A Holistic Optimisation - Success Mantra for HPC Performance

Ashish Bisht, Deepika H.V, Haribabu P, S A Kumar, S D Sudarsan
Centre for Development of Advanced Computing, Bengaluru, India
{ashishbisht, deepikahv, hari, sakumar, sds}@cdac.in

***Abstract*—High Performance Computing (HPC) aids in solving numerous complex scientific problems such as weather forecasting, drug discovery, physical simulations, molecular modeling, nuclear research, cryptanalysis, oil and gas exploration. To scale these kinds of applications across the nodes of a cluster or supercomputer Message Passing Interface (MPI) is used. The performance of MPI is one of the key aspects to achieve the expected speedup in the application performance on a HPC cluster, which in turn depends on the architecture of the servers and hence it is important to understand the same for HPC practitioners. Application performance is dependent not only on an applications' capability to scale but also on the efficiency of application to execute on an architecture. In this paper, we compare the performance of applications using MPI on two architectures of based on Intel and ARM. This is important as the former is a predominant architecture and the latter is one of the upcoming architectures among the supercomputers. We have used benchmarks which cover a wide variety of applications belonging to the Berkley dwarfs. This will help to analyze the comprehensive behavior of HPC applications on both the architectures. Finally, we present our observations in terms of computation, communication, data size and functional behavior of the applications.**

***Keywords*—High Performance Computing, Supercomputers, Berkley dwarfs, MPI applications, ARM A64FX, Intel Xeon, OpenMPI, TAU**

I. Introduction

High Performance Computing (HPC) solves a wide range of complex problems. Solving these problems require a lot of computational resources and time. In order to get the solution faster, parallel programming is required. In parallel programming the problem is divided into multiple smaller problems or data sets and these are worked upon individually and simultaneously. Execution of these tasks are done by exploiting message passing techniques, shared and distributed memory. Message Passing Interface (MPI) [1] is a prevalent communication library used for passing messages between processes. MPI is used for spawning the tasks across the nodes since a single node may not be able to accommodate the full computation.

A. Application Classification

The HPC applications can be classified into 13 categories using the Berkley dwarfs [2]. Each of the dwarfs address a different class of computational problem along with a different pattern of the communication among the processes.

Classification of the applications is important to avoid benchmarking similar kinds of algorithms for the hardware. It also helps to group similar kind of applications based on the communication and computation pattern used among them. This in turn assists in evaluating hardware architectures [3]. Berkley dwarfs is a widely accepted algorithmic method based classification. It was first classified by Phil Colella, who identified seven numerical methods [4] .The dwarfs are specified at a high level of abstraction based on their behaviour across a range of applications. Programs of a particular class can be implemented differently and the underlying numerical methods may change over time, but the underlying patterns have persisted through generations and will remain important into the future.

The Berkley dwarfs can be used to find the performance of new architectures and novel programming model across all dwarfs to find the suitability for a broad range of applications. The characteristics summary of the original 7 dwarfs is described below:

- Dense linear algebra consists of dense matrix and vector operations. It has a high ratio of math-to-load operations and a high degree of data interdependency among the threads.
- Sparse linear algebra solves the same problem as dense linear algebra but has matrices with few more non-zero entries. To reduce space and computation, such algorithms store and operate on a list of values and indices rather than proper matrices, resulting in more indirect memory accesses.
- Spectral methods work on transformation of data from spatial to temporal domain or vice versa. The execution profile is typically characterized by multiple stages of processing, where dependencies within a stage form a "butterfly" pattern of computation.
- MapReduce or Monte-carlo captures the repeated independent execution of a "map" function and results are aggregated at the end via a "reduce" function. No communication is required between processes in the map phase, but the reduce phase requires global communication. These kinds of problems usually fall into the category of 'embarrassingly parallel'.
- Structured grids organize data in a regular multidimensional grid, where computation proceeds as a series of grid updates. For each grid update, all points are updated using values from a small neighbourhood around each point. The algorithm determines the data distribution and showcases specific update patterns.
- Unstructured grids possess data structures such as a linked list of pointers, that keep track of the location and 'neighbourhood' of points that are used to update the location. Data access and distribution do not showcase any specific pattern and are highly irregular.
- N-body methods calculate interactions between many discrete points and are characterized by large numbers of independent calculations within a timestamp, followed by all-to-all communication between the timestamp.

B. Architectures

In Top 500 [5], majority of HPC clusters use CISC based processors, but there are other computer architectures also which show promise in the HPC workspace. ARM based processors occupy about 1.2% in the Top500 list. Fugaku cluster at RIKEN Center for Computational Science uses an ARM based processor. One of the major reasons for looking at ARM based processors for HPC clusters is because of its less power consumption than other CISC based processors such as Intel. Our study here aims to find the applications that work well on these computer architectures. Through our work we compare the performance of these architectures when they are exposed to different classes of HPC applications.

C. Message Passing Interface

Message Passing Interface specifies how data is moved from address space of one process to another. It acts as a standard for different vendors and users. There are multiple implementations of MPI such as: Intel MPI, OpenMPI, MPICH, MVAPICH. OpenMPI is one of the implementations of MPI which is an open-source software and has a huge user base. Hence, we use OpenMPI for our evaluation. Our aim is to compare the performance of applications using OpenMPI based on the original seven Berkley dwarfs on two different architectures: ARM and Intel.

The further sections of the document are classified as follows: section 2 deals with related works, section 3 explains the experimental setup and our methodology, section 4 lists the observations and our inferences with section 5 concluding our work with future prospects.

II. Related Works

Banchelli et. al [6] have presented the evaluation of CTE-Arm, a Fugaku like system. For the evaluation they use fined tuned micro-benchmarks as well as five other scientific applications being GROMACS, Alya, NEMO, OpenIFS and WRF. These applications were not fine-tuned to the system. The focus of their paper was to evaluate the performance for the CTE-Arm system by comparing the results with an Intel based HPC system. The CTE-Arm system was performing worse than other systems, the authors suspected this was cause of CTE-Arm was not able to leverage the vectorization properly.

Odajima et. al [7] have evaluated the performance of A64FX using seven different HPC benchmarks/applications. They compared the performance of A64FX based on these benchmarks with Marvell (Cavium) ThunderX2 processor and Intel Xeon Skylake processor. The results showed high performance in memory intensive applications due to its high memory bandwidth. But in other applications it underperformed due to lack of out-of-order resources.

Jackson et. al [8] have investigated the performance for complex scientific applications on A64FX across single node as well as multiple nodes. They compared the performance of A64FX with other HPC systems. They found certain benchmarks to perform better on A64FX even without specific application optimizations. Though, there were benchmarks where A64FX was underperforming than other systems. OpenSBLI had the worst performance drop when compared to other benchmarks.

III. Experimental Setup And Methodology

We use two different HPC clusters for our evaluation. PARAM Utkarsh is an Intel based cluster situated at CDAC Bengaluru. PARAM Neel is an ARM (A64FX) based cluster situated at CDAC Pune. For ARM based architecture we selected Fujitsu's A64FX processor as it is currently ranked second in Top 500 [9]. Details of both the clusters have been summarized in Table 1.

There is a plethora of compilers, profilers, libraries available. In order to do fair comparison, we executed using the same version of open-source softwares' on both the architectures as the optimized Fujitsu compiler was not available on the ARM cluster. For profiling OpenMPI's internal functions we use Tuning and Analysis Utilities (TAU) [10]. It gathers information using instrumentation as well as event-based sampling method. Since it supports both x86_64 and aarch64 architecture it seemed like the best fit for our cause. We use GNU compiler collection (GCC) [11] version 12.2 along with OpenMPI version is 4.1.1 for compiling the benchmarks. As network topology for both the clusters is different, we use nodes that are connected to a single switch on the Intel cluster to have fair comparison of the execution runs.

TABLE I. DETAILS OF HPC CLUSTERS

Configuration	**PARAM Utkarsh**	**PARAM Neel**
Processor	Intel Xeon platinum 8268	Fujitsu A64FX
Memory type	DDR-4	HBM-2
Memory / node	192 GB	32 GB
Clock speed	2.9 GHz	1.8 GHz
No. of cores/socket	24	48
No. of socket/node	2	1
No of nodes	150	38
Interconnect	Mellanox InfiniBand	Mellanox InfiniBand
Peak bandwidth	100 Gbps	200 Gbps
Cluster topology	Fat - tree	Star
L1 cache size/core	1.5 MiB	6MiB
L2 cache size/core	24 MiB	32 MiB
L3 cache size/core	35.75 MiB	-
SIMD extension	AVX, AVX2, AVX-512	Neon, SVE

A. Comparing latencies

Since MPI communicates between nodes/processors, it becomes imperative that we take network topology and latency of the cluster network into account while doing our experiments. Hence, we start by comparing the latencies for MPI functions on both ARM and Intel clusters. Our aim here is to keep an eye on any difference caused by network latency observed in any of the clusters. We transmit a message from one process to another in a round robin fashion and finally compute the time taken for the message transmitted to be received by the original sender. The process ranked p transmits a data to a process ranked $p+1$. Then the process ranked $p+1$ forwards the data received to process ranked $p+2$ which forwards it to process ranked $p+3$. This forwarding of message continues till we reach a process ranked $N-1$ which will transmit the data back to process ranked 0. Where N is the total number of processes spawned.

For our experiment we spawn 2, 4, 8 and 16 processes with one process per node. On performing the experimentation, we find that ARM cluster shows less round-trip time than Intel as shown in Fig 1. We find that the performance varies anywhere between 2.2x to 6.99x. In terms of latency ARM performs better than Intel. To a great extent we can attribute this difference to the higher bandwidth available on ARM cluster. We observe there is a linear curve in time taken when the number of nodes is increased. On ARM cluster we find that initially time decreases when the number of nodes is increased but gradually the time taken increases linearly.

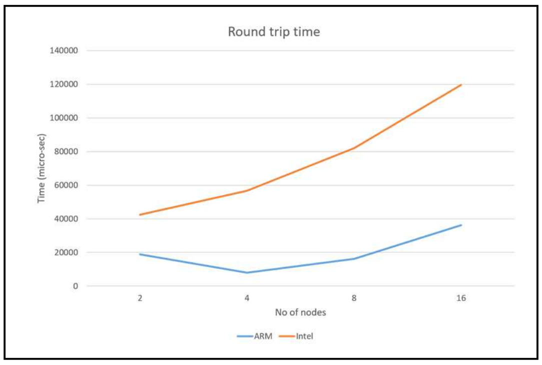

Fig. 1. Round trip time for Intel and ARM clusters

B. *Berkley dwarfs*

We benchmark the performance of MPI applications based on the original seven Berkley dwarfs. For five of the Berkley dwarfs, we use NPB benchmarks/kernels[12]. For N-body methods and unstructured grids we use GROMACS[13] and OpenFOAM [14] respectively. Table 2 lists the dwarfs and the benchmarks chosen for the experimentation.

TABLE II. BENCHMARKS CHOSEN FOR A DWARF

Sr. no	Berkley dwarf	*Benchmark chosen*
1	Dense linear algebra	LU, BT
2	Sparse linear algebra	CG
3	Spectral methods	FT
4	Map Reduce/Monte carlo	EP
5	Structured grids	MG, SP
6	Unstructured grids	OpenFOAM
7	N-body methods	GROMACS

For comparison between Intel and ARM we use Class C problem size for all the benchmarks as it contains acceptable data size / number of iterations that are commonly observed in HPC applications. Since our Intel cluster has 2 sockets per node we choose to spawn 2 processes per node. For CG, EP, FT, IS, LU and MG we use 16 nodes with 2 process per node (represented by 16N x 2P) while for BT and SP we use 16 nodes with 4 processes per node (represented by 16N x 4P), since BT and SP require number of processes to be a perfect square. We also make sure to bind the processes onto a core in order to avoid any core switching during the execution runs.

Since NPB does not cover MPI based benchmarks for n-body methods and unstructured grid we use GROMACS and OpenFOAM. For GROMACS we use one of the available benchmark suites "HECBioSim" [15]. We use Intel MKL [16] and FFTW [17] for building GROMACS. OpenFOAM uses the classical example of a motorbike but with cell count set to 8.5 million [18]. As vector instructions provide better performance on both Intel [19] and ARM [20], we executed the default benchmarks along with a version where vector extensions for both architectures were enabled. We used Advanced Vector Extensions (AVX-512) for Intel and Scalable Vector Extension (SVE) for ARM.

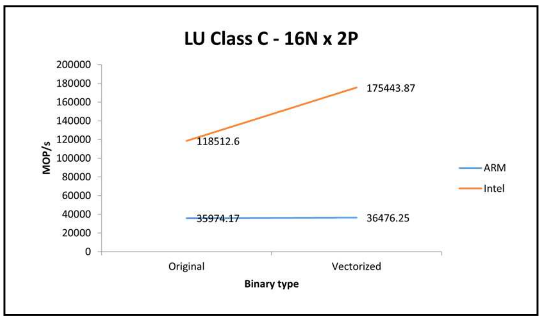

Fig. 2. LU

Under Dense algebra, we have benchmarked it using LU decomposition (LU) and Block Tri-diagonal solver (BT). Performance for Lower-Upper Gauss-Seidel solver represented in the Fig 2. We observe that Intel performs 3.29x better than ARM when using the base benchmark. This difference further increases to 4.8x when vector instructions are used. Using vector instructions in ARM did not result in significant performance improvement.

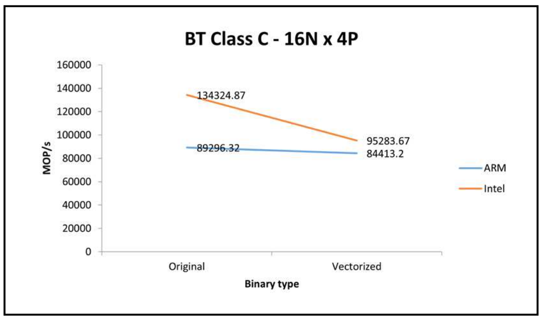

Fig. 3. BT

Fig. 3 presents the performance observed for Block Tri-diagonal solver. We observe that Intel performs 1.5x times better than ARM. Though on using vectorization we see a decrease in the performance on both the architectures. Intel takes a heavy toll on the performance when compared to ARM using vectorization.

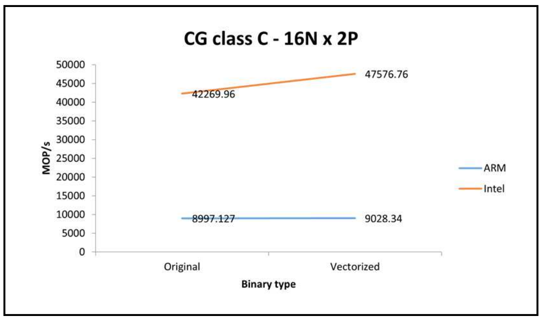

Fig. 4. CG

We have chosen Conjugate Gradient (represented by Fig. 4) for sparse linear algebra. For CG, Intel performs 4.69x better than ARM. On using vectorization, we observe performance increase of 1.12x and 1.003x for Intel and ARM respectively.

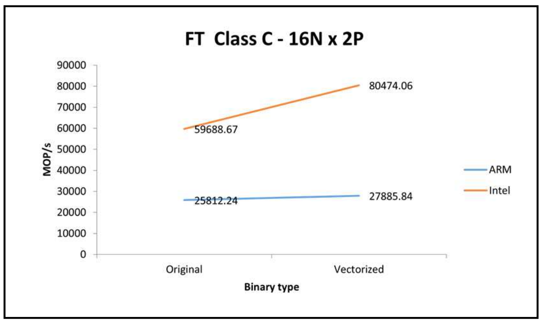

Fig. 5. FT

Fig. 5 represents data for discrete 3D fast Fourier Transform. We observe that Intel performs 2.31x times better than ARM for the original benchmark. When vectorization is used the difference increases to 2.88x. Using vectorization on ARM results in 1.08x increase in performance as contrary to 1.34x for Intel.

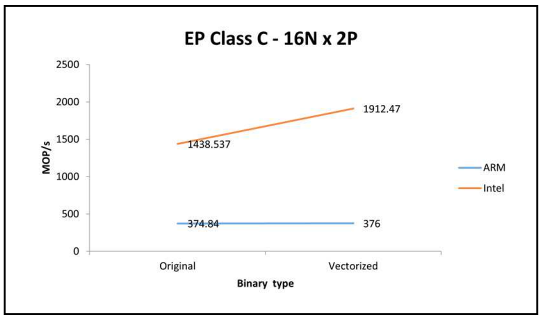

Fig. 6. EP

Fig. 6 showcases the performance for Embarrassingly Parallel kernel. When base kernel is used, we observe 3.83x performance of ARM in Intel. On using vectorization, we observe the performance gap to widen by 5.08x.

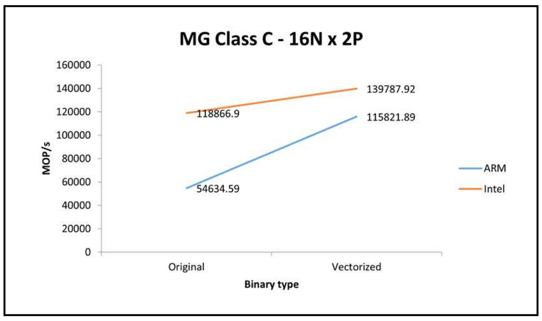

Fig. 7. MG

Performance data for Multi-Grid is represented by Fig. 7. When using the original benchmark, we observe 2.17x performance better on Intel, though on using vectorization we observe decrease in the performance gap. Using vectorization in ARM results in 2.11x increase in performance contrary to Intel's 1.17x.

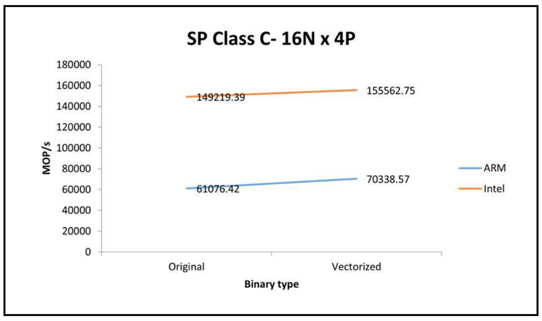

Fig. 8. SP

Scalar Penta-diagonal solver performance data is represented in Fig. 8. We observe Intel performing 2.44x better when compared to ARM though on using vectorization this gap decreases to 2.21x.

TABLE III. OPENFOAM PERFORMANCE DATA

Number of Process	ARM	Intel
16	19.8 hours	7.76 hours
32	18.94 hour	7.71 hours
64	77 sec	19 sec

Table 3 represents the performance of OpenFOAM using 8.5 million cell count as a dataset. We observe Intel performs 4.52x better than ARM. When increasing the process count, we see much more performance gap between both the clusters.

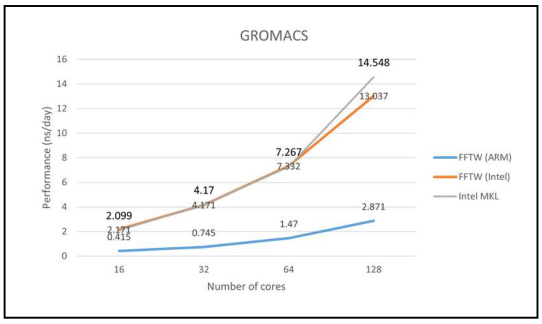

Fig. 9. GROMACS

GROMACS test results are represented by Fig. 9. We find GROMACS performs better on Intel when using the FFTW library. Initially, the performance gap was approximately 5.23x times between ARM and Intel platforms. It decreases to 4.54x times as the number of cores is increased to 128. We also observed, on increasing the number of cores, MKL library performs better when compared to FFTW on the Intel platform.

From the above experiments, Intel shows higher performance for all the benchmarks that were tried. However, in MG when using vector instructions, we see much higher gradient in ARM than in Intel indicating better performance increase. In majority of the benchmarks vector instructions made slight difference in the performance in ARM, though in Intel significant performance was observed on the usage of vectorization. On further analysis we found that NAS Parallel

Benchmarks are not optimized for a particular architecture hence it does not use vectorization and out-of-order execution capabilities of A64FX processor. Both GROMACS and OpenFOAM use FFTW library, which does not use SVE on ARM. We also observe better performance of Intel MKL than FFTW, when core count is increased on Intel. This is due to the fact that Intel MKL is optimized to efficiently use the capabilities of Intel hardware. On using all the architecture capabilities of A64FX we have confidence that dwarfs will perform much better than the current scenario.

C. *Behavior analysis of common MPI functions*

Since applications use MPI, it is imperative that we investigate the performance of MPI functions. The most common MPI functions were chosen and benchmarked. We look at three different categories of MPI functions namely: point-to-point communication, file input or output and collective MPI calls. Other than these categories we also look at *MPI_Init()* and *MPI_Finalize()* as they are responsible for initialization and termination of the MPI environment. For point to point communications we chose *MPI_Send()* and *MPI_Recv()* as a representative. For file handling we chose *MPI_File_open(), MPI_File_close()* and *MPI_File_write_all() while MPI_Scatter(), MPI_Gather(), MPI_Bcast() and MPI_Reduce()* were chosen to represent collective calls.

For point-to-point communication calls we send/receive 1MB, 10 MB and 100 MB of data to observe the trend of performance in relation to data for both the architectures. As for the file-based operations, we write 10 MB of data to a file. For *MPI_Bcast(), MPI_Scatter()* and *MPI_Gather()* we execute a single precision, general matrix multiply (SGEMM) program with matrix size set to 6400x6400. As for *MPI_Reduce()* we use Monte-Carlo PI calculation program. We compile these programs using "-O2 -gdwarf-4" flag with OpenMPI v4.1.1 using GCC v12.2.0 as the base compiler and execute the binary obtained using TAU profiler.

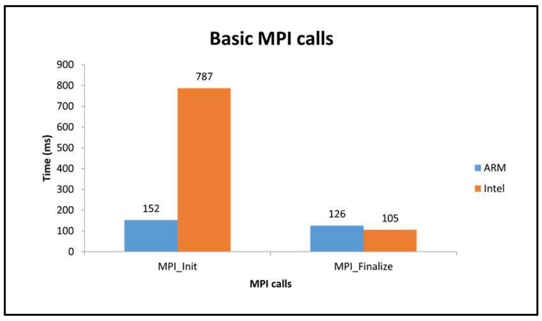

Fig. 10. Basic MPI calls

The performance data for *MPI_Init()* and *MPI_Finalize*() is represented in Fig. 10. We observe that MPI execution environment is much faster on ARM than Intel. Though the termination of MPI environment is somewhat similar in Intel.

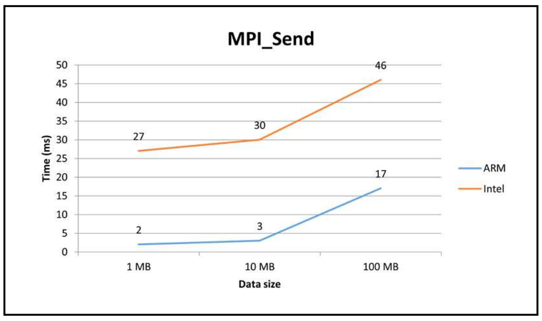

Fig. 11. MPI_Send Performance

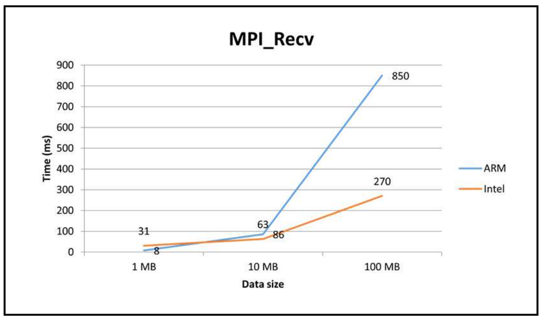

Fig. 12. MPI_Recv Performance

For *MPI_Send()* and *MPI_Recv()* performance data refer Fig. 11 & 12 respectively. We observe that *MPI_Send()* calls on ARM perform better than Intel for all three data sizes. *MPI_Recv()* initially performs better for ARM but then increases exponentially as the data size increases.

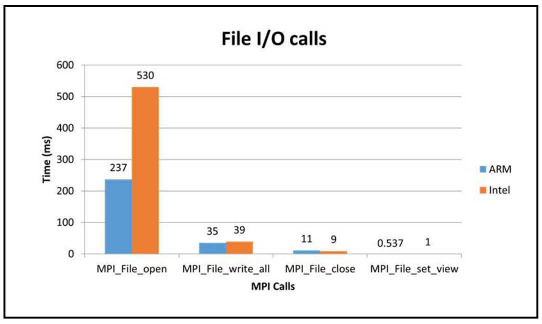

Fig. 13. File IO calls

For file operations the performance data is represented by Fig. 13. Opening a remote file on ARM was 2.23x faster than Intel. For closing a file and writing data to a remote file Intel and ARM performed similar. *MPI_File_set_view()* on ARM performed 1.86x better than Intel.

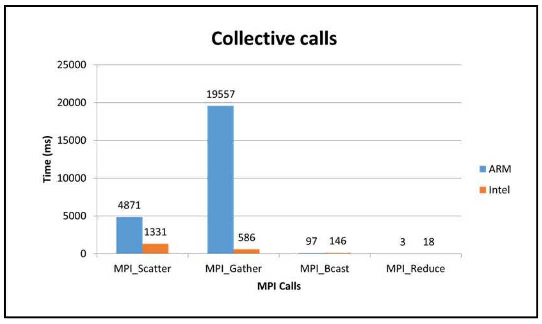

Fig. 14. Collective calls

Performance of collective MPI calls is represented in Fig. 14. We observe that Intel performs 3.65x and 33x better than ARM for *MPI_Scatter()* and *MPI_Gather()* respectively. *MPI_Bcast()* performs 1.5x better on ARM than Intel. *MPI_Reduce()* performs 6x times better on ARM than on Intel.

From the above experiments, we observe that Intel's performance is better than ARM for majority of the MPI calls, the only exceptions being the *MPI_Init(), MPI_Send(), MPI_File_open(), MPI_Bcast() and MPI_Reduce()*. We observe that currently, OpenMPI is one of the major stumbling blocks for ARM's performance. Researchers have optimized *MPI_Reduce()* performance for ARM by using SVE [21]. There is a scope for improvement in OpenMPI for ARM architecture.

IV. Conclusion And Future Work

In this paper, we have evaluated the performance of HPC clusters consisting of Xeon Platinum 8268 processors and A64FX processors using MPI. We considered Berkley dwarfs to cover a wide variety of applications that are executed on these clusters. This led to using NPB, GROMACS and OpenFOAM for evaluating the two architectures. These benchmarks helped us to understand the communication-computation performance gap, impact of increasing data size, and network latency between the nodes.

Through the experiments we found that the performance on ARM cluster lags when compared to the Intel cluster with the only exception of structured grids where speedup achieved on ARM was better with vectorization. In case of OpenMPI, the process spawning is faster on ARM, while some collective calls execute better on Intel. On further analysis, we found that the FFTW library on ARM was not finetuned for A64FX due to which the performance was lower compared to Xeon. On observing holistically, we conclude that the architectural based optimization of libraries and MPI will enhance the application performance on a target architecture.

V. Acknowledgement

We would like to thank Shamjith K V and Prachi Pandey for the paper review and all others without whom this work would not have been possible.

References

[1] "MPI Documents", [online], Available: https://www.mpi-forum.org/docs/, July 2023

[2] Asanovic, K., Bodik, R., Catanzaro, B. C., Gebis, J. J., Husbands, P., Keutzer, K., ... & Yelick, K. A, "The landscape of parallel computing research: A view from berkeley", 2006.

[3] Laion F. Manfroi, Mariza Ferro, André M. Yokoyama, A. Mury, B. Schulze, "A Walking Dwarf on the Clouds", In IEEE/ACM 6th International Conference on Utility and Cloud Computing, 2013.

[4] Kaltofen, E. L, The "seven dwarfs" of symbolic computation (pp. 95-104). Springer Vienna, 2012.

[5] "Top 500 JUNE 2023 list", [online], Available: https://www.top500.org/lists/top500/2023/06, July 2023

[6] Banchelli, F., Peiro, K., Ramirez-Gargallo, G., Vinyals, J., Vicente, D., Garcia-Gasulla, M., & Mantovani, F, "Cluster of emerging technology: evaluation of a production HPC system based on A64FX," In 2001 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 741-750). IEEE, September 2021.

[7] Odajima, T., Kodama, Y., Tsuji, M., Matsuda, M., Maruyama, Y., & Sato, M, "Preliminary performance evaluation of the Fujitsu A64FX using HPC applications," In 2020 IEEE international conference on cluster computing (cluster) (pp. 523-530). IEEE, September 2020.

[8] Jackson, A., Weiland, M., Brown, N., Turner, A., & Parsons, M, "Investigating Applications on the A64FX", In 2020 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 549-558). IEEE, September 2020

[9] "Supercomputer Fugaku", [online], Available: https://www.top500.org/system/179807/, July 2023

[10] "Tuning and Analysis Utilities", [online], Available: https://www.cs.uoregon.edu/research/tau/home.php, July 2023

[11] "GCC, the GNU Compiler Collection", [online], Available: https://gcc.gnu.org, July 2023"

[12] "NAS Parallel Benchmarks", [online], Available: https://www.nas.nasa.gov/software/npb.html, July 2023

[13] "GROMACS", [online], Available: https://www.gromacs.org, July 2023

[14] "OpenFOAM", [online], Available: https://www.openfoam.com/, July 2023

[15] "The HECBioSim Benchmarks", [online], Available: https://www.hecbiosim.ac.uk/access-hpc/benchmarks, July 2023

[16] "Intel MKL", [online], Available: https://www.intel.com/content/www/us/en/docs/onemkl/get-started-guide/2023-0/overview.html, July 2023

[17] "FFTW", [online], Available: https://www.fftw.org/, July 2023

[18] "OpenFOAM HPC benchmark suite", [online], Available: https://develop.openfoam.com/committees/hpc, July 2023

[19] Somaia Awad Hassan, A.M. Hemeida, Mountasser M.M. Mahmoud, "Performance Evaluation of Matrix-Matrix Multiplications Using Intel's Advanced Vector Extensions (AVX)", 2016

[20] Y. Kodama, T. Odajima, M. Matsuda, M. Tsuji, J. Lee and M. Sato, "Preliminary Performance Evaluation of Application Kernels Using ARM SVE with Multiple Vector Lengths," 2017 IEEE International Conference on Cluster Computing (CLUSTER), Honolulu, HI, USA, 2017, pp. 677-684, doi: 10.1109/CLUSTER.2017.93.

[21] D. Zhong et al., "Using Arm Scalable Vector Extension to Optimize OPEN MPI," 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), Melbourne, VIC, Australia, 2020, pp. 222-231, doi: 10.1109/CCGrid49817.2020.00-71.