MPI1-Benchmark-Analysis

ℹ️ Synopsis

This is an in dept analysis and modeling of MPI-Collective calls on Fritz using the IMB-MPI1 benchmarks.

✅ Todo :Collectives

- [x] ~~Allgather~~
- [ ] Scatter
- [x] ~~Alltoall~~
- [ ] Scatterv
- [ ] Allgatherv
- [ ] Gatherv
- [ ] Bcast
- [ ] Gather
- [ ] Reduce_Scatter

- [ ] Allreduce
- [ ] Reduce

🗺️ Fritz's Network Topology

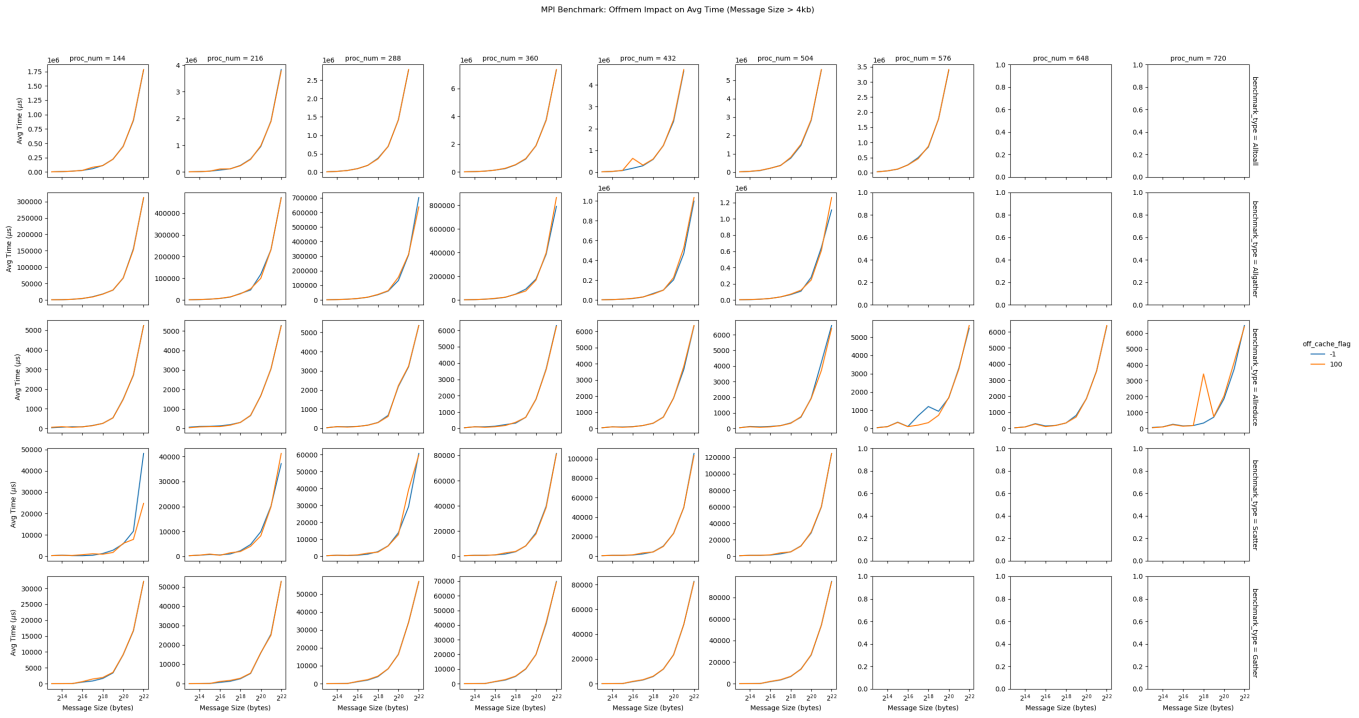
Fritz uses blocking HDR100 Infiniband with up to 100 GBit/s bandwidth per link and direction. There are islands with 64 nodes (i.e. 4.608 cores). The blocking factor between islands is 1:4. Fritz uses unmanaged 40 port Mellanox HDR switches.
This means that for any island each node can exchange data with any other node at full 12.5 GBps (light-speed).

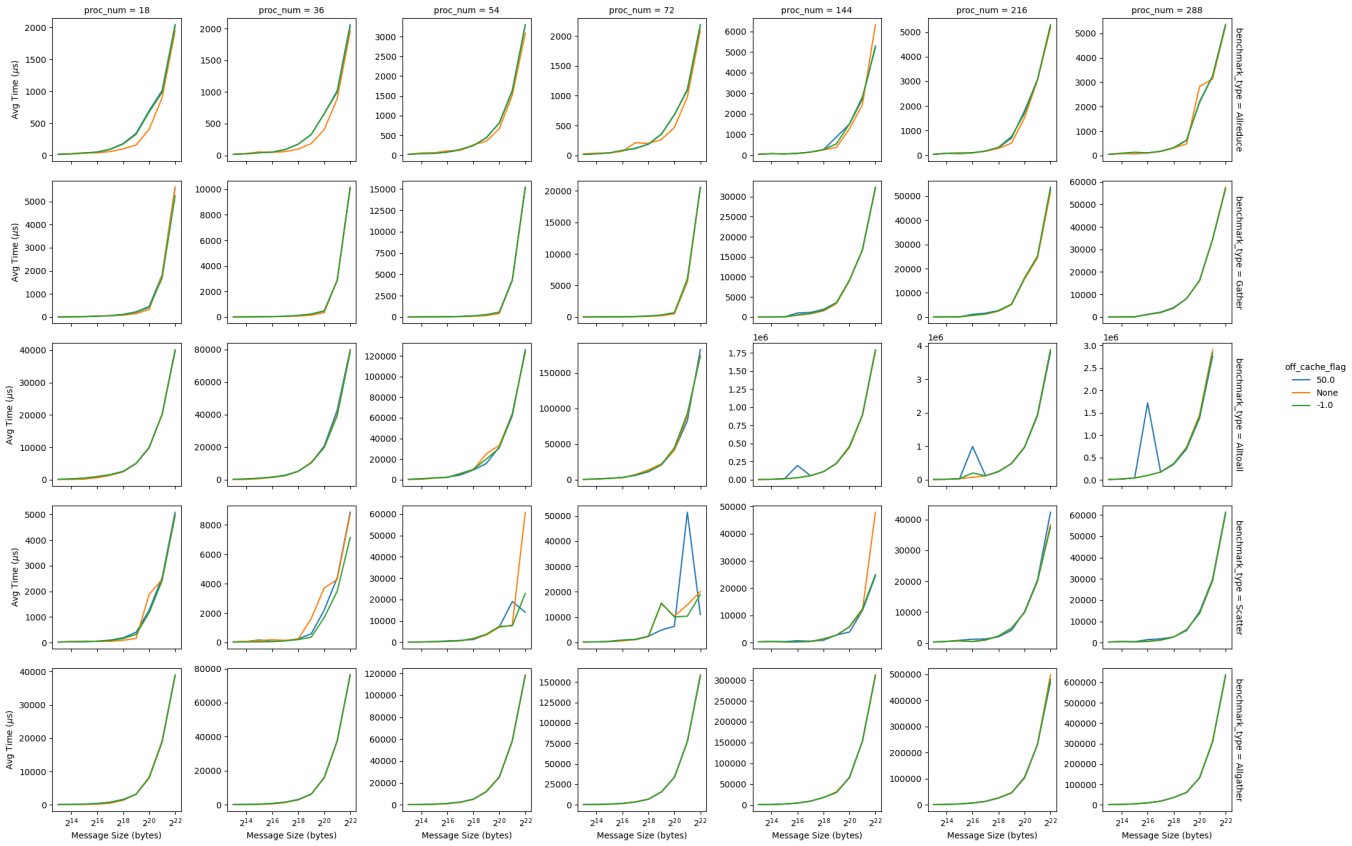
📊Analysis

ℹ️ About the -off_cache flag

The IMB-MP1 benchmarks allow the user to supply a flag that enables mitigation of caching effects. If it has value "-1" the details of such mitigation are left to the implementations, otherwise it shall be a positive integer. If such a positive integer is supplied message buffers are not reused but message src are multiple and shifted by x MB were x is the integer passed.

Having a look at the effects of this flag yield mixed results:





It is clear that the flag has some effect on the collected data but none of the flags used seems to offer a consistent lower bound for the measurement. However values of "-1" and "100" seemed to behave more coherently for different node numbers and benchmarks.
In the following the value of this flag used to collect the data will be always reported.

🌐 Alltoall

The benchmark for the MPI_Alltoall function. In the case of n number of processes, every process inputs $x \cdot n$ bytes (x for each process) and receives $x \cdot n$ bytes (x from each process).

⚛️ Modelling (Multi-node)

Lets assume k processes per node and m nodes taking part into the communication at fixed message size x.
For comparison we take the second slowest data-path in Fritz → Socket-Socket communication, in particular we make the simplification that $k_{\text{socket}} = k/2$ as it reflects Fritz's topology.

Unit type	Direction	Total	Over Socket	Over Network
process	SEND	$x \cdot m \cdot k$	$x \cdot k/2$	$x \cdot (m-1) \cdot k$
process	RECV	$x \cdot m \cdot k$	$x \cdot k/2$	$x \cdot (m-1) \cdot k$
node	SEND	$x \cdot m \cdot k^2$	$x \cdot k^2/2$	$x \cdot (m-1) \cdot k^2$
node	RECV	$x \cdot m \cdot k^2$	$x \cdot k^2/2$	$x \cdot (m-1) \cdot k^2$

From the table is clear that we most data moving over the slowest data-path. Furthermore the $\text{data}_{\text{in}} = x \cdot (m-1) \cdot k^2 = \text{data}_{\text{out}}$, so we are limited by the one-way speed of light of the network interconnect (12.5 GBps).

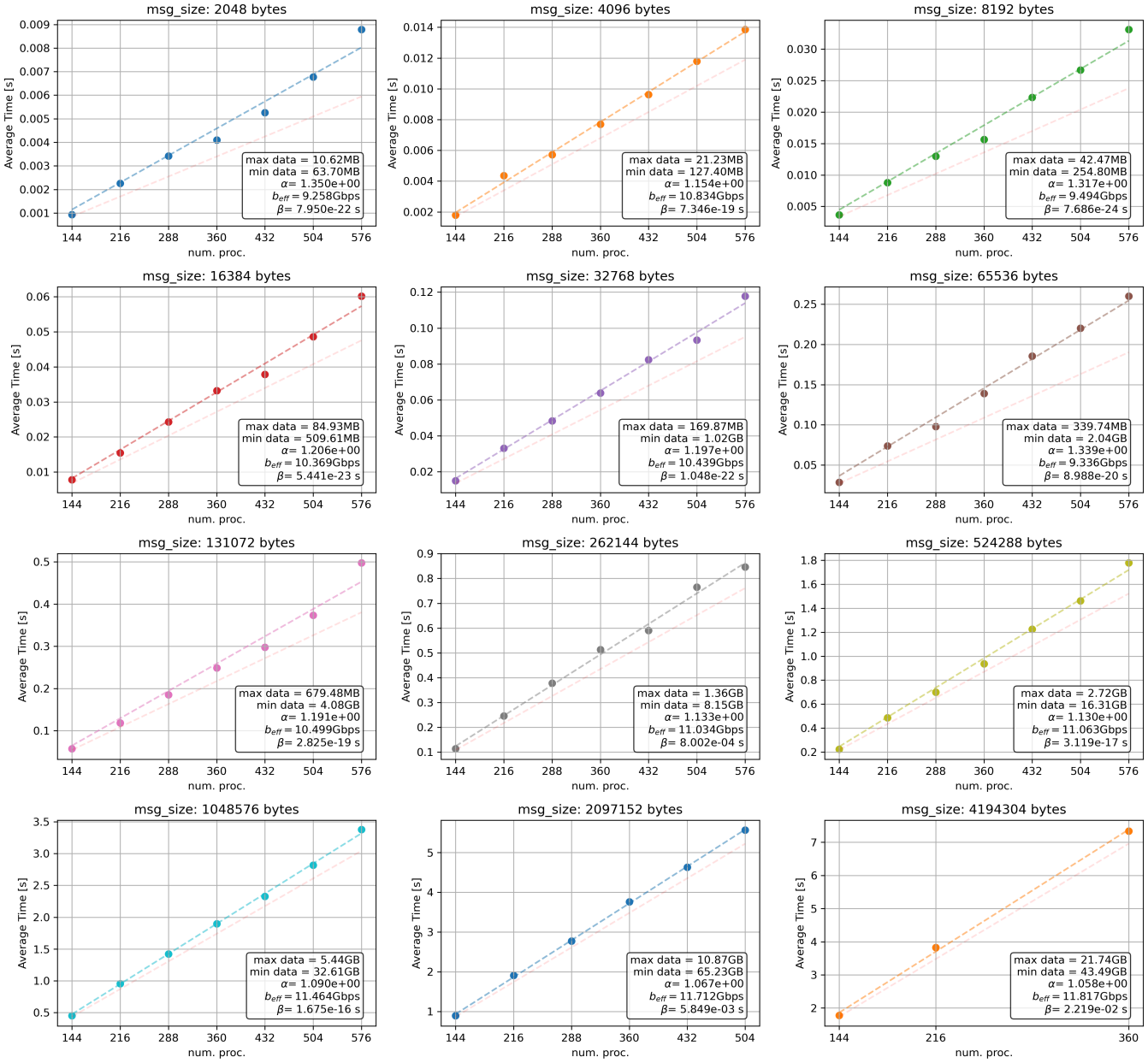
One can expect the call time to scale as follows:

$$t_{\text{avg}}(x, k, m) = \frac{\alpha x (m-1) k^2}{b} + \beta$$

Where b is the connection speed-of-light bandwidth, and α, β are interpolation parameters.

- α represents a "slowdown" factor to the bandwidth ($1/\alpha \leq 1$), in average.
- β represents cumulative startup latency of the communication, in average.

Results



The model successfully fits the the collected data with rational fitting parameters. The faint red line symbolizes the speed-of-light for the case ($\alpha = 1, \beta = 0$). As one can see from the plot the calls are operating very close to speed of light!

msg_size_bytes	alpha	beta $[s]$	inv_alpha
2048	1.350220	7.950002e-22	0.740620
4096	1.153722	7.345616e-19	0.866760
8192	1.316655	7.685776e-24	0.759501
16384	1.205501	5.440666e-23	0.829531
32768	1.197387	1.048284e-22	0.835152
65536	1.338850	8.987548e-20	0.746910
131072	1.190638	2.825286e-19	0.839886
262144	1.132900	8.001526e-04	0.882691
524288	1.129881	3.119047e-17	0.885049
1048576	1.090405	1.674706e-16	0.917091
2097152	1.067311	5.849285e-03	0.936934
4194304	1.057813	2.218755e-02	0.945346

The bandwidth usage increases as expected with message size as well as the startup latency.

🌐 Allgather

Every process inputs x bytes and receives the gathered $x \cdot n$ bytes, where n is the number of processes.

⚛️ Modelling (Multi-node)

Lets assume k processes per node and m nodes taking part into the communication at fixed message size x.
Again, for comparison we take the second slowest data-path in Fritz → Socket-Socket communication, with $k_{\text{socket}} = k/2$.

Unit type	Direction	Total	Over Socket	Over Network	
process	SEND	x	x	x	
process	RECV	$x \cdot m \cdot k$	$x \cdot k/2$	$x \cdot (m-1) \cdot k$	
node	SEND	$x \cdot k$	$x \cdot k/2$	$x \cdot k$	
node	RECV	$x \cdot m \cdot k$	$x \cdot k/2$	$x \cdot (m-1) \cdot k$	

Most data moving over the slowest data-path. Furthermore, since $\text{data}_{\text{in}} = x \cdot (m-1) \cdot k^2 > x \cdot k = \text{data}_{\text{out}}$, we are limited by the one-way speed of light of the network interconnect (12.5 GBps) on the incoming pathway.

As before, we can expect the call time to scale as follows:

$$t_{\text{avg}}(x, k, m) = \frac{\alpha x (m-1) k}{b} + \beta$$

Where b is the connection speed-of-light bandwidth, and α, β are interpolation parameters.

- α represents a "slowdown" factor to the bandwidth ($1/\alpha \leq 1$), in average.
- β represents cumulative startup latency of the communication, in average.

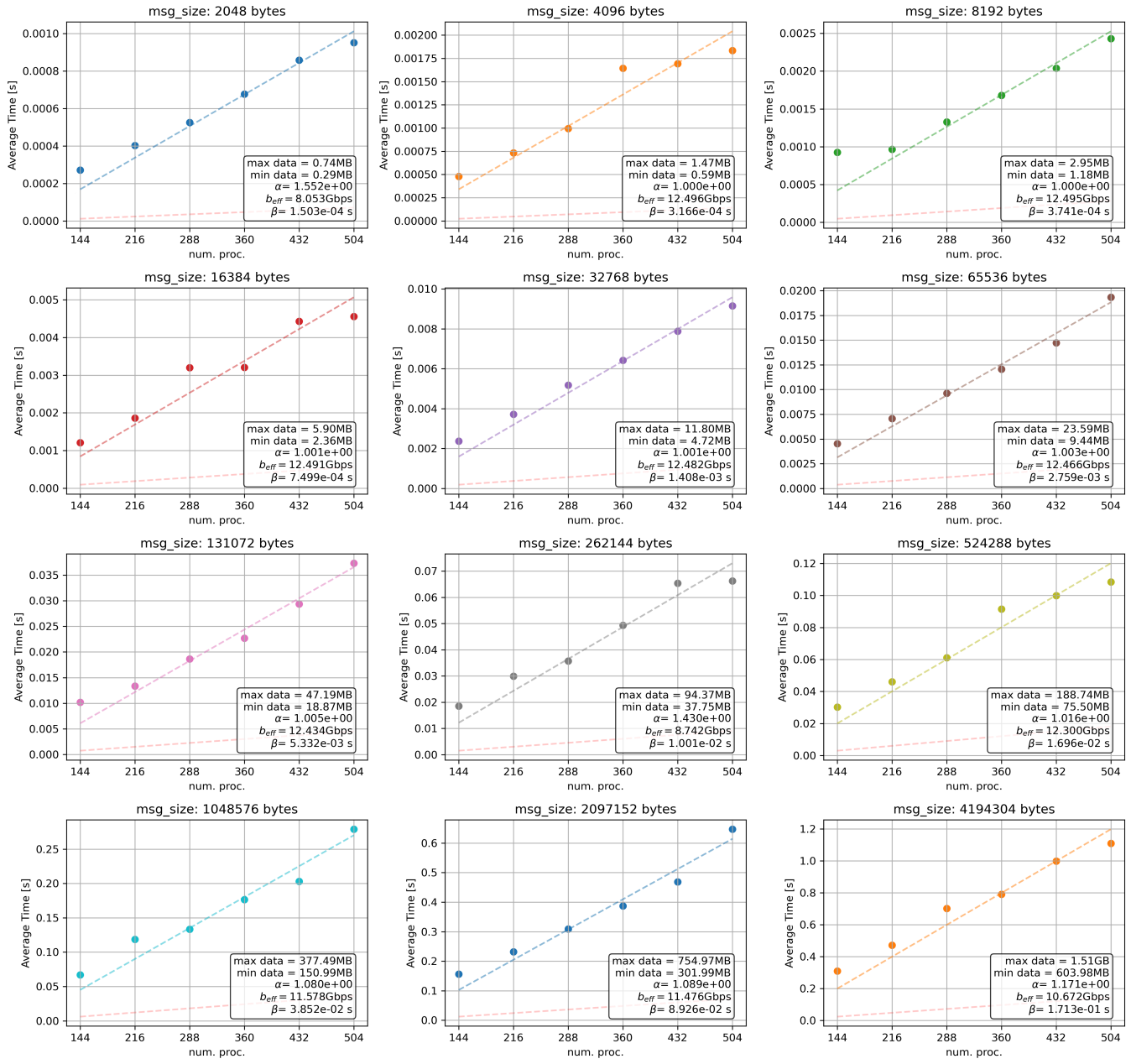
This model was however incapable of fitting the data. It seems that *Allgather* may utilize a ring exchange pattern under the hood.
To model this we assume that each node outputs $x \cdot k$ data per ring iteration, for a total of $m - 1$ iteration, and a setup latency, due to synchronization, applies at every iteration.

The modelling equation changes to

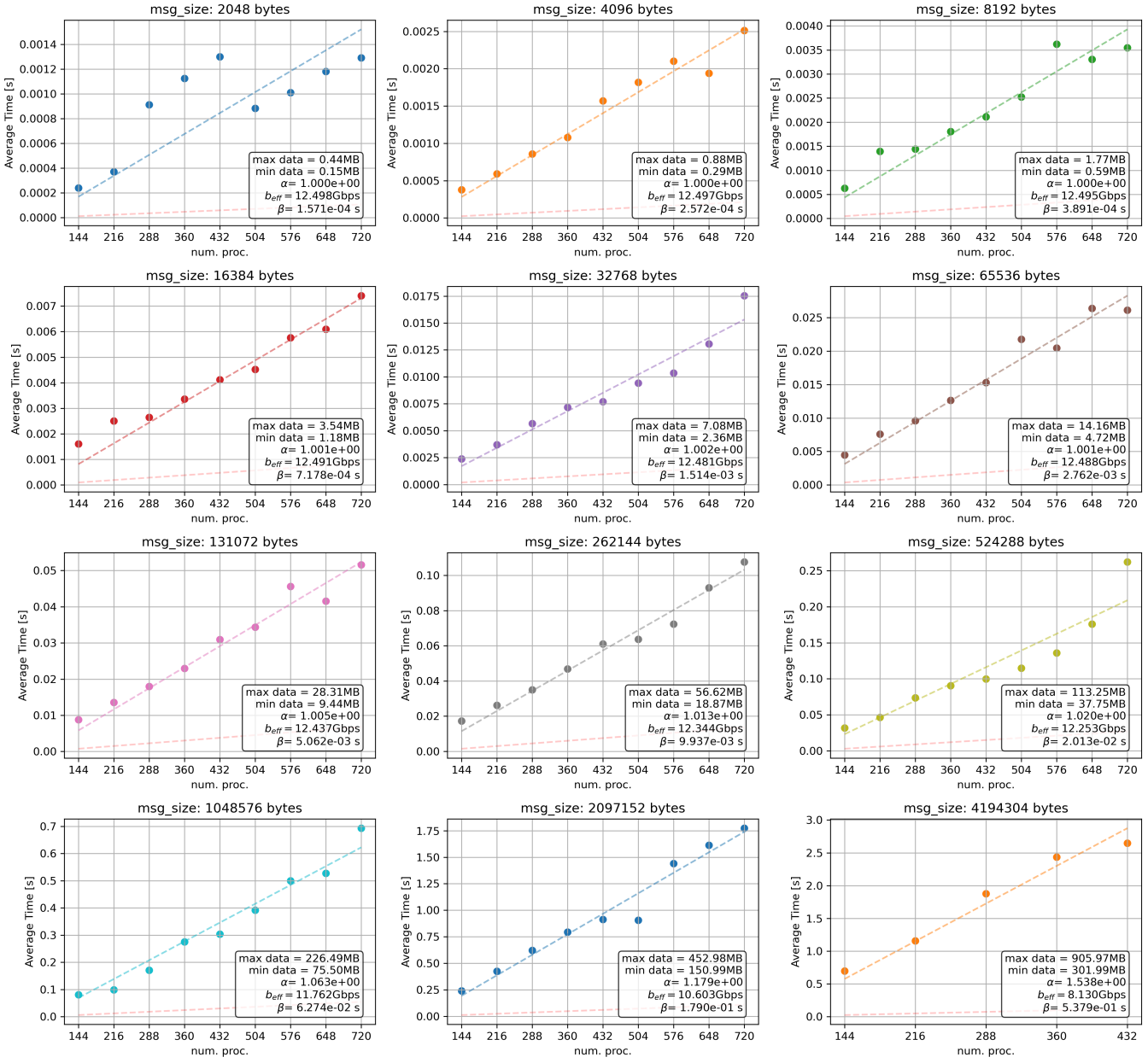
$$t_{\text{avg}}(x, k, m) = (m-1)\left(\frac{\alpha x k}{b} + \beta\right)$$

Results

In the following plots we will observe that the model gives a satisfying trend-line for the observed data but does not perfectly fit our observations. The reason for this is that the *Intel MPI* implementation is not guaranteed to use a ring pattern under the hood, some of the alternatives scale non linearly in the number of nodes. This can however be fixed with some flags, the experiment should be repeated for different communication patterns.



Running the benchmark again for slightly different parameter marks the utilization of different communication patters better (particularly in the last 2 rows):



At first glance this could be attributed to a *Recursive Doubling* pattern, but further investigation is required.