMMPS, Quantum ESPRESSO**

I. Introduction

Most modern HPC machines are comprised of multiple interconnected computer clusters, each with many cores. This is the case for the ARCHER2 HPE Cray EX supercomputer which has 128 cores per node, provided by dual AMD 7742 EPYC ™ 64-core processors. This is a 5,860 node (750,080 core) machine based at the University of Edinburgh, and is the UK national supercomputing service.

Having many cores per node may change the way in which applications should be run to ensure that they make best use of the available resources, and also perform and scale well. Efficient use of these machines may warrant moving away from the traditional "pure MPI" approach of allocating a single core per MPI process, as this results in many MPI processes on a node which all share the same network links, and so sending many messages causes congestion on the network. Furthermore the memory per process is limited when the total memory on a node is shared amongst a large number of MPI processes. Instead it may be advantageous to consider the use of the node as a whole rather than the individual cores. The use of MPI+OpenMP, where OpenMP is used between cores on a node and MPI is used between nodes, may be a good fit for taking advantage of the resources on ARCHER2 nodes.

MPI+OpenMP is available in many of the core ARCHER2 applications, including VASP, CASTEP, CP2K, GROMACS, LAMMPS, and Quantum ESPRESSO. For a user, using MPI+OpenMP may be more complex than using MPI alone, as the balance of threads and processes affects the performance, and this adds a further parameter than must be tuned for a particular use case. The performance achieved using MPI+OpenMP depends on a variety of factors including the node-level architecture (NUMA regions), the application, the number of nodes it is run on, and the particular use case itself. Choosing a sensible configuration for running with MPI+OpenMP is therefore extra work for the user which adds a barrier to its usage. It is therefore desirable to be able to provide clear guidance for users to achieve good performance when using MPI+OpenMP.

This paper will look at the MPI+OpenMP performance for a variety of applications on ARCHER2. For these applications the optimum configuration of threads and processes will be determined for a variety of test cases. These test cases will be selected to cover different scales of problem size. These results will aim give users a greater understanding of how MPI+OpenMP can be used to improve the performance of these key applications on ARCHER2. This paper will focus on a selection of the most used codes on ARCHER2 including CP2K, CASTEP, LAMMPS and GROMACS. The main aim of this paper is to find for which applications MPI+OpenMP may be advantageous and understand what factors influence this. A secondary objective is to be able to provide sensible suggestions to users on how to efficiently use MPI+OpenMP on ARCHER2 to hopefully increase its use across the system.

The paper is organised as follows: It will begin with an overview of the Cray EX architecture and highlight how this will relate to the use of MPI+OpenMP. It will then cover the applications discussed in this paper, giving information about them and their support for MPI+OpenMP. This will also include descriptions the test cases that will be used to benchmark the MPI+OpenMP performance for each application. The

results of MPI+OpenMP usage in these applications will then be presented. Finally this paper will conclude with a real user example case where a user was able to improve the performance of their simulation considerably using MPI+OpenMP on CP2K.

II. Cray EX architecture

ARCHER2 nodes are formed of two sockets each with a 64 core AMD EPYC 7742 processor which has a clock speed of 2.25 GHz [1]. There is 256 GB of memory per standard node (2GB per core) and 512 GB of memory per high memory node (4 GB per core). Nodes are connected together with the HPE Cray Slingshot interconnect. The node-level architecture is shown in Fig. 1.

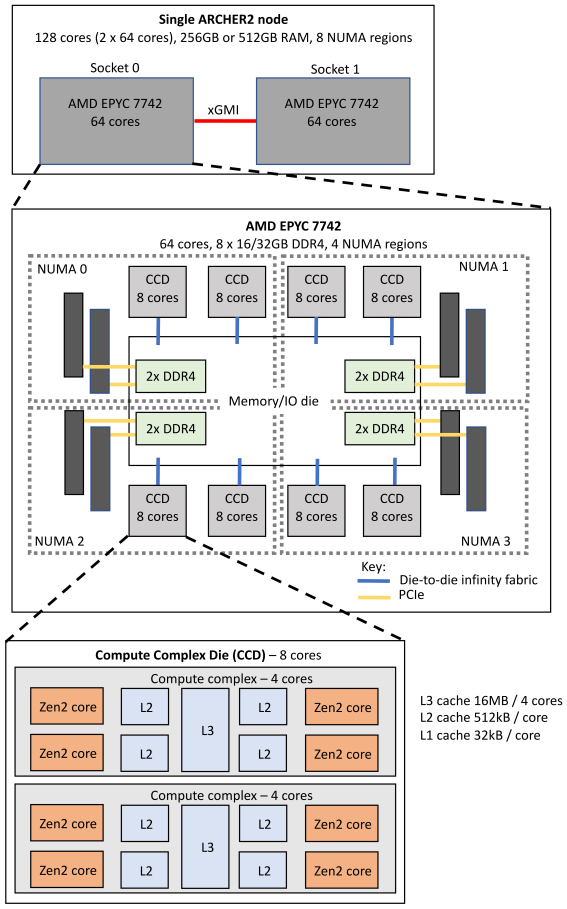

Fig. 1. The architecture of an ARCHER2 node. This has two sockets each with a 64 core AMD EPYC 7742 processor. Taken from [1].

The node is split into 8 non-uniform memory access (NUMA) regions of 16 cores (4 NUMA regions per processor). Groups of 4 compute cores form Compute Complexes (CCX) which share 16 GB of L3 cache. Two CCXs form a Compute complex Die (CDD) of 8 cores which are connected with die-to-die infinity fabric. Each core has its own 32kB L1 cache and 512 kB L2 cache.

For use of MPI+OpenMP this memory structure means that thread values of 1, 2, 4, 8, 16 are sensible options, so that the threads are confined to a single NUMA region and so do not share memory across different memory regions on a node. Additionally the fact that L3 cache is shared between 4 cores means that for up to 4 threads the threads will share the cache.

Compared to ARCHER2's predecessor, ARCHER, the compute nodes have many more cores (128 vs 24) and have a more complex memory layout. ARCHER was an Cray XC machine with nodes containing two 2.7 GHz, 12-core E5-2697 v2 (Ivy Bridge) series processors [2]. Standard nodes on ARCHER had 64 GB of memory, which equates to roughly 2.6 GB per core. Therefore the switch from ARCHER to ARCHER2 has meant in a reduction in the amount of memory per core. For users running memory intensive applications this has meant a greater likelihood that out of memory errors will occur when running with one process assigned to each core.

One solution to this is to underpopulate the nodes with processes, so that each process has more memory available to it. However, this can be undesirable as the processing power of the cores is not fully utilised. As an alternative, using MPI+OpenMP shifts the balance so that there is more memory available per process, whilst still ensuring each node is fully populated.

III. Applications

The ARCHER2 team support a number of core scientific applications. User demand for these varies. Fig. 2 shows the most used applications by node hours used for January 2022 as reported in [3].

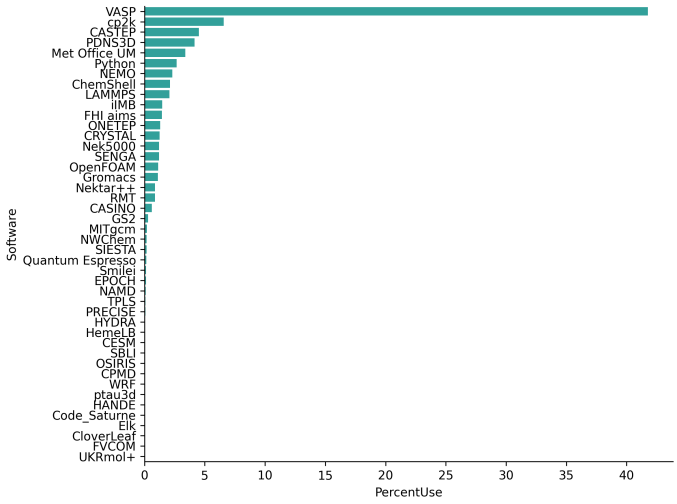

Fig. 2. The usage of applications on ARCHER2 for January 2022. The results are reported as the percentage use of node hours used [3]

By far the most used application in terms of node hours is VASP, followed by CP2K and CASTEP. These were also in the top 5 most used codes on ARCHER2's predecessor ARCHER along with GROMACS [4].

This study will investigate the performance of MPI+OpenMP in CP2K, GROMACS, CASTEP, LAMMPS, and Quantum ESPRESSO. These applications are well used

on ARCHER2 and are supported by the ARCHER2 team. The usage MPI+OpenMP in them is also well documented.

Results for VASP will not be presented due to reasons mentioned below, however using MPI+OpenMP in VASP on ARCHER2 was investigated.

A. CP2K

CP2K is a quantum chemistry and solid state physics package [5], [6]. At the core of CP2K is the Quickstep method for density functional theory (DFT). This uses the mixed Gaussian and plane waves approaches (GPW and GPAW). CP2K can perform a variety of different simulations such as molecular dynamics, metadynamics, Monte Carlo and energy minimisation. It can also simulate solid state, liquid, molecular, periodic, material, crystal, and biological systems. CP2K is the among the top 5 most used codes on ARCHER2 [3], and is freely available under the GPL licence.

CP2K is parallelised primarily though the use of MPI, however it has OpenMP parallel regions throughout its code base which allows it to exploit MPI+OpenMP parallelism. OpenMP is used in fast-Fourier transforms (FFTs), the realspace to planewave transfer, and the collocate and integrate routines [7]. CP2K is memory intensive and makes large use of MPI collectives.

1) Test Cases: This paper considers 3 different test case systems for CP2K. These are as follows:

- H2O-64 - This is a Born-Oppenheimer molecular dynamics (MD) simulation of 64 water molecules [8]. The QuickStep DFT method is used and for the basis sets a TZV2P basis is used. For the Exchange-Correlation energy the Local Density Approximation (LDA) is used and the planewave cutoff is set to 280 Ry. 10 MD steps are performed. This test case represents a very small system designed to scale to just a few ARCHER2 nodes (a few hundred cores).
- H2O-512 - This is a molecular dynamics simulation of 512 water molecules. The setup and parameters are the same as the 64 molecule case. This test case represents a system that should scale to around a few thousand ARCHER2 cores.
- LiH-HFX - This calculation uses the hybrid Hartree-Fock exchange (HFX) to calculate the single-point energy of a 216 atom Lithium-Hydride crystal. The plane wave energy cutoff is 300 Ry. This calculation is many more times computationally intensive than a local density calculation. The Hartree-Fock exchange calculation is also memory intensive. The MAX_MEMORY input parameter can be tuned at run time to set the amount of available memory for the Hartree-Fock module. This value depends on the available memory per MPI process, the MAX_MEMORY should be less than this to prevent out of memory errors. Using MPI+OpenMP is therefore a good option for this test case as it will increase the memory per MPI process. This test case scales to a few hundred ARCHER2 nodes (10,000 cores). Note that a minimum number of nodes are required in order to satisfy the memory requirements.

2) Installation and Usage: Here version 8.2 of CP2K will be used, which is built with GCC version 11.2. For the the scientific linear algebra routines the Intel MKL libraries [9] are used. Version 8.1.9 of Cray-mpich is used for the MPI library. The MPI-OFI communications protocol is used for most calculations, however the Mellanox UCX MPI protocol is used at larger node counts for the LiH-HFX system as this was better performing at large scales, when run on more than 32 nodes. The Cray-fftw library (v3.3.8.11) is used for the FFTW routines [10]. CP2K is built with the Libint, Libxc and ELPA libraries enabled [11]–[13], although only the use of ELPA relates to the performance, whereas the others add more scientific features such as additional exchange-correlation functionals (Libxc) and the Hartree-Fock exchange (Libint).

The hybrid MPI+OpenMP `cp2k.psmp` executable is used for all runs. The run time is recorded as the reported time from the CP2K timing report printed at the end of a calculation.

B. CASTEP

CASTEP is a density functional theory software package for calculating the electronic structure properties of solids, surfaces, molecules and liquids [14]. It uses plane-wave basis sets and supports geometry optimisations and molecular dynamics simulations among others. CASTEP is among the top 5 most used codes on ARCHER2. It requires a licence in order to use.

CASTEP makes use of MPI and OpenMP parallelism. Calculations of larger systems in CASTEP can often be memory intensive and use of MPI+OpenMP has been shown to be one strategy of reducing the memory requirements [15].

1) Test Cases: This paper considers 2 different test case systems for CASTEP. These are as follows:

- Al3x3 - This calculates the single point energy of a aluminium oxide 3x3 sapphire surface. For the exchange-correlation energy the LDA approximation is used and the plane wave cutoff energy is set to 400 eV. 2 k-points are used. This is a small test case which should scale to around 10 ARCHER2 nodes (1,000 cores).
- DNA - This calculates the energy of a poly-A strand DNA. The generalised gradient approximation (GGA) is used for the exchange-correlation energy, and the plane-wave cutoff energy is 1000 eV. Only the Gamma point is used to sample the Brillion zone. This is a large system with significant memory requirements. It should scale to a few hundred ARCHER2 nodes (10,000 cores).

2) Installation and Usage: Version 20.11 of CASTEP is used in these experiments. It is compiled with GCC version 10.2 and uses Intel MKL version 19 [9] for its linear algebra algorithms. The Cray-fftw library is used for the FFTW routines [10] and the MPI-OFI communications protocol is used with Cray-mpich version 8.1.4.

The `num_proc_in_smp` option in CASTEP allows control of the number of processes shared memory segments. This

allows these processes to communicate by shared memory rather than by the interconnect in order to speed up 3D FFTs. For these experiments `num_proc_in_smp` is set to 16 for the DNA system as this is the size of a NUMA region on ARCHER2. For the Al3x3 system this option is unset.

C. GROMACS

GROMACS is a molecular dynamics package that is used for simulating biological systems such as proteins, lipids and nucleic acids [16]. GROMACS is among the most used codes on ARCHER2 and is freely available to use under the GNU Lesser General Public license.

GROMACS is MPI and OpenMP parallelised. From version 4.6 multithreading was added to the particle mesh Ewald (PME) calculations [17], [18].

1) Test Cases: This paper will consider 2 different test case systems for GROMACS. These are MD simulations taken from the HecBioSim benchmark suite [19]. The performance of MD simulations in GROMACS are mainly dependent on the size of the system. These are as follows:

- 1.4M atom system - A MD simulation of a pair of human epidermal growth factor receptor (hEGFR) dimers. There are 1,403,182 atoms total. This simulation performs 10,000 MD steps.
- 3M atom system - A MD simulation of a pair of hEGFR tetramers. There are 2,997,924 atoms total. This simulation performs 10,000 MD steps.

2) Installation and Usage: In this paper version 2021.3 of GRAOMCS is considered. This is built with GCC 11.2. Cray-mpich version 8.1.9 is used with the OFI communications protocol.

Performance is reported as the GROMACS performance for the run in nanoseconds per day (ns/day).

D. LAMMPS

LAMMPS is a molecular dynamics code which is mostly used for materials modelling [20], [21].

LAMMPS is MPI parallelised for general use. However it can also be built with its OpenMP package [22] which provides multi-threaded versions for many pair interaction styles.

1) Test case: The LAMMPS test case is the same 3M atom system used for the GROMACS test case which was taken from the HecBioSim benchmark suite [19]. This does This does 10,000 MD steps and writes a restart file every 1,000 steps.

2) Installation and Usage: The stable version of LAMMPS as of January 2022 is tested. This is built with GCC version 10.2. The Cray-fftw library is used for the FFTW routines [10] and the MPI-OFI communications protocol is used with Cray-mpich version 8.1.4.

Performance is reported as the predicted nanoseconds per day reported at the end of the LAMMPS simulation.

E. Quantum ESPRESSO

Quantum ESPRESSO (QE) is collection of codes for electronic-structure calculations [23]. It is based on density-functional theory, plane waves, and pseudopotentials. It can perform variety of calculations of different features such as ground-state calculations (with a variety of functionals and VdW corrections), as well as structural optimisations, molecular dynamics, nudged elastic band calculations and more. It is fully open-source.

QE is MPI parallelised throughout its main algorithms. OpenMP loops have been added to its space integrals and point function evaluations, furthermore OpenMP is used in the 3D FFT and linear algebra libraries (SCALPACK and BLAS) [24].

1) Test cases: For Quantum ESPRESSO the following test cases will be used. These are taken from the QE benchmark suite [25]:

- GRIR443 - This is self-consistent (scf) calculation of graphene layer on a iridium surface. There are 443 atoms and 4 k-points are used.
- CNT - This is a scf calculation of a carbon nanotube functionalised with porphyrins. There are 1532 atoms total. A single k-point (the Gamma point) is used. This is a computationally and memory intensive calculation, which should scale to around 10 ARCHER2 nodes, which is equivalent to over 1000 cores.

2) Installation and Usage: In these tests version 6.8 of Quantum ESPRESSO is examined. This is built with GCC 11.2. The Cray-libsci library (v21.08.1.2) is used for the linear algebra libraries [26]. The MPI parallel version of the cray-hdf5 library is used for HDF5 support. For the MPI cray-mpich version 8.1.4 is used along with the MPI-OFI communications protocol.

The run time of a calculation is reported as the PWSCF Wall time. For the FFT a 2D slab decomposition is used when there are enough planes for each process, however in some cases a pencil decomposition is used when there are more processes than slabs.

For the GRIR system the `-npool` option (which controls how the k-points are shared amongst processes) is set to be equal to the number of k-points (`-npool 4`).

F. VASP

MPI+OpenMP is present in VASP version 6. Versions 6.2.1 and 6.3.0 are both present on ARCHER2. Version 6.2.1 is is built with GCC version 11.2 and uses Cray-libsci v21.04.1.1 [26]. Version 6.3.0 is also built with GCC version 11.2 and but uses the AMD Optimizing CPU Libraries (AOCL) version 3.1 [27] for BLAS and SCALAPACK. Both versions use Cray-fftw [10] and default OFI communications protocol.

The use MPI+OpenMP was tested in VASP for a TiO_2 test case found here [28]. However the total energies reported when running MPI+OpenMP differed from the non-threaded run. The cause of this will be investigated in the future, but because of this performance results for VASP will not be reported here.

IV. Methodology

A. Thread options

Thread values are chosen to ensure that the 16 core NUMA regions are evenly divided. This prevents threads being spawned across different memory regions and accessing memory from different memory regions. The thread values which satisfy this condition are 1, 2, 4, 8 and 16. To ensure that the nodes are fully occupied the number of MPI processes is set so that $threads \times processespernode = 128$. The number of processes is set with the Slurm `--tasks-per-node` option, and the `--cpus-per-task` option is set to equal the number of threads used to ensure the correct placement of processes.

B. Thread placement

The following options are added to the Slurm `srun` command in the job submission script at run time:

- `--hint=nomultithread` - Only use physical cores (avoids use of SMT/hyperthreads)
- `--distribution=block:block` - Allocate processes to cores in a sequential fashion.

Additionally the `OMP_PLACES=cores` environment variable is set in order to to generate the correct thread pinning.

C. Switching MPI implementation

HPE Cray MPICH can use two different low-level protocols to transfer data across the network. The default is the Open Fabrics Interface (OFI), but it is possible to switch to the UCX protocol from Mellanox at run time. This can be done by adding the following into the job script:

```
module swap craype-network-ofi craype-network-ucx
module swap cray-mpich cray-mpich-ucx
```

D. Results collection

The results from these experiments can be found at [29]. The time and performance results are usually averaged over 3 separate runs of the same calculation, with minimal deviation between results observed. For long running calculations (over 1 hour) a single run is performed.

V. Results

A. CP2K

Fig. 3 shows the run time for the CP2K H2O-64 test case on ARCHER2. The time is reported for up to 4 nodes and with 1, 2, 4, 8 and 16 threads per MPI process. This test case is small and is not meant to scale beyond a couple of nodes. However it does benefit from MPI+OpenMP across all scales. As the number of nodes increases the run time of the single threaded version increases and its performance becomes increasingly worse compared to the runs using multiple threads per process. Hence without MPI+OpenMP this benchmark would show no increase in performance when running on more than one node. Using 2 threads gives the best performance on one node, and on 2 and 3 nodes 4 threads is better, with this increasing to 8 threads on 4 nodes.

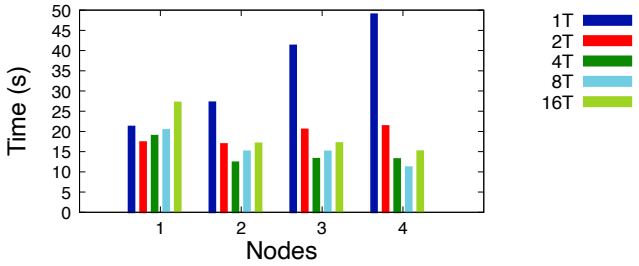

Fig. 3. The run times of the CP2K H2O-64 test case with MPI+OpenMP on ARCHER2.

From looking at the CP2K timing report the main contributors to the run time in the single threaded version of this benchmark are the calls to `MPI_Alltoall`, which become increasingly dominant on multiple nodes. This is shown in Table I. The main contributor for 1 threads is `mp_alltoall_l11v` which does an all-to-all exchange between all processes for long integers as is called once per MD step. Using MPI+OpenMP reduces the run time of these calls and offsets some of the communication cost. On ARCHER this same benchmark did not benefit from multi-threading [7] likely because the smaller nodes meant that MPI communications were less dominant.

TABLE I
THE CUMULATIVE TIME FOR MPI ALLTOALL CALLS FOR 1 AND 4 THREADS PER PROCESS.

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

Fig. 4 shows the run times for the CP2K H2O-512 test case. This is a larger version of of the 64 H2O molecule test case. The effect of MPI+OpenMP is similar for this benchmark where the use of multiple threads per process allows for further scaling beyond the single threaded version, which does not scale beyond 4 nodes. Using 2 or 4 threads per process usually gives the best performance. MPI calls also dominate the run time of this benchmark on multiple nodes, which pushes up the run time of the single threaded calculation. This effect is reduced with MPI+OpenMP.

Fig. 5 shows the run times of the LiH-HFX test case with MPI+OpenMP. This test case is uses a hybrid functional which involves the calculation of the Hartree-Fock exchange and is both computationally and memory intensive. The results show that this test case also benefits from the use of MPI+OpenMP. Using 4 or 8 threads per process give the best performance across the node numbers tested. On 32 and 64 nodes the increase in performance with multiple threads is less drastic than in the previous test cases as this test case is dominated more by the computation of the integrals for the Hartree-Fock

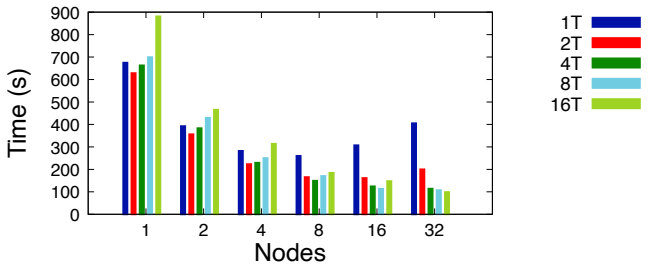

Fig. 4. The run times of the CP2K H2O-512 test case with MPI+OpenMP on ARCHER2.

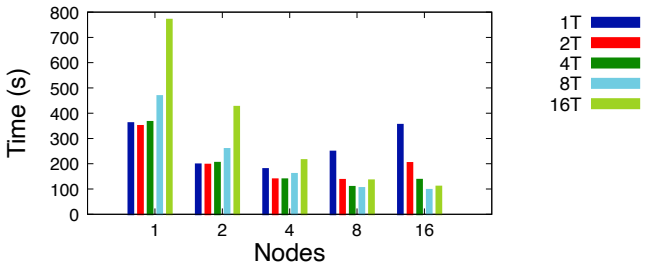

Fig. 6. The run times of the CASTEP Al3x3 test case with MPI+OpenMP on ARCHER2.

exchange than the communication costs. On 256 nodes the run time of the 8 thread calculation is nearly 3 times faster than the single threaded calculation. Using multiple threads for this test case has the advantage that the available memory is shared between fewer processes. Therefore integrals can be stored in memory rather than having to be recomputed.

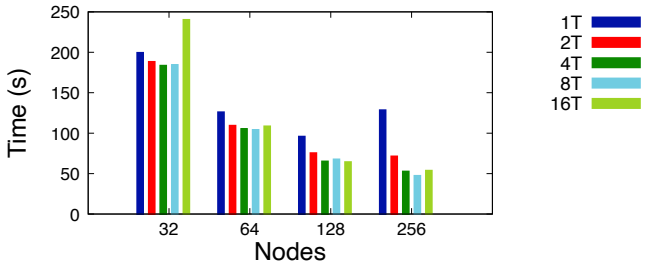

Fig. 5. The run times of the CP2K LiH-HFX test case with MPI+OpenMP on ARCHER2.

The run time of CP2K on ARCHER2 is dominated by MPI_alltoall calls. As ARCHER2 has many cores per node, these calls become costly as the number of nodes is increased. The results of the experiments in this paper have shown that MPI+OpenMP can improve the performance when running across all node counts, but especially on more nodes when the communications make up a greater fraction of the overall run time. MPI+OpenMP allows for aggregation of MPI messages which reduces the communication time. This can help extend the scaling out to greater node counts than with just MPI alone.

B. CASTEP

Fig. 6 shows the run time of the CASTEP Al3x3 test case on ARCHER2 with multi-threading. For this system using MPI+OpenMP with 2 or 4 threads per process gives a performance similar to the single threaded version. On 8 and 16 nodes the run time of the single threaded version increases, however using MPI+OpenMP improves the performance at this scale. The run time of CASTEP has been shown to be dominated by all-to-all communications when running on many nodes.

This test case has 2 k-points, so when run on more than 2 nodes the k-points are spread across multiple nodes, with the plane waves spread across the cores within a node. On 4 nodes the 88,000 plane waves in this calculation are spread so that there are around 344 plane waves per core, which is less than the suggested amount for good parallelism [30].

Fig. 7 shows the performance of the DNA test case on 1, 2 and 4 threads per MPI process, note that a larger value here represents better performance. 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.

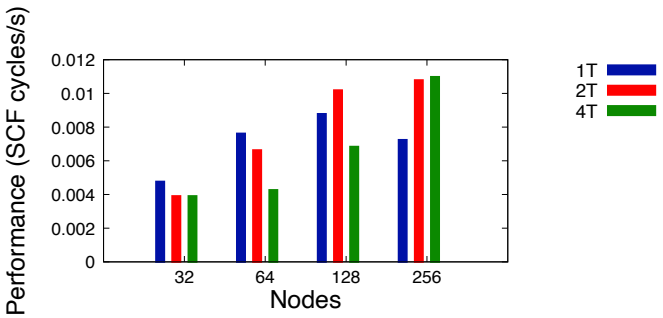

Fig. 7. The performance of the CASTEP DNA test case with MPI+OpenMP on ARCHER2.

C. GROMACS

Fig. 8 shows the GROMACS performance in nanoseconds per day for the 1400k atom test case. This test case scales well up to 16 nodes on ARCHER2 when using a single thread per process. On 32 nodes the PME communications start to take up a greater fraction of the run time. For example on 32 nodes (with 1 thread per process) the PME Wait and Receive took 14 seconds, whereas on 16 nodes (with 1 thread per process) this took just under 1 second. The GROMACS documentation states that these communications can become a limiting factor when the parallelism is high and when the nodes contain many cores [17]. In this case MPI+OpenMP can benefit the performance. This is shown for 2 threads per process on 32 nodes, however the performance is still less than the 16 node single threaded case.

For the 3000k atom test case in Fig. 9 the performance is clearly improved by MPI+OpenMP on 32 and 64 nodes. On 32

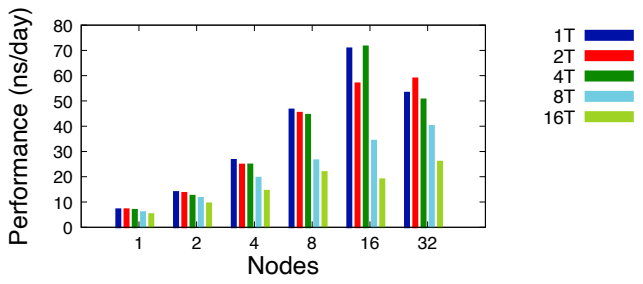

Fig. 8. The performance in nanoseconds per day of the GROMACS 1400k atom test case with MPI+OpenMP on ARCHER2.

node 2 threads per process gives a performance of 56.8 ns/day compared with 44.2 ns/day for a single thread and on 64 nodes the performance is 65 ns/day for 4 threads per process, which is again greater than the 55.1 ns/day reported for a single thread per process. This is due to the difference in time taken for the PME Wait and Receive communications which is 5.3 seconds for the single threaded version and 1.2 seconds with 4 threads per process.

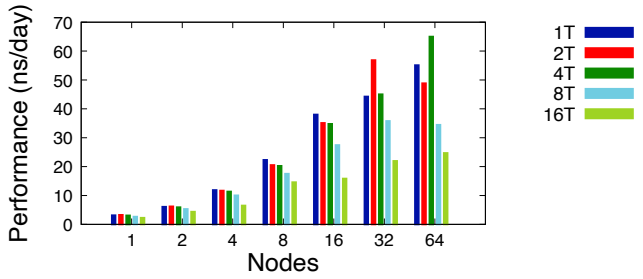

Fig. 9. The performance in nanoseconds per day of the GROMACS 3000k atom test case with MPI+OpenMP on ARCHER2.

For GROMACS using MPI+OpenMP does not show much of an improvement to the performance in general. The communications in GROMACS are generally quite efficient and only start to contribute significantly to the performance at large scales. However, once the calculations start to reach the limit of their scaling using MPI+OpenMP can allow for slight improvements in the performance compared to using MPI alone. This is because using multiple threads can reduce the communication cost. In these cases using 2 or 4 threads per process can improve the performance, but any greater than this then the performance gets considerably worse. This is likely because if there are more than 4 threads the threads are no longer sharing L3 cache on the ARCHER2 nodes. GROMACS is not memory intensive so having threads which span more than one cache will negatively affect the performance.

D. LAMMPS

Fig. 10 shows the performance in ns/day for the LAMMPS 3000k atom test case on 1, 2, 4 and 8 threads. Up to 32 nodes there is no benefit from using multiple threads, and the performance decreases with more threads. Only on 64 and 128 nodes does using MPI+OpenMP improve the performance.

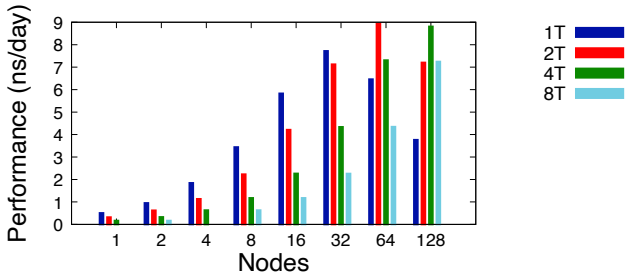

Fig. 10. The performance in nanoseconds per day of the LAMMPS 3000k atom test case with MPI+OpenMP on ARCHER2.

The MPI task timing breakdown shows that the writing of the output (the restart file) takes over 50% of the total run time on 64 nodes with a single thread per process. This write time is reduced when using multiple threads per process. The performance for the MPI-IO in LAMMPS on ARCHER2 is poor when using the OFI protocol with a large number of processes, and using multiple threads reduces the total number of processes. This is a known issue that can be overcome by using UCX. This will be investigated in the future based on MPI-IO investigations ongoing in the ARCHER2 CSE team. Again, as with GROMACS, using more than 4 threads per process results in poor performance due the arrangement of L3 cache on ARCHER2.

E. Quantum ESPRESSO

The run times for the QE GRIR system on 1, 2, 4 and 8 threads per process and across 2, 4, 8 and 12 nodes is shown in Fig. 11. On 1 node this calculation fails with an out of memory (OOM) error across all thread values examined. On 2 nodes with a single thread per process the calculation also fails with OOM, but using MPI+OpenMP allows it to run successfully. However using MPI+OpenMP does not appear to benefit the performance of this calculation. Only on 12 nodes does using 2 threads slightly improve the run time, but at this point the scaling is poor so it is not worthwhile running on as many nodes.

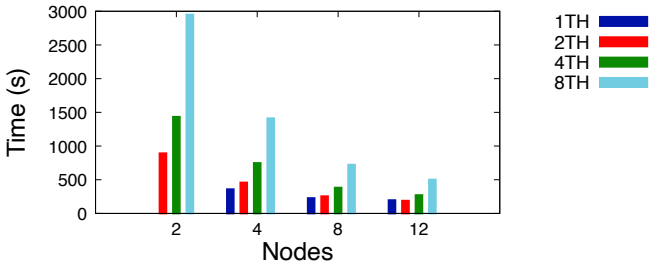

Fig. 11. The run time of the QE GRIR433 test case with MPI+OpenMP on ARCHER2. OOM errors are reported when running on 1 node, and for 2 nodes with a single thread.

The QE CNT test case has even higher memory requirements and fails with a OOM error on 1 and 2 nodes regardless of how many threads are used. On 4 nodes when only one threads is used (1T) the calculation also fails with OOM, however this is not the case for when MPI+OpenMP is used. This is because there is more memory per process available which is key for this test case.

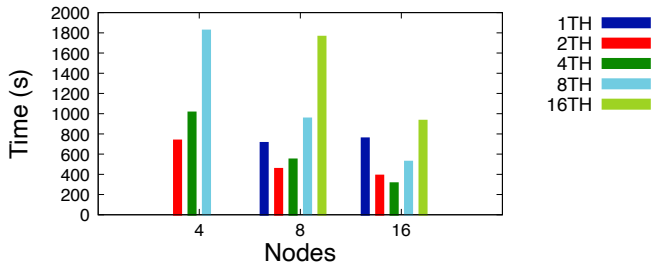

Fig. 12. The run time of the QE CNT test case with MPI+OpenMP on ARCHER2. OOM errors are reported when running on 1 or 2 nodes, and for 4 nodes with a single thread.

Fig. 12 shows the run time for the CNT test case on 4, 8 and 16 nodes. On 4 nodes MPI+OpenMP does not appear to improve the performance, however it does prevent the calculation running out of memory as noted above. On 8 nodes using 2 threads per process is around 1.5 times faster than using single thread and on 16 nodes using 4 threads per process is over 2 times faster than the single threaded case. This is due to the increase in memory per process. As shown in Table II below, even underpopulating with 64 processes per node gives significant performance increase on 8 nodes. Using 2 threads per process only improves this slightly.

TABLE II
THE RUN TIMES FOR THE CNT SYSTEM ON 8 NODES FOR DIFFERENT CONFIGURATIONS OF THREADS AND PROCESSES PER NODE (PPN).

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

Of the QE systems that were investigated both had sizable memory requirements which prevented them from running on a single node. In both cases using MPI+OpenMP was able to prevent them from failing with OOM by increasing the available memory per process compared to the single threaded case. This is particularly useful on ARCHER2 because the memory per process is only roughly 2 GB if one process per core is allowed. For the large CNT system the performance was also significantly improved with MPI+OpenMP. This is because this calculation is memory limited.

VI. USER EXAMPLE

In some circumstances using MPI+OpenMP can result in big differences to the overall run time of a calculation. For CP2K it has already been shown that using MPI+OpenMP can offer performance benefits compared to using MPI alone when running the CP2K performance benchmarks. This section will look at the affect of using MPI+OpenMP for a real user simulation. This is a long running molecular dynamics simulation of around 100 water molecules which uses the isothermal-isobaric ensemble. The QuickStep method and the DZVP basis sets are used. For the exchange-correlation energy the generalised gradient approximation (GGA) is used, and the planewave cutoff is 1000 Ry. Switching using MPI+OpenMP with 4 threads per process was found to greatly speed up the users simulation.

Fig. 13 shows the time per MD step when using 1, 2, 4, 8 and 16 threads per process as calculated from averaging over 1,000 MD steps. The calculation was run on 4 ARCHER2 nodes.

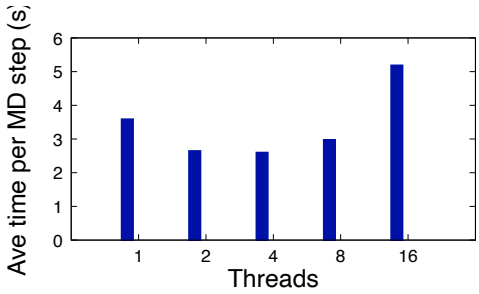

Fig. 13. The time per MD step for different values of threads per process.

For 4 threads per process the time taken per MD step is around 70% of that of the pure MPI case. For a long running MD simulation this will add up a big difference in the run time. Not only this but the compute resources consumed will be reduced by this same factor. This is very beneficial for users.

The large reduction seen the run time when using MPI+OpenMP is mainly due to the time taken for the MPI calls. These are summarised in Table III.

TABLE III
THE CUMULATIVE TIME FOR CP2K AND DIFFERENT FOR MPI CALLS ON 1 AND 4 THREADS PER PROCESS. THE CALCULATION IS DONE FOR 1,000 MD STEPS.

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%

The largest reduction in time when using 4 threads per process is seen in the calls to mp_sum_d, which is reduced by 90%. However there are also large reductions in the time taken for the other MPI calls, with these now taking around half of the time. The cost of these communications, which have been found to be dominant in CP2K on ARCHER2, can be greatly reduced by using MPI+OpenMP.

VII. Conclusions

This paper explored the use of MPI+OpenMP on a Cray EX system - ARCHER2. A range of scientific applications were examined and for each of these applications different test cases were investigated in order to cover different user usage scenarios. Applications were chosen based on their usage on the system. The results showed that use of MPI+OpenMP has performance benefits over using single threaded "MPI only" for almost all applications tested.

It was found that CP2K and CASTEP can benefit from using MPI+OpenMP for all test cases examined and whether run on a few or many nodes. These applications are often dominated by MPI communications such as `MPI_Alltoall` and using MPI+OpenMP reduces the communications costs by allowing for message aggregation which results in fewer messages overall. Using 2 or 4 threads per process is usually a good choice for getting good performance across a range of scales.

For Quantum ESPRESSO using MPI+OpenMP was able to help with fulfilling the memory requirements. When using pure MPI on ARCHER2 it is possible that the available memory per process is not large enough and the calculation will crash with out of memory errors. For the larger CNT system the increase in memory with MPI+OpenMP was also able to help improve the performance. Using MPI+OpenMP with 2 or 4 threads per process is therefore a good idea for these memory intensive calculations.

For the classical molecular dynamics codes that were investigated, GROMACS and LAMMPS, using MPI+OpenMP was found to have no performance benefit unless at the scaling limit. Classical molecular dynamics codes usually have lower memory requirements than quantum DFT based codes and also have fewer collective communications. This means that they can cannot benefit as much from using MPI+OpenMP. The larger MD systems tested here typically scale to around 32 nodes when one thread per process is used. By using MPI+OpenMP with 2 or 4 threads per process the scaling can be extended out slightly.

Finally this paper looked at a real user use case where MPI+OpenMP was able to improve the performance of a CP2K simulation by nearly 40%. This clearly demonstrates that using MPI+OpenMP can be of large benefit to users.

References

[1] "ARCHER2 Hardware," https://docs.archer2.ac.uk/user-guide/hardware/, accessed: 2022-04-12.

[2] "ARCHER Hardware," http://www.archer.ac.uk/about-archer/hardware/, accessed: 2022-04-12.

[3] "Software usage data on ARCHER2," https://www.archer2.ac.uk/news/2022/02/07/software-usage-data.html, accessed: 2022-04-12.

[4] "ARCHER code use," https://www.archer2.ac.uk/news/2021/02/04/archer-code-use.html, accessed: 2022-04-12.

[5] T. D. Kühne, M. Iannuzzi *et al.*, "CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations," *The Journal of Chemical Physics*, vol. 152, no. 19, p. 194103, 2020. [Online]. Available: https://doi.org/10.1063/5.0007045

[6] "CP2K," https://www.cp2k.org, accessed: 2022-04-12.

[7] I. Bethune, F. Reid, and A. Lazzaro, "CP2K Performance from Cray XT3 to XC30," 2014, cray User Group (CUG) 2014 ; Conference date: 06-05-2014 Through 08-05-2014.

[8] "CP2K performance," https://github.com/cp2k/cp2k/tree/master/benchmarks/QS/, accessed: 2022-04-12.

[9] "Intel Math Kernel Library," https://www.intel.com/content/www/us/en/developer/articles/guide/intel-math-kernel-library-intel-mkl-2019-getting-started.html, accessed: 2022-04-12.

[10] "FFTW," http://www.fftw.org, accessed: 2022-04-12.

[11] E. F. Valeev, "Libint: A library for the evaluation of molecular integrals of many-body operators over Gaussian functions," http://libint.valeyev.net/, version 2.6.0.

[12] A. Marek, V. Blum *et al.*, " The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science," *Journal of Physics: Condensed Matter*, vol. 26, p. 21, 2014. [Online]. Available: https://doi.org/10.1088/0953-8984/26/21/213201

[13] S. Lehtola, C. Steigemann *et al.*, "Recent developments in libxc — a comprehensive library of functionals for density functional theory," *SoftwareX*, vol. 7, pp. 1–5, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711017300602

[14] "CASTEP," http://www.castep.org/, accessed: 2022-04-12.

[15] E. Higgins, M. Probert *et al.*, "Hybrid OpenMP and MPI within the CASTEP code," *ARCHER eCSE Techn. Rep. TR-eCSE01–017*, 2015.

[16] P. Bauer, B. Hess, and E. Lindahl, "GROMACS 2022 Manual," Feb. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6103568

[17] "Acceleration and parallelization," https://www.gromacs.org/Documentation/Acceleration_and_parallelization, accessed: 2022-04-12.

[18] F. Affinito, "Performance Analysis and Petascaling Enabling of GROMACS," Sep. 2012. [Online]. Available: https://doi.org/10.5281/zenodo.814480

[19] "HecBioSim Benchmark Suite," https://www.hecbiosim.ac.uk/access-hpc/benchmarks, accessed: 2022-04-12.

[20] "LAMMPS," https://www.lammps.org, accessed: 2022-04-12.

[21] A. P. Thompson, H. M. Aktulga *et al.*, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[22] "LAMMPS OpenMP Package," https://docs.lammps.org/Speed_omp.html, accessed: 2022-04-12.

[23] "Quantum ESPRESSO," https://www.quantum-espresso.org, accessed: 2022-04-12.

[24] C. Cavazzoni, "PRACE workshop on application porting and performance tuning," https://materials.prace-ri.eu/100/1/porting_workshp_cc.pdf, accessed: 2022-04-12.

[25] "Quantum ESPRESSO benchmarks," https://github.com/QEF/benchmarks/tree/master/, accessed: 2022-04-12.

[26] "Cray Scientific and Math Libraries (CSML)," https://support.hpe.com/hpesc/public/docDisplay?docId=a00113947en_us&docLocale=en_US&page=Cray_Scientific_and_Math_Libraries_CSML_.html, accessed: 2022-04-12.

[27] "AMD Optimizing CPU Libraries," https://developer.amd.com/amd-aocl/, accessed: 2022-04-12.

[28] "VASP benchmarks," https://github.com/hpc-uk/archer-benchmarks/tree/main/others/VASP, accessed: 2022-04-12.

[29] "ARCHER2 MPI+OpenMP benchmarking," https://github.com/holly-t/ARCHER2_hybrid_benchmarking, accessed: 2022-04-12.

[30] "CASTEP on Parallel Computers," http://www.castep.org/CASTEP/FAQGeneralProblems, accessed: 2022-04-12.