Performance Analysis of OpenMPI Collective Operations on the ORFEO Cluster

High Performance Computing 2023
Exercise 1

Giovanni Coronica
Matricola: IN2000215
31/01/24

1. Introduction

Optimizing collective operations is essential for maximizing the performance of high-performance computing (HPC) applications. In MPI-based applications, operations such as broadcast (`MPI_Bcast`) and gather (`MPI_Gather`) are everywhere, their efficiency directly impacting overall execution time. This report presents a detailed evaluation of these operations within the OpenMPI library, conducted on the ORFEO cluster.

Utilizing the OSU MPI benchmarks, a standard for MPI performance assessment, the study compares the default OpenMPI implementation against selected algorithms across a range of message sizes. The benchmarks were executed on two fully utilized THIN nodes of the ORFEO cluster, simulating a demanding parallel environment that is representative of real-world scientific computations.

The analysis aims to shed light on the performance characteristics of various OpenMPI algorithms, providing a comparison in terms of latency. The chosen algorithms for the broadcast operation were `scatter_allgather_ring` and `pipeline`, while for the gather operation, `binomial` and `basic_linear` algorithms were selected. These were benchmarked to establish a performance baseline and identify the most efficient algorithms for the given HPC architecture.

The resulting data serves not only to determine the optimal choices for collective operations but also as a foundation for developing performance models that account for the specificities of the network architecture in the ORFEO cluster. By examining the efficiency of each algorithm under different conditions, the report contributes insights into their suitability and guides the selection process for similar HPC applications.

In the following sections, the report will outline the methodology used in the benchmarking process, present and analyze the results, and discuss the implications of the findings, culminating in a set of recommendations for collective operation optimization in HPC systems.

2. Methodology

2.1 Benchmarking Tools

The OSU Micro-Benchmarks suite, version 7.3, was utilized for benchmarking the MPI collective operations. No patches or modifications were applied to the benchmarks, ensuring that the results were based on the standard implementation provided by the benchmark suite.

2.2 Cluster and MPI Configuration

The benchmarks were conducted on the ORFEO HPC cluster, specifically using two THIN nodes. Each node was configured with 24 tasks per node to fully utilize the available cores. The nodes were allocated using the Slurm workload manager with the following command:

```
salloc -N2 --tasks-per-node=24 -p THIN --time=0:30:00
```

The OpenMPI module version 4.1.5 compiled with GNU compilers version 12.2.1 was loaded to provide the MPI environment for the benchmarks.

2.3 Benchmark Execution

For both the broadcast and gather operations, the OpenMPI dynamic rules for algorithm selection were enabled. The specific algorithms were selected using the `--mca` parameter. The command-line format for the benchmark execution was as follows:

```
mpirun --mca coll_tuned_use_dynamic_rules true --mca
coll_tuned_<algorithm name>_algorithm <algorithm selection>
        <benchmark program> -x 100 -i 1000 -f
```

The `-x` and `-i` parameters were used to specify 100 warm-up iterations and 1000 total iterations, respectively, for each message size tested. The benchmark suite automatically tested a predefined set of message sizes ranging from 1 byte to 1.048.576 bytes.

2.4 Data Collection and Preliminary Analysis

Performance data was collected directly from the standard output, which reported the mean, minimum, and maximum latencies over 1000 iterations for each message size and algorithm. No additional statistical analysis was performed on the mean values, as it was determined that calculating the mean and standard deviation of these means would not accurately represent the true distribution of the latency values.

2.5 Comparative Analysis

The comparative analysis of performance data involved a two-pronged approach: visual comparison and performance ratios calculation. For each collective operation tested—broadcast and gather—benchmarks were performed using the baseline implementation and two alternative algorithms.

Visual Comparison:

Line graphs were constructed to provide a visual representation of the latency across the range of message sizes tested. Each algorithm, including the baseline, was plotted with message size on the x-axis and mean latency on the y-axis. These plots allowed for an immediate visual assessment of the performance trends and highlighted any message sizes where significant latency differences occurred between algorithms.

Performance Ratios:

To quantify the performance differences, performance ratios were calculated for each algorithm relative to the baseline. This was done for each message size by dividing the mean latency of the algorithm by the mean latency of the baseline. Ratios below 1 indicated that the algorithm outperformed the baseline, whereas ratios above 1 indicated that the algorithm was outperformed by the baseline.

By synthesizing the visual insights from the graphs with the quantified performance ratios, a comprehensive picture of each algorithm's efficiency relative to the baseline was established. The analysis provided a clear basis for selecting the optimal algorithm for different sizes of messages in the context of the ORFEO cluster's network architecture.

3. Results

The benchmarking results are presented in this section, showcasing the performance of different OpenMPI algorithms for the broadcast and gather operations. Performance ratios were calculated to compare each algorithm against the baseline implementation, and graphical representations were created to visualize the latency trends across various message sizes.

3.1 Broadcast Operation

For the broadcast operation, the performance of the baseline implementation was compared against the `scatter_allgather_ring` and `pipeline` algorithms.

3.1.1 Performance Tables

The following tables summarize the mean latency measurements for each message size, along with the minimum and maximum latencies observed during the benchmarking process:

Size (bytes)	AvgLatency(us)	MinLatency(us)	MaxLatency(us)
1	3,19	0,84	5,26
2	3,05	0,82	5,01
4	3,07	0,85	5,03
8	3,02	0,82	4,96
16	2,98	0,82	4,90
32	3,27	0,93	5,37
64	3,54	0,91	5,67
128	4,25	1,71	6,35
256	4,63	1,92	6,91
512	4,87	1,99	7,29

1.024	5,73	2,37	8,53
2.048	7,63	3,61	11,41
4.096	10,43	5,17	15,62
8.192	15,12	7,80	22,44
16.384	21,37	12,10	28,74
32.768	33,73	20,86	46,74
65.536	62,01	39,25	87,26
131.072	125,57	79,74	146,07
262.144	255,78	160,30	319,45
524.288	487,27	306,02	619,10
1.048.576	872,29	536,79	1.070,54

Table 1: Baseline Broadcast Operation Latencies

Size (bytes)	AvgLatency(us)	MinLatency(us)	MaxLatency(us)
1	5,23	0,45	8,53
2	5,07	0,40	8,14
4	5,04	0,42	8,07
8	4,97	0,42	7,94
16	4,97	0,37	7,99
32	5,48	0,47	8,60
64	16,02	15,01	17,83
128	16,25	15,09	17,95
256	16,69	15,42	18,45
512	17,24	15,76	19,15
1.024	17,79	16,32	19,56
2.048	23,29	21,41	25,59
4.096	23,71	22,12	25,48
8.192	31,62	29,35	34,16
16.384	54,83	49,68	60,54
32.768	51,75	48,47	54,56
65.536	68,22	63,33	72,24
131.072	95,29	89,67	99,80
262.144	163,00	150,90	173,26
524.288	304,88	268,81	337,07
1.048.576	569,83	506,84	617,40

Table 2: Scatter AllGather Ring Broadcast Operation Latencies

Size (bytes)	AvgLatency(us)	MinLatency(us)	MaxLatency(us)
1	10,49	0,21	17,55
2	10,04	0,21	16,99
4	9,93	0,21	16,90
8	9,87	0,20	16,76
16	9,87	0,21	16,74
32	11,62	0,21	20,37
64	11,55	0,22	20,13
128	15,82	0,25	28,86
256	15,84	0,25	29,06
512	16,55	0,26	30,53
1.024	29,89	0,30	57,65
2.048	25,76	0,38	48,22
4.096	37,75	0,69	71,74
8.192	56,34	0,97	108,18
16.384	125,75	4,01	240,15
32.768	177,88	5,72	340,10
65.536	274,20	9,43	524,63
131.072	472,31	16,37	901,64
262.144	864,46	30,55	1.650,74
524.288	1.652,92	56,57	3.160,64
1.048.576	3.345,41	128,54	6.442,69

Table 3: Pipeline Broadcast Operation Latencies

3.1.2 Performance Ratios

Performance ratios were calculated for the scatter_allgather_ring and pipeline algorithms relative to the baseline. These ratios are presented in:

Algorithm \ Size (bytes)	1	2	4	8	16	32	64	128	256	512	1.024	2.048
Scatter AllGather Ring	1,64	1,66	1,64	1,65	1,67	1,68	4,53	3,82	3,60	3,54	3,10	3,05
Pipeline	3,29	3,29	3,23	3,27	3,31	3,55	3,26	3,72	3,42	3,40	5,22	3,38

Algorithm \ Size (bytes)	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576
Scatter AllGather Ring	2,27	2,09	2,57	1,53	1,10	0,76	0,64	0,63	0,65
Pipeline	3,62	3,73	5,88	5,27	4,42	3,76	3,38	3,39	3,84

Table(s) 4: Broadcast Performance Ratios

3.1.3 Graphical Representation

A line graph was created to illustrate the latency trends for the broadcast operation:

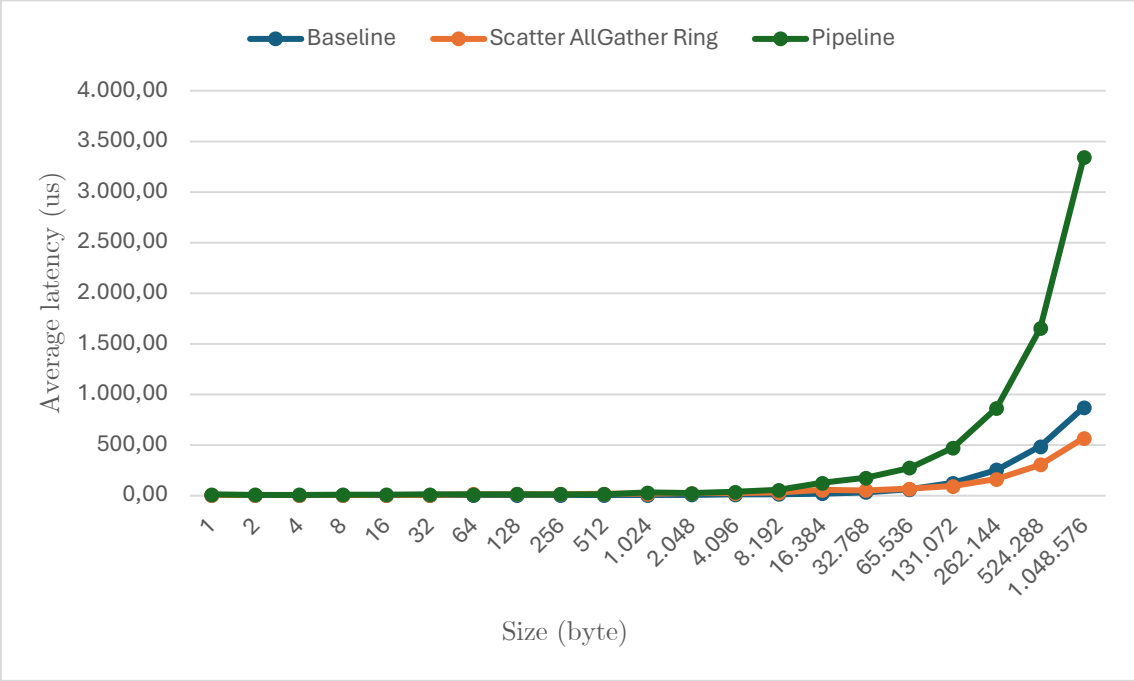

Figure 1: Broadcast Operation Latency Comparison

3.2 Gather Operation

Similarly, for the gather operation, the performance of the baseline implementation was compared against the binomial and `basic_linear` algorithms.

3.2.1 Performance Tables

The mean latency measurements, along with the observed minimum and maximum latencies for the gather operation, are displayed in the tables below:

Size (bytes)	AvgLatency(us)	MinLatency(us)	MaxLatency(us)
1	0,77	0,12	3,72
2	0,74	0,11	3,66
4	0,76	0,11	3,69
8	0,78	0,10	4,61
16	0,79	0,09	4,18
32	0,96	0,10	5,07
64	1,02	0,10	5,56
128	1,13	0,13	6,04
256	1,33	0,15	7,96
512	1,51	0,15	10,08

1.024	1,99	0,16	15,49
2.048	2,93	0,19	23,68
4.096	4,75	0,35	41,72
8.192	7,89	0,64	71,30
16.384	23,19	1,81	186,05
32.768	27,63	3,26	216,99
65.536	57,90	6,96	416,77
131.072	119,13	13,40	817,12
262.144	270,08	47,41	1.791,23
524.288	668,34	111,96	4.555,65
1.048.576	3.528,83	498,98	17.055,41

Table 5: Baseline Gather Operation Latencies

Size (bytes)	AvgLatency(us)	MinLatency(us)	MaxLatency(us)
1	0,75	0,09	3,60
2	0,74	0,09	3,60
4	0,74	0,09	3,55
8	0,75	0,09	4,51
16	0,78	0,09	4,05
32	0,94	0,10	4,96
64	1,01	0,10	5,40
128	1,14	0,14	6,01
256	1,25	0,15	7,50
512	1,52	0,15	10,22
1.024	2,08	0,17	15,89
2.048	2,90	0,19	23,69
4.096	4,71	0,34	41,70
8.192	7,89	0,64	71,96
16.384	25,77	1,82	224,57
32.768	27,70	3,34	217,96
65.536	55,97	6,98	413,48
131.072	118,75	13,47	811,98
262.144	274,53	47,87	1.808,37
524.288	666,98	112,93	4.644,81
1.048.576	3.543,27	510,34	17.134,42

Table 6: Binomial Gather Operation Latencies

Size (bytes)	AvgLatency(us)	MinLatency(us)	MaxLatency(us)
1	0,53	0,08	5,11
2	0,51	0,08	4,58
4	0,50	0,08	4,48
8	0,49	0,08	4,21
16	0,50	0,08	4,09
32	0,49	0,08	4,11
64	0,50	0,08	4,82
128	0,67	0,09	8,23
256	0,68	0,11	9,05
512	0,68	0,13	10,07
1.024	0,69	0,17	14,21
2.048	1,06	0,24	21,71
4.096	1,50	0,34	32,25
8.192	2,30	0,49	53,82
16.384	6,56	1,08	158,09
32.768	12,27	1,99	213,99
65.536	28,52	4,08	438,34
131.072	65,53	8,43	970,44
262.144	890,42	72,89	1.630,83
524.288	2.170,69	196,58	3.598,58
1.048.576	5.394,52	479,71	8.384,75

Table 7: Basic Linear Gather Operation Latencies

3.2.2 Performance Ratios

Performance ratios for the `binomial` and `basic_linear` algorithms were calculated in comparison to the baseline:

Algorithm \ Size (bytes)	1	2	4	8	16	32	64	128	256	512	1.024	2.048
Binomial	0,97	1,00	0,97	0,96	0,99	0,98	0,99	1,01	0,94	1,01	1,05	0,99
Basic Linear	0,69	0,69	0,66	0,63	0,63	0,51	0,49	0,59	0,51	0,45	0,35	0,36

Algorithm \ Size (bytes)	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576
Binomial	0,99	1,00	1,11	1,00	0,97	1,00	1,02	1,00	1,00
Basic Linear	0,32	0,29	0,28	0,44	0,49	0,55	3,30	3,25	1,53

Table(s) 8: Gather Performance Ratios

3.2.3 Graphical Representation

A line graph depicting the latency trends for the gather operation is provided:

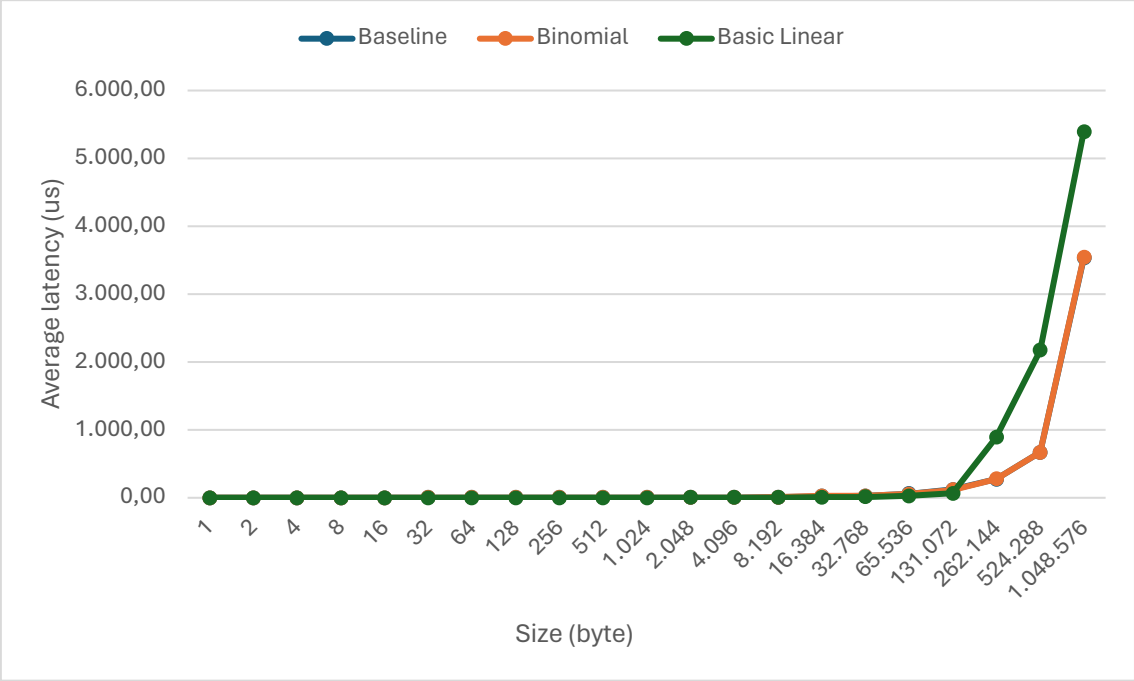

Figure 2: Gather Operation Latency Comparison

4. Discussion

The benchmarking results, visually represented in the graphs and quantitatively analyzed through performance ratios, offer insights into the behavior of different collective operation algorithms under various conditions. The discussion focuses on these findings and their implications in the context of the ORFEO cluster's architecture.

4.1 Broadcast Operation

The benchmarking results for the broadcast operation on the ORFEO cluster present a clear performance distinction between the `scatter_allgather_ring` and `pipeline` algorithms.

4.1.1 Scatter AllGather Ring Benchmark

For small message sizes, the algorithm does not exhibit a clear advantage over the baseline. The performance ratios suggest that the algorithm's setup and execution overhead may not be offset by the benefits of parallel dissemination and collection at these smaller scales.

However, as message sizes grow, the algorithm begins to outperform the baseline, with performance ratios steadily decreasing to below 1, reaching as low as 0.63. The efficiency for larger message sizes of this approach in the ORFEO cluster can be attributed to several key factors:

- The network's full non-blocking capabilities allow multiple segments of the message to be communicated simultaneously without interference, effectively leveraging the high bandwidth provided by the 100 Gbit Infiniband infrastructure.

- Full-duplex communication of the Infiniband technology is well-suited for the `scatter_allgather_ring` algorithm, as it can handle concurrent send and receive operations, thus reducing the communication time.
- The algorithm's parallel dissemination and collection of message segments mean that it can scale effectively with the number of processes, maintaining consistent performance as message sizes grow.

4.1.2 Pipeline Benchmark

In stark contrast, the pipeline algorithm exhibits less favorable performance, particularly for larger messages. This algorithm operates by passing the message sequentially from one process to the next in a pipeline manner, causing several disadvantages:

- The sequential transfer of data fails to utilize the network's full non-blocking and full-duplex communication capabilities, resulting in suboptimal performance.
- As message sizes increase, the cumulative delay introduced by the sequential handoff of data across processes becomes more pronounced, leading to higher overall latencies.
- The pipeline algorithm's linear communication pattern does not scale well with the number of processes or the size of the message, as each step in the pipeline introduces additional latency.

4.2 Gather Operation

The `binomial` and `basic_linear` algorithms are two different approaches to conducting gather operations in MPI, and their performance on the ORFEO cluster can be significantly affected by the network's characteristics and the number of processes involved.

4.2.1 Binomial Benchmark

The binomial algorithm for gather operations uses a tree-based approach, where data is gathered in stages, halving the number of communicating nodes at each step until the root node has all the data. This method has several features that can contribute to its performance being comparable to the baseline on the ORFEO cluster:

- Logarithmic steps: The binomial algorithm reduces the number of steps required to gather the data as the number of processes increases, which can lead to less time spent in communication compared to linear methods.
- Overlap of communication: The tree structure allows for overlapping communication, where different branches of the tree can communicate simultaneously, leveraging the full-duplex capabilities of the Infiniband network.

4.2.2 Basic Linear Benchmark

For smaller message sizes, ranging from 1 byte to 64 bytes, the `basic_linear` algorithm has performance ratios less than 1, indicating a lower latency compared to the baseline. This suggests that the simplicity of the algorithm might be beneficial when handling very small amounts of data due to potentially less overhead in setting up more complex communication patterns.

However, as the message size grows, the algorithm's performance relative to the baseline worsens, with performance ratios jumping to values as high as 3.30. The sharp increase in performance ratios for larger message sizes can be explained by the following factors:

- Inefficient communication pattern: The sequential nature of the `basic_linear` algorithm fails to utilize the full potential of the ORFEO cluster's Infiniband network, especially as the amount of data to gather increases.

- Increased latency: The compounded latency from the serial collection of data from each process becomes more pronounced with larger message sizes, leading to a substantial increase in total gather time.
- Scalability issues: The algorithm's performance deteriorates as the number of processes increases, with each additional process adding more time to the sequential gathering sequence.

5. Conclusions

The comparative analysis underscores the importance of algorithm selection in HPC environments, particularly when dealing with varying message sizes. While some algorithms may perform well across the board, others exhibit size-dependent behavior that can either benefit or hinder performance.

The `scatter_allgather_ring` and `binomial` algorithms have demonstrated their effectiveness on the ORFEO cluster, making them suitable candidates for applications that require their respective collective operations. However, the `pipeline` and `basic_linear` algorithms may require careful consideration and potentially avoidance for applications involving large message transmissions.

These findings emphasize the need for careful benchmarking and evaluation when selecting collective operation algorithms for use in HPC applications, considering both the network architecture and the typical data volumes to be communicated.