Assignment 1

Final Report submitted by Nooshin Behrooz

ORFEO username nbehrooz

Problem: Compare different openMPI algorithms for collective operations

- ## Introduction

We want to evaluate the latency of the default implementation within the openMPI library for collective operations **Bcast** and **Scatter**. This entails examining how quickly messages are transmitted between processes, considering variations in both the number of processes involved and the size of the messages exchanged. Furthermore, we will contrast these findings with latency values derived from employing different algorithms within the library. Essentially, the goal is to understand how efficiently the default implementation, which dynamically adjusts its collective communication algorithms based on the runtime conditions (like message size and number of processes), performs compared to alternative approaches under various conditions.

For the **Bcast** communication, we will evaluate three different algorithms: **basic_linear**, **binary_tree**, and **knomial tree**. Similarly, for **Scatter** collective operations, we will apply **basic linear**, **binomial**, and **linear_nb** (non-blocking linear) algorithms.

Here we will introduce very briefly these algorithms respectively, for **Bcast operation**:

1. **Basic_Linear**: In the basic linear algorithm for the Bcast communication, each process sends its data directly to every other process in a linear sequence, resulting in $O(N)$ communication complexity.

2. **Binary_Tree**: In the binary tree algorithm, processes are organized in a binary tree structure where each process sends data to its parent and eventually to all other processes, reducing communication complexity to $O(log\ N)$.

3. **Knomial Tree**: The knomial tree algorithm for Bcast communication groups processes into a k-ary tree, where each process sends data to its k-1 children, resulting in improved scalability over binary tree with communication complexity $O(log_k N)$.

For **Scatter operation**:

1. **Basic_Linear**: In the basic linear algorithm for the Scatter communication, the root process sends portions of the data to each process in a linear sequence, resulting in $O(N)$ communication complexity.

2. **Binomial:** The binomial algorithm for Scatter communication organizes processes in a binomial tree structure, where each process receives data from multiple sources, reducing communication complexity to $O(log\ N)$.

3. **Linear_nb** (non-blocking Linear): In the linear nb algorithm, each process initiates non-blocking sends to receive data from the root process simultaneously, allowing overlapping communication and computation, enhancing performance for Scatter operations.

▪ OSU Benchmark Algorithm

The **OSU Benchmarks** provide a standardized set of tests for assessing MPI performance across various platforms and configurations. They cover a range of communication patterns and operations, including point-to-point and collective operations. The goal of using OSU benchmarks is to evaluate and compare MPI implementations, optimize code, and diagnose performance issues in parallel computing environments.

In this regard, we will examine two collective operations: Bcast and Scatter communications, each potentially employing various algorithms for different message sizes or process counts. Our aim is to explore different algorithms for these communicators and compare their results with those of the default algorithm, which is dynamically selected at runtime based on internal tuning rules. The number of MPI processes used in the benchmark can significantly affect the results, particularly in distributed memory systems where inter-node communication results in overhead.

Some information about OSU benchmarks: they have similar overall structure, while they are different in specific communication operations.

- **benchmark osu_bcast.c and osu_scatter.c :**

1. **Initialization:** The code begins with including necessary headers, defining constants and structures.

2. **Main Function:** initializes variables, MPI, and options, then it proceeds with the benchmarking process.

3. **Benchmarking Loop**: The code iterates over different message sizes and MPI data types, performing the broadcast operation multiple times. Latency metrics are calculated for each combination of size and data type.

4. **Output:** Latency statistics, such as average, minimum, and maximum latency, are printed. Optionally, graphs may be generated to visualize the results.

To run this benchmark, Open MPI libraries need to be installed. Detailed instructions on how to use benchmark operations, compile, and run the code are provided in the *slurm.job* files and the *Readme* pdf. They can be download from the link provided at the end of the report.

▪ Performance and Results of OSU Benchmark

Rsult1:

Using OSU benchmarks, we investigate the behavior of various algorithms for collective operations, focusing on Bcast and scatter communications. We compare these algorithms with the default algorithm, which is dynamically chosen at runtime based on message size and number of the core for optimal performance.

a. Bcast communication

For the Bcast operator, we examined four different algorithms: **default**, **basic_linear, binary_tree**, and **knomial.** Latency among nodes was measured by incrementing message size and number of the core across the range of (2, 4, 8, 16, 32, 64, 128, 256). Results are depicted in figures 1a, 1b, 2a, 2b, 3a, 3b, 4a, and 4b.

0. Default Algorithm

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.15	1	0.57	1	1.59	1	1.77
2	0.14	2	0.56	2	1.49	2	1.73
4	0.16	4	0.56	4	1.64	4	1.74
8	0.14	8	0.56	8	1.66	8	1.72
16	0.15	16	0.56	16	1.59	16	1.78
32	0.15	32	0.66	32	1.92	32	2.16
64	0.15	64	0.66	64	1.70	64	2.17
128	0.21	128	0.87	128	2.83	128	3.22
256	0.22	256	0.91	256	2.52	256	3.29
512	0.23	512	1.11	512	2.97	512	3.40
1024	0.26	1024	1.21	1024	3.33	1024	3.91
2048	0.29	2048	1.86	2048	4.79	2048	5.37
4096	0.37	4096	2.65	4096	6.28	4096	7.01
8192	0.54	8192	4.47	8192	10.06	8192	10.57
16384	4.40	16384	8.73	16384	18.40	16384	22.44
32768	5.14	32768	10.49	32768	26.48	32768	38.42
65536	6.70	65536	16.26	65536	57.80	65536	93.96
131072	10.02	131072	31.18	131072	128.77	131072	173.00
262144	21.57	262144	78.42	262144	285.26	262144	341.98
524288	40.98	524288	167.66	524288	576.55	524288	611.58
1048576	80.02	1048576	394.05	1048576	881.01	1048576	589.01

1a)

np=32		np=64		np=128		np=256	
Size	Avg Latency(us)	# Size	Avg Latency(us)	Size	Avg Latency(us)	# Size	Avg Latency(us)
1	2.01	1	2.02	1	2.15	1	1.14
2	2.04	2	1.98	2	3.42	2	1.21
4	2.06	4	1.98	4	3.34	4	1.19
8	2.06	8	1.98	8	3.41	8	1.18
16	2.08	16	1.97	16	4.01	16	1.15
32	2.42	32	2.56	32	3.78	32	1.56
64	2.46	64	2.55	64	3.91	64	1.51
128	3.53	128	3.74	128	6.24	128	2.67
256	3.59	256	3.87	256	6.36	256	3.27
512	3.92	512	4.23	512	6.85	512	4.12
1024	4.24	1024	4.60	1024	7.61	1024	4.93
2048	6.08	2048	7.16	2048	10.93	2048	8.75
4096	9.16	4096	11.55	4096	15.01	4096	18.26
8192	15.94	8192	20.82	8192	24.33	8192	33.57
16384	27.40	16384	35.54	16384	207.38	16384	111.70
32768	47.95	32768	67.67	32768	86.46	32768	136.87
65536	99.66	65536	141.73	65536	275.45	65536	263.25
131072	238.97	131072	315.84	131072	589.14	131072	573.03
262144	509.41	262144	778.61	262144	1091.66	262144	1177.22
524288	1092.34	524288	1655.12	524288	2109.80	524288	2218.22
1048576	1892.10	1048576	2761.77	1048576	4224.90	1048576	4119.46

1b)

1. Basic_linear

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)		
1	0.14	1	0.35	1	1.99	1	2.38
2	0.14	2	0.34	2	2.00	2	2.42
4	0.14	4	0.35	4	2.03	4	2.36
8	0.14	8	0.35	8	1.99	8	2.41
16	0.13	16	0.33	16	1.99	16	2.37
32	0.12	32	0.38	32	2.03	32	2.73
64	0.14	64	0.37	64	2.17	64	2.67
128	0.19	128	0.82	128	3.49	128	4.90
256	0.20	256	0.76	256	3.56	256	5.43
512	0.21	512	1.01	512	3.71	512	6.11
1024	0.26	1024	1.02	1024	3.97	1024	7.00
2048	0.29	2048	1.96	2048	5.70	2048	10.02
4096	0.38	4096	2.56	4096	7.99	4096	13.60
8192	0.56	8192	4.49	8192	12.12	8192	21.62
16384	4.48	16384	9.13	16384	17.02	16384	26.49
32768	5.40	32768	10.48	32768	32.09	32768	60.96
65536	6.78	65536	15.06	65536	63.33	65536	117.92
131072	9.00	131072	31.84	131072	130.18	131072	172.55
262144	20.01	262144	74.70	262144	285.59	262144	333.65
524288	40.55	524288	165.80	524288	575.53	524288	742.21
1048576	79.81	1048576	383.40	1048576	972.54	1048576	1563.75

2a)

np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	4.53	1	8.13	1	15.27	1	40.58
2	4.60	2	8.19	2	15.71	2	39.89
4	4.55	4	8.19	4	16.74	4	40.50
8	4.60	8	8.41	8	16.08	8	40.24
16	4.90	16	8.58	16	17.14	16	41.31
32	5.12	32	10.09	32	17.08	32	41.32
64	5.02	64	9.93	64	17.69	64	42.67
128	10.89	128	23.60	128	48.42	128	81.85
256	11.57	256	24.62	256	48.55	256	86.53
512	12.06	512	23.89	512	49.76	512	86.14
1024	13.26	1024	25.24	1024	51.96	1024	93.43
2048	19.44	2048	39.93	2048	79.53	2048	135.10
4096	25.75	4096	51.96	4096	105.62	4096	182.94
8192	40.76	8192	81.65	8192	165.72	8192	305.03
16384	56.99	16384	104.71	16384	224.81	16384	255.35
32768	116.04	32768	235.51	32768	452.70	32768	419.15
65536	253.35	65536	520.36	65536	997.23	65536	717.31
131072	519.46	131072	1065.04	131072	2114.53	131072	1436.96
262144	1076.51	262144	2049.14	262144	4315.91	262144	2873.30
524288	2143.84	524288	4259.46	524288	8480.02	524288	6500.77
1048576	3852.62	1048576	8196.32	1048576	16629.83	1048576	15360.29

2b)

2. Binary_tree

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.15	1	0.49	1	1.70	1	1.83
2	0.15	2	0.49	2	1.69	2	1.84
4	0.15	4	0.49	4	1.73	4	1.86
8	0.16	8	0.50	8	1.69	8	1.86
16	0.15	16	0.50	16	1.70	16	2.27
32	0.16	32	0.57	32	2.22	32	2.44
64	0.16	64	0.57	64	1.97	64	2.38
128	0.22	128	0.79	128	2.91	128	3.49
256	0.22	256	0.83	256	2.94	256	3.94
512	0.27	512	0.96	512	3.33	512	4.26
1024	0.25	1024	1.09	1024	3.55	1024	5.17
2048	0.31	2048	1.62	2048	4.82	2048	6.82
4096	0.40	4096	2.34	4096	6.58	4096	9.06
8192	0.57	8192	3.80	8192	9.77	8192	12.88
16384	4.45	16384	10.44	16384	20.83	16384	28.28
32768	5.21	32768	13.45	32768	33.22	32768	46.65
65536	7.04	65536	20.59	65536	60.23	65536	84.36
131072	9.30	131072	35.69	131072	114.35	131072	151.65
262144	21.35	262144	76.81	262144	215.30	262144	269.25
524288	40.78	524288	146.11	524288	352.22	524288	507.61
1048576	80.08	1048576	279.34	1048576	600.58	1048576	887.11

np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)			# Size	Avg Latency(us)
1	2.39	1	2.80	1	3.30	1	6.17
2	2.40	2	2.88	2	3.49	2	5.57
4	2.39	4	2.88	4	3.35	4	6.36
8	2.42	8	2.87	8	3.44	8	6.28
16	2.45	16	2.86	16	3.45	16	5.57
32	2.84	32	3.64	32	3.77	32	5.70
64	2.84	64	3.65	64	3.88	64	7.75
128	4.43	128	5.37	128	6.19	128	10.34
256	4.45	256	5.77	256	6.53	256	10.99
512	4.87	512	6.34	512	6.88	512	13.64
1024	5.18	1024	6.71	1024	7.70	1024	16.27
2048	7.72	2048	9.29	2048	11.29	2048	25.47
4096	10.26	4096	12.21	4096	15.02	4096	41.19
8192	15.40	8192	19.30	8192	24.35	8192	81.08
16384	34.34	16384	43.09	16384	52.59	16384	185.39
32768	56.77	32768	69.58	32768	87.03	32768	440.89
65536	100.36	65536	123.88	65536	153.47	65536	491.47
131072	189.88	131072	235.91	131072	294.12	131072	1025.09
262144	364.08	262144	455.28	262144	577.72	262144	2283.96
524288	653.52	524288	850.44	524288	1095.41	524288	4928.99
1048576	1097.49	1048576	1372.71	1048576	1826.61	1048576	10917.53

3. Knomial

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.15	1	0.36	1	1.67	1	1.69
2	0.15	2	0.37	2	1.62	2	1.68
4	0.16	4	0.37	4	1.58	4	1.72
8	0.15	8	0.36	8	1.63	8	1.67
16	0.16	16	0.37	16	1.60	16	1.70
32	0.15	32	0.44	32	1.84	32	2.08
64	0.16	64	0.39	64	1.88	64	2.06
128	0.21	128	0.74	128	2.96	128	3.01
256	0.22	256	0.77	256	2.83	256	3.02
512	0.24	512	1.00	512	3.17	512	3.23
1024	0.29	1024	1.04	1024	3.40	1024	3.50
2048	0.32	2048	1.80	2048	4.68	2048	5.60
4096	0.40	4096	2.59	4096	6.53	4096	7.79
8192	0.57	8192	4.47	8192	9.86	8192	12.66
16384	4.69	16384	9.12	16384	17.57	16384	20.49
32768	5.23	32768	11.09	32768	29.18	32768	34.56
65536	6.70	65536	17.70	65536	54.40	65536	70.95
131072	9.24	131072	34.59	131072	112.59	131072	159.05
262144	22.43	262144	82.43	262144	241.86	262144	330.53
524288	42.04	524288	179.55	524288	494.04	524288	726.26
1048576	84.13	1048576	380.44	1048576	912.03	1048576	1312.49

4a)

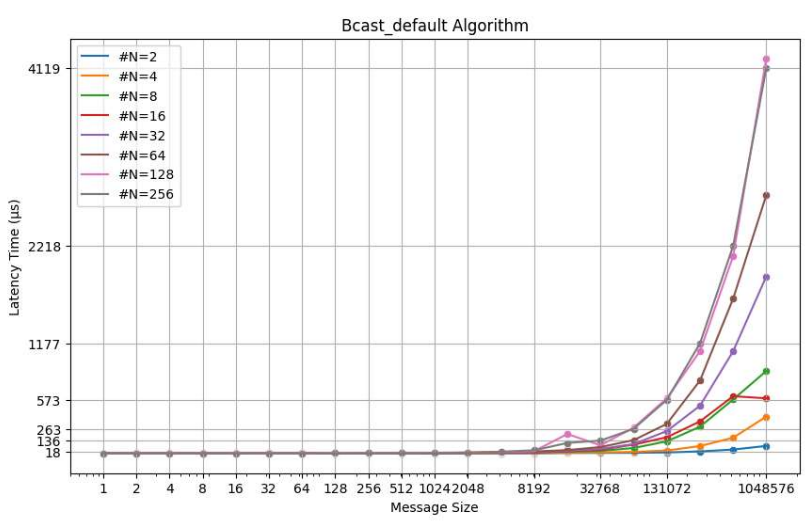
np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	2.01	1	2.11	1	2.11	1	1.38
2	2.69	2	2.16	2	2.46	2	1.41
4	2.12	4	2.12	4	2.08	4	1.40
8	2.85	8	2.10	8	2.16	8	1.40
16	2.27	16	2.11	16	2.23	16	1.38
32	3.17	32	2.78	32	2.33	32	1.77
64	2.57	64	2.73	64	2.30	64	1.72
128	3.66	128	4.14	128	4.49	128	3.02
256	3.83	256	4.32	256	4.74	256	3.71
512	3.99	512	4.95	512	5.06	512	4.75
1024	4.27	1024	5.12	1024	5.58	1024	5.65
2048	6.39	2048	7.67	2048	8.89	2048	9.24
4096	10.04	4096	12.37	4096	17.87	4096	20.57
8192	15.64	8192	20.72	8192	33.78	8192	36.38
16384	25.63	16384	36.26	16384	59.10	16384	66.26
32768	57.10	32768	68.74	32768	127.18	32768	133.06
65536	101.78	65536	148.15	65536	238.45	65536	265.68
131072	225.60	131072	333.02	131072	551.04	131072	571.95
262144	508.92	262144	758.28	262144	1194.93	262144	1237.29
524288	1069.23	524288	1533.30	524288	2438.84	524288	2288.65
1048576	1981.62	1048576	2795.78	1048576	4038.25	1048576	4107.63

4b)

These outputs demonstrate that, across all distinct algorithms and varying core counts, average latency increases with message size. This aligns with the expected behavior in network communication, where larger messages typically demand more transmission time and revealing the scaling of latency with message size.

With closer examination of the data, it becomes apparent that as the message size increases beyond 8192 bytes, the average latency time approximately doubles. Conversely, for message sizes within the interval [1-4096], the corresponding latency remains nearly constant or increases with a mild slope.

To be more clear, we plot these results only for default algorithm:



b. Scatter communication

For the Scatter operator, we followed the same methodology as with Bcast, extending it to include the OSU Benchmark for the scatter communication across four different algorithms: **default**, **basic_linear, binomial,** and **linear_nb.** Latency among nodes was assessed by varying message size and and number of cores within the range of (2, 4, 8, 16, 32, 64, 128, 256). The results, respectively are illustrated in figures 5a, 5b, 6a, 6b, 7a, 7b, 8a, and 8b.

0. Default Algorithm

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.16	1	0.57	1	1.59	1	1.63
2	0.16	2	0.56	2	1.62	2	1.65
4	0.16	4	0.57	4	1.57	4	1.64
8	0.15	8	0.56	8	1.60	8	1.63
16	0.15	16	0.55	16	1.62	16	1.63
32	0.15	32	0.65	32	1.91	32	1.99
64	0.16	64	0.65	64	1.78	64	2.04
128	0.23	128	0.99	128	2.78	128	2.94
256	0.23	256	0.90	256	2.67	256	2.95
512	0.26	512	1.13	512	3.02	512	3.17
1024	0.28	1024	1.20	1024	3.33	1024	3.41
2048	0.31	2048	1.87	2048	4.65	2048	6.47
4096	0.40	4096	2.63	4096	6.31	4096	8.48
8192	0.54	8192	4.47	8192	9.74	8192	10.54
16384	4.44	16384	10.36	16384	19.29	16384	21.94
32768	5.27	32768	11.66	32768	27.58	32768	36.91
65536	6.70	65536	17.59	65536	56.06	65536	77.04
131072	10.20	131072	34.50	131072	129.14	131072	156.09
262144	19.49	262144	74.73	262144	278.65	262144	351.73
524288	41.63	524288	146.33	524288	571.74	524288	667.53
1048576	85.59	1048576	395.30	1048576	930.54	1048576	594.80

5a)

np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	1.88	1	2.31	1	2.20	1	1.34
2	1.91	2	1.83	2	3.61	2	1.34
4	1.90	4	1.87	4	3.58	4	1.36
8	1.89	8	1.80	8	3.58	8	1.37
16	1.97	16	1.80	16	3.61	16	1.34
32	2.31	32	2.39	32	4.00	32	1.74
64	2.31	64	2.34	64	4.20	64	1.72
128	3.46	128	3.40	128	6.51	128	2.91
256	3.52	256	3.67	256	6.80	256	3.67
512	3.92	512	4.26	512	7.45	512	4.63
1024	4.25	1024	4.45	1024	8.13	1024	5.42
2048	6.13	2048	6.83	2048	11.45	2048	8.96
4096	9.54	4096	11.63	4096	17.38	4096	19.42
8192	15.47	8192	20.28	8192	25.42	8192	36.30
16384	25.49	16384	37.23	16384	201.97	16384	113.40
32768	45.55	32768	66.74	32768	87.56	32768	140.63
65536	93.08	65536	145.96	65536	271.63	65536	271.39
131072	206.11	131072	310.67	131072	513.73	131072	559.96
262144	484.57	262144	648.36	262144	1154.34	262144	1170.83
524288	1078.73	524288	1353.84	524288	2361.59	524288	2266.66
1048576	1951.16	1048576	2398.98	1048576	4399.62	1048576	4068.63

5b)

1. Basic_linear

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.14	1	0.34	1	2.04	1	2.43
2	0.13	2	0.34	2	2.01	2	2.40
4	0.14	4	0.33	4	2.02	4	2.44
8	0.14	8	0.35	8	2.02	8	2.40
16	0.13	16	0.33	16	2.05	16	2.47
32	0.14	32	0.36	32	2.06	32	2.71
64	0.14	64	0.35	64	2.13	64	2.74
128	0.19	128	0.74	128	3.45	128	4.91
256	0.21	256	0.85	256	3.37	256	5.68
512	0.24	512	0.93	512	3.54	512	6.31
1024	0.28	1024	1.05	1024	3.74	1024	6.98
2048	0.32	2048	1.78	2048	5.74	2048	10.11
4096	0.40	4096	2.54	4096	7.93	4096	14.48
8192	0.57	8192	4.51	8192	12.11	8192	22.83
16384	4.46	16384	9.45	16384	15.42	16384	28.12
32768	5.16	32768	10.56	32768	30.76	32768	57.93
65536	6.89	65536	16.35	65536	63.41	65536	126.76
131072	11.08	131072	32.72	131072	130.00	131072	270.22
262144	22.37	262144	83.15	262144	286.61	262144	552.64
524288	46.67	524288	193.02	524288	569.13	524288	861.05
1048576	96.58	1048576	396.06	1048576	1014.36	1048576	1693.72

6a)

np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	4.51	1	7.70	1	15.11	1	40.29
2	4.65	2	7.98	2	15.56	2	39.88
4	4.53	4	7.99	4	16.30	4	40.33
8	4.64	8	8.26	8	15.87	8	39.87
16	4.53	16	9.42	16	16.80	16	40.97
32	5.25	32	9.54	32	17.01	32	40.96
64	4.95	64	9.41	64	17.43	64	42.27
128	11.17	128	23.87	128	50.60	128	80.33
256	11.33	256	24.64	256	50.08	256	86.85
512	12.51	512	24.15	512	50.26	512	86.30
1024	13.77	1024	25.27	1024	52.27	1024	94.78
2048	20.06	2048	40.28	2048	80.22	2048	134.92
4096	26.35	4096	52.16	4096	106.39	4096	183.38
8192	41.75	8192	82.27	8192	163.50	8192	303.07
16384	52.53	16384	100.82	16384	227.52	16384	252.46
32768	122.50	32768	233.13	32768	461.82	32768	421.09
65536	264.57	65536	462.34	65536	1020.15	65536	701.06
131072	533.39	131072	706.04	131072	2135.55	131072	1435.65
262144	1061.66	262144	1350.03	262144	4223.65	262144	2873.20
524288	2112.44	524288	3057.79	524288	8532.56	524288	6636.84
1048576	3885.65	1048576	7028.45	1048576	16139.47	1048576	15192.11

6b)

2. Binomial

np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.16	1	0.36	1	1.71	1	1.73
2	0.16	2	0.38	2	1.69	2	1.70
4	0.15	4	0.36	4	1.74	4	1.73
8	0.15	8	0.38	8	1.67	8	1.71
16	0.16	16	0.36	16	1.68	16	1.72
32	0.16	32	0.43	32	1.94	32	2.19
64	0.16	64	0.38	64	1.96	64	2.18
128	0.21	128	0.82	128	2.95	128	3.14
256	0.23	256	0.82	256	2.99	256	3.82
512	0.23	512	0.96	512	3.27	512	4.09
1024	0.24	1024	1.05	1024	3.44	1024	4.45
2048	0.31	2048	1.92	2048	4.88	2048	6.21
4096	0.41	4096	2.57	4096	6.41	4096	8.41
8192	0.59	8192	4.47	8192	9.62	8192	13.15
16384	4.43	16384	10.32	16384	17.34	16384	28.58
32768	5.22	32768	12.33	32768	31.23	32768	48.95
65536	6.79	65536	19.03	65536	60.46	65536	93.92
131072	10.57	131072	38.38	131072	118.00	131072	182.65
262144	21.90	262144	98.79	262144	235.25	262144	350.22
524288	49.41	524288	182.13	524288	470.63	524288	623.47
1048576	98.43	1048576	381.31	1048576	892.85	1048576	1081.19

np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	2.69	1	5.74	1	17.67	1	17.91
2	2.69	2	5.73	2	12.71	2	18.74
4	2.70	4	5.68	4	12.43	4	19.34
8	2.74	8	5.72	8	12.42	8	17.93
16	2.69	16	5.78	16	13.96	16	18.00
32	4.18	32	7.62	32	15.58	32	21.42
64	3.42	64	7.50	64	14.77	64	21.37
128	4.80	128	9.71	128	19.15	128	26.22
256	5.59	256	9.71	256	18.89	256	26.47
512	5.34	512	10.73	512	21.35	512	32.14
1024	5.95	1024	11.35	1024	25.04	1024	36.12
2048	8.66	2048	15.55	2048	30.40	2048	51.98
4096	12.25	4096	21.01	4096	41.49	4096	74.93
8192	20.64	8192	33.83	8192	66.14	8192	119.71
16384	47.56	16384	80.03	16384	147.08	16384	269.10
32768	80.59	32768	135.34	32768	236.94	32768	425.74
65536	149.97	65536	243.55	65536	397.54	65536	764.91
131072	285.65	131072	454.26	131072	730.31	131072	1375.48
262144	525.86	262144	765.66	262144	1362.37	262144	2484.59
524288	874.40	524288	1470.63	524288	2490.29	524288	4393.51
1048576	1610.05	1048576	2705.42	1048576	4705.67	1048576	7532.47

7a) **7b)**

3. Linear_nb

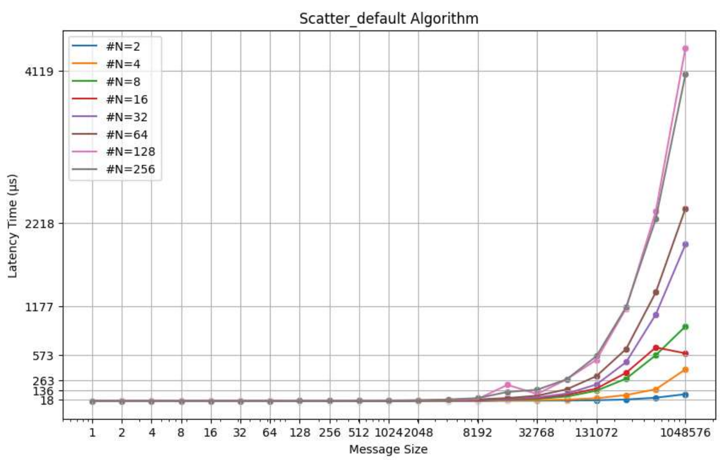
np=2		np=4		np=8		np=16	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	0.16	1	0.79	1	3.04	1	4.74
2	0.15	2	0.79	2	3.07	2	4.71
4	0.16	4	0.78	4	3.02	4	4.71
8	0.15	8	0.80	8	3.03	8	5.08
16	0.16	16	0.79	16	3.06	16	5.46
32	0.16	32	0.95	32	3.41	32	6.98
64	0.17	64	0.94	64	3.49	64	6.83
128	0.21	128	1.19	128	4.02	128	10.70
256	0.22	256	1.22	256	4.06	256	9.30
512	0.23	512	1.48	512	4.38	512	9.18
1024	0.26	1024	1.60	1024	5.07	1024	11.76
2048	0.31	2048	2.39	2048	6.80	2048	13.42
4096	0.40	4096	3.34	4096	9.02	4096	17.93
8192	0.58	8192	5.47	8192	13.71	8192	29.46
16384	4.55	16384	16.00	16384	31.53	16384	60.63
32768	5.24	32768	22.90	32768	49.51	32768	95.61
65536	6.84	65536	36.60	65536	85.20	65536	164.85
131072	10.55	131072	63.74	131072	152.14	131072	297.85
262144	24.49	262144	113.64	262144	273.23	262144	563.02
524288	50.04	524288	207.66	524288	506.95	524288	998.89
1048576	98.95	1048576	399.92	1048576	1027.50	1048576	1906.96

np=32		np=64		np=128		np=256	
# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)	# Size	Avg Latency(us)
1	10.41	1	24.79	1	53.92	1	79.38
2	10.32	2	26.51	2	55.54	2	88.96
4	11.20	4	25.04	4	58.01	4	78.16
8	11.58	8	24.94	8	54.22	8	73.55
16	10.70	16	24.58	16	51.55	16	71.16
32	14.57	32	29.98	32	59.39	32	86.12
64	13.25	64	29.63	64	58.64	64	104.37
128	16.65	128	34.51	128	70.28	128	109.68
256	15.40	256	34.55	256	76.33	256	108.96
512	18.00	512	40.23	512	85.74	512	143.26
1024	19.64	1024	41.90	1024	91.14	1024	144.94
2048	27.80	2048	55.22	2048	116.61	2048	208.63
4096	36.46	4096	72.63	4096	161.23	4096	288.73
8192	55.91	8192	113.29	8192	235.88	8192	459.31
16384	122.56	16384	238.06	16384	498.66	16384	980.97
32768	191.54	32768	383.88	32768	777.65	32768	1754.47
65536	324.69	65536	642.79	65536	1335.49	65536	2638.92
131072	584.72	131072	1169.17	131072	2360.08	131072	4758.95
262144	1062.11	262144	2120.04	262144	4283.08	262144	8723.14
524288	1961.08	524288	3852.99	524288	7864.79	524288	15824.92
1048576	3746.24	1048576	7280.20	1048576	14869.87	1048576	30141.12

8a) **8b)**

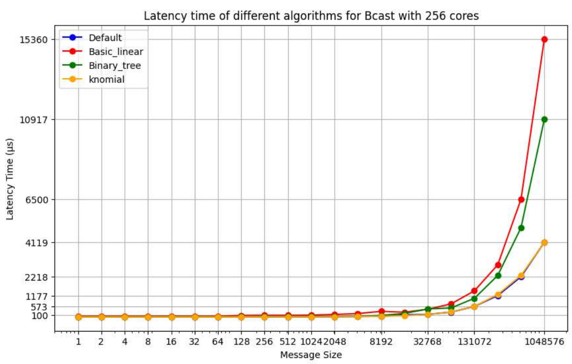
These outputs also confirm the previous findings obtained for various Bcast communication algorithms, demonstrating that larger messages generally require more transmission time, revealing the scaling of latency with message size. Additionally, they indicate a sharp increase in latency after the message size reaches 8192 bytes.

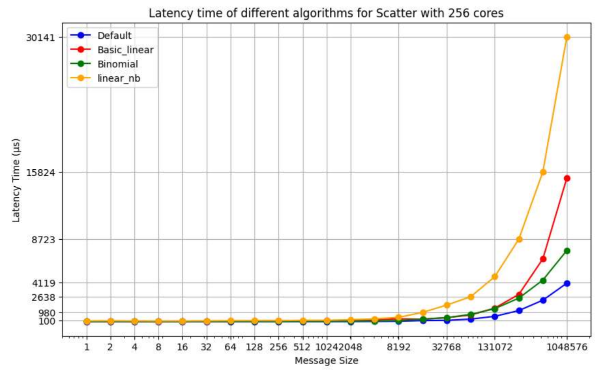
We plot these results only for default algorithm:



- **Result 2: Compare different algorithms for Bcast and Scatter Communication with Fixed Number of Cores**

By fixing the Number of Cores (256) for **Bcast** communication, **Knomial** algorithm has the most similar behavior in comparison with default, while in **Scatter** communication, **Binomial** Al has this property when we change the size of message.





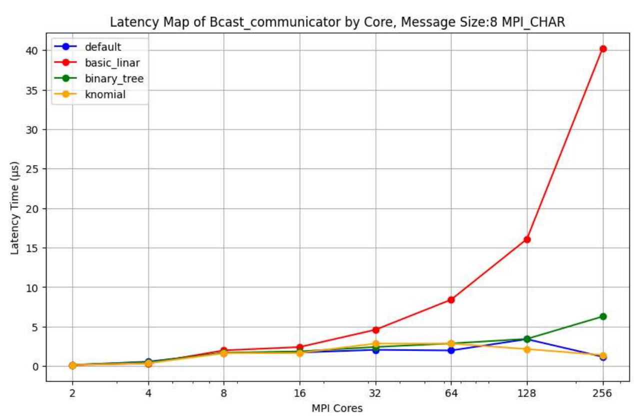
- **Result 3: Latancy Map**

To compare different algorithms for the collective communications Bcast and Scatter, we fix the message size. For each of the four algorithms, we will consider varying numbers of MPI cores within the range of (2, 4, 8, 16, 32, 64, 128, 256) on two Epyc Nodes and plot the latency maps.

- **Bcast communication:**

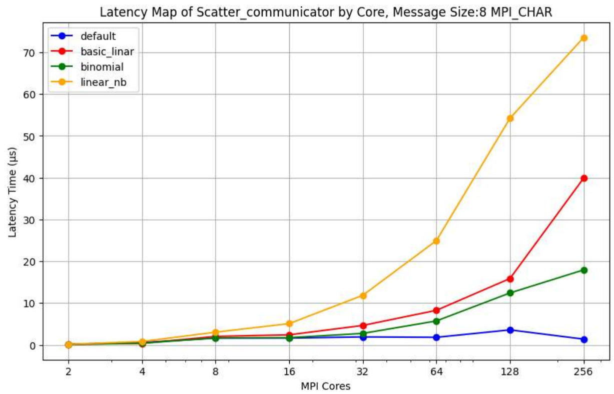
As illustrated in the following image, for the **default algorithm** (blue color), the latency time sincreases with the number of nodes until 128 nodes, after which the latency starts to decrease as the number of nodes increases to 256. The **Knomial** algorithm (yellow color) exhibits behavior similar to the default algorithm, but with the distinction that latency begins to decrease earlier, specifically from 64 nodes onward.

Both the **Basic_linear** (red color) and **binary_tree** (green color) algorithms demonstrate increasing latency as more nodes are added. However, the overall behavior of the binary_tree algorithm is closer to that of the default algorithm when compared to the basic_linear algorithm. Conversely, the basic_linear algorithm exhibits the most distinct behavior in comparison to the others.



- **Scatter communication:**

In the image below, we observe that for the default algorithm (blue), latency increases with the number of nodes until 128, followed by a decrease up to 256, very similar to the behavior of the Bcast default algorithm. However, the three verified algorithms, binomial (green), basic-linear (red), and linear_nb (yellow) exhibit distinct patterns from the default algorithm. While binomial data closely resembles the default algorithm, linear_nb demonstrates the most divergent behavior.



▪ Conclusion

- **Result 1:**

The outputs obtained from **four Bcast** communication algorithms and **four Scatter** communication algorithms reveal that **larger messages** generally require **more transmission time**, indicating the scaling of latency with message size. Moreover, they indicate a sharp increase in latency once the message size reaches approximately 8192 bytes.

- **Result 2:**

By fixing Number of Cores (256): For **Bcast**, **Knomial** algorithm has the most similar behavior in comparison with default, while in **Scatter**, **Binomial** Al has this property when we change the size of message.

- **Result 3:**

Latency map for the **Bcast communication** shows that, as the number of nodes increases, the **knomial** algorithm exhibits behavior closest to the default algorithm. Both experience latency increases with added cores, followed by a decrease beyond 64 and 128 nodes, respectively. Conversely, the **basic linear** algorithm consistently demonstrates increasing latency with node count and shows the most divergent behavior compared to the default algorithm.

Latency map for **scatter communication** reveals that, default algorithm's latency increases until 128 nodes, then decreases to 256, very close to Bcast default behavior, while all other three algorithms, Binomial, basic-linear and linear_nb show a constantly increasing trend. Among them, linear_nb displays the most divergent behavior.

The Codes, Slurms and Readme file are available on **Github** at:

https://github.com/SNB-Cs-Ds/hpc_projects/tree/main/project_1_OSU_Benchmark