Broadcast

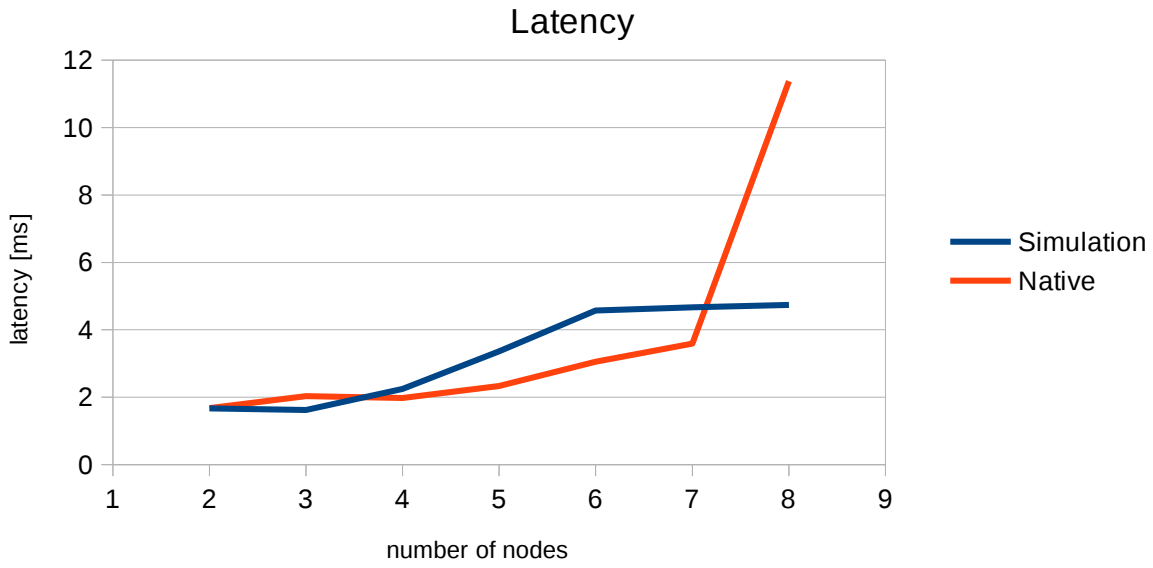
Efficiency by number of nodes

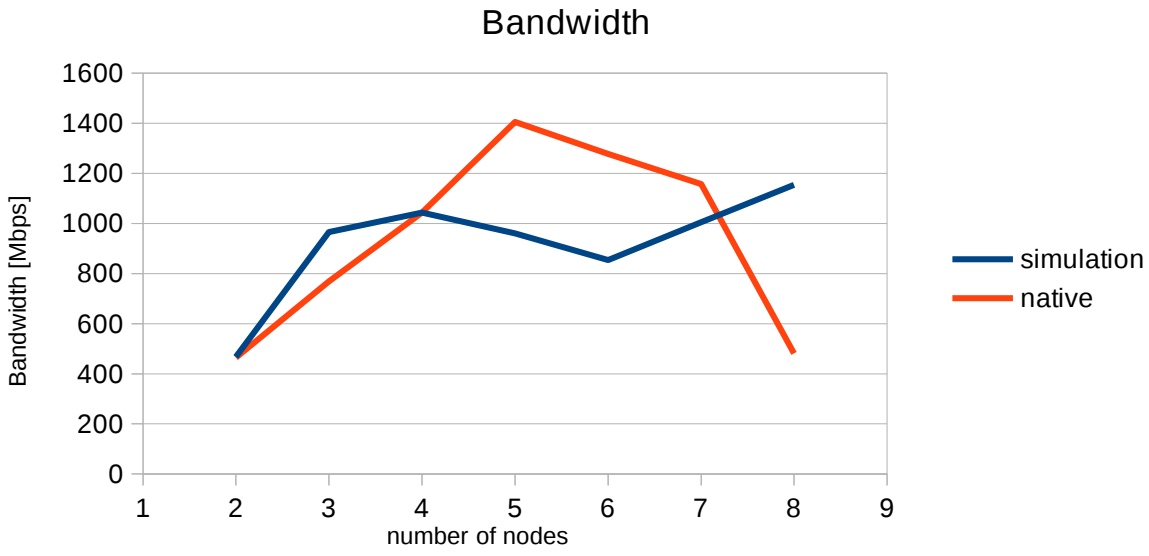
Constant message size = 100 kB

Results:

	latency [ms]		bandwidth [Mbps]	
no. of nodes	simulation	native	simulation	native
2	1.671	1.6811	467.49	464.7
3	1.6191	2.03165	965.26	769.08
4	2.2477	1.9749	1042.71	1042.73
5	3.3639	2.335	960.68	1404.55
6	4.5749	3.0577	853.84	1277.51
7	4.6647	3.5882	1004.87	1157.28
8	4.7373	11.370054	1154.37	480.98

Charts:





Conclusion:
native MPI_Bcast implementation seems to be slightly better, especially communication costs grows slower with number of nodes than in my implementation. From the other hand native implementation gets really slower for 8 nodes, it is recurrent situation.

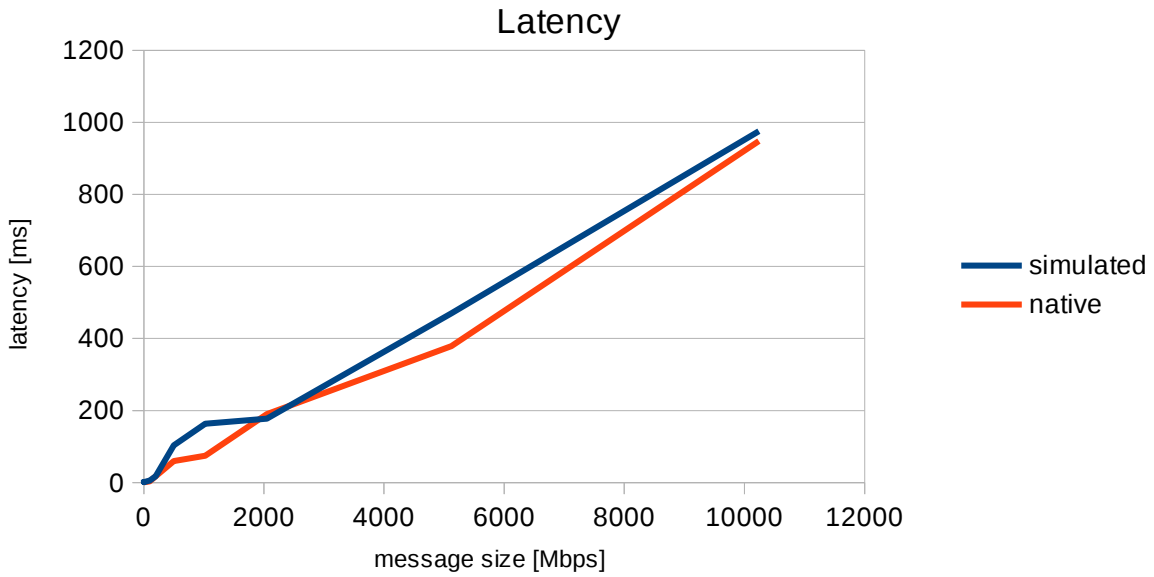
Efficiency by message size

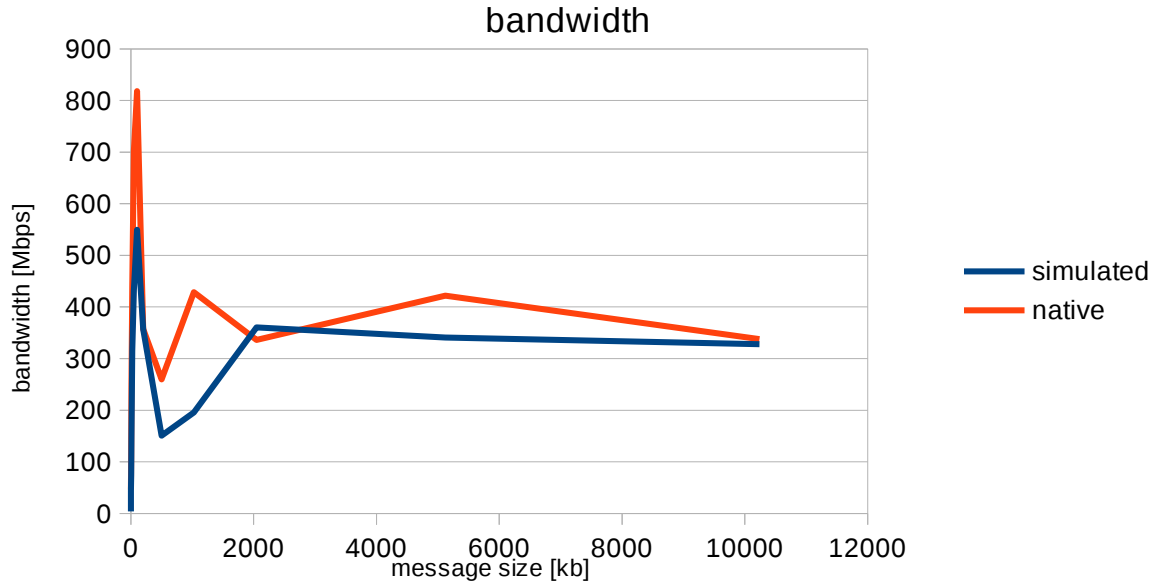
Constant number of nodes = 5

Results:

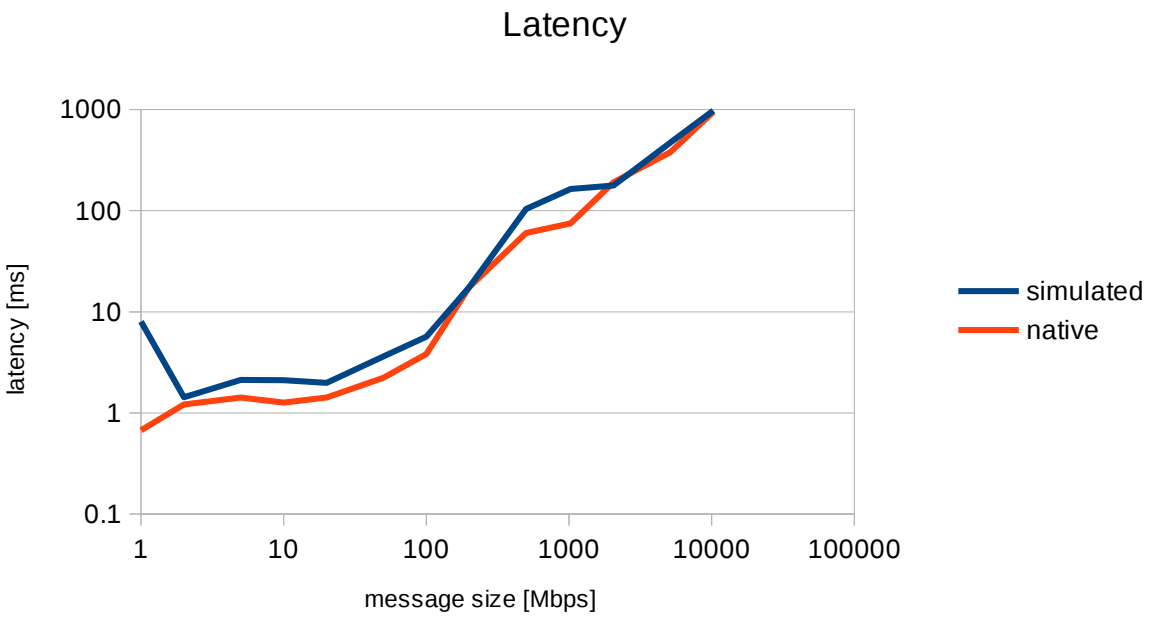
	latency[ms]		bandwidth[Mbps]	
message size [kB]	simulated	native	simulated	native
1	8.022499	0.672984	3.895295	46.434975
2	1.423907	1.210594	43.893307	51.627541
5	2.120614	1.42231	73.681488	109.856514
10	2.100515	1.26369	148.77301	247.291656
20	1.980996	1.421189	315.497833	439.772522
50	3.61619	2.226996	432.084595	701.617676
100	5.685997	3.819203	549.595764	818.233459
200	17.507887	17.492081	356.981964	357.304565
500	103.514603	60.166908	150.944885	259.694244
1024	163.759903	74.66819	195.408035	428.562683
2048	177.56601	190.344711	360.429352	336.232086
5120	469.813812	379.056793	340.560455	422.100342
10240	975.340088	948.446289	328.090668	337.39389

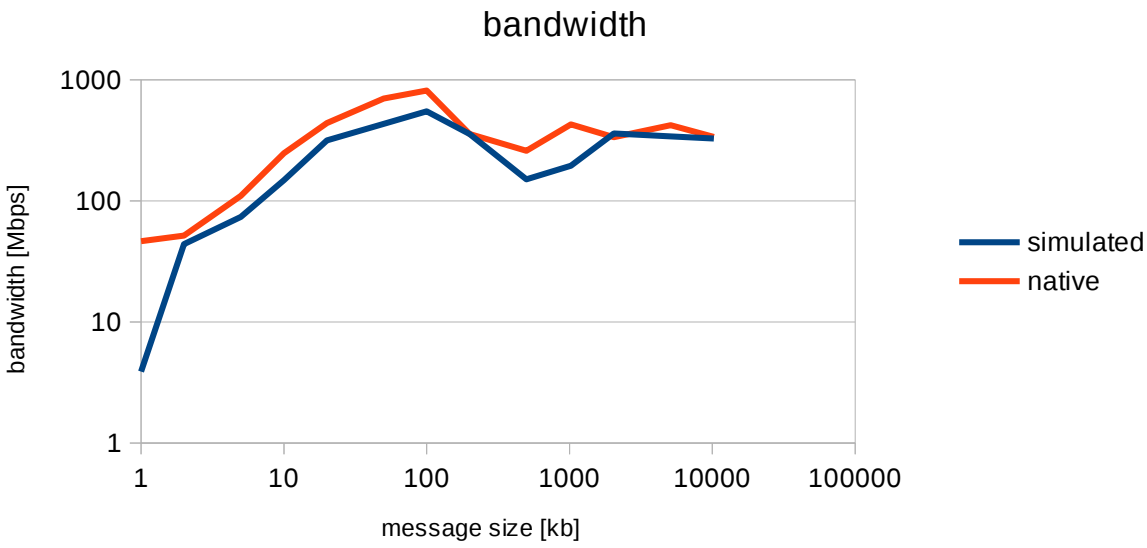
charts:





same data, logarithmic scale:





Conclusion:
In matter of efficiency both implementation behave very similar for small number of nodes. Unfortunately I have got no opportunity to test it with big number of nodes.

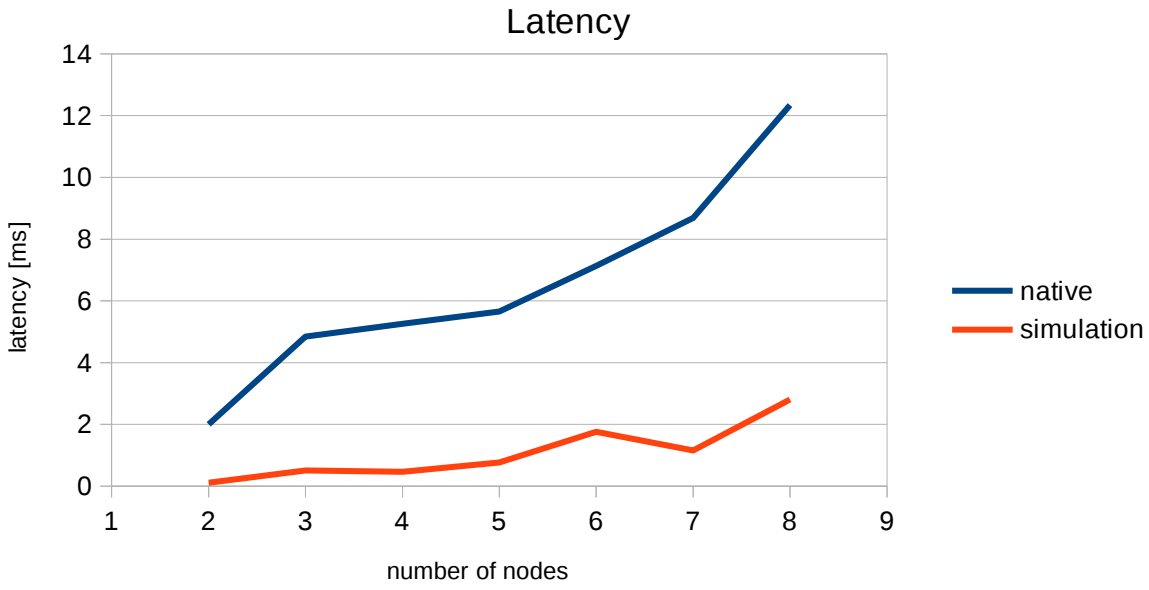
Barrier

Efficiency by number of hosts

Results:

number of nodes	native	simulation
2	2.000183	0.106443
3	4.843018	0.510336
4	5.260961	0.463259
5	5.658088	0.766366
6	7.133344	1.760902
7	8.687277	1.155371
8	12.342397	2.80749

Chart:



Conclusion: My implementation was much quicker. Results even got better trend but probably for thousands of nodes native MPI_Barrier performance would be better, another drawback of my implementation is its centrality.

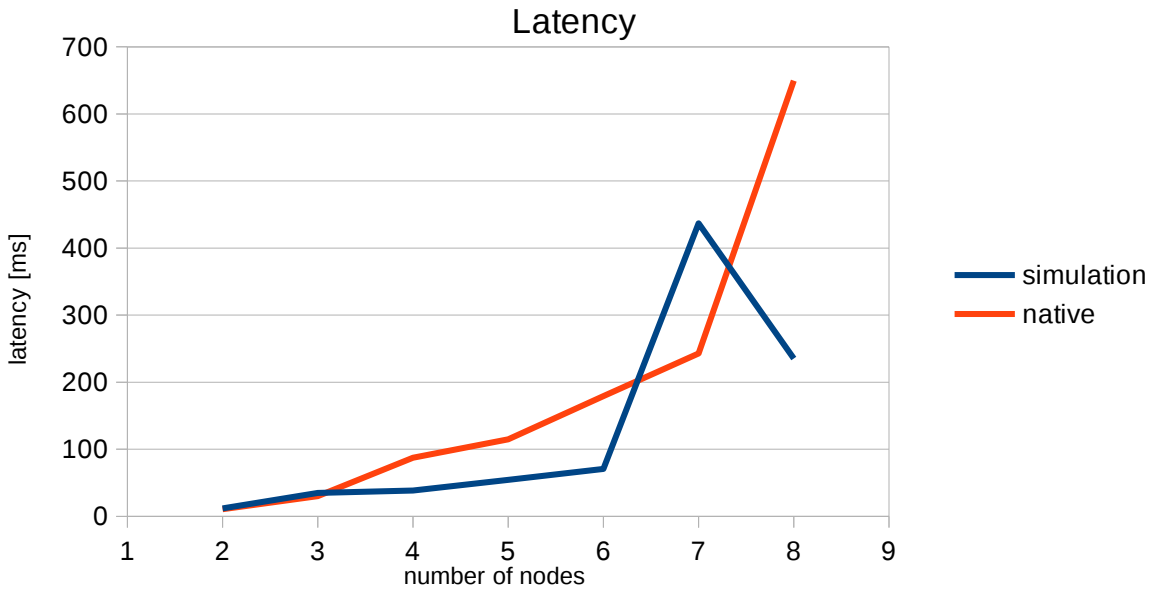
Allgather

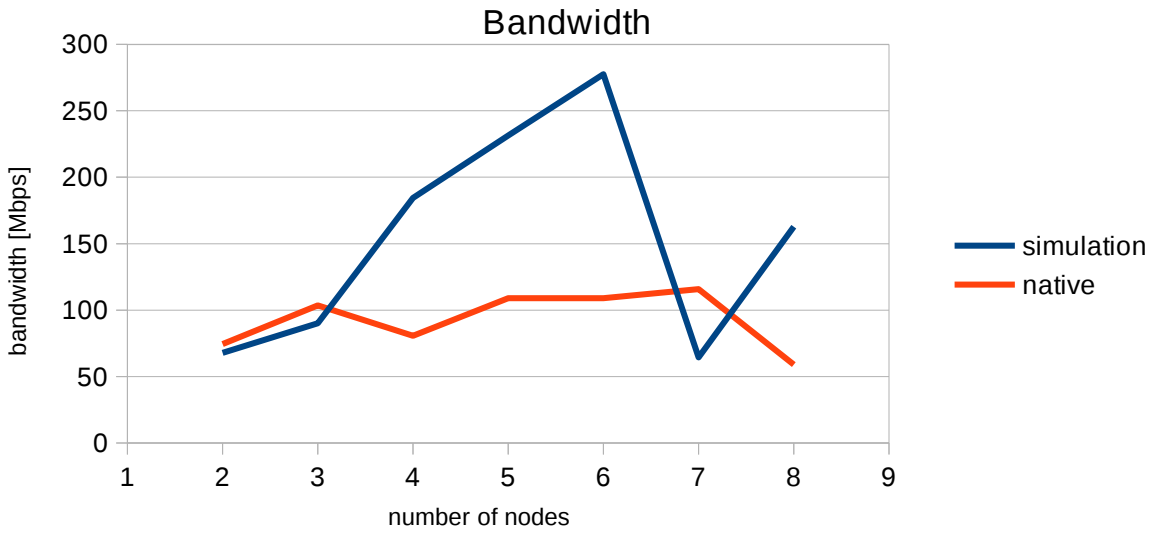
Efficiency by number of nodes

Constant chunk size = 100 kB

Results:

number of nodes	latency simulation	latency native	bandwidth simulation	bandwidth native
2	11.529803	10.482287	67.759178	74.530487
3	34.702991	30.147409	90.049873	103.657333
4	38.109612	87.081001	184.500702	80.74379
5	54.007103	114.74469	231.45105	108.937508
6	70.377205	179.2397	277.522369	108.967209
7	437.027008	242.702606	64.355293	115.882561
8	235.218185	649.513184	162.747833	58.93837





Conclusion: Results are similar to previous ones. Again we can see strange latency peak of native MPI_Allgather implementation for 8 nodes.

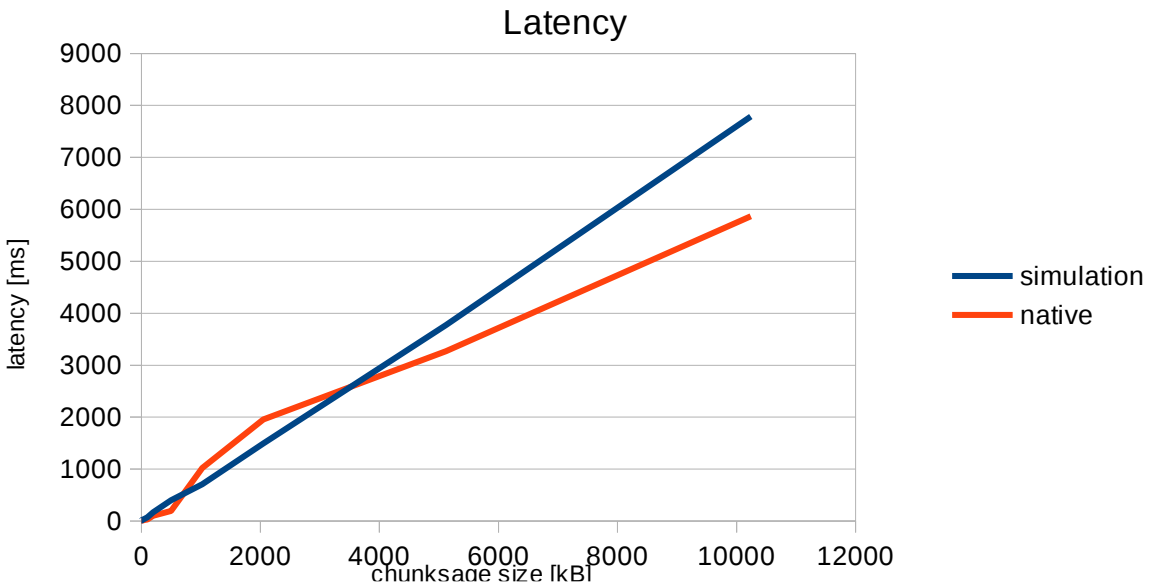
Efficiency by chunk size

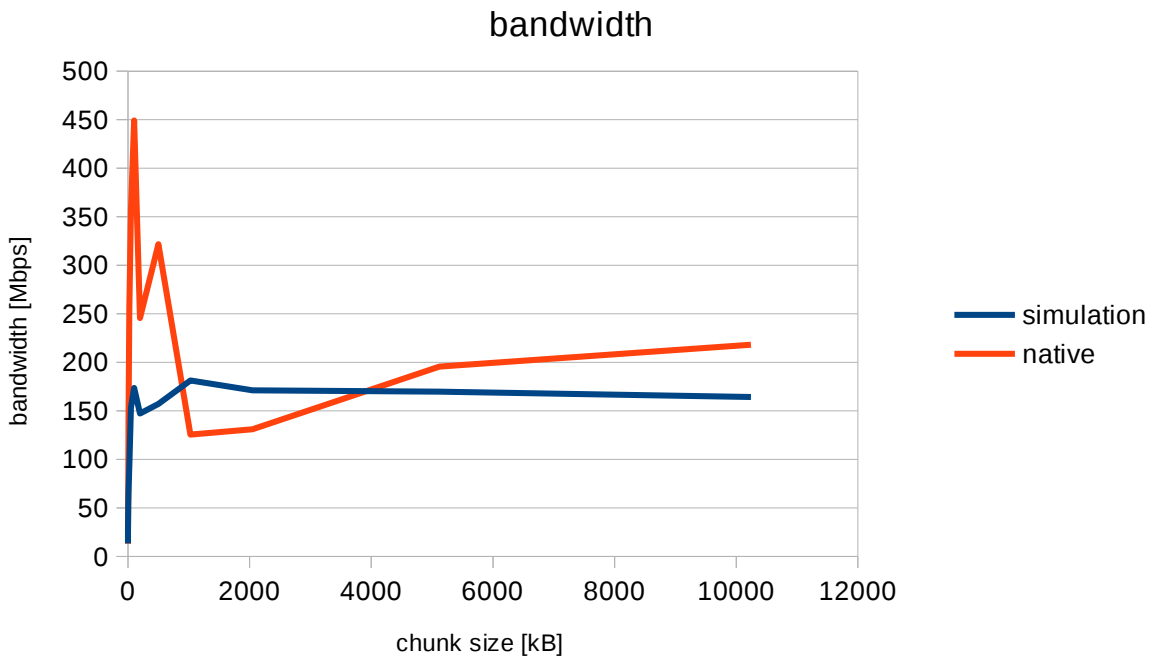
Constant number of nodes = 5

Results:

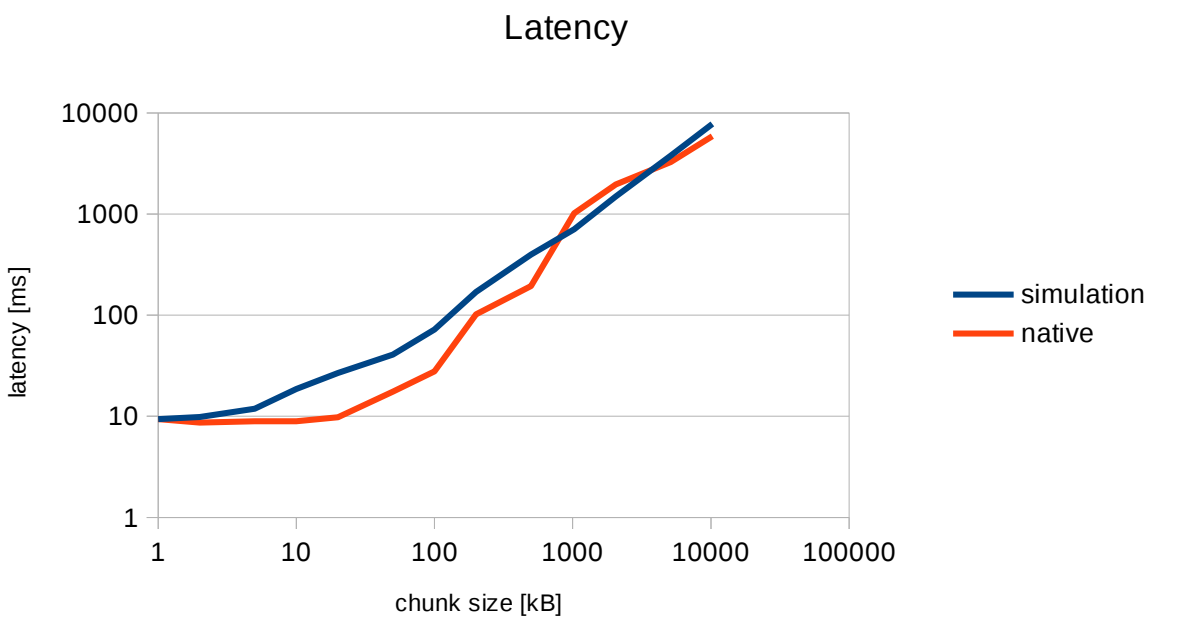
chunk size [kB]	latency [ms] simulation	latency [ms] native	bandwidth [Mbps] simulation	bandwidth [Mbps] native
1	9.375405	9.396005	13.332757	13.303527
2	9.831309	8.676887	25.428963	28.812178
5	11.92162	8.958912	52.425758	69.762939
10	18.677902	8.925796	66.924004	140.043533
20	26.761103	9.769201	93.419167	255.906281
50	40.758087	17.551899	153.343811	356.086823
100	72.007301	27.817202	173.593521	449.362244
200	169.711487	101.736191	147.308823	245.733597
500	397.578186	194.277908	157.201782	321.704102
1024	706.26239	1018.518311	181.235764	125.672752
2048	1494.720459	1954.041626	171.26947	131.010513
5120	3770.473633	3270.132568	169.739944	195.710724
10240	7784.272949	5863.815918	164.434113	218.287888

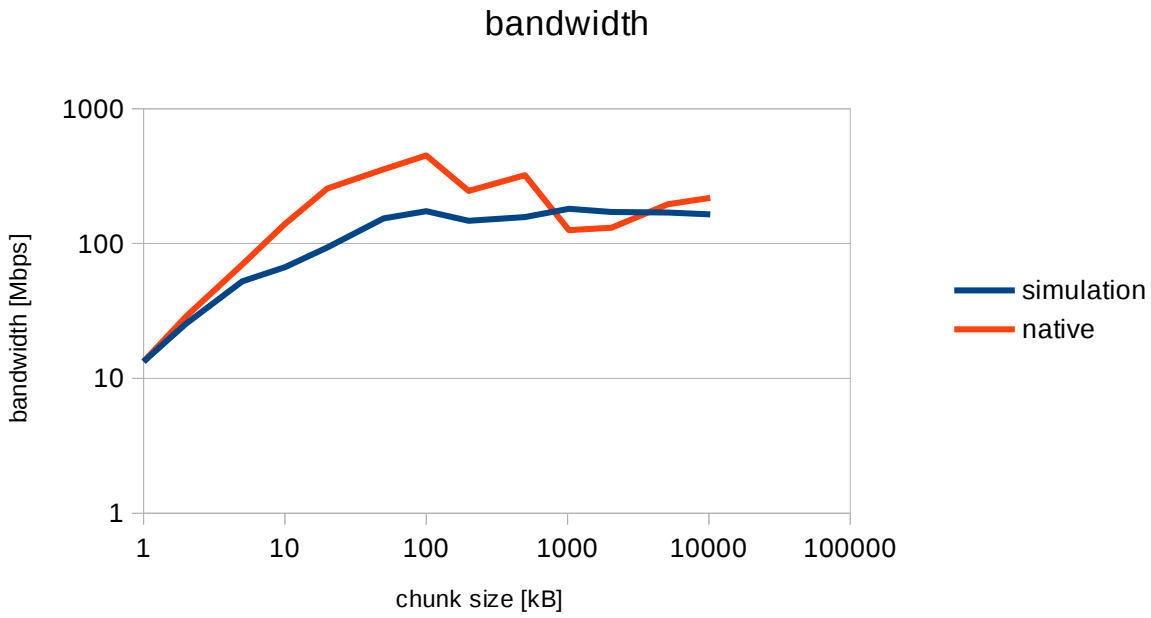
Charts





Same data logarithmic scale:





Conclusion:
This time native implementation is slightly better even for small number of nodes.