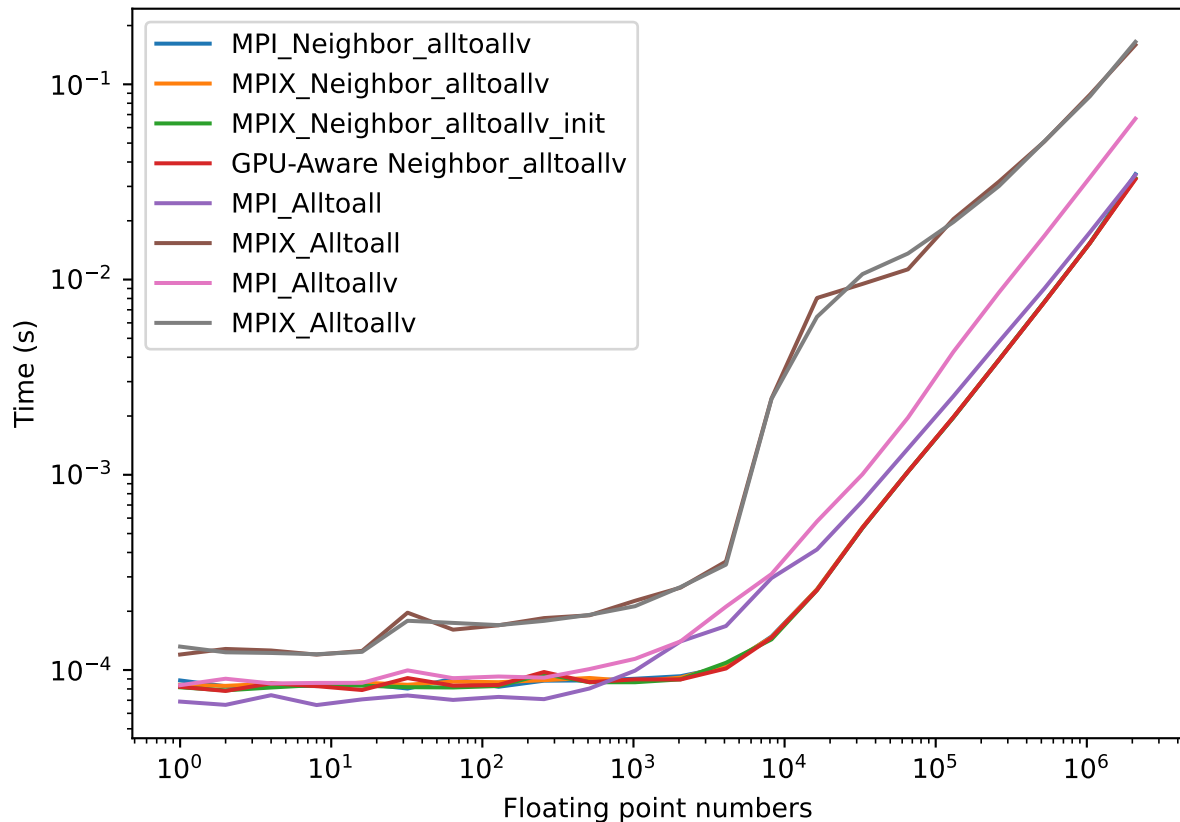
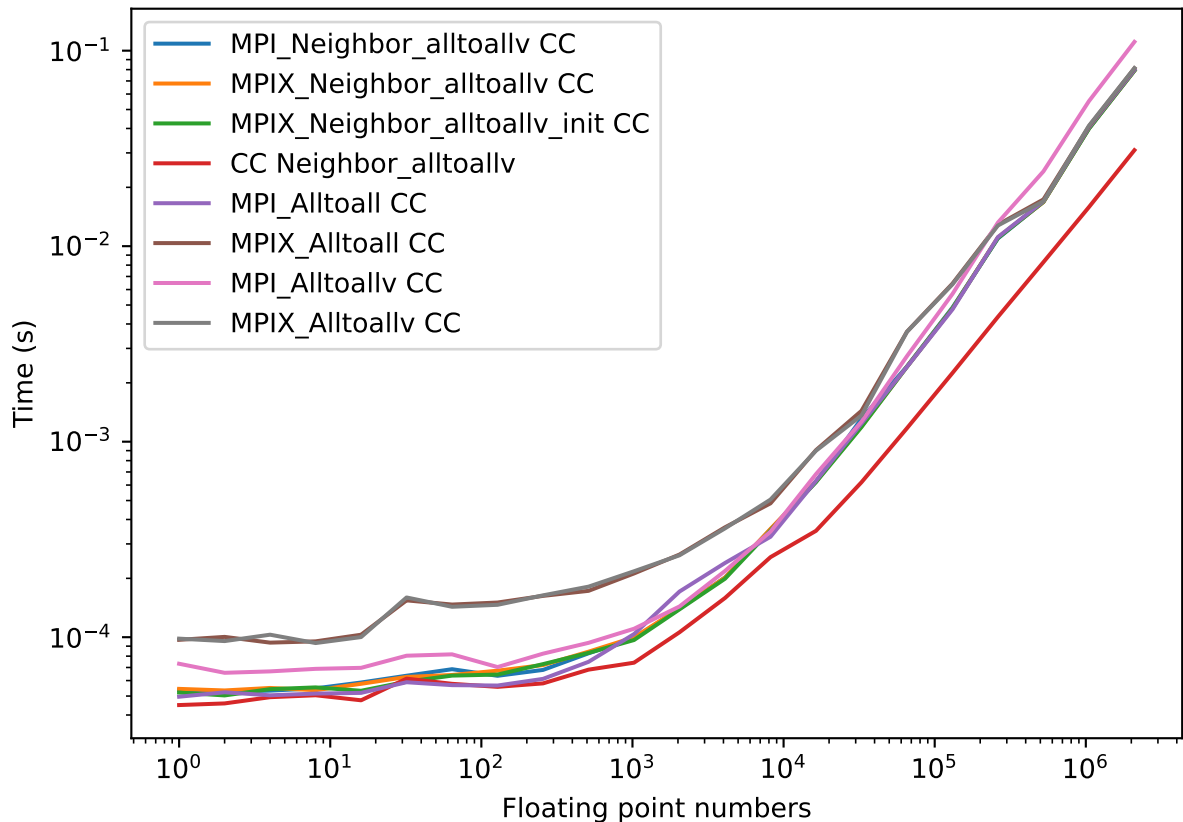


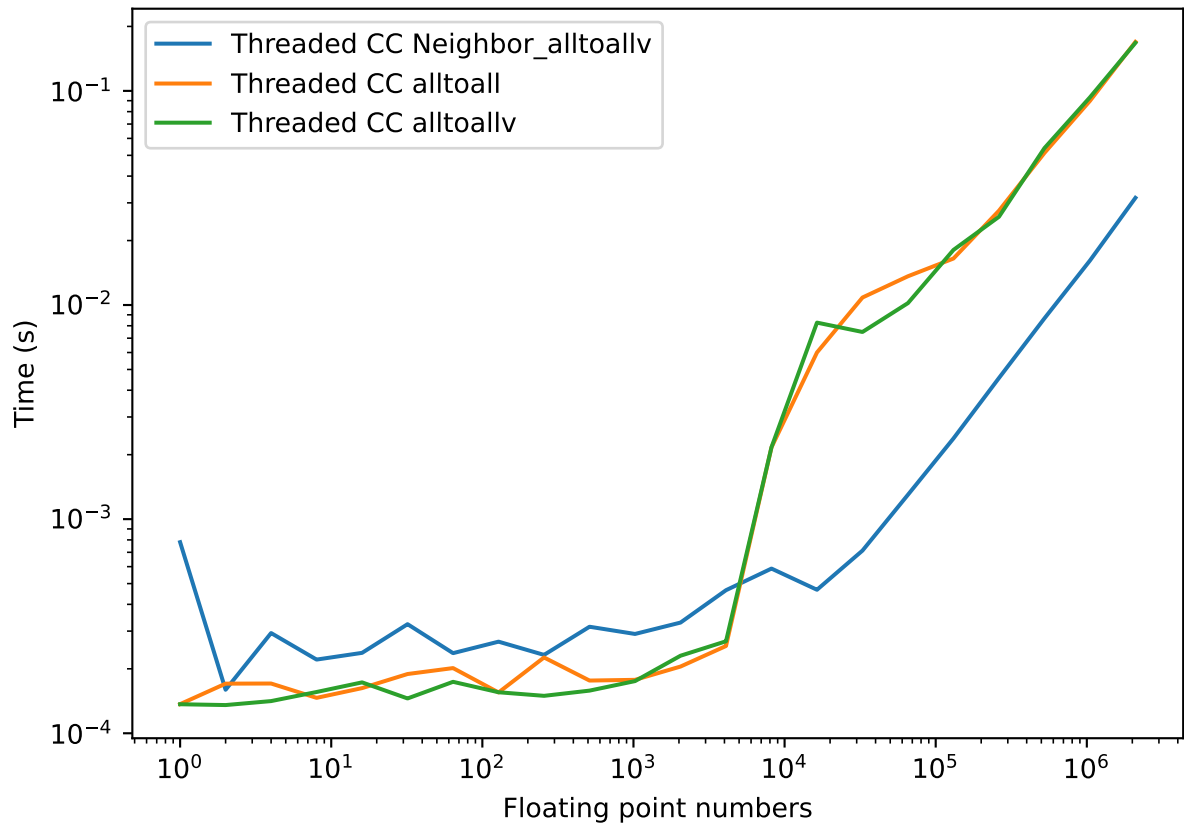
GPU Aware



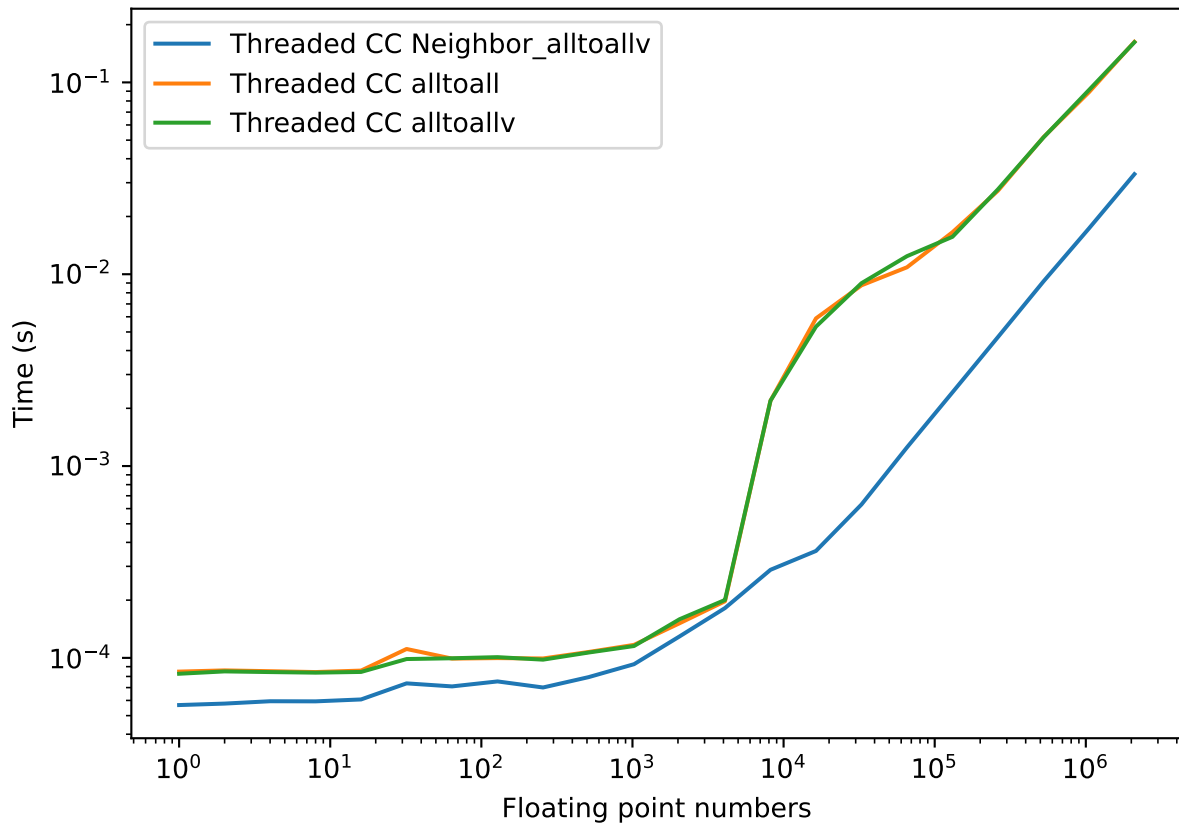
Copy-to-CPU



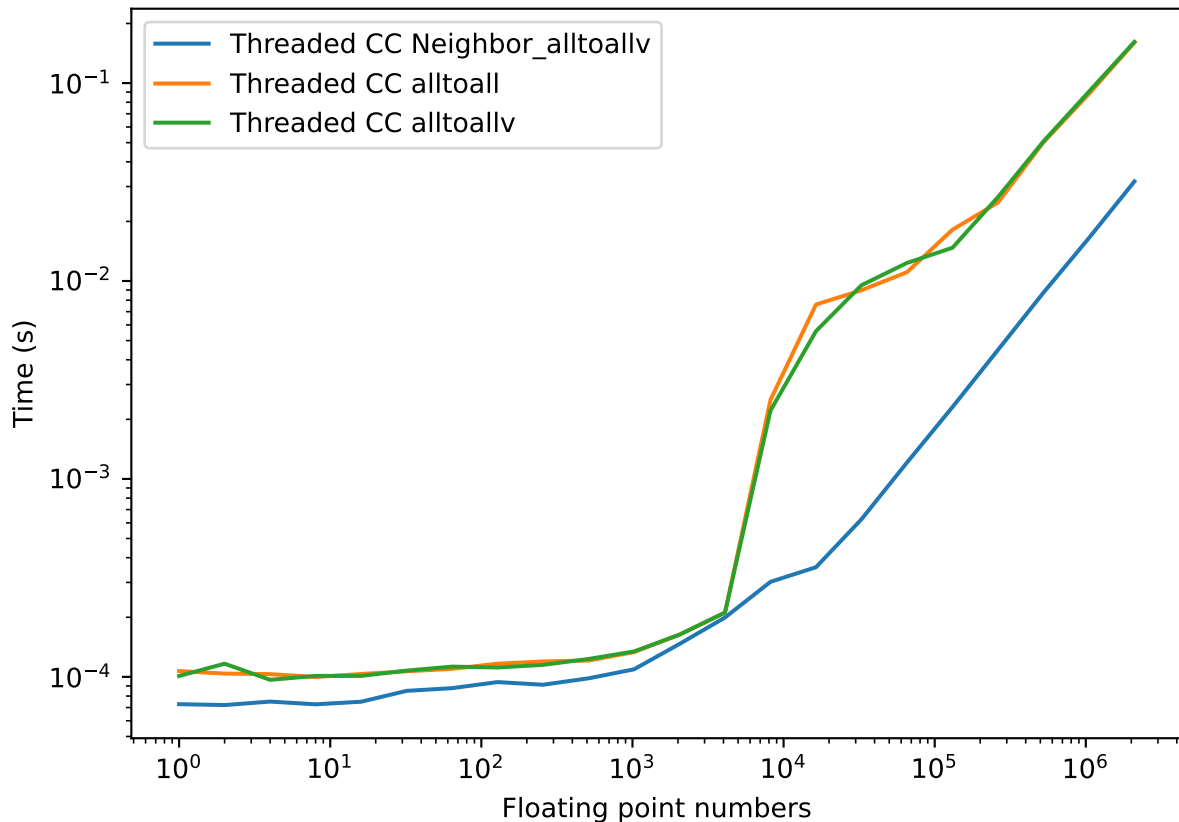
Threaded: 32



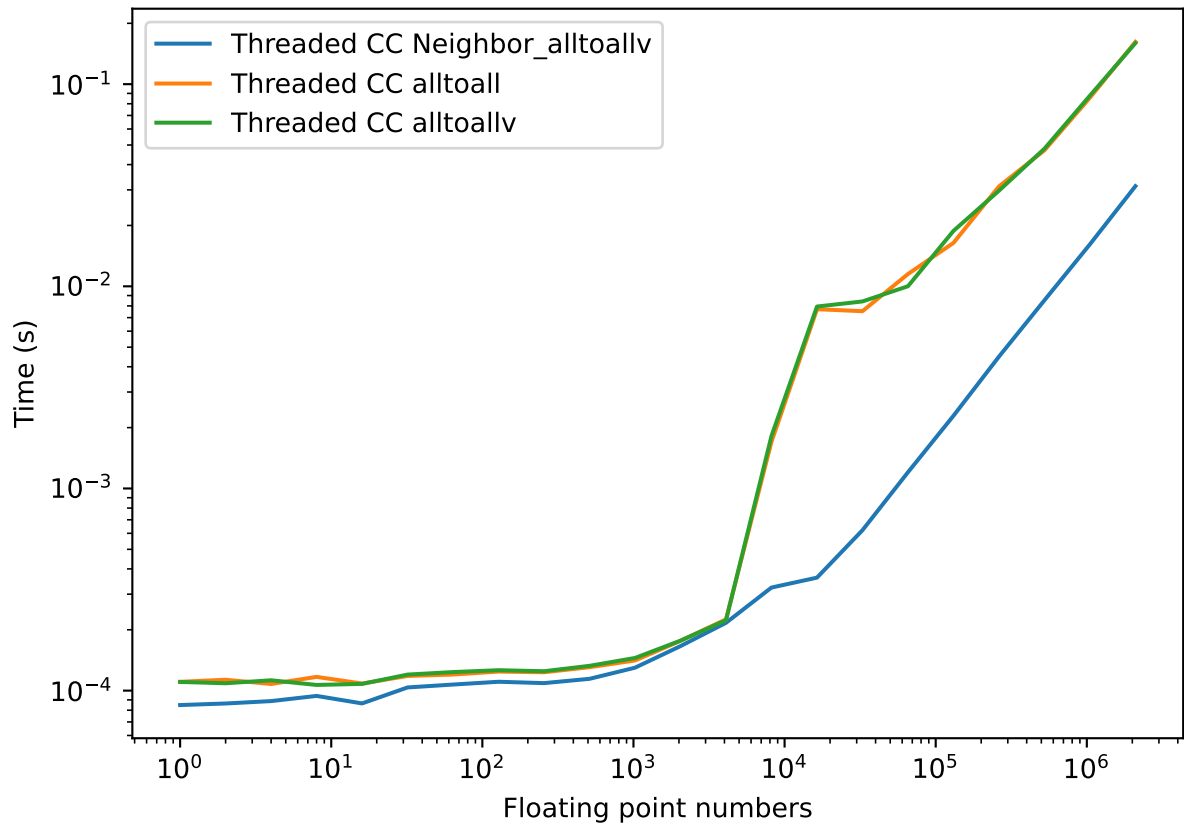
Threaded: 2



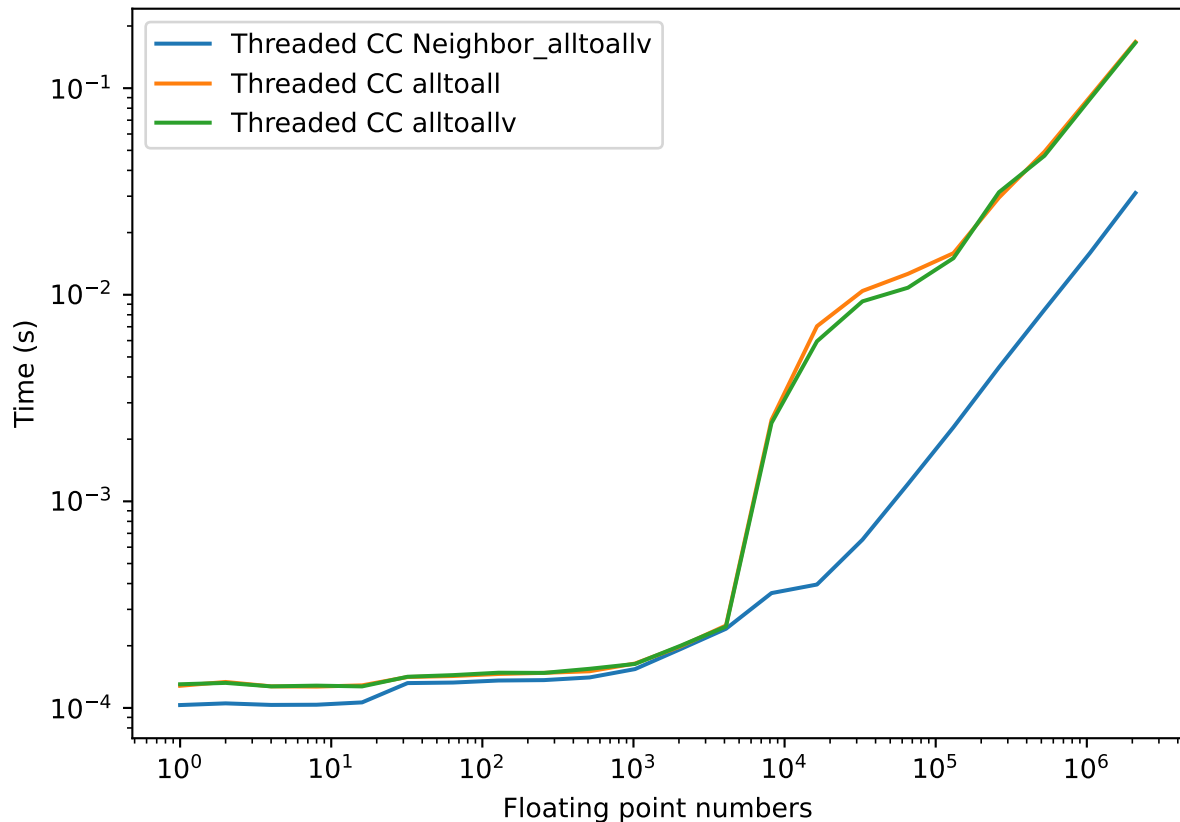
Threaded: 4



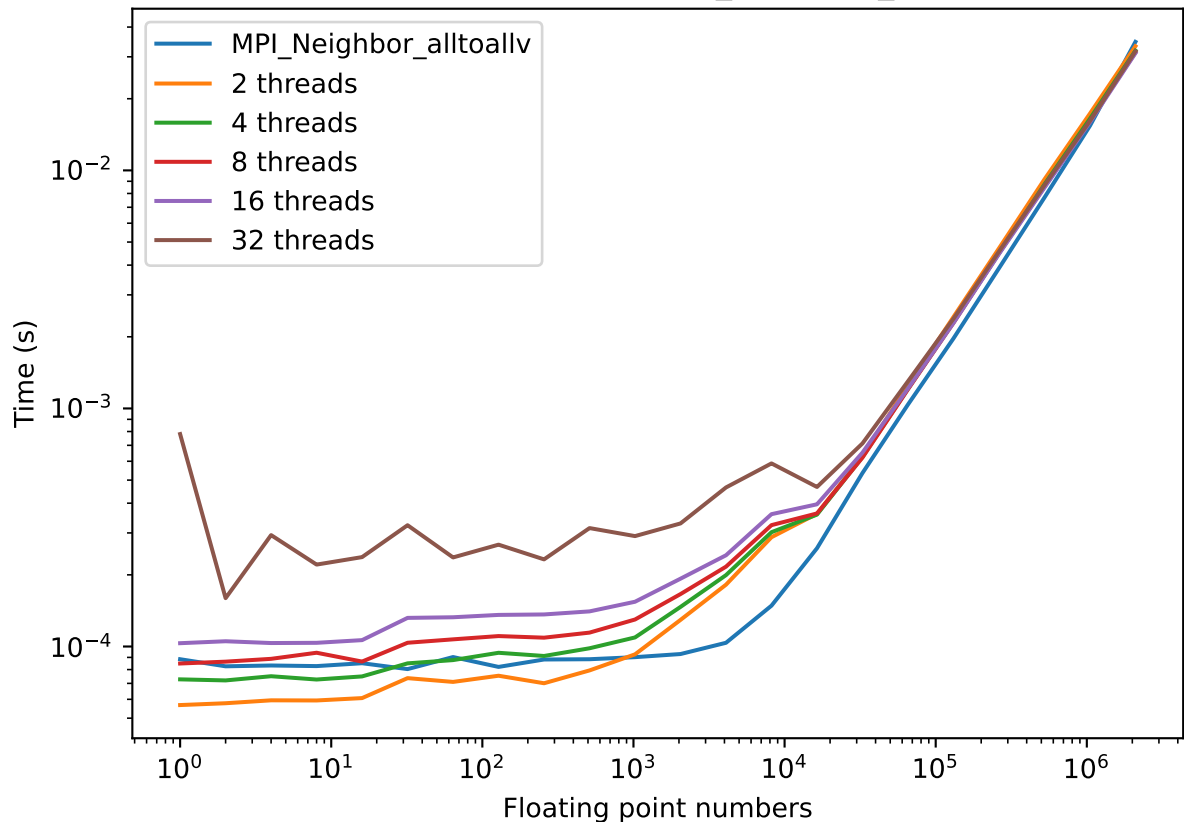
Threaded: 8



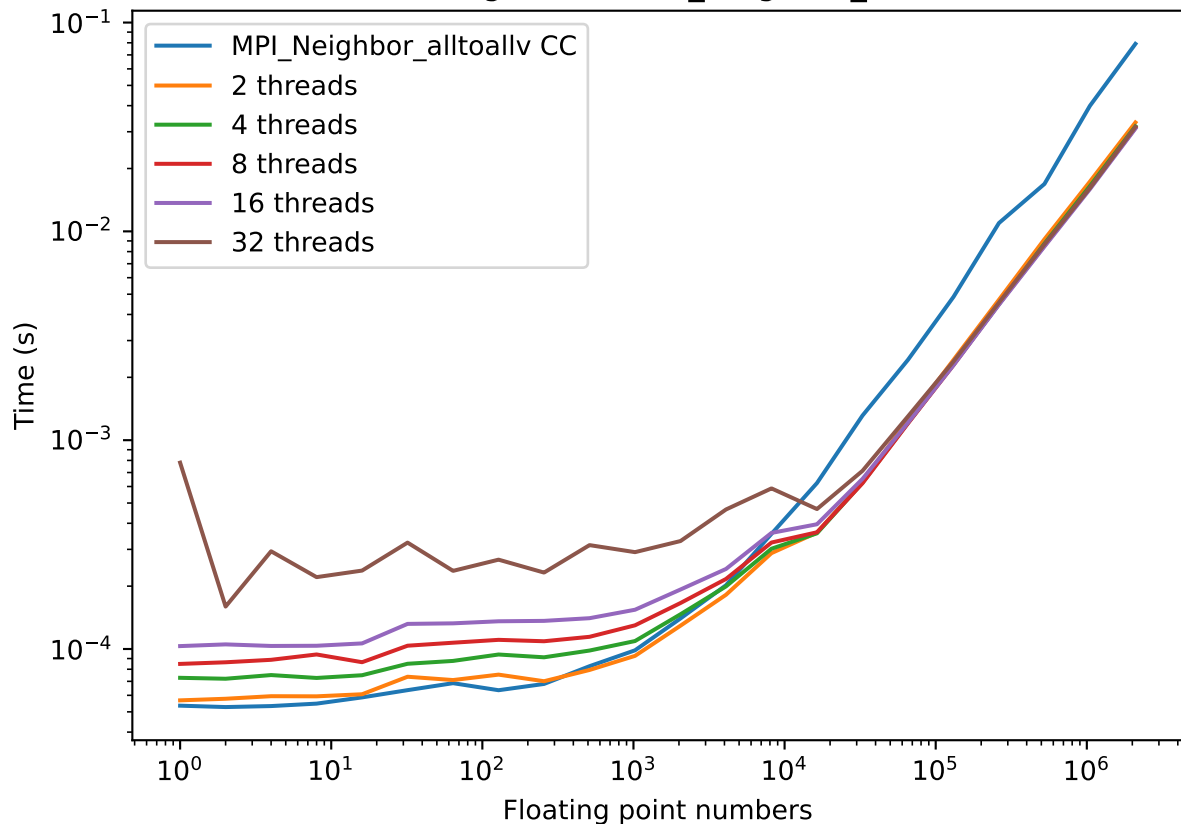
Threaded: 16



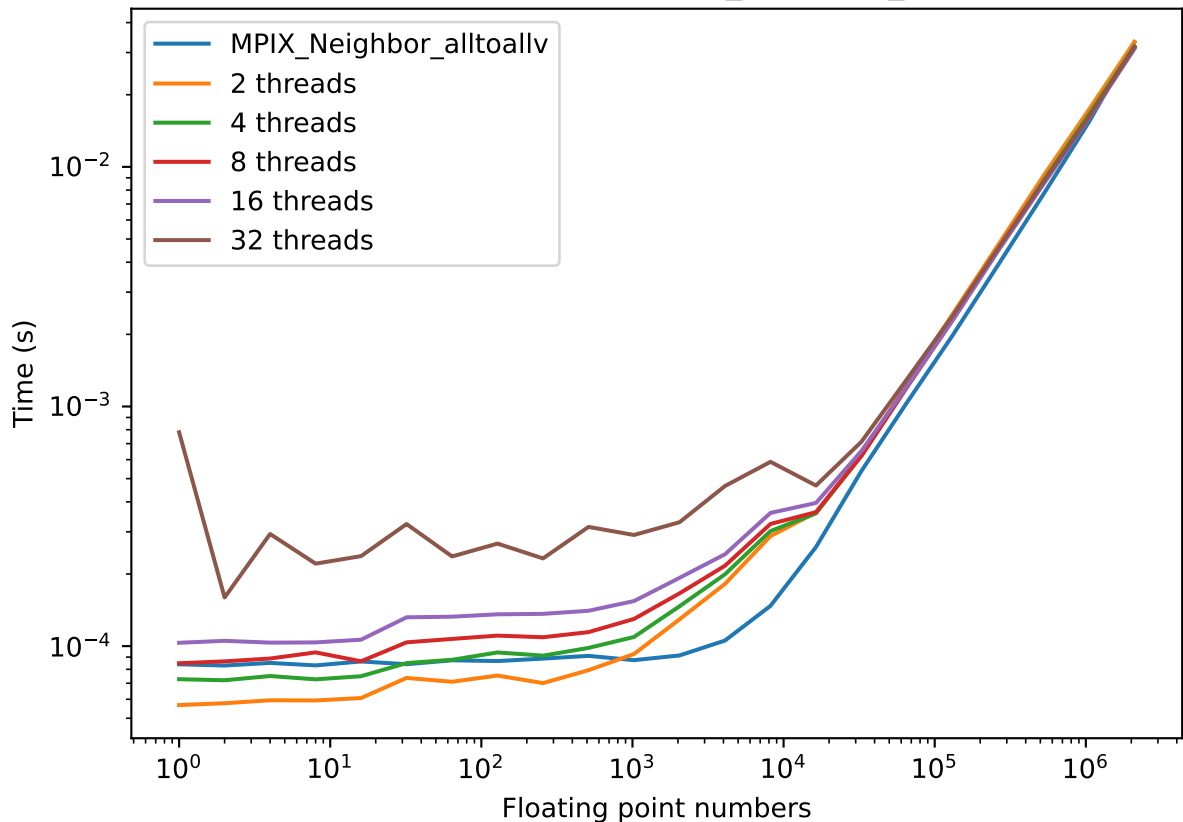
Threaded Neighbor vs MPI_Neighbor_alltoallv



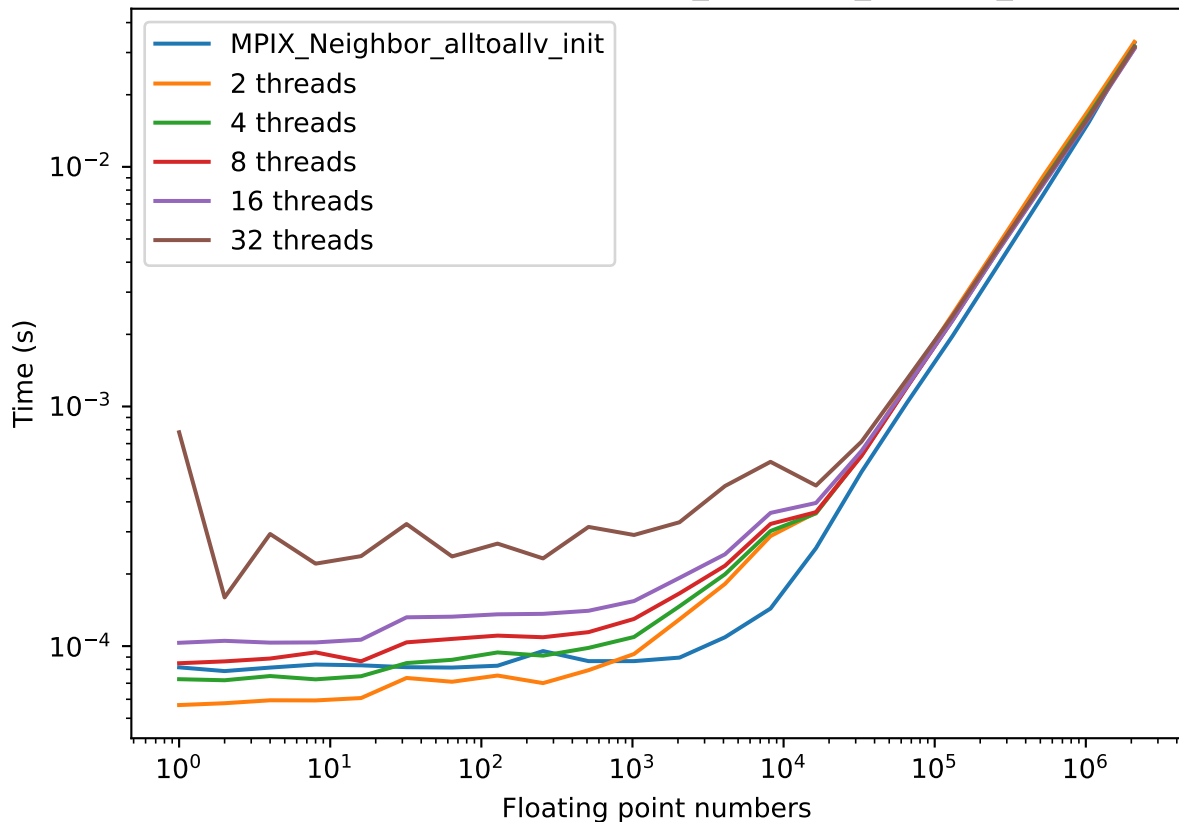
Threaded Neighbor vs MPI_Neighbor_alltoallv CC



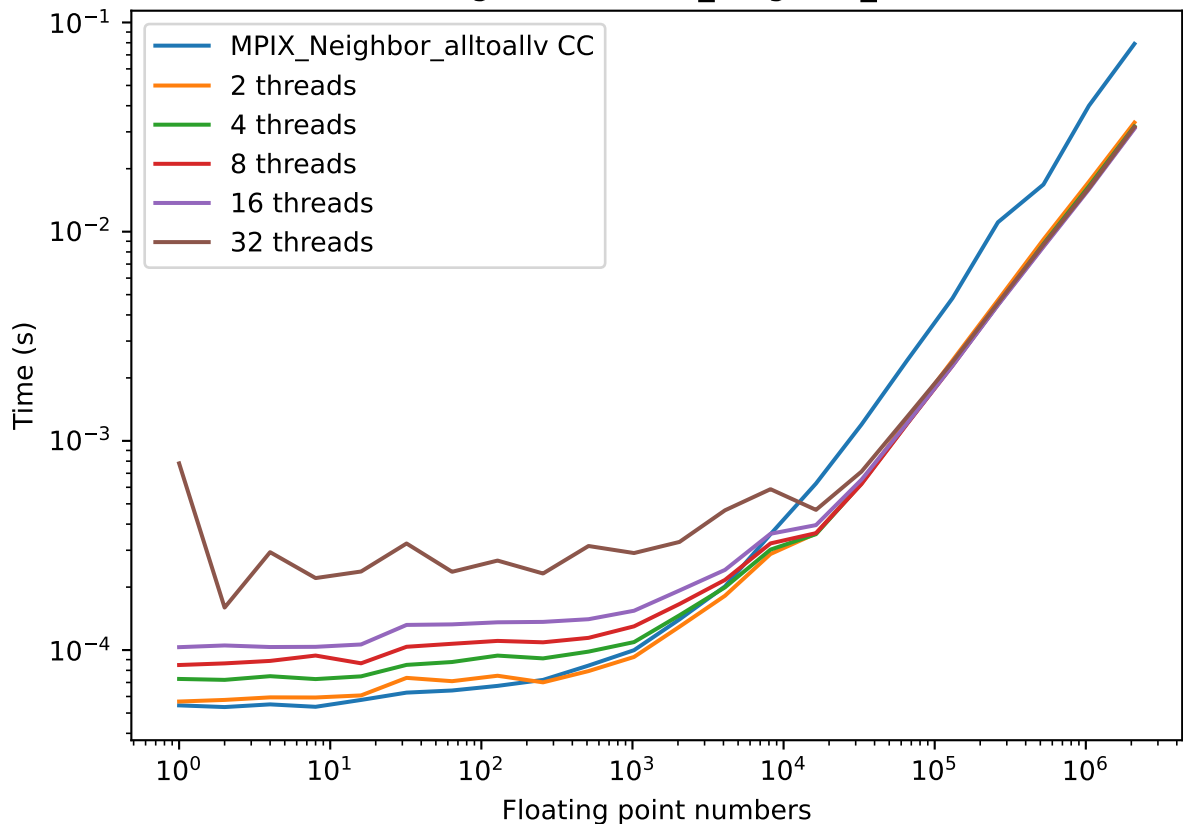
Threaded Neighbor vs MPIX_Neighbor_alltoallv



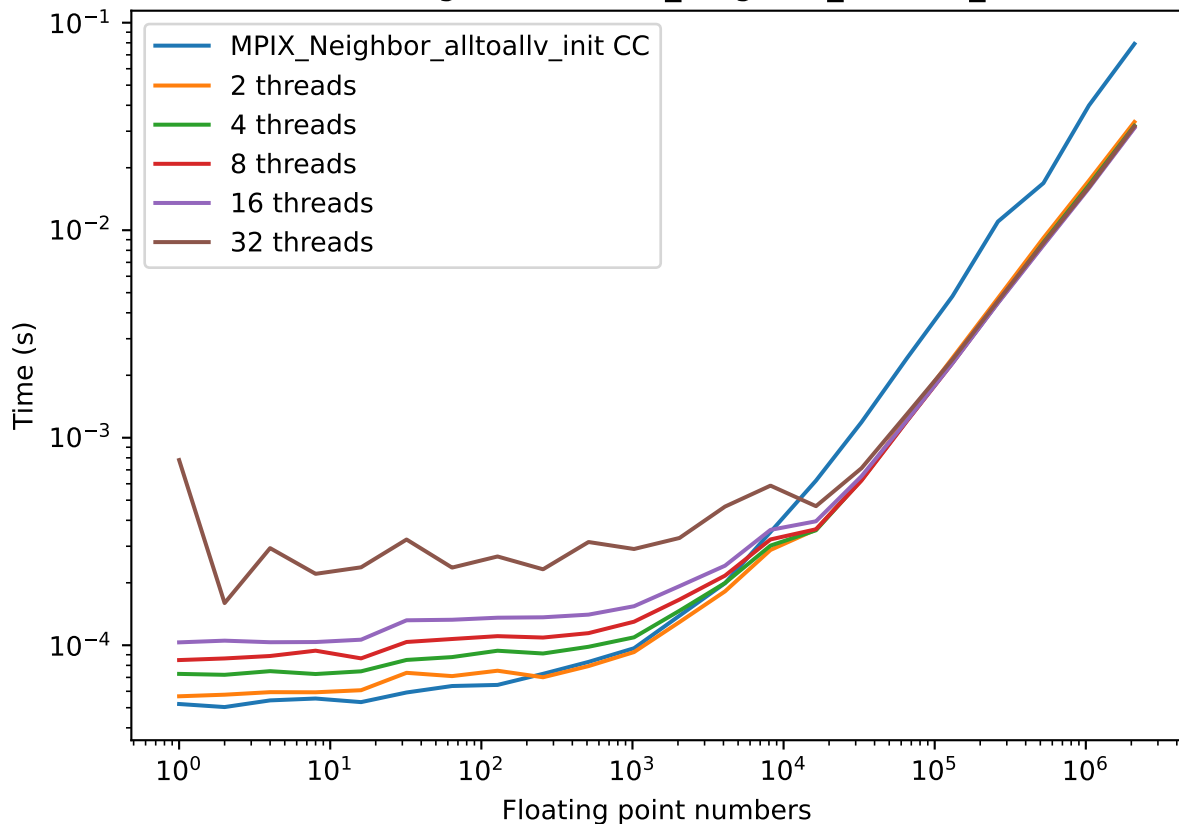
Threaded Neighbor vs MPIX_Neighbor_alltoallv_init



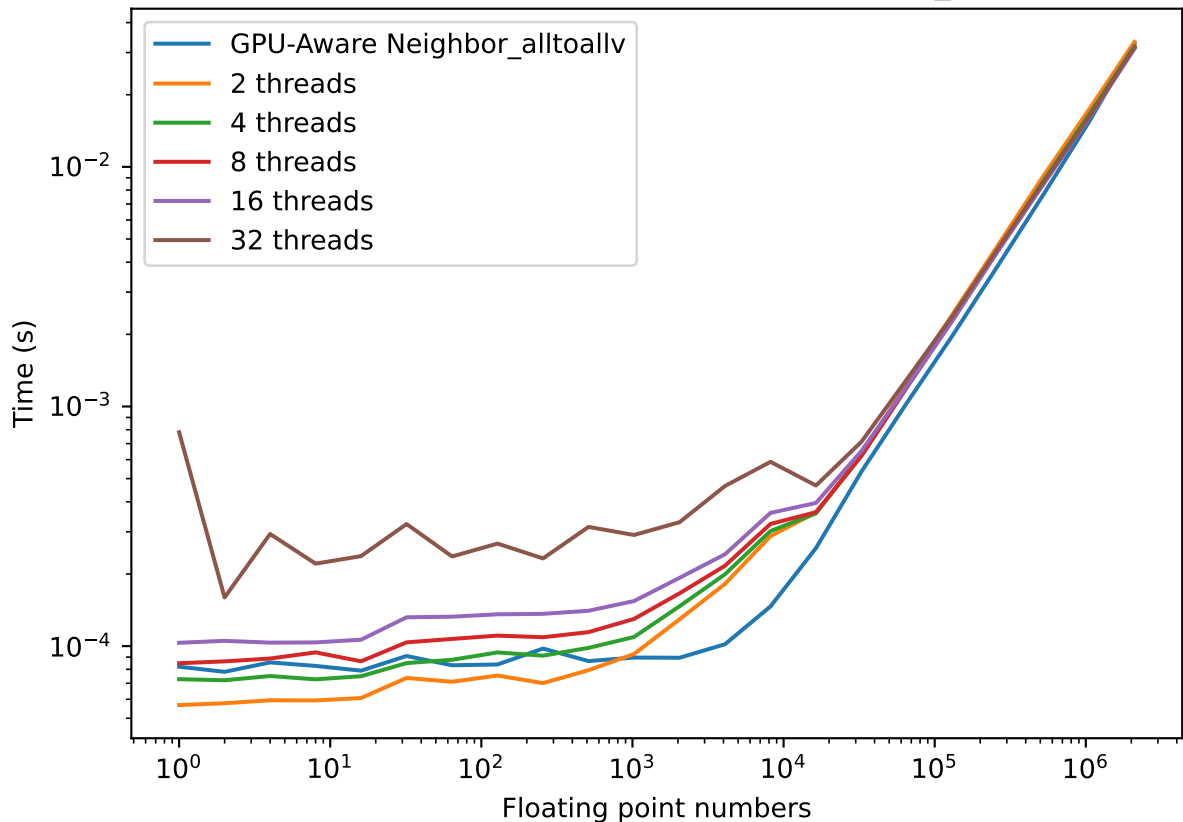
Threaded Neighbor vs MPIX_Neighbor_alltoallv CC



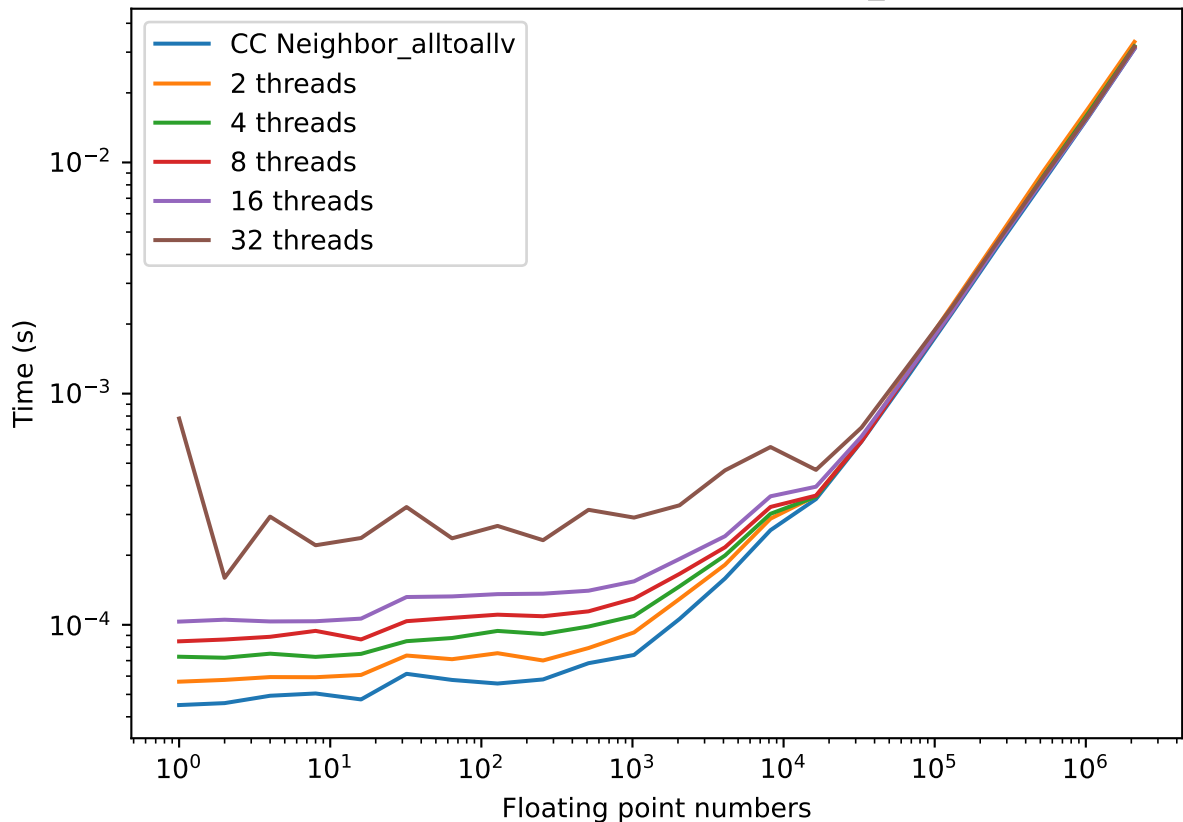
Threaded Neighbor vs MPIX_Neighbor_alltoallv_init CC



Threaded Neighbor vs GPU-Aware Neighbor_alltoallv



Threaded Neighbor vs CC Neighbor_alltoallv



Threaded Neighbor vs MPI_Alltoall

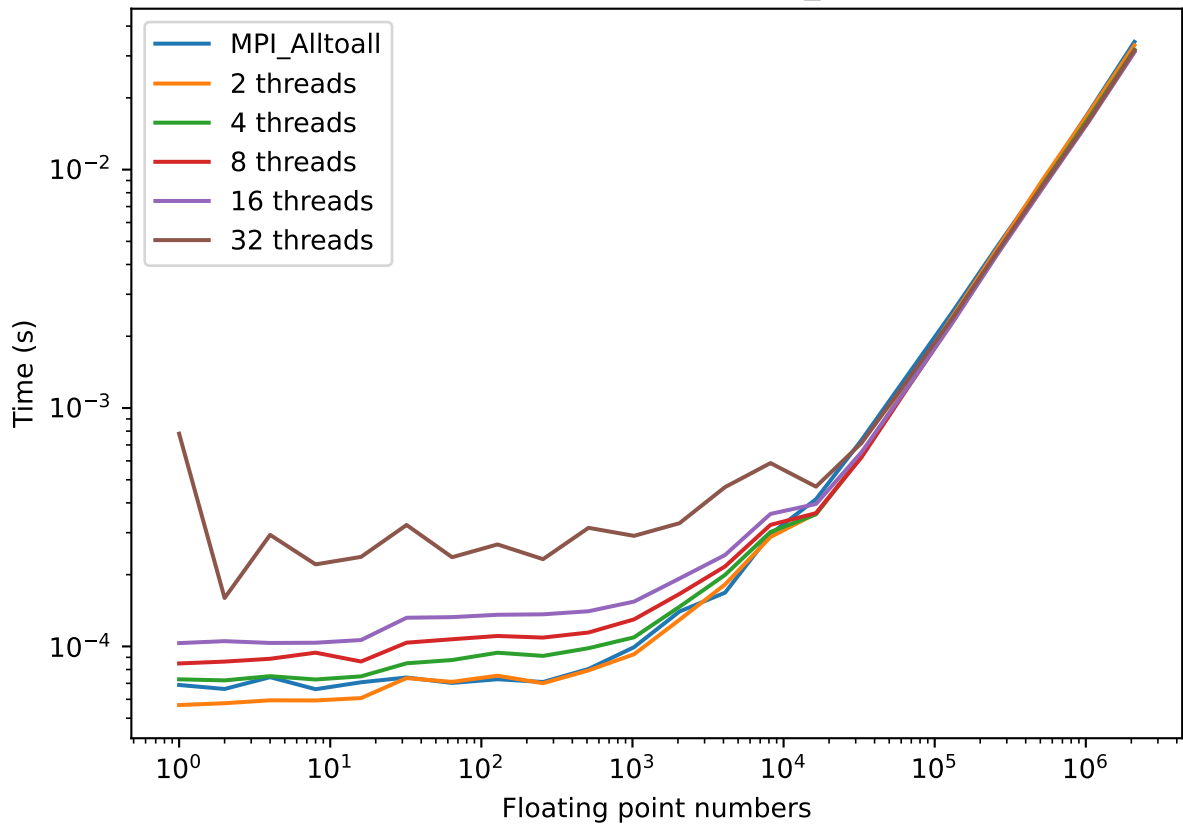
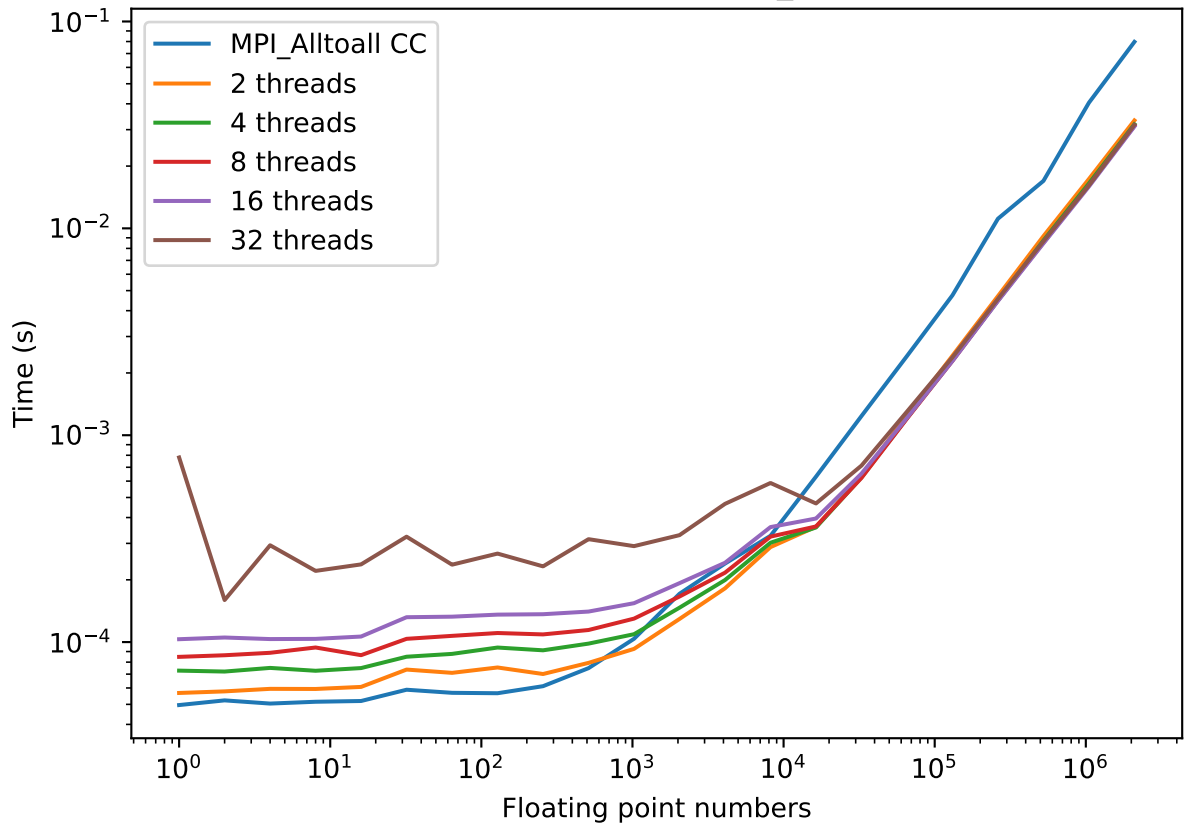
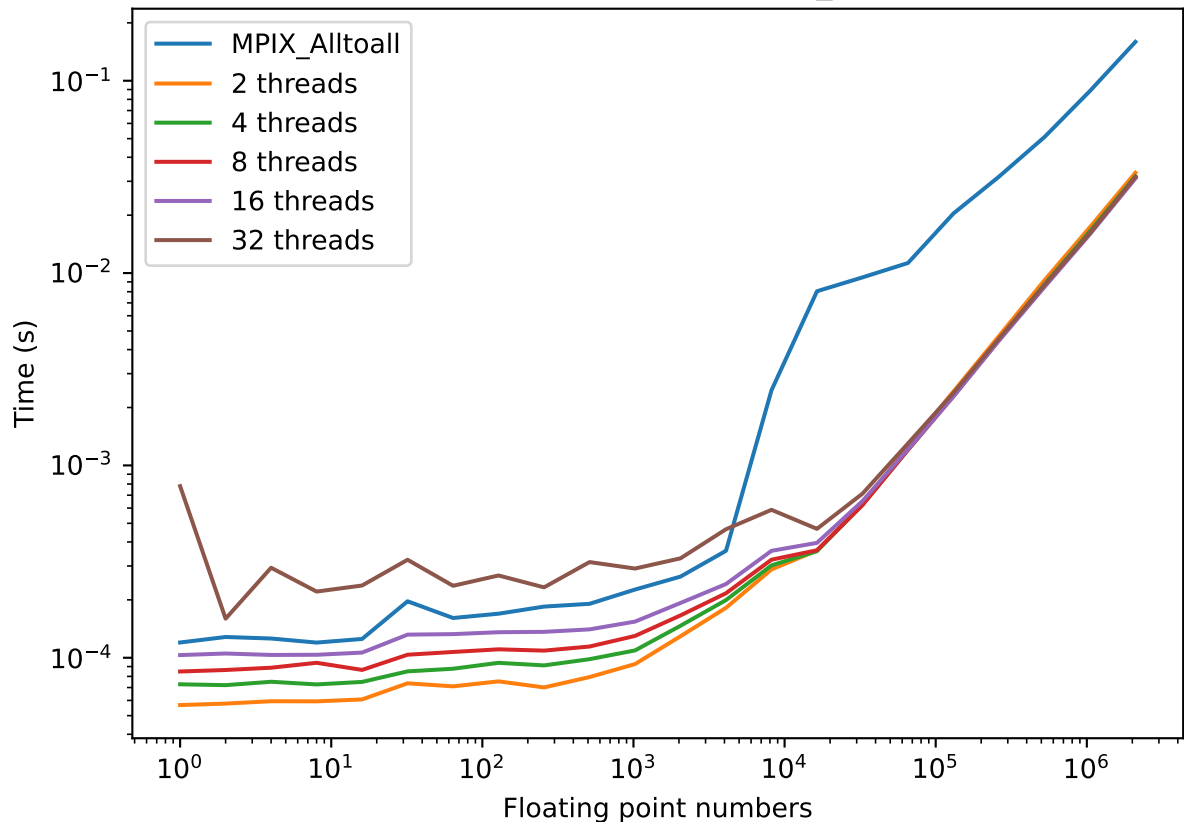


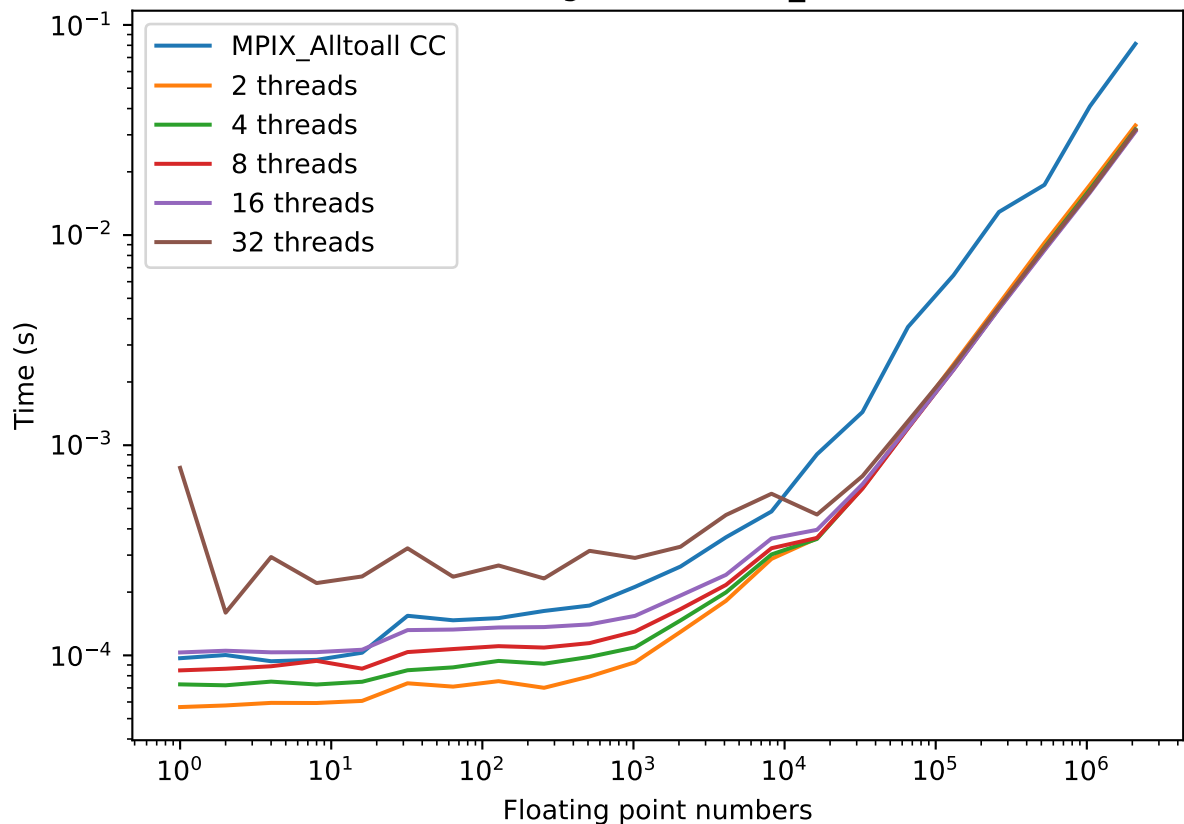
Figure 1 is a log-log plot showing the scaling of the proposed algorithm. The x-axis represents the number of floating point operations, ranging from 10^0 to 10^6 . The y-axis represents the number of threads, ranging from 10^0 to 10^4 . The plot compares the performance of the proposed algorithm (MPI_Alltoall CC) against 2, 4, 8, 16, and 32 threads. The proposed algorithm shows a significant increase in performance as the number of threads increases, reaching a peak of approximately 10^4 operations at 32 threads. The other algorithms show a much slower increase in performance, reaching a peak of approximately 10^3 operations at 32 threads.



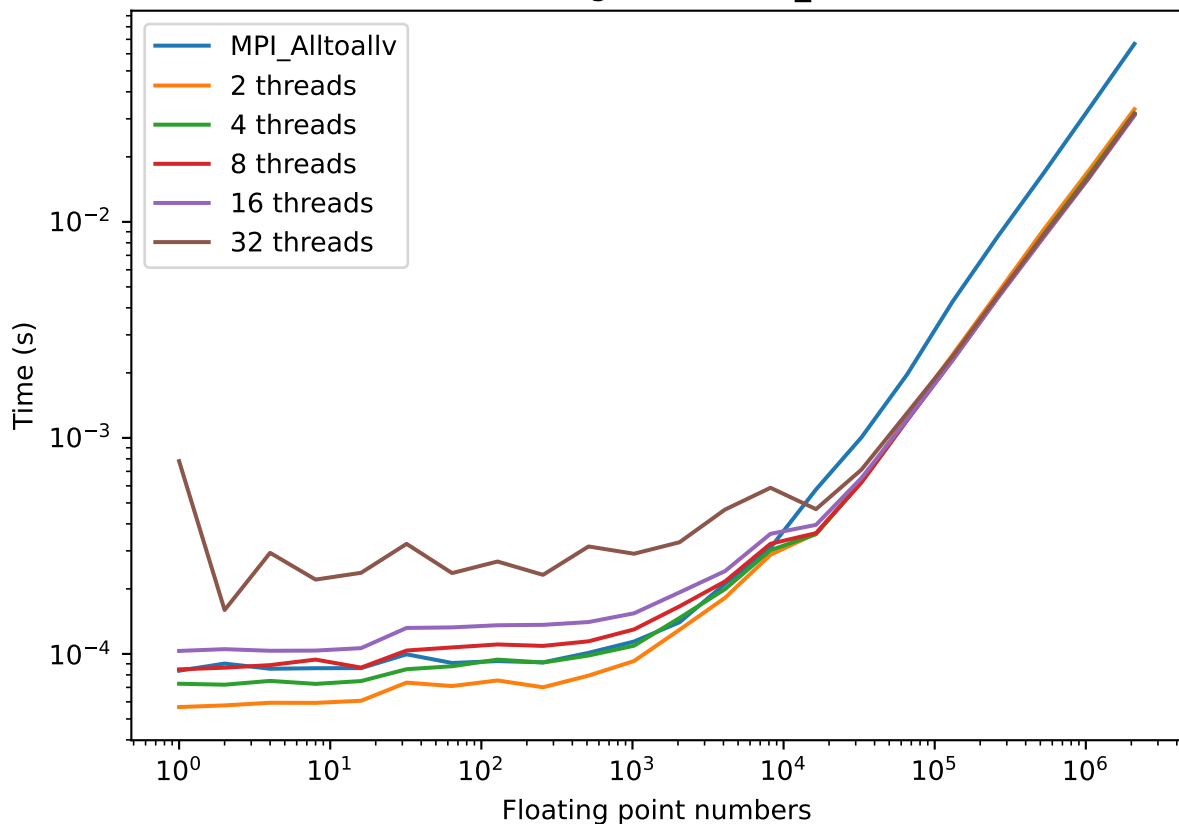
Threaded Neighbor vs MPIX_Alltoall



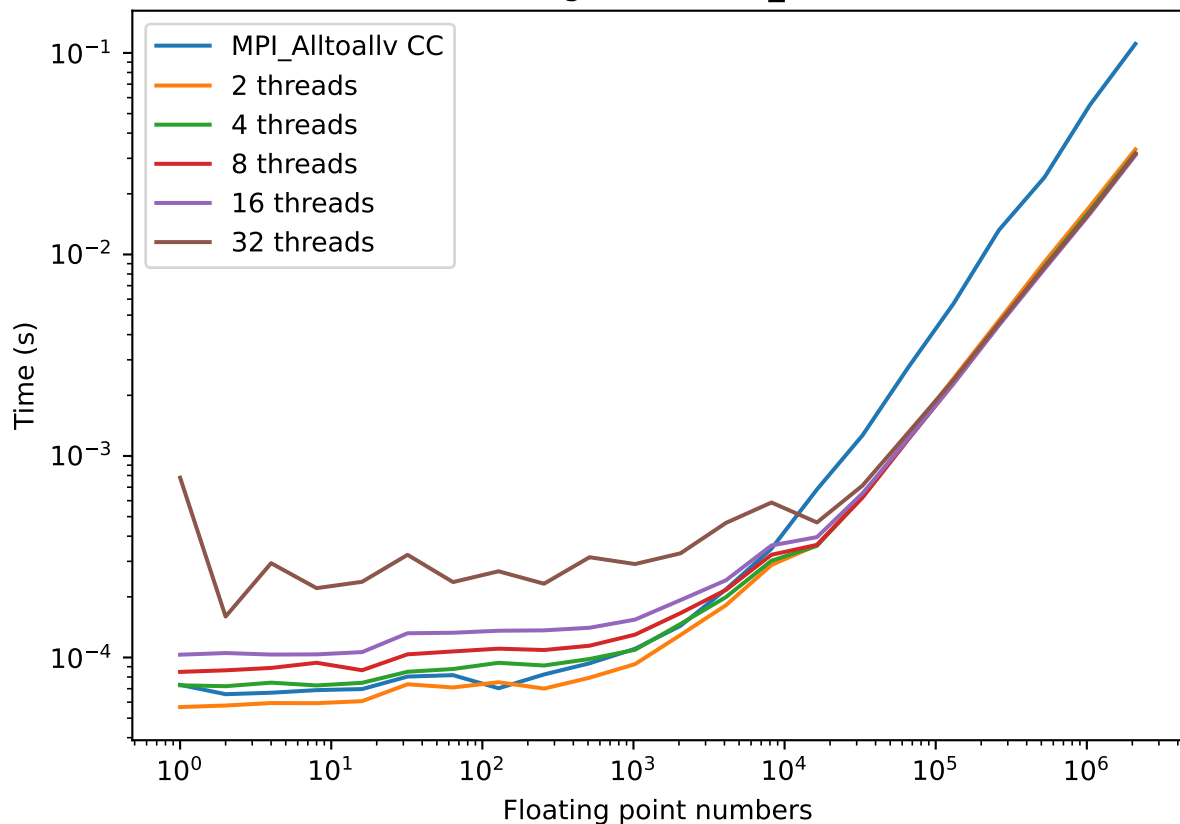
Threaded Neighbor vs MPIX_Alltoall CC



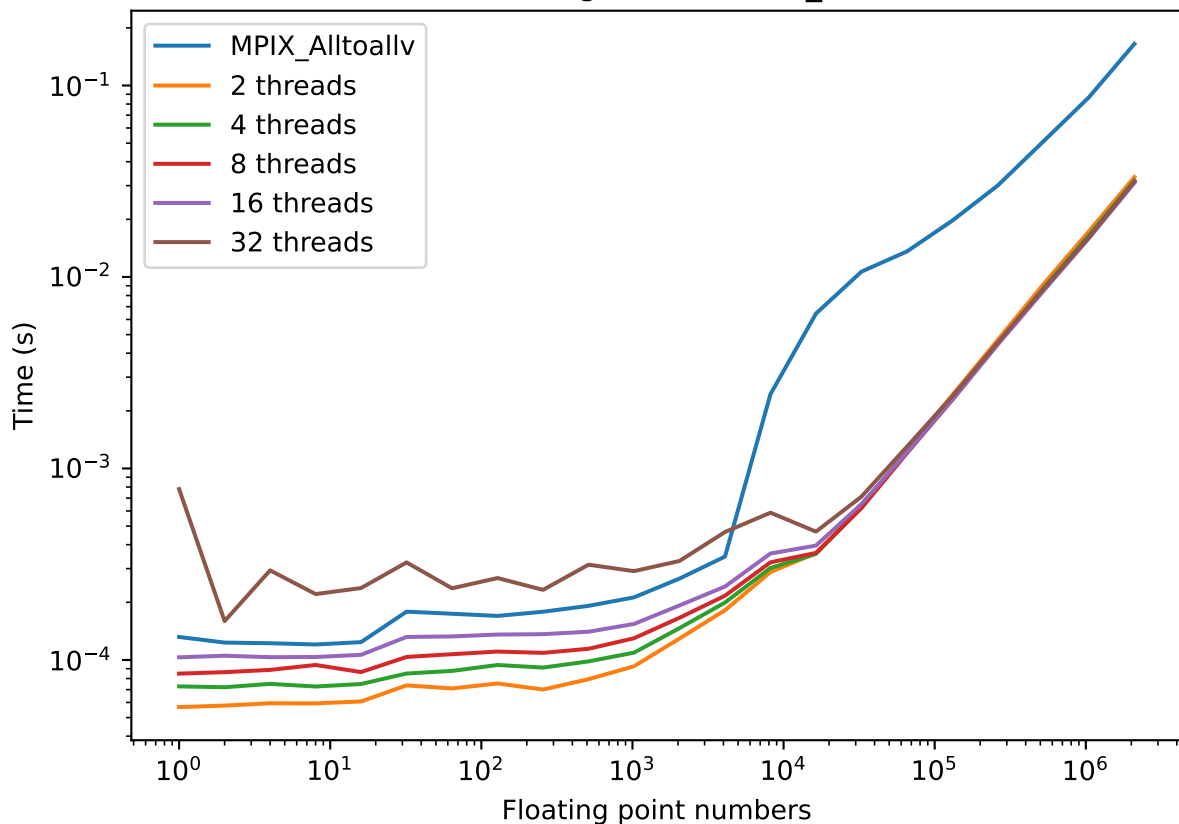
Threaded Neighbor vs MPI_Alltoallv



Threaded Neighbor vs MPI_Alltoallv CC



Threaded Neighbor vs MPIX_Alltoallv



Threaded Neighbor vs MPIX_Alltoallv CC

