**Performance Analysis**

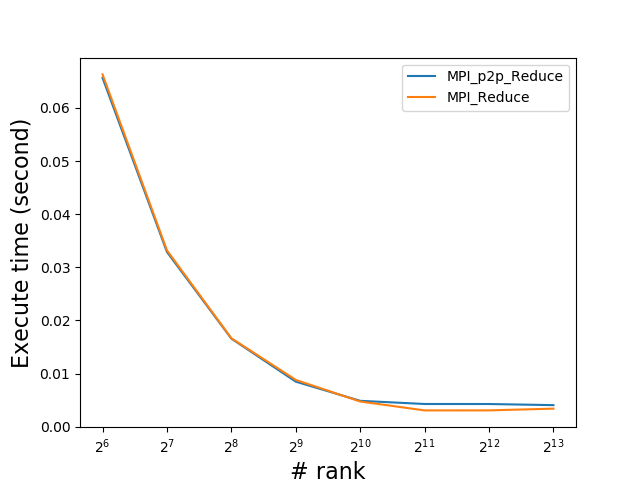


Figure 1 Graph of Runtime Under Different Rank

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Rank | 64 | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 |
| MPI\_p2p\_Reduce (s) | 0.065659 | 0.032880 | 0.016572 | 0.008467 | 0.004860 | 0.004263 | 0.004258 | 0.004049 |
| MPI\_reduce (s) | 0.066313 | 0.033183 | 0.016640 | 0.008796 | 0.004730 | 0.003057 | 0.003061 | 0.003386 |

Figure Chart of Runtime

Note: Time of MPI\_Reduce () was measured from calculating local sum to the completion of MPI\_Reduce. Or I think that it would be unfair for MPI\_p2p\_Reduce because it needs to calculate the local sum inside the function, which would cost a lot of I/O time.

**Analysis**

First, in a small rank, both methods have almost the same execute time. This is because that since I/O time, when calculating the local sum of one rank would consume a lot of time, is much bigger than the time spending on CPU. Therefore, the speedup is almost 100% when number of ranks is doubled. There This is also because that both methods use the tree structure (<https://github.com/open-mpi/ompi/blob/master/ompi/mca/coll/base/coll_base_reduce.c>) to achieve high percentage of parallel part.

Second, when number of ranks becomes extremely large, the default MPI method (MPI\_reduce) cost less time. This might be because that the number of synchronizations needed by MPI\_reduce is less than MPI\_p2p\_Reduce, or MPI\_reduce is optimized for the computing system to achieve higher performance. Also, since most of the local array in each rank is almost stored in the cache, the speedup by I/O part is no longer a factor.