Analyzing the performance of hierarchical collective algorithms on ARM-based multicore clusters

Gladys Utrera
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
gutrera@ac.upc.edu

Marisa Gil
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
marisa@ac.upc.edu

Xavier Martorell
Computer Architecture Department
Universitat Politècnica de Catalunya
Barcelona, Spain
xavim@ac.upc.edu

***Abstract*— MPI is the de facto communication standard library for parallel applications in distributed memory architectures. Collective operations performance is critical in HPC applications as they can become the bottleneck of their executions. The advent of larger node sizes on multicore clusters has motivated the exploration of hierarchical collective algorithms aware of the process placement in the cluster and the memory hierarchy. This work analyses and compares several hierarchical collective algorithms from the literature that do not form part of the current MPI standard. We implement the algorithms on top of OpenMPI using the shared-memory facility provided by MPI-3 at the intra-node level and evaluate them on ARM-based multicore clusters. From our results, we evidence aspects of the algorithms that impact the performance and applicability of the different algorithms. Finally, we propose a model that helps us to analyze the scalability of the algorithms.**

Keywords—MPI, shared-memory, collectives, HPC

I. Introduction

The message-passing model is the foundation of HPC performance and MPI [1], the de facto library now and in the future decade as it seems. The performance of collective operations is under continuous research as they can become the performance bottleneck of HPC applications. For example, in their work, Chunduri et al. [2] characterize HPC applications running on a production supercomputer and identify *MPI_Allreduce* and *MPI_Bcast* as the two most time-consuming operations. In addition, nodes in multicore clusters are becoming significantly large enough to be treated separately within hierarchical collective algorithms. In particular, the availability of a shared-memory and faster interconnection network within a node makes the adaptation of those algorithms attractive.

There are some proposals done in the area which have not been included yet in the MPI standard [3][4][5]. The design of the algorithms is aware of the process placement in the cluster and try to take advantage of shared memory and the available parallelism within a node. In this work, we analyze those proposals for the Broadcast operation (*MPI_Bcast*). Our main objective is to detect the parameters that make them more suitable depending on the scenario. In addition, we define a model to compare the different strategies and explain their behaviour. We build the implementations on top of OpenMPI [6] and use the shared-memory facility provided by the MPI-3 [1] at the intra-node level.

This work has been supported by the Spanish Ministry of Education (PID2019-107255GB-C22) and the Generalitat de Catalunya (2017-SGR-1414).

ARMs [7] architectures are designed to optimize the execution with low energy costs, providing many cores by a node to achieve this goal without performance loss. For this reason, we considered this architecture as an exciting target to perform our evaluations.

In summary, the main contributions of this work are:

- Implementation, evaluation and comparison of three different hierarchical collective algorithms: Multi-leader, Multi-lane and Hier. We evaluate their performance varying several parameters like number of leaders, number of lanes, message sizes. The comparisons are made against the native OpenMPI broadcast algorithm (Binomial in our case).
- Implementation of a hierarchical algorithm that makes use of the MPIWin facility from MPI-3 standard.
- Construction of a model to compare the algorithms and explain their behavior.

Our first results seem to confirm that hierarchical algorithms that are aware of the underlying platform are needed. We confirm that the currently available MPI libraries demonstrate severe problems in that sense [5]. However, the hierarchical algorithm needs to adapt to parameters other than the size of nodes, like memory hierarchy and interconnection networks.

The rest of the document is organized as follows: first, we present other works related to the design and evaluation of collective algorithms. Then, we explain our methodology and describe the algorithms implemented and evaluated as well as our model proposal. After that, we describe the evaluations carried out and our findings. Finally, we present our conclusions and future work.

II. Related work

There is a vast amount of research on the design of MPI collective algorithms. We limit to enumerate some relevant works related to the algorithms we compare and other characteristics like MPI-3 shared memory. The authors in [3] suggest reducing the bandwidth for the collectives, using leader processes within a node. These leader processes are in charge of collecting the information from all the group processes through shared memory (implemented via *mmap*). The approach works well for messages up to 256 Kbytes. The mechanism is improved later for larger message sizes by introducing pipelining [4].

The work presented by [5] indicates considerable room for improvement with the MPI collectives in current libraries, using multi-lane communication. Zhou et al. [8] demonstrate that the use of shared-memory windows introduced by the

 DOI 10.1109/PDP55904.2022.00043

MPI-3 standard may improve the performance of the MPI collective operations. Mamidala et al. [9] propose sharing buffers for intra-node and inter-node communication to avoid extra copies of data. They claim that they also achieve some kind of overlap with network communication. The basic idea seems promising. They show improvements of up to 43% for messages up to 16 Kbytes.

III. Methodology

In this section, we describe first the algorithms evaluated and our proposed model. Then we present the experimental platform.

A. Hierarchical algorithms evaluated

We explore recently proposed algorithms that consider at least one of the following characteristics: large node sizes, shared-memory facility, use of available parallelism. The selected algorithms are:

1) Multi-leader. This algorithm proposed by Bayatpour et al. [3][4] organize collectives in groups of processes with a leader process in charge of the "inter-group communication". In their work, they suggest adjusting the size of the group to a socket. Communication within the processes of the group is performed through shared memory. Figure 1 shows a graphical representation of the algorithm with two groups with four processes each. The entire message is copied onto the shared-memory region and also sent to the other groups. Non-root processes copy back the result from the shared-memory region.

2) Multi-lane. This algorithm proposed by J. Traff et al. [5] suggests having multiple lanes per group. The message is partitioned and scattered among the processes of the group. Then each partition is broadcasted between homologous processes in other groups. Finally, the original message is built within each group by an allgather operation. Figure 2 shows the mechanism.

3) Hier. This algorithm is another alternative to Multi-lane proposed in the work by J. Traff et al. [5] as well. It is very similar to Multi-leader, but without using explicitly shared memory. The collective is decomposed into two broadcast operations: one "intra-group" and another "inter-group". Figure 3 shows how the algorithm works.

We explored several possibilities of group sizes within a node for each case: one group (96 processes) to 16 groups (6 processes each group). We studied their performance and selected the best configuration for the comparison.

B. Model

The motivation to construct a model is to compare the algorithms and to explain the differences in the performance of them. In addition, the model serves us to study scalability to large clusters.

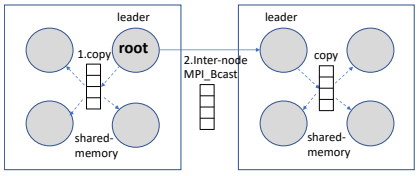

Figure 1. Multi-leader representation with two groups and four processes per group

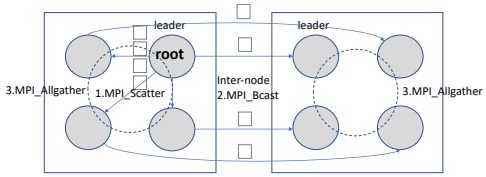

Figure 2. Multi-lane representation with two groups and four processes per group (4 lanes)

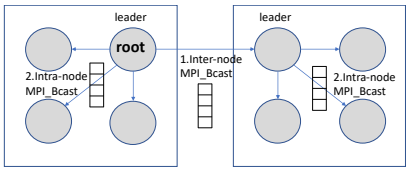

Figure 3. Hier representation with two groups and four processes per group.

C. Experimental platform

The experiments are carried out on a mini-cluster with 4 ARM-based nodes, each with two Cavium Thunder-X 88XX CPUs, supporting the Arm AArch64 architecture and fully compliant with the ARMv8 (64 bit) instruction set. Each CPU has 48 cores at a frequency of 1.9 GHz, being the core's cache coherent by using the Cavium Coherent Processor Interconnect. Each core has a private level 1 instruction cache of 78 Kbytes and a private data cache of 32Kbytes. Each chip has a 16 Mbytes shared L2 cache for instructions and data. The node has a single NUMA node with 32 GB of RAM. Nodes are interconnected through the integrated 10GBit Ethernet devices provided by Cavium on the node boards.

The node runs Ubuntu 20.04.1 LTS, and for the experiments, we have used gcc version 9.3.0 to compile the applications, and OpenMPI version 3.2 To test the *MPI_Bcast* collective algorithms, we use the Intel MPI. Benchmarks version 2019 with the option "cache off" to avoid distortion of results from repeated executions. We repeat at least four times each experiment.

IV. Experimental results

In this section, we present the results of our evaluations. Let's us first take a look at the performance results when selecting the best configuration for each algorithm. After that, we proceed with the comparison.

A. Configuration at intra-node level of the collective algorithms evaluated

We describe which configuration we choose for each of the collective algorithms considered and the reasons for that. In addition, we present an expression for each algorithm to model the execution time of the critical path.

1) Hier

The critical path of this algorithm is composed of two broadcast operations. Both operations are of the entire message, so the lesser the number of broadcasts, the better the performance.

In this configuration, we can describe the time T_h taken by this operation using Eq. 1, where m is the message size, s the number of allocated processes per node and n the total number of nodes. The sum of the functions *intraNodeBcast(x, y)* and *interNodeBcast(x, y)* is the cost of *MPI_Bcast,* a

message of *x* size to *y* processes, at intra- and inter-node level, respectively.

$$T_h = 2\ x\ intraNodeBcast\ (m,s) + interNodeBcast\ (m,n) \quad \textbf{(Eq. 1)}$$

2) Multi-leader

This algorithm in our experiments showed its best performance when configuring one leader per node and one shared-memory group per node. Consequently, communication at the intra-node level is performed only through shared memory.

The critical path executing this algorithm is composed of one memory copy from the root process to the shared-memory region, a broadcast operation to all the other leaders, a copy of the message to the shared-memory region and then a copy back of the result by each process of the group. In this way, there are always three memory copies of the whole message, no matter the number of processes (no matter then, the number of leaders within a node). The impact on scalability is determined only by the number of target processes in the broadcasting (the number of nodes). As expected, the lesser the number, the better the performance.

The broadcast operation does not suffer a lack of scalability as the allgather and scatter operations when varying the number of processes at the intra-node level. For example, the degrading in performance for the *MPI_Bcast* between 6 and 16 processes for message sizes up to 4Mbytes is 3x.

Eq. 2 shows the expression to model the critical path of this operation, where *m* is the message size, and *n* is the total number of nodes. The *memcpy (x)* function is the cost of copying a message of size *x* bytes and *interNodeBcast(x, y)* as before, the cost of time of MPI_Bcast of a message of size *x* to *y* processes in different nodes.

$$T_r = 3\ x\ memcpy\ (m) + interNodeBcast\ (m,n) \quad \textbf{(Eq. 2)}$$

3) Multi-lane

In this case, the best configuration we obtained was when choosing 16 groups of 6 processes each.

The critical path of the multi-lane algorithm is composed of one scatter, one allgather and a broadcast operation. The scatter and allgather operations can become the performance bottleneck at the intra-node level when increasing the number of participating processes. We have observed for *MPI_Allgather* between 6 and 16 processes at the intra-node level a variation up to 50x in the performance.

For this reason, these two operations perform notably better on small groups, even though the message size would be more significant.

In this configuration, Eq. 3 is the expression that models the execution time of the collective using the multi-lane algorithm. The sum of the functions *scatter(x, y)* and *allgather(x, y)* represents the cost of *MPI_Scatter* and *MPI_Allgather* respectively of a message of size *x* and *y* processes. The variable *s* is the total number of processes per node, and *g* is the number of groups per node. The sum of the function *intraNodeBcast (x, y)* and *interNodeBcast (x, y)* represents the cost of *MPI_Bcast* of a message of size *x* bytes to *y* processes intra and inter-node level, respectively.

$$T_l = scatter\ \left(m,\frac{s}{g}\right) + allgather\ \left(m,\frac{s}{g}\right) + intraNodeBcast\ \left(\frac{m}{g},\frac{s}{g}\right) + interNodeBcast\ \left(\frac{m}{g},n\right) \quad \textbf{(Eq. 3)}$$

B. Performance comparison results

This section presents the comparison results of the three collective algorithms using their best configuration described in Section A and the native OpenMPI algorithm (Binomial) from the current MPI library in use.

We can see in Figure 4, Figure 5, Figure 6 the average execution times for the *MPI_Bcast* operation when using the IMB Benchmarks, for the message ranges: 4bytes-256bytes, 512bytes-64Kbytes and 128Kbytes-4Mbytes respectively. The hierarchical Hier algorithm works better for message sizes up to 2Kbytes. As can be seen in the figures, the Multi-leader algorithms obtain the best performance for message sizes between 4Kbytes and 64Kbytes; the Multi-lane gets the best performance for message sizes greater or equal 128Kbytes.

Let's analyze closer the results by comparing their critical path. In the case of the Hier and Multi-leader algorithms, the difference is related to the message distribution among processes at the intra-node level. There is a broadcast operation using the MPI native library in the first case, so performance is strictly dependent on the MPI current library in use. In the second case, the distribution is performed through shared memory. Moreover, the copy back of the messages are done in parallel, i.e. each process within a group makes its copy from shared-memory to the output buffer. For larger message sizes, a parallel copy seems to be a better option than broadcasting.

The comparison between the performance of the multi-leader and multi-lane can be divided into two parts. During the first part (up to message size 64Kbytes), we observe that multi-leader overperforms multi-lane. There is a cost that depends only on the message size and is related to the operations at the intra-node level: memory copies (multi-leader) and the scatter + allgather operations (multi-lane). The other operations (broadcast) depend on the message size and the number of inter-node-level processes. In addition, taking a look at the performance of individual executions of the participating collectives, we observe that for message sizes lesser or equal to 64 Kbytes, the cost of the scatter and the allgather cannot compensate for the difference in broadcasting a smaller message size than in the multi-leader case. For message sizes over 64 Kbytes, the cost of broadcasting larger message sizes makes multi-lane more attractive. Recall that in the multi-lane algorithm, the message is partitioned among the processes members of the group, having parallel "lanes" of smaller messages to broadcast.

We perform our evaluations in a mini-cluster with only four nodes, which prevent us from making conclusions about scalability. However, from the results and the model already presented, we can extrapolate some reasoning. When incrementing the number of nodes, for a message size given, at the intra-node level, the cost of time of the operations is constant. For this reason, the impact depends on the inter-node operations exclusively. When comparing multi-leader and multi-lane, we can guess that the tendency of having better performance with multi-lane is consolidated, as this option always has smaller message sizes. There would be a trade-off in terms of bandwidth, but as stated in [5], many modern, high-performance systems can offer the facility for multi-lane communication.

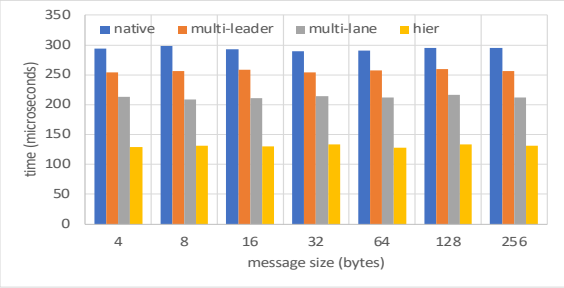

Figure 4. MPI_Bcast average execution time comparison (more minor is better) for messages sizes between 4 bytes and 256 bytes using 384 processes distributed in 4 nodes.

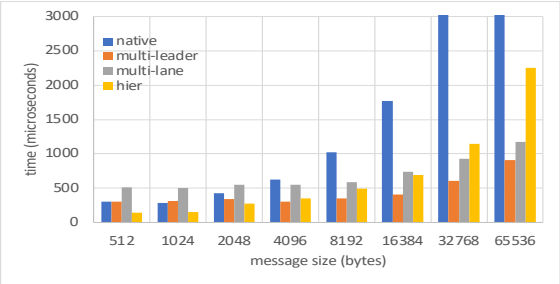

Figure 5. MPI_Bcast average execution time comparison (more minor is better) for messages sizes between 512 bytes and 64 Kbytes using 384 processes distributed in 4 nodes.

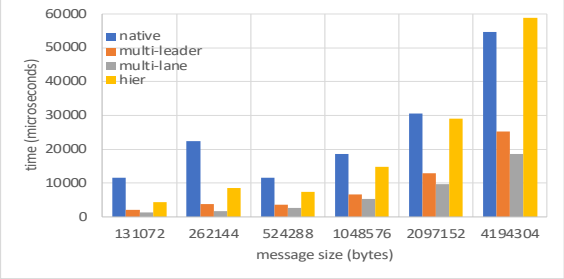

Figure 6. MPI_Bcast average execution time comparison (more minor is better) for messages sizes between 128 Kbytes and 4 Mbytes using 384 processes distributed in 4 nodes.

V. Conclusions and Future Work

There is a lot of research focused on the performance of MPI collective operations. They have been demonstrated to be the bottleneck of HPC applications in high-performance computing systems. In addition, the increasing number of cores per node in multicore clusters deserve to have a unique role in the hierarchy of the collective algorithms. The availability of shared memory and faster interconnection networks within a node claim to adapt those existing algorithms.

Recently there have been many efforts in that direction. In this work, we focused on the broadcasting operation. To that aim, we have selected three hierarchical approaches, which we implemented, evaluated and compared with the OpenMPI default algorithm (Binomial). They are Multi-leader, Multi-lane and Hier collective algorithms. We evaluated several configurations for them, varying, for example, the number of intra-node groups. The first one at the intra-node level communicates through shared memory which we implemented using the MPIWin facility from MPI-3. The other two just decompose the collective into other collectives differentiating between intra- and inter-node communication.

Our evaluations on an ARM-based mini-cluster demonstrated that Hier is a good option for small message sizes with 2x over the native. At the same time, Multi-leader can be a good choice for medium-sized messages with an average gain of 60% over the second-best. The Multi-lane approach works very well with large message sizes (30% better than the second-best), as the communication performs in parallel by partitioning the message.

We have built an analytical model to express the different elements that impact the performance of the different approaches and explain the algorithms' behavior. Using this model, we also make some reasonings about the scalability of the techniques. In particular, we confirm the tendency of the Multi-lane approach to overperform the other algorithms evaluated when increasing the number of nodes, especially for large message sizes.

We plan to extend our evaluations to a more significant number of nodes and other MPI libraries. In addition, we would like to increase the accuracy of our model by including architectural parameters like bandwidth and memory latencies.

Acknowledgement

The authors acknowledge the support of the BSC (Barcelona Supercomputing Centre), especially to Filippo Mantovani and his team.

References

[1] MPI. Forum. Available at: https://www.mpi-forum.org (2021).

[2] Chunduri,S.,Parker,S.,Balaji,P.,Harms,K.,Kumaran, K.: Characterization of mpi usage on a production supercomputer. In: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 386–400 (2018). DOI 10.1109/SC.2018.00033

[3] Bayatpour, M., Chakraborty, S., Subramoni, H., Lu, X., Panda, DKD: Scalable reduction collectives with data partitioning-based multi-leader design. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17. Association for Computing Machinery, New York, NY, USA (2017). DOI 10.1145/3126908.3126954. URL https://doi.org/10.1145/3126908.3126954

[4] Bayatpour, M., Maqbool Hashmi, J., Chakraborty, S., Subramoni, H., Kousha, P., Panda, DK: Salar: Scalable and adaptive designs for large message reduction collectives. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 12–23 (2018). DOI 10.1109/CLUSTER.2018.00014

[5] J. L. Träff and S. Hunold, "Decomposing MPI. Collectives for Exploiting Multi-lane Communication," *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 270-280, doi: 10.1109/CLUSTER49012.2020.00037.

[6] OpenMPI. Available at: https://www.open-mpi.org// (2021).

[7] ARM architecture. Available at: https://www.arm.com (2021).

[8] Zhou, H., Gracia, J., Schneider, R.: Mpi collectives for multicore clusters: Optimized performance of the hybrid mpi+mpi parallel codes. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3339186.3339199. URL https://doi.org/10.1145/3339186.3339199

[9] Mamidala, A.R., Vishnu, A., Panda, DK: Efficient shared memory and rdma based design for mpi allgather over infiniband. In: B. Mohr, J.L. Tr¨aff, J. Worrin- gen, J. Dongarra (eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface, pp. 66–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)