Scalability and Performance of OFI and UCX on ARCHER2

Jaffery Irudayasamy
EPCC
The University of Edinburgh, UK
J.Irudayasamy@epcc.ed.ac.uk

Juan F. R. Herrera
EPCC
The University of Edinburgh, UK
J.Herrera@epcc.ed.ac.uk

Evgenij Belikov
EPCC
The University of Edinburgh, UK
E.Belikov@epcc.ed.ac.uk

Michael Bareford
EPCC
The University of Edinburgh, UK
M.Bareford@epcc.ed.ac.uk

***Abstract*—OpenFabrics Interfaces (OFI) and Unified Communication X (UCX) are both transport protocols that underlie the HPE Cray MPICH library on HPC systems like ARCHER2. They can be selected at runtime by users. This paper presents the scalability and performance of OFI and UCX transport layer protocol implementations on ARCHER2, an HPE Cray EX system that features Slingshot 10 interconnect. We use ReproMPI microbenchmarks to study the performance of MPI collectives and run experiments using some of the most commonly used applications on ARCHER2. The results show that in most cases OFI and UCX performance is comparable on under 32 nodes (16384 cores) but for larger number of nodes OFI runs more reliably. Ultimately, when it comes to applications there is no one-size-fits-all solution and profiling can facilitate tuning for best performance.**

I. Introduction

The performance of MPI collective communication primitives directly affects many commonly used HPC applications that heavily rely on those primitives. When multiple underlying transport layer protocol implementations such as OFI (OpenFabrics Interfaces) [1] and UCX (Unified Communication X) [2] are available on a system, we need to understand performance and scalability trade-offs to be able to advise the users on most efficient setup choices.

This paper presents the results of investigating scalability and performance of OFI and UCX transport protocol implementations used by the HPE Cray MPICH library on ARCHER2, an HPE Cray EX system with Slingshot 10 interconnect, following a major software upgrade in 2023. The results build on and extend previous work that has demonstrated that for various applications and node counts either OFI or UCX may result in better performance [3]. We also complement *Khorassani et al.*'s comprehensive study [4] by using larger number of nodes and larger applications, whilst they also provide detailed results on GPU-aware communication and also discuss measurements using Slingshot 11.

To obtain reproducible and statistically sound results we use the ReproMPI microbenchmark suite [5], [6] and methodology advocated by its authors. Additionally, we run various test cases for some of the most commonly used applications on ARCHER2 such as VASP, CASTEP, GROMACS, and NEMO. Compared to previous work [3], the number and scale of runs is substantially increased and, where relevant, we also report the dispersion of the measurements. Additionally, ReproMPI allows us to vary messages sizes in a similar way to the OSU benchmark suite which is used for comparison.

We find that following the 2023 upgrade on ARCHER2 OFI performs much more reliably and in most cases matches or outperforms UCX, which may require some tuning when run on larger node counts. However, whilst we believe OFI is a reasonable default, the best setting will depend on the application and profiling can assist with performance tuning.

In particular, for a medium-sized CASTEP case, OFI and UCX perform similarly for up to four nodes when highest performance is reached. For GROMACS OFI matches UCX on up to and including 16 nodes and outperforms UCX on larger node counts. For NEMO we observe similar performance on up to 512 nodes, where OFI starts to outperform UCX and the difference in performance appears to increase with increasing node count. For VASP, performance is similar up to 16 nodes, but overall UCX achieves better scaling resulting in lower elapsed runtime. Further studies are needed to investigate the effects of various MPI-related environment variable settings that may have a substantial effect on performance.

This paper is structured as follows. The methodology employed and experimental environment are described in II. The applications chosen for the performance comparison are listed in Section III. Section IV shows the performance results and Section V dicusses the MPI profiles of the applications. Section VI concludes.

II. Methodology

We build and run several microbenchmarks and selected applications on the UK National Supercomputing Service ARCHER2.

A. Experimental Environment

ARCHER2 is an HPE Cray EX system with Slingshot 10 interconnect consisting of 5,860 nodes (a total of 750,080

cores). Each node comprises two 64-core 2.25 GHz AMD EPYC 7742 Rome (Zen2) processors with the default frequency set to 2 GHz, whilst 1.75 GHz and and 2.25 GHz options are available via Slurm scheduler and module settings. The standard nodes have 256 GiB DDR-4 DRAM, whilst each of the 584 high-memory nodes has 512 GiB. HPE Cray Slingshot 10 interconnect utilises interconnect system software version 2.0.2 and uses two bi-directional 100 Gbps links per node[1]. The Lustre parallel file system with support for striping is used (`/work`, HPE Cray ClusterStor with 14.5 PB capacity supported by 12 disk units (Object Storage Targets or OSTs)).

Each node provides non-uniform memory access (NUMA): each NUMA region contains 16 cores subdivided into two Core Complex Dies (CCD), which are themselves split into two Core Complexes (CCX). Each CCX comprises 4 cores sharing 16MB L3 cache, with private L2 cache of 512KB and private L1 cache of 32KB per core.

During end of May and beginning of June of 2023 ARCHER2 underwent a major software upgrade including improvements to interconnect performance and reliability. This upgrade also made available more recent versions of compilers, libraries and various tools (e.g., CrayPat profiler).

The compute nodes are running Cray OS 2.4.109 (based on SUSE Linux Enterprise Server 15 SP4) with Slingshot interconnect system software version 2.0.2 and HPE Cray Management Software (CMS) 1.3.1. Slurm is used to manage the queues and schedule jobs. Prior to the upgrade a socket referred to 16 cores that share a DRAM memory controller, whilst after the upgrade the setting of a socket now refers to a CCX (i.e. 4 cores sharing and L3 cache). Various compiler suites are supported: the HPE Cray Compiling Environment (CCE) 15.0.0, the GNU Compiler Collection (GCC) 11.2.0, and AMD Optimizing Compiler Collection (AOCC) 3.2. Cray MPICH 8.1.23 (based on MPICH 3.4a) is used as communication library, which supports OFI (OpenFabrics Interfaces) and UCX (Unified Communication X) transport layer protocol implementations. Among other libraries Cray LibSci 22.12.1.1 is available, along with FFTW 3.3.10.3 and CrayPat profiler version 22.12.0.

B. Measurement procedure

For the measurements of several blocking MPI collectives we rely on ReproMPI [5], [6], which is a microbenchmarking suite and tool that allows for statistically sound benchmarking by running a sufficient number of repetitions and by using appropriate non-parametric statistical tests for analysis. It is also possible to extract the raw data in addition to a summary. We run ReproMPI for the following blocking collectives: `MPI_Bcast`, `MPI_Scatter`, `MPI_Allgather`, `MPI_Gather`, `MPI_Allreduce`, `MPI_Scan` and `MPI_Alltoall`. We vary messages size between 1 Byte and 1 Megabyte from 4 up to 512 nodes (256 in most cases). Within a run the repetition varies from 5 to 500, 300 being the `nrep` repetition count setting used for most of the function calls. Due to larger run-time `nrep` is set to 5 for `MPI_Alltoall` (and we use only 5 runs). Each ReproMPI run is performed 10 times (`srun` calls). We report the mean (across runs) of median runtimes (reported by ReproMPI for a set of repetitions of each run). In some cases largest message sizes are omitted due to memory limitations. We also use OSU microbenchmarks for comparison, limited to up to 128 nodes (64 in most cases). Please refer to the Appendix section for the discussion of the OSU results.

Application runs are setup in different ways. Please refer to each of the subsections below for the respective description of the setup and figures of merits used to represent performance.

[1]see https://docs.archer2.ac.uk/user-guide/hardware/#interconnect-details

III. Applications

This section will describe the applications used to carry out the performance study.

A. CASTEP

CASTEP is a leading code for calculating the properties of materials from first principles [7]. Using density functional theory, it can simulate a wide range of material properties such as energetics, structure at the atomic level, vibrational properties, electronic response properties etc. In particular, it has a wide range of spectroscopic features that link directly to experiment, such as infra-red and Raman spectroscopies, NMR, and core level spectra.

The present CASTEP code is written in Fortran 2003 using a modular structure. CASTEP employs three levels of parallelism: G-vectors (i.e. basis-set), k-points, and bands.

The `al3x3` simulation cell comprises a 270-atom (108 Al, 162 O) sapphire surface with a vacuum gap. There are only 2 k-points, so it is a good test of the performance of CASTEP's other parallelisation strategies.

CASTEP was run on ARCHER2 using node counts of 1, 2, 4, 8, and 16. The testcase was executed 5 times and the best performance per node count has been considered and plotted.

B. GROMACS

GROMACS (GROningen Machine for Chemical Simulations) [8] is a versatile package to perform molecular dynamics, i.e., simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins, lipids and nucleic acids that have a lot of complicated bonded interactions, but since GROMACS is extremely fast at calculating the non-bonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers.

The `benchPEP` benchmark is part of the GROMACS benchmark set, which includes typical simulation systems from various research projects and covers a wide range of system sizes from 6k to 12M atoms. `benchPEP` specifically simulates 12 milliom atoms, representing peptides in water, with a 2 femtosecond (fs) time step. All bonds in the system are constrained, which means that the update step has to be done on the CPU.

CASTEP was run on ARCHER2 using node counts of 1, 2, 4, 8, 16, 32, 64, 128, and 256. The nodes were fully populated, so 128 MPI processes per node. The testcase was executed 5 times and the best performance per node count has been considered and plotted.

C. NEMO

Nucleus for **E**uropean **M**odelling of the **O**cean (NEMO) is a modelling framework for running simulations of the Earth's oceans [9]. The NEMO code is written in Fortran 2008 and is comprised of three core engines, namely, OCE (ocean dynamics), SI^3 (sea-ice dynamics) and TOP (biogeochemistry).

The NEMO software also includes many reference configurations and tests. One of the tests, called BENCH [10] is ideal for benchmarking performance since it requires simple text-based input files (or namelists), and generates minimal output. The impact of file system load, which can be significant on a large shared system such as ARCHER2, is therefore minimised.

Essentially, the BENCH test allows you to run any NEMO configuration with idealized grid and initial state. For this paper we ran BENCH using the ORCA12 grid. It has a horizontal resolution of 1/12 degree and 75 levels in the vertical, making 991.6 million points in total.

BENCH was run on ARCHER2 using node counts of 32, 64, 128, 256, 512 and 1024. The nodes were always half populated with ocean tasks arranged such that was one idle core between each process. And so, for ARCHER2, there were 64 ocean processes per node. In the text that follows, NEMO namelist parameters will be indicated using a monospaced font.

For the smallest node count (32), the number of simulation steps (`nn_itend`) was set to 1000. The simulation time per step is 300 seconds if using the ORCA12 grid. Running for 300k simulation seconds (or 3.47 simulation days) took around 30 mins when using 32 nodes. The number of simulation steps was doubled when moving to the next node count in order to prevent the run times from becoming too short (e.g., `nn_itend` $= 32000$ for 1024 nodes).

In NEMO, the horizontal dimensions are labelled i and j. At 32 nodes then the number of processors in the i direction (`jpni`) was set to 64 and the number of processors in the j direction (`jpnj`) was 32, i.e., `jpni` $\times$ `jpnj` $= 64 \times 32 = 2048$, the total number of ocean processes. Table I shows the values of the aforementioned namelist parameters for each node count.

TABLE I: NEMO namelist values

node count	nn_itend	jpni	jpni
32	1000	64	32
64	2000	64	64
128	4000	128	64
256	8000	128	128
512	16000	256	128
1024	32000	256	256

By default, the BENCH configuration features the PISCES component, which is an initialism for Pelagic Interaction Scheme for Carbon and Ecosystem Studies. This circulation and bio-geochemistry model requires substantial memory bandwidth however, and could interfere with detecting differences in communications performance between OFI and UCX. We remove this possibility by deactivating the PISCES component and instead run with just an age tracer, which tracks the time-dependent spread of surface waters into the ocean interior. We configure NEMO in this way by setting two parameters in the `namelist_top_cfg` file.

```
ln_pisces = .false.
ln_age = .true.
```

D. VASP

VASP (Vienna Ab initio Simulation Package) is a powerful software tool for simulating materials at the atomic level. It performs calculations like electronic structure and quantum-mechanical molecular dynamics, all based on fundamental physical principles. VASP leverages parallelization techniques like MPI for efficient use of computing resources. Additionally, it can utilize OpenMP threading and OpenACC for potential acceleration on GPUs. Notably, VASP is the most popular code used on the ARCHER2 computing machine.

We investigated the performance difference between OFI and UCX for the VASP application on ARCHER2. We used the pre-installed VASP 6 executable on ARCHER2 and the example input files for H2O adsorption on TiO2. We ran the application with both OFI and UCX communication libraries. For OFI, we additionally tested the impact of the `MPICH_OFI_RMA_STARTUP_CONNECT` flag. This flag sets up connections between all processes (ranks) on each node during MPI initialization. This can be beneficial for applications using remote memory access (RMA) with an all-to-all communication pattern within each node.

To comprehensively assess performance, we executed the application with these three different configurations, utilizing 1, 2, 4, 8, 16, 32, 64, 128, and 256 nodes. Each node utilized 64 tasks, with 2 cores assigned per task. We repeated these experiments for each framework and each configuration for three iterations. The results, including various plots depicting LOOP+ (ionic step) performance, were obtained.

For profiling, we employed the CrayPat Lite module, available on ARCHER2. We ran the application using 8, 32, and 64 nodes across three configurations, and MPI profiles were obtained for comparative analysis.

IV. Experimental results

In this section we present our measurement results, starting with microbenchmarks run using ReproMPI, followed by application performance. Note the logarithmic scale for the ReproMPI figures.

A. *ReproMPI Microbenchmark*

We run ReproMPI for various collectives and observe that there is no single winning configuration: for different collectives, message sizes and number of nodes either OFI or UCX may perform better as shown below. However, after the upgrade OFI performs more reliably and in most cases outperforms or matches UCX. UCX tends to behave less robustly with increasing number of nodes and message sizes.

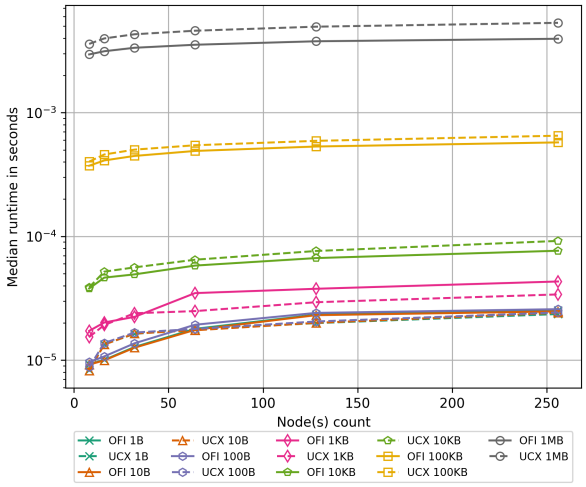

Fig. 1: MPI_Bcast performance

Figure 1 demonstrates performance of `MPI_Bcast` for up to 256 nodes. We observe that for message sizes larger than 1000 bytes OFI consistently outperforms UCX by up to 38% (for messages for size 10^6 on 256 nodes). However, for smaller messages UCX performs slightly better. A somewhat similar picture can be observed in Figures 2 and 3 for `MPI_Scatter` and `MPI_Gather`, respectively. For these collectives OFI slightly outperforms UCX in most cases, except for smallest message size of 1 byte and smaller node counts of under 128 nodes. Additionally the difference is close to $2\times$ in favour of OFI on 256 nodes for largest message size of 10^5 bytes. For `MPI_Scan`, Figure 4 illustrates OFI outperforming UCX for message sizes over 10^4 by up to $3.2\times$ (form message size of 10^5 bytes, dropping to $1.7\times$ for 10^6-byte messages), whilst for smaller messages using UCX consistently results in lower run times, with differences up to $2.5\times$ (for 1-byte messages).

Figure 5 illustrates the behaviour of the `MPI_Allgather` collective, where in most cases OFI outperforms UCX, by up to around 33% (for 512 nodes and message size of 100 bytes). Figure 6 depicts the performance of `MPI_Allreduce` for which we observe UCX outperforming OFI for in all cases, except for message size of 10^5 using 128 or more nodes and for message sizes of 10^3 or fewer bytes for node counts of fewer or equal to 32. Finally, Figure 7 presents the performance of `MPI_Alltoall`, which leads to highest absolute run times and scales less well than other collectives. In part this is why we only used up to 128 nodes and messages sizes of up to 10^5 for these runs. We observe OFI performing on par with or slightly better than UCX in most cases.

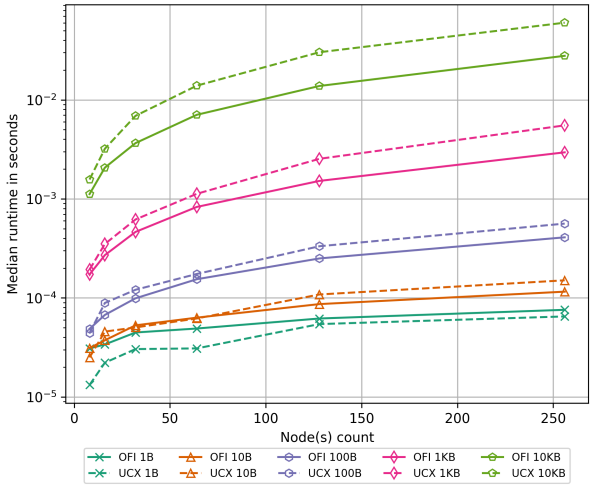

Fig. 2: MPI_Scatter performance

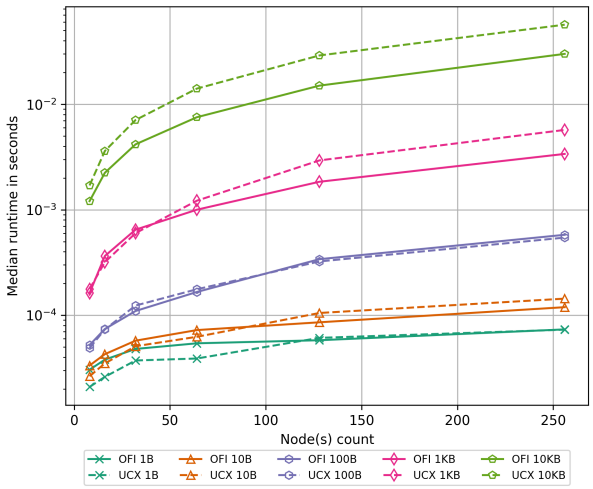

Fig. 3: MPI_Gather performance

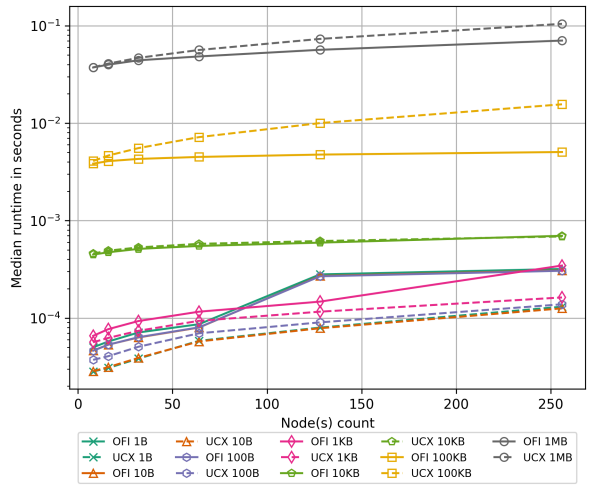

Fig. 4: MPI_Scan performance

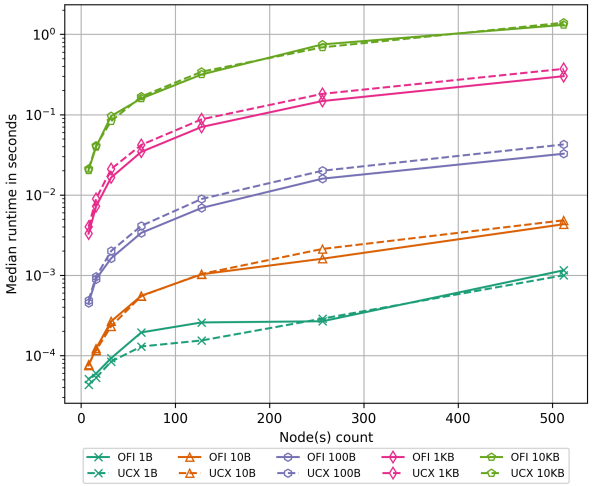

Fig. 5: MPI_Allgather performance

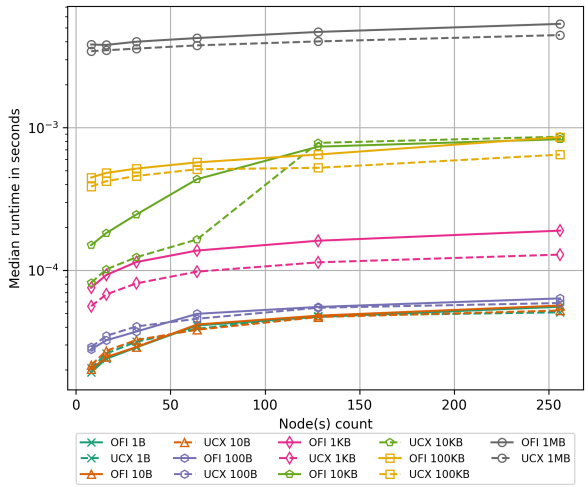

Fig. 6: MPI_Allreduce performance

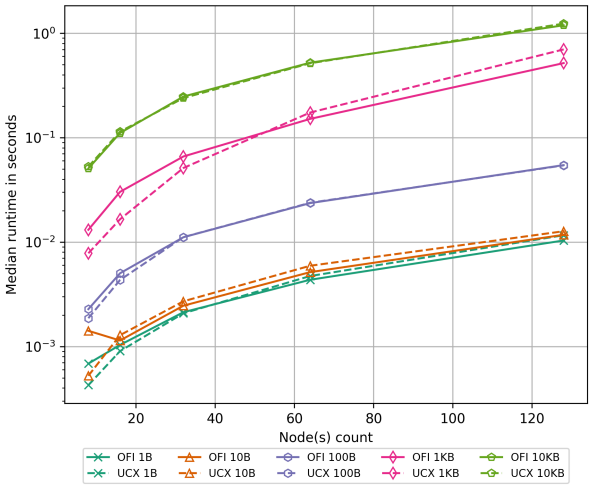

Fig. 7: MPI_Alltoall performance

B. CASTEP

Figure 8 shows the performance of OFI and UCX when CASTEP is run using up to 16 nodes, a total of 2,048 MPI processes. The blue solid line represents the performance of OFI, while the orange dashed line represents UCX. The performance is measured in SCF cycles per second.

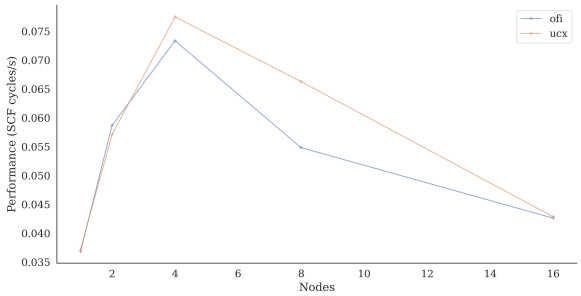

Fig. 8: CASTEP al3x3 testcase

Given the medium size of the benchmark, the best performance is achieved when using 4 nodes (512 MPI processes), and decreases when the number of nodes is 8 and 16. There is not a significant difference between the performance of OFI vs UCX independent of the number of nodes, although UCX performs slightly better when the node count is higher or equal to 4 nodes (512 MPI processes).

C. GROMACS

Figure 9 shows the performance of OFI and UCX when GROMACS is run using up to 256 nodes, a total of 32,768 MPI processes. The blue solid line represents the performance of OFI, while the orange dashed line represents UCX. The performance is measured in nanoseconds (ns) per day.

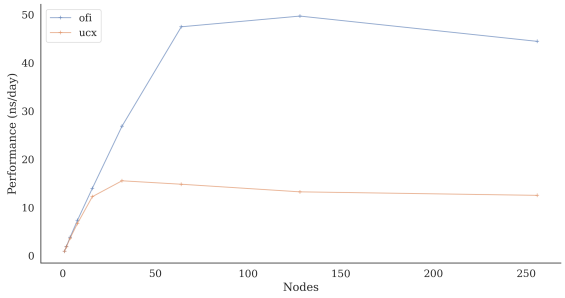

Fig. 9: GROMACS benchPEP testcase

GROMACS scales well up to 64 when OFI is used. On the other hand, the performance does not improve when using UCX with a node count larger than 32. Compared to the performance comparison reported in [3], OFI demonstrates a better performance when the node count is higher than 16.

D. NEMO

Figure 10 shows for NEMO 4.2.2, the OFI vs UCX performance in terms of simulation years per day (SYPD) of runtime. The NEMO BENCH test was used to run an idealised configuration that incorporated the ORCA12 grid. The NEMO executable was generated using the HPE CCE15 compiler.

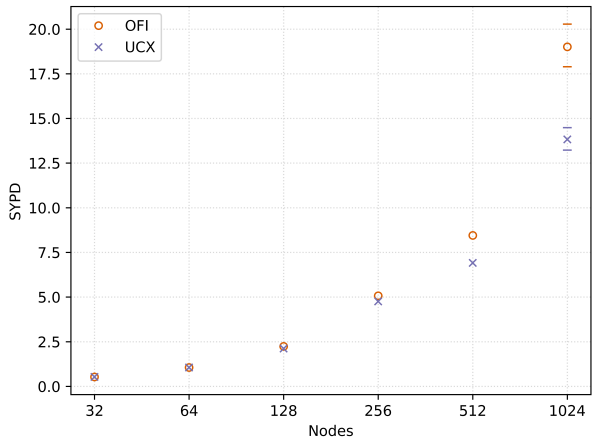

Fig. 10: NEMO 4.2.2 BENCH (ORCA12) has been run three times for a range of node counts on ARCHER2 using OFI and UCX. The plot points are the performance averages in **S**imulation **Y**ears **P**er **D**ay of runtime. The plot points for the 1024-node runs are accompanied by horizontal lines indicating the minimum and maximum performance. At other node counts, those lines are too close to the average to be properly distinguished.

For the UCX runs it was necessary to set two environment variables in order to guarantee job completion.

```
export UCX_IB_REG_METHODS=direct
export UCX_TLS=rc,dc,self,sm
```

The first setting avoids memory allocation errors [11]. The second variable, `UCX_TLS`, was set so as to exclude the Unreliable Datagram (UD) transport protocol, otherwise the NEMO UCX runs start to fail with UD endpoint (unhandled timeout) errors for node counts of 128 or above.

The results show that OFI and UCX performance is very similar until node counts of 512 and 1024, at which point OFI begins to lead. This performance gap appears to grow with node count. At 1024 nodes, the OFI performance is 19 SYPD, whereas UCX achieves 13.8 SYPD, and so, compared to UCX, OFI gives a speedup of 1.38 (at 512 nodes the speedup is 1.23). An identical trend is seen with GCC-compiled NEMO, albeit with an SYPD performance that is approximately 10% lower.

E. VASP

In Figure 11, we observe the performance of LOOP+ represented as ionic calculations per second across various communication frameworks. The error bar are standard deviations. Notably, after 16 nodes, runs utilizing UCX and OFI

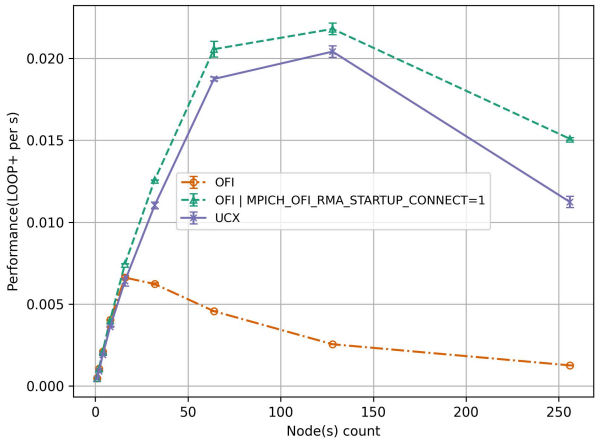

Fig. 11: VASP LOOP+ performance: OFI vs UCX

with the RMA startup flag enabled exhibit superior scaling, while standard OFI runs demonstrate comparatively poorer scaling. Overall, runs employing OFI with the RMA startup flag enabled demonstrate the best scaling, indicating that the LOOP+ step primarily involves all-to-all communication calls.

When we see Figure 12 showing the elapsed time of the VASP applcation runs with error bars indicating the standard deviation. we see that for larger node the runs using OFI with RMA flag enabled runs longer that the UCX runs, even though it exibited better LOOP+ performance scaling than the UCX runs in Figure 11.

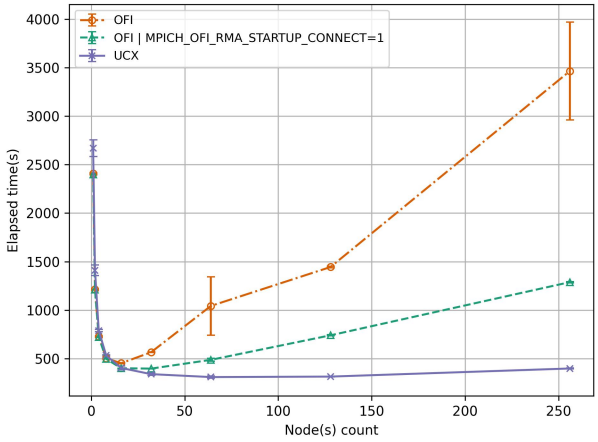

Fig. 12: VASP elapsed time: OFI vs UCX

V. MPI Profiling

In this section we present CrayPat application profiles based on sampling using default reporting settings. The profiles show the fraction of time spent in MPI communication with a breakdown by MPI call, excluding calls that contribute less than 1% to the execution time.

A. NEMO

We profiled the 32 and 256 node runs for OFI and UCX using CrayPat Lite in sampling mode. Predictably, the percentage of the runtime spent doing MPI operations grows with node count. At 32 nodes, approximately 10% of the runtime is spent in MPI — this is the case for both OFI and UCX. The MPI runtime percentages differ at 256 nodes however: for OFI it is 20%, but for UCX it is 32%.

The majority of NEMO's MPI operations involve point-to-point communications (i.e., halo swapping). There are some all reduce operations also, which take up a diminishing portion of the runtime as the node count is increased. It is clear from the profiling results that the greater MPI runtime percentage seen for UCX is driven by an increase in the time it is taking to do point-to-point MPI calls such as `MPI_RECV` and `MPI_ISEND`. At 32 nodes, those two operations account for around 7% of the runtime for both OFI and UCX. At 256 nodes, that percentage increases to 14% for OFI, but for UCX, it rises to 25%. Figure 10 indicates that this divergence widens for the higher node counts of 512 and 1024.

B. CASTEP

Table II shows the percentage of the MPI runtime with respect to the total runtime. Results with 1 and 2 nodes are not included in the table due to its similarity.

TABLE II: CASTEP MPI profile

Nodes	4		8		16	
	OFI	UCX	OFI	UCX	OFI	UCX
MPI (%)	67.1	38.1	83.8	32.9	91.6	66.3
`MPI_ALLTOALLV`	29.1	9.0	50.0	10.5	65.4	21.5
`MPI_BARRIER`	13.3	4.4	15.1	3.0	12.0	14.0
`MPI_ALLREDUCE`	12.5	9.8	9.7	7.9	7.8	16.3
`MPI_GATHER`	10.6	13.7	10.6	10.5	5.6	12.5

We can observe that the percentage of MPI runtime is larger than 80% when OFI is used on 8 and 16 nodes. MPI function `MPI_ALLTOALLV` is predominant all cases except when UCX is used with 4 nodes. When OFI is used with 16 nodes, 65.4% of the samples are related to `MPI_ALLTOALLV`.

C. GROMACS

Table III shows the MPI profile obtained using CrayPat Lite for 8 and 64 nodes.

When using 64 nodes, `MPI_Scatterv` takes most of the time with OFI, but doesn't appear for UCX (i.e., the sampling percentage is less than 0.95% of the total).

D. VASP

Tables IV, V, and VI present the percentage of total runtime used by different MPI function calls. Across eight nodes, the profiles for all three configurations are quite similar. However, when running on 32 and 64 nodes, we notice that in the default OFI configuration (Table IV), the MPI all-to-all communication step takes up a larger portion of the runtime compared to the other configurations. This, along with Figure 11, suggests that the all-to-all collective communication negatively affects scaling under the default OFI setting.

TABLE III: GROMACS MPI profile

Nodes	8		64	
	OFI	UCX	OFI	UCX
MPI (%)	36.0	36.9	88	77.0
`MPI_Recv`	3.0	2.2	18.5	25.0
`MPI_Waitall`	12.4	12.8	1.6	15.1
`MPI_Sendrecv`	5.7	9.5	2.4	11.3
`MPI_Alltoall`	2.3	4.0	1.5	10.4
`MPI_Comm_split`	6.6	6.6	9.9	9.6
`MPI_Bcast`	3.4	1.4	1.8	4.3
`MPI_Scatterv`	2.0	–	51.6	–

TABLE IV: VASP MPI OFI profile

Nodes	8	32	64
MPI (% runtime)	25.2	46.3	56.8
`MPI_ALLTOALLV`	6.6	23.8	38.2
`MPI_BCAST`	8.4	9.9	7.7
`MPI_BARRIER`	5.8	6.9	4.9
`MPI_ALLREDUCE`	3.3	5.2	5.5

TABLE V: VASP MPI OFI (RMA startup enabled) profile

Nodes	8	32	64
MPI (% runtime)	22.6	33.8	38.7
`MPI_BCAST`	9.7	16.1	19.7
`MPI_BARRIER`	4.2	10.8	12.6
`MPI_ALLTOALLV`	4.0	2.1	1.7
`MPI_ALLREDUCE`	3.6	4.2	4.1

TABLE VI: VASP MPI UCX profile

Nodes	8	32	64
MPI (% runtime)	27.5	43.7	50.7
`MPI_BCAST`	10.6	19.9	24.1
`MPI_ALLTOALL`	5.8	9.4	10.2
`MPI_BARRIER`	4.1	9.3	11.3
`MPI_ALLTOALLV`	3.9	3.4	1.5
`MPI_REDUCE`	-	-	3.5

When employing OFI with the RMA startup flag enabled, the percentage of runtime consumed by the all-to-all communication call appears to be significantly reduced (Table V), likely contributing to the improved scaling seen in Figure 11. Table VI reveals that the default UCX configuration manages collective communication effectively, showing improved scaling compared to default OFI runs as depicted in Figure 11.

VI. Conclusion

Although the best transport protocol and its settings depend on various factors such as the number of nodes and the specific application being run, we find that after the major

2023 upgrade on ARCHER2 and compared to the 2022 results, OFI performs much more reliably and can be recommended as default choice, whilst UCX may require some tuning when run on larger node counts. This is supported by the results measuring blocking collectives using ReproMPI.

Using various popular applications we make several further observations. For a medium-size CASTEP case, OFI and UCX perform similarly up to four nodes when highest performance is reached and then UCX scales somewhat better; this is also confirmed by the profiles visible as difference in the percentages spent in MPI. For GROMACS, OFI matches UCX up to and including 16 nodes and outperforms UCX on larger node counts (speedup of ca. $3.2\times$). For NEMO, we observe similar performance up to 512 nodes, where OFI starts to outperform UCX and the difference in performance appears to increase with node count from then on (speedup of $1.23\times$ on 512 nodes and $1.38\times$ on 1024). Profiles show that the fraction spent in MPI increases faster for UCX, which also requires aforementioned changes to environment variables to enable the runs on 128 nodes and above. For VASP, performance seems to be similar up to 16 nodes, where OFI requires an additional environmental variable setting which allows it to reach higher performance in LOOP+, however, overall UCX achieves better scaling (up to 32-64 nodes range as opposed to 16 nodes for OFI) resulting in a lower elapsed runtime.

In summary, choosing the right transport protocol may be critical for achieving optimal performance in parallel applications, but there is not a one-size-fits-all solution. Users should consider their specific context and requirements when making this decision, whilst OFI appears a suitable default.

Future investigations will continue to explore the impact of routing protocols on performance and study the performance of inter-node GPU-aware communication.

Acknowledgements

This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk). We gratefully acknowledge the valuable advice of many members of the ARCHER2 CSE team to carry out this project.

References

[1] P. Grun, S. Hefty, S. Sur, D. Goodell, R. D. Russell, H. Pritchard, and J. M. Squyres, "A brief introduction to the OpenFabrics Interfaces - a new network API for maximizing high performance application efficiency," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 34–39.

[2] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, "UCX: an open source framework for HPC network APIs and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 40–43.

[3] M. Bareford, D. Henty, W. Lucas, and A. Turner, "OpenFabrics and UCX: Performance on the ARCHER2 HPE Cray EX system," CUG 2022.

[4] K. S. Khorassani, C.-C. Chen, B. Ramesh, A. Shafi, H. Subramoni, and D. K. Panda, "High performance MPI over the Slingshot interconnect," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 128–145, 2023.

[5] S. Hunold and A. Carpen-Amarie, "Reproducible MPI benchmarking is still not as easy as you think," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3617–3630, 2016.

[6] S. Hunold, "Verifying Performance Guidelines for MPI Collectives at Scale," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1264–1268.

[7] S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. I. J. Probert, K. Refson, and M. C. Payne, "First principles methods using CASTEP," *Zeitschrift für Kristallographie - Crystalline Materials*, vol. 220, no. 5-6, pp. 567–570, 2005. [Online]. Available: https://doi.org/10.1524/zkri.220.5.567.65075

[8] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1, pp. 19–25, 2015.

[9] NEMO Consortium, "NEMO community ocean model," https://www.nemo-ocean.eu, 2024.

[10] G. Irrmann, S. Masson, Éric Maisonnave, D. Guibert, and E. Raffin, "Improving ocean modeling software NEMO 4.0 benchmarking and communication efficiency," *Geoscientific Model Development*, vol. 15, pp. 1567—-1582, 2022. [Online]. Available: https://doi.org/10.5194/gmd-15-1567-2022

[11] "ARCHER2 Known Issues," https://docs.archer2.ac.uk/known-issues, 2024.

Appendix: Reproducibility

Full details of the benchmarks are available on GitHub: *https://github.com/ARCHER2-HPC/performance_ofi-ucx*

The repository includes:

- Benchmark descriptions and input decks
- Job submission scripts
- Build instructions and versions of software used
- Raw results and outputs from benchmark runs
- Profiling outputs for different node counts
- Analysis scripts

Appendix: OSU results

For comparison we ran the OSU MPI Benchmarks for matching blocking collectives (except `MPI_Scan`) on 8, 16, 32, 64, and in some cases 128 nodes for various message sizes (we report measurements for sizes of 4, 128, 1024, 8192 and 32768 bytes). The runtime is reported in microseconds (note the logarithmic y-axis scale). Overall we mostly observe similar trends to the ReproMPI results.

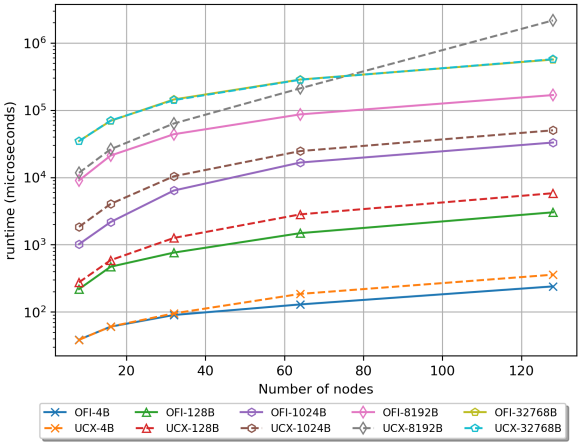

Fig. 13: OSU: MPI_Allgather performance

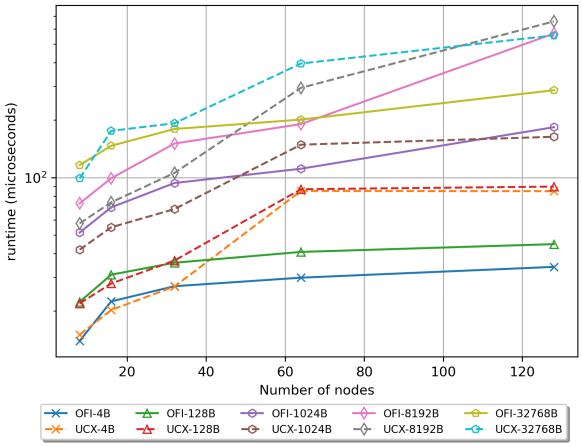

Fig. 14: OSU: MPI_Allreduce performance

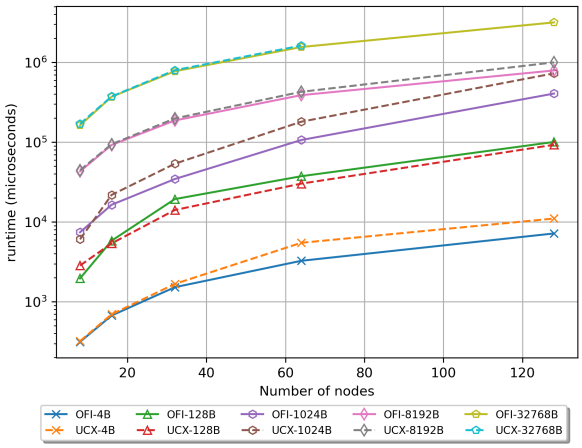

Fig. 15: OSU: MPI_Alltoall performance

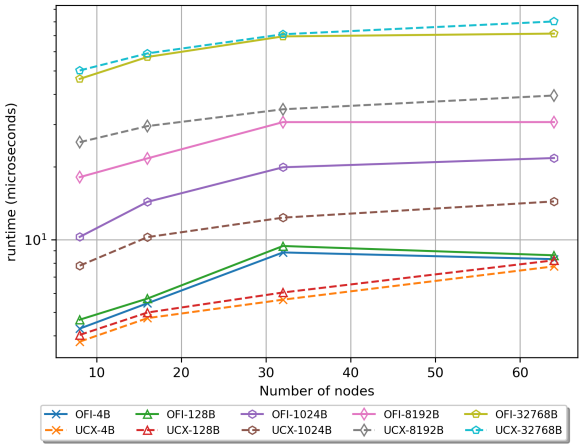

Fig. 16: OSU: MPI_Bcast performance

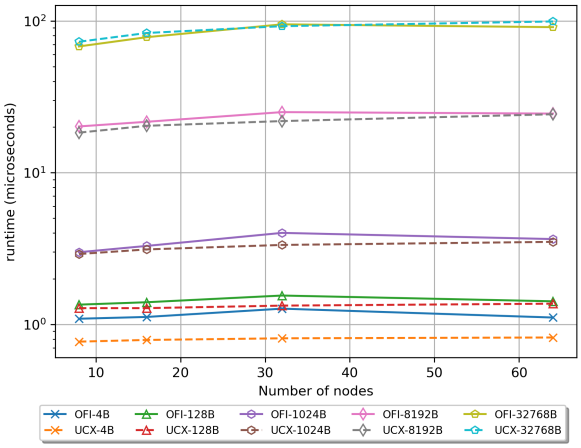

Fig. 17: OSU: MPI_Gather performance

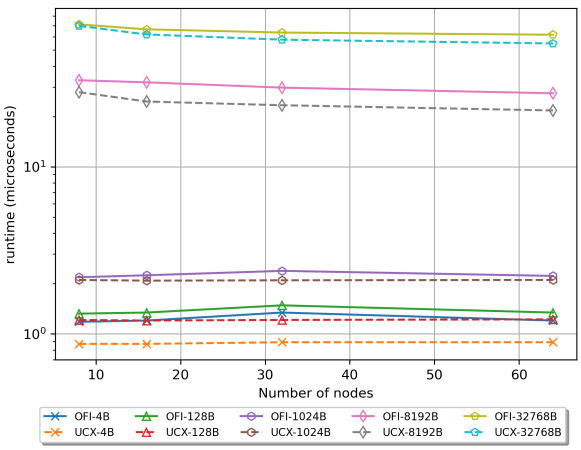

Fig. 18: OSU: MPI_Reduce performance

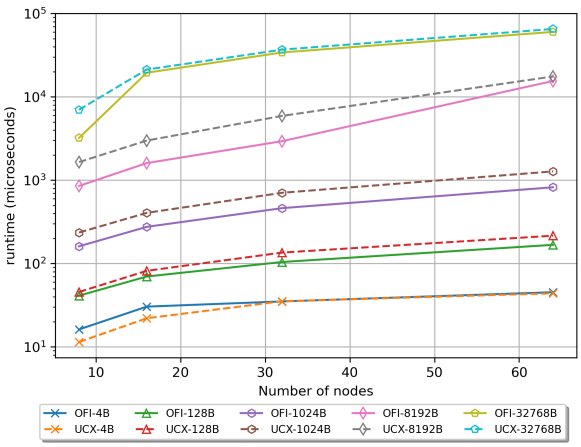

Fig. 19: OSU: MPI_Scatter performance