FFT Communication Optimization

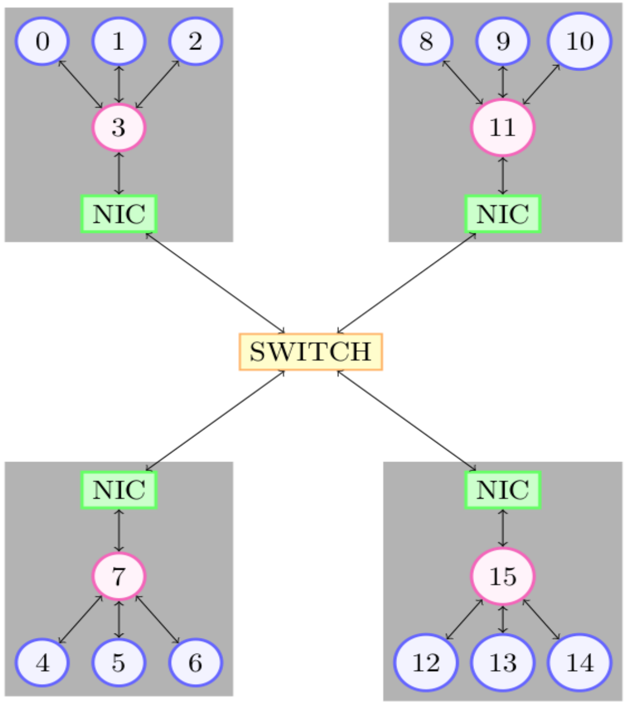
Mohit Kumar

Supervisor:- Prof. Preeti Malakar

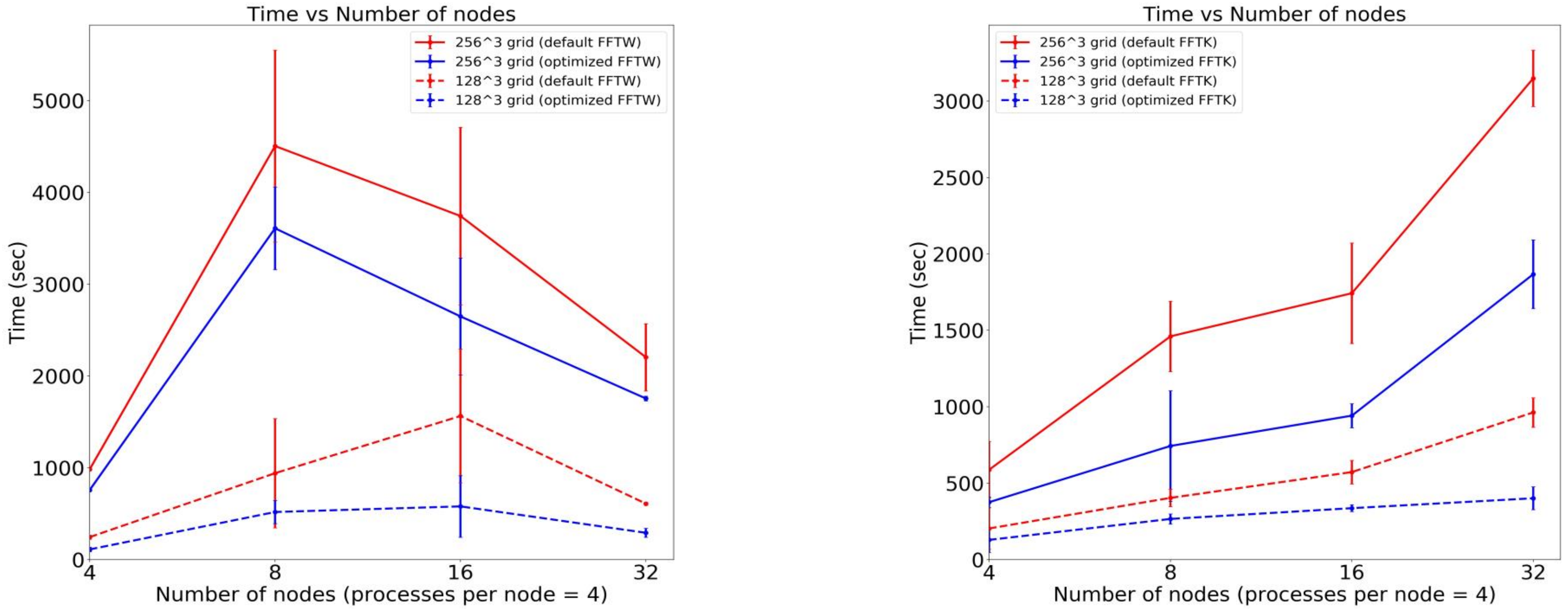
Co-supervisor:- Prof. Mahendra Verma

One Leader Case

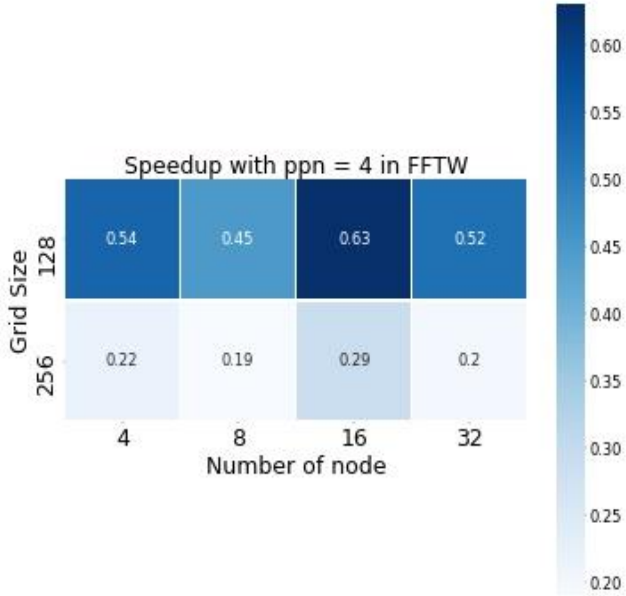
- Profiled FFTW and FFTK code by using TAU profiler and HPCToolkit.
- Finds the bottleneck reason:- MPI_Sendrecv() in FFTW and MPI_Alltoall() in FFTK.
- Converts the blocking call into the non-blocking call.
- Implemented hierarchical communication technique at the node level.
- Tested the non-blocking and hierarchical version of FFTW and FFTK on the IITK CSE Lab cluster.
- Above implementation reduces the communication time on the IITK CSE Lab cluster.

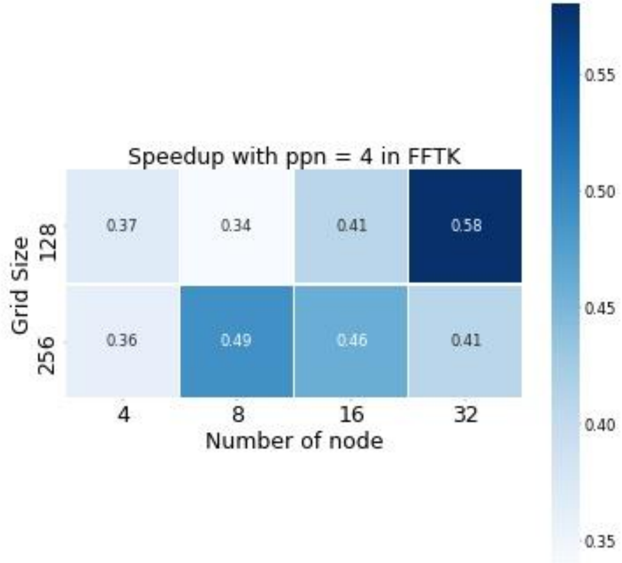


Result of node-level communicator on CSEWS Cluster



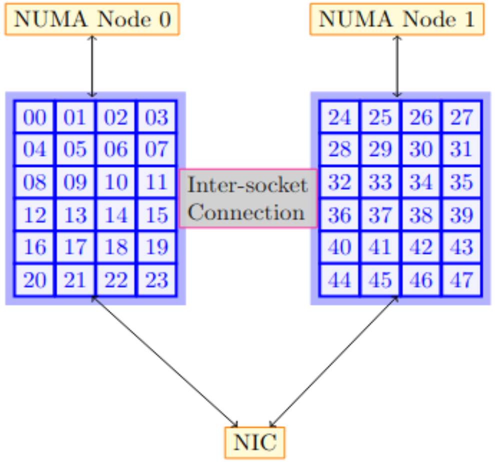
Result of node-level communicator on CSEWS Cluster

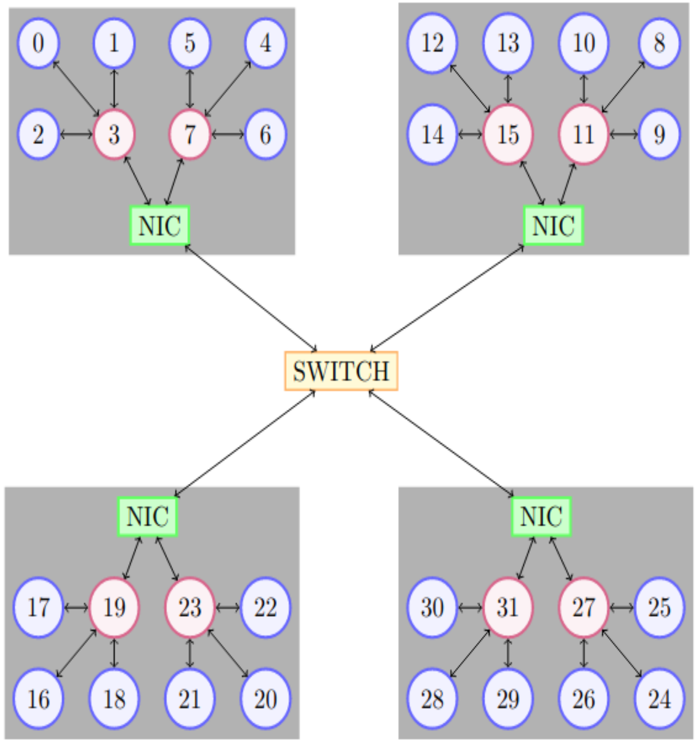




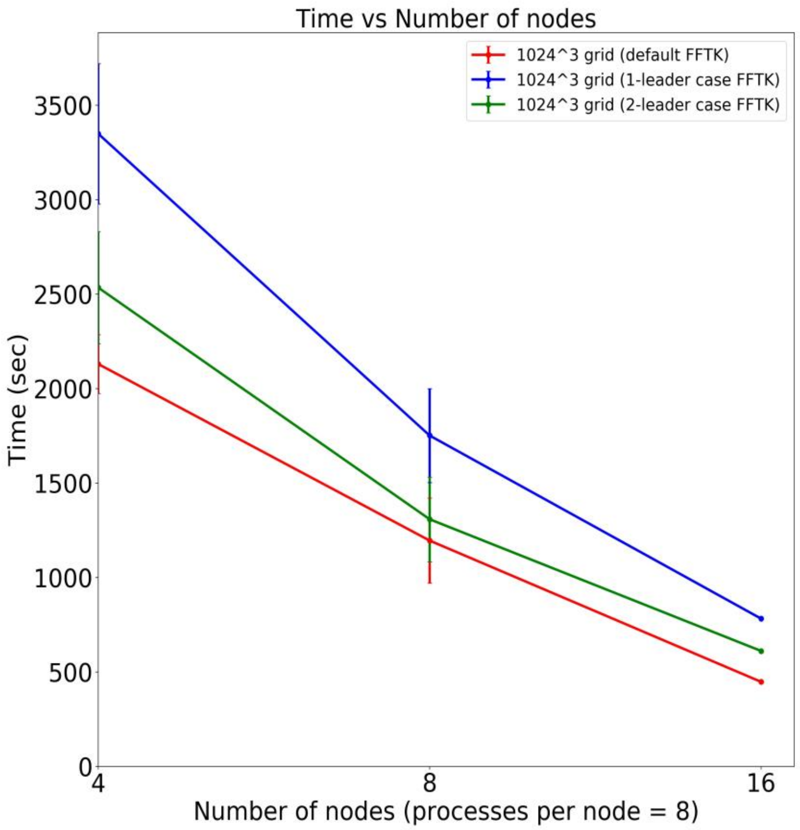
Node-level communicator on PARAM

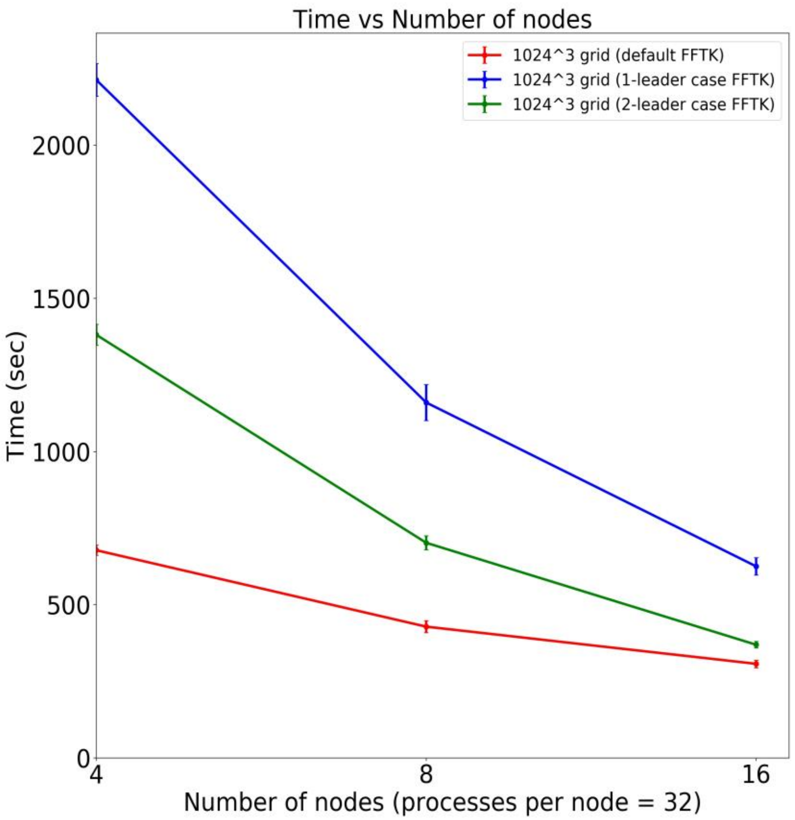
- The number of Numa Nodes on PARAM is 2.
- PARAM has InfiniBand interconnect, which is very fast compared to interconnect speed at the IITK CSE Lab cluster.
- Set the number of leaders per node to 2.
- Used process pinning for placement of processes.



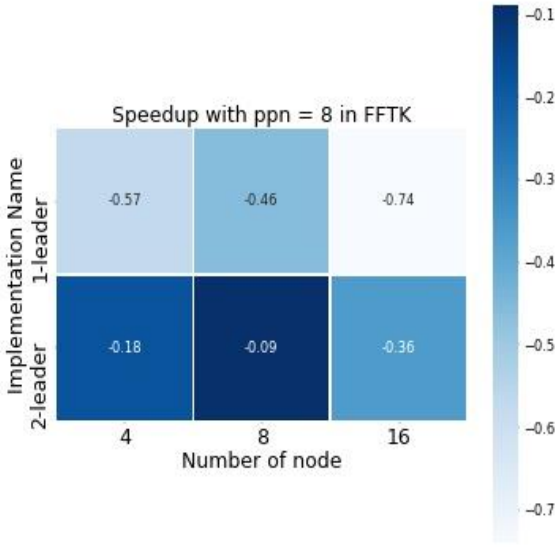


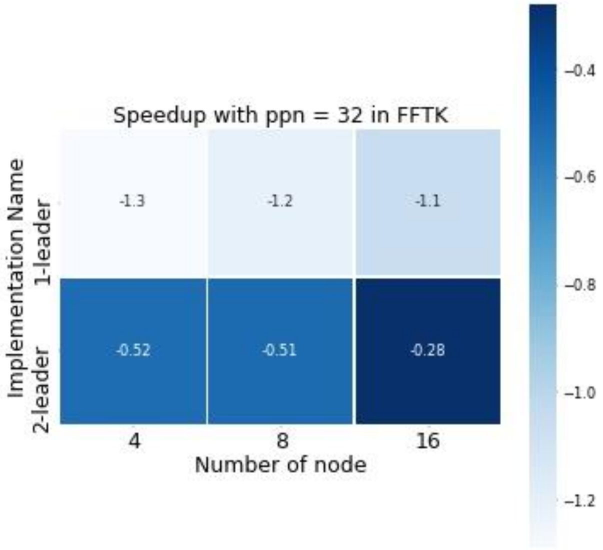
Result of node-level communicator on PARAM (FFTK)



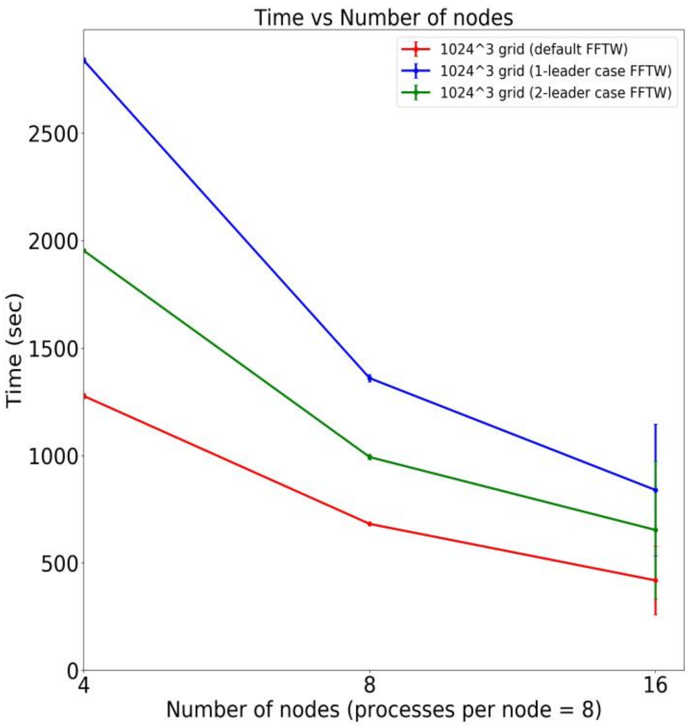


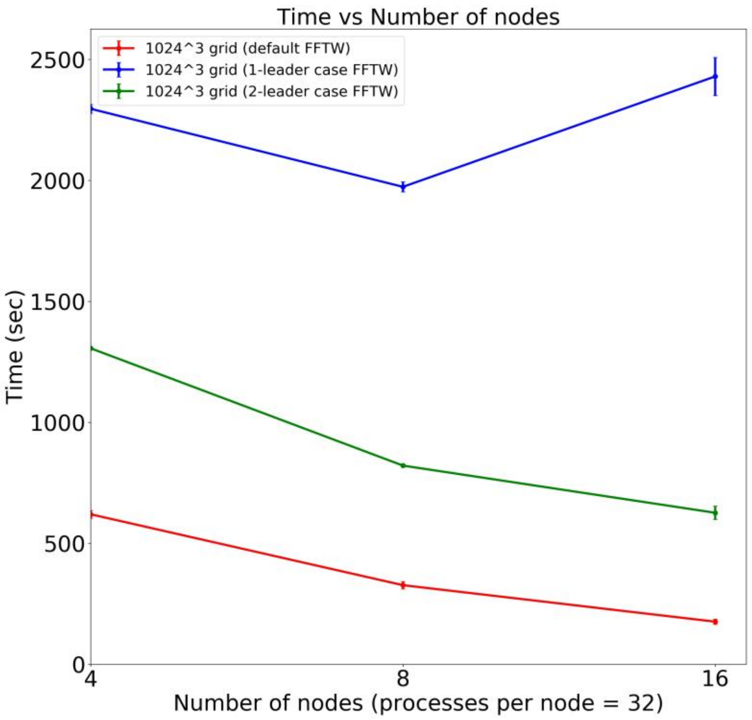
Result of node-level communicator on PARAM (FFTK)



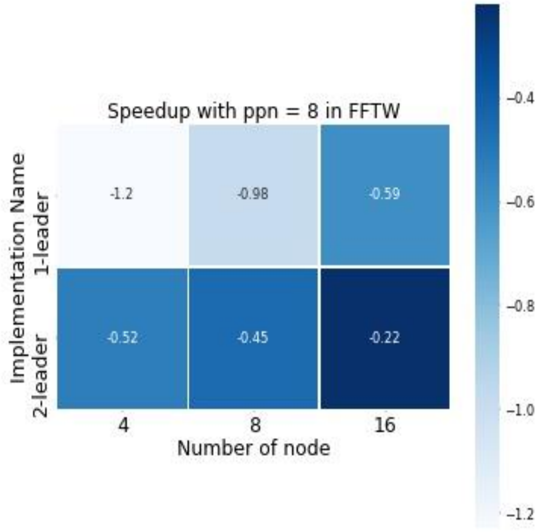


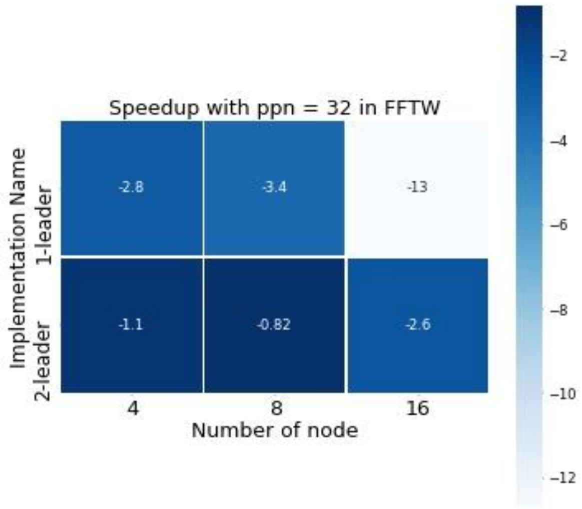
Result of node-level communicator on PARAM (FFTW)





Result of node-level communicator on PARAM (FFTW)



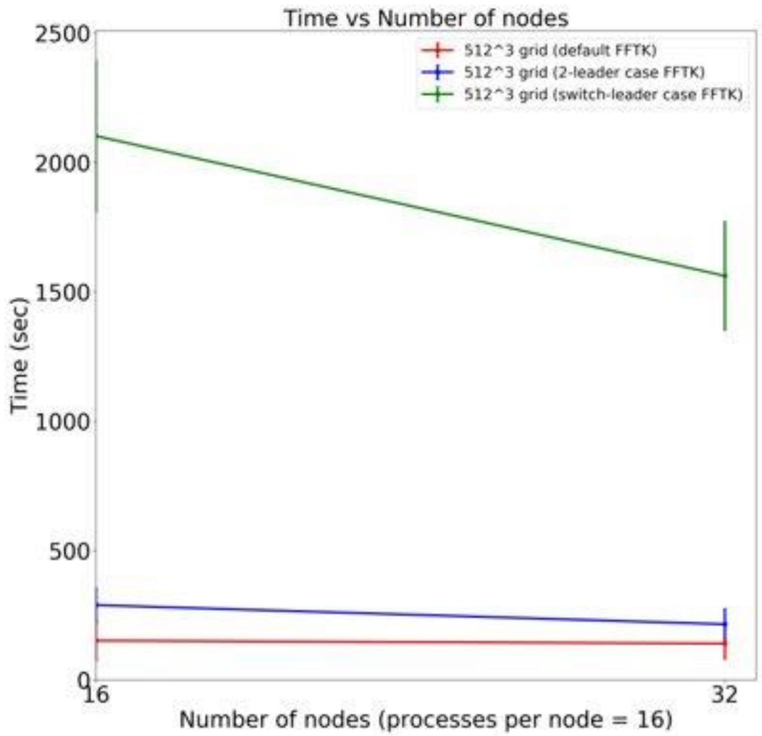


Switch Leader Communicator

- Node leader is created at every node.
- Every non-leader process present at each node will send and receive data from their node leader.
- One node leader is selected as the switch leader among all the nodes present in a switch.
- So, out of N node leaders present in a switch, one is the switch leader, and the rest are the non-switch leader.
- Every node leader will send the data to their switch leader. It may be possible that the number of nodes present in every switch is not equal.
- All the leaders of each switch will communicate and exchange data.
- The switch leader will send the exchanged data to the non-switch leader.
- Then, a non-switch leader who is also a node-level leader will distribute the data to the non-leader process present on the same node.
- Here, we have implemented two hierarchy levels, i.e., node level and switch level.

Switch Leader Result

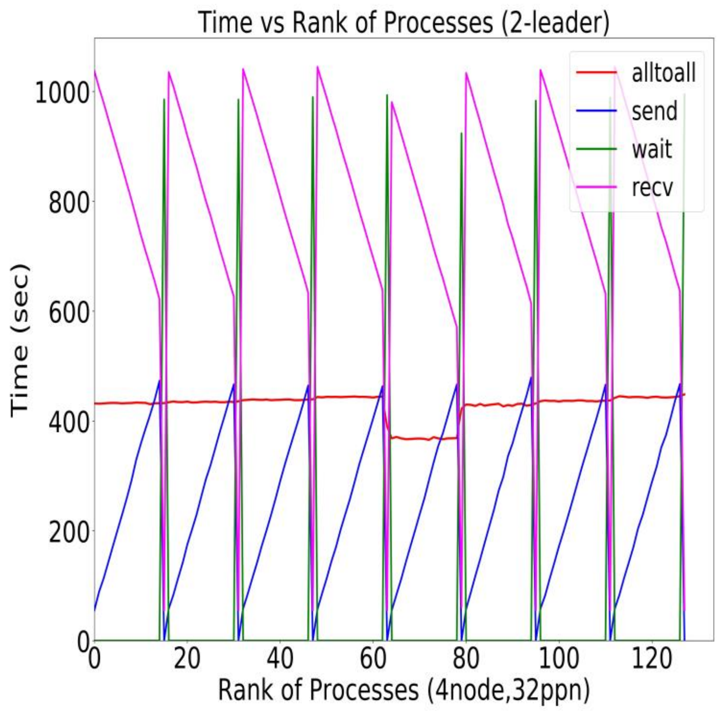
- On PARAM, node allocation is done by SLRUM.
- Node allocation is not done in equal distribution of nodes among switches.
- For example, if N=16, it may be possible that seven nodes are allocated in switch 1, 5 in switch 2, and 4 in switch 3.
- The number of nodes leader varies per switch. So, every switch-leader may have different amount of data.
- Our result shows that the switch leader code performs worse than the default case.

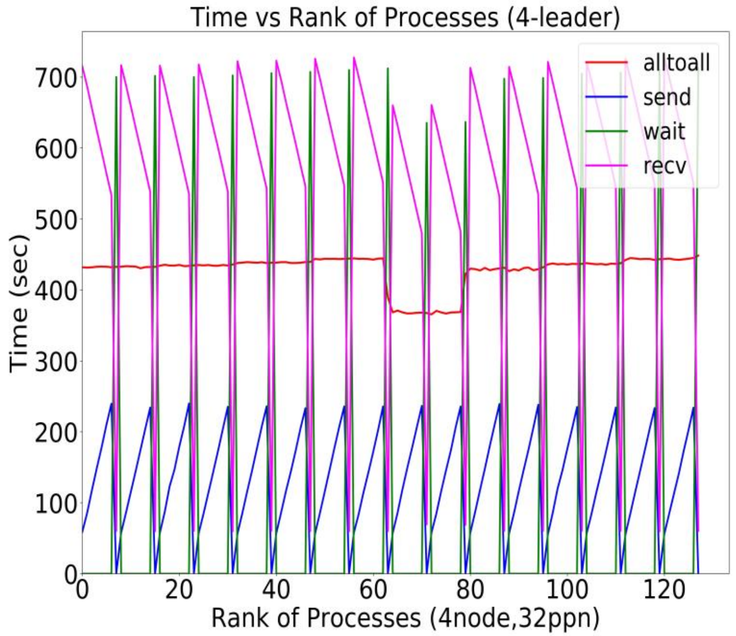


Increase In Number of Leader per Node

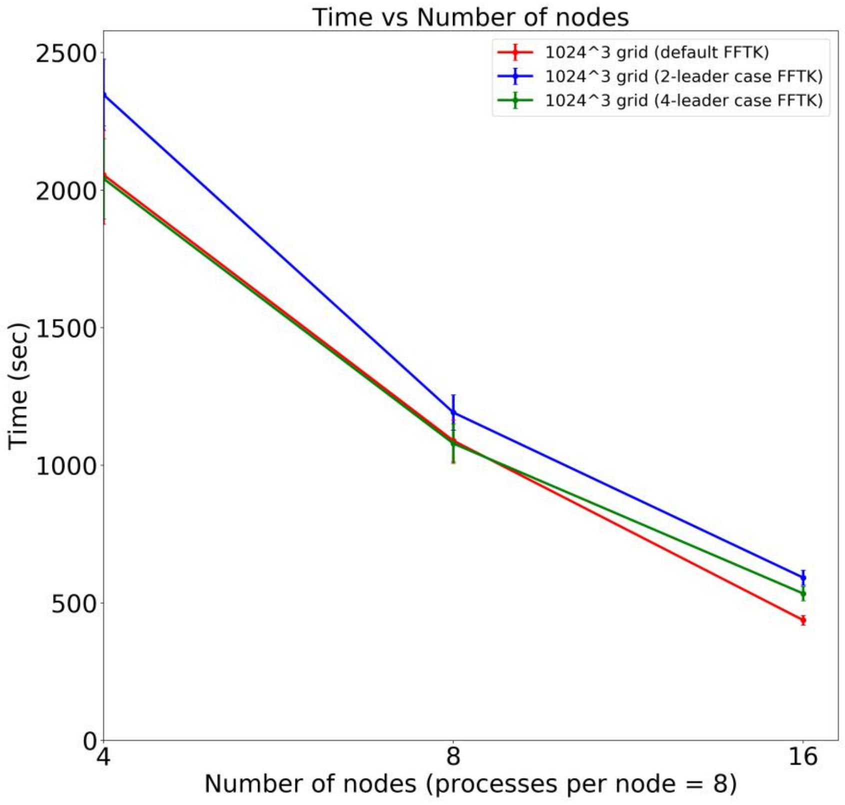
- Observed that the leader process takes a significant portion of time to gather the data from the non-leader process.
- The number of leaders per node is increased to reduce the data gathering time.
- Increase the number of leaders in multiple of 2 for the equal number of non-leader processes per leader.
- Used process pinning for placement of processes for using both NUMA nodes of PARAM.

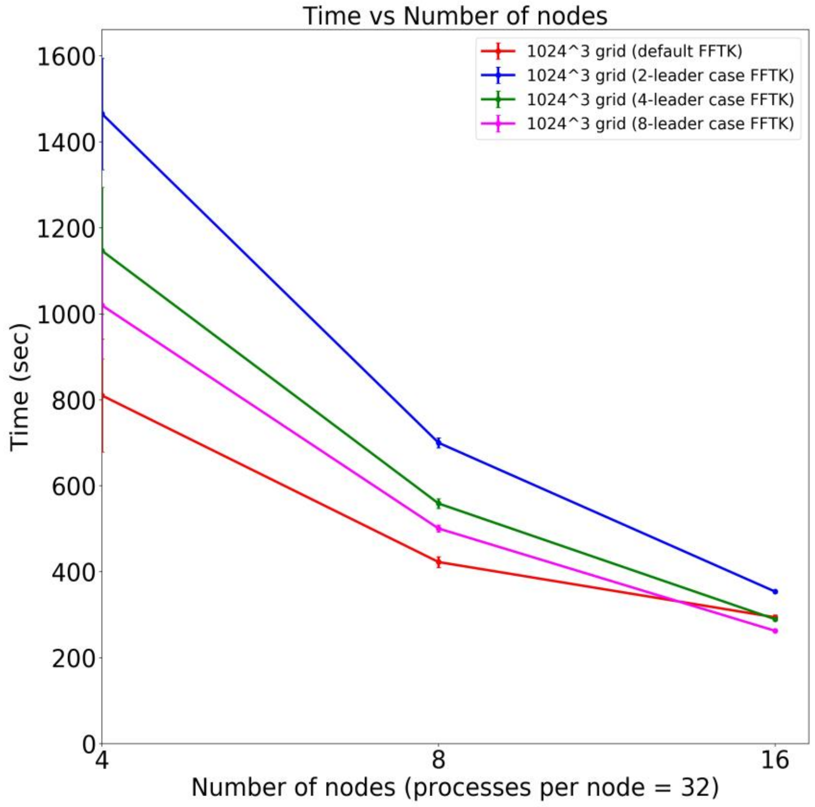
Increase In Number of Leader per Node



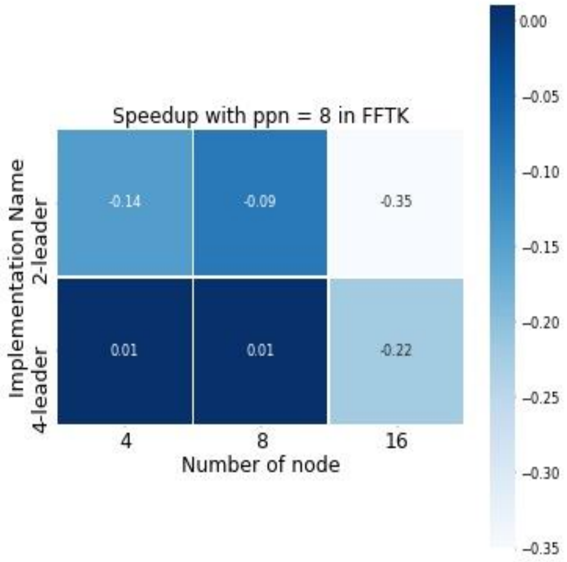


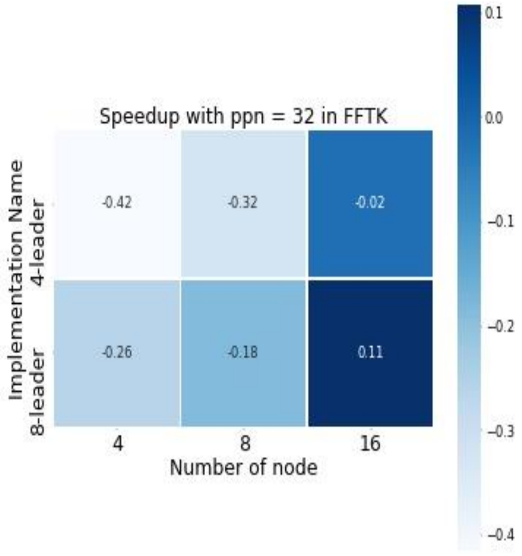
Result of multiple node-level communicator on PARAM(FFTK)



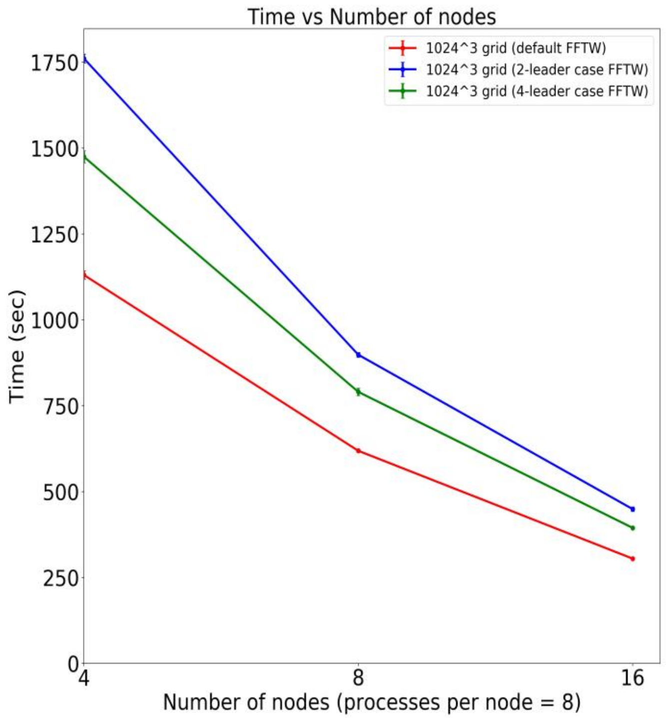


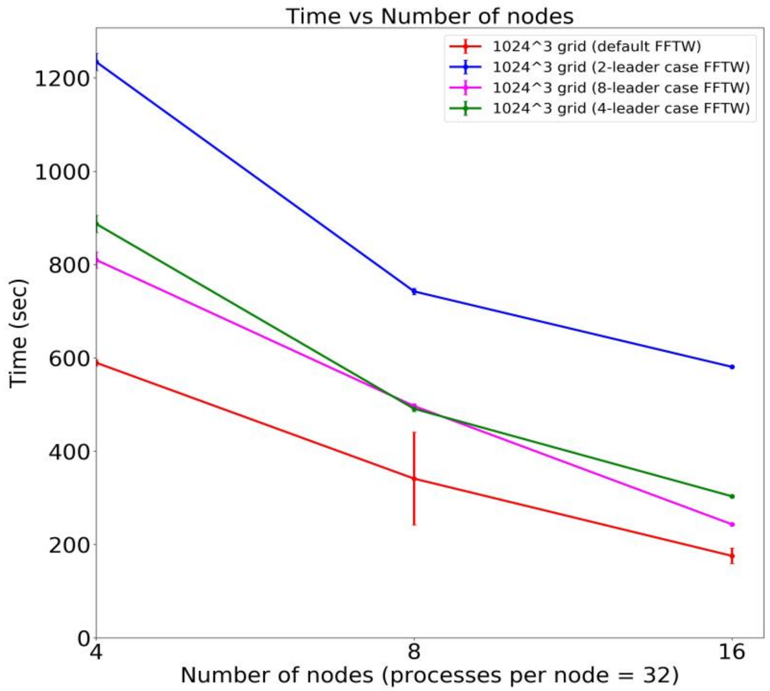
Result of multiple node-level communicator on PARAM(FFTK)



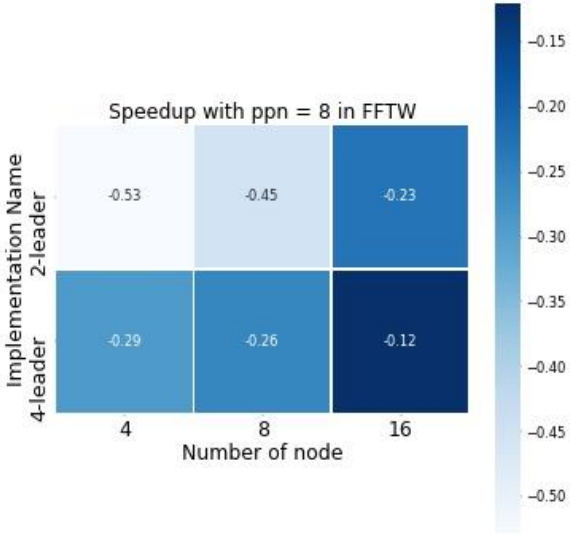


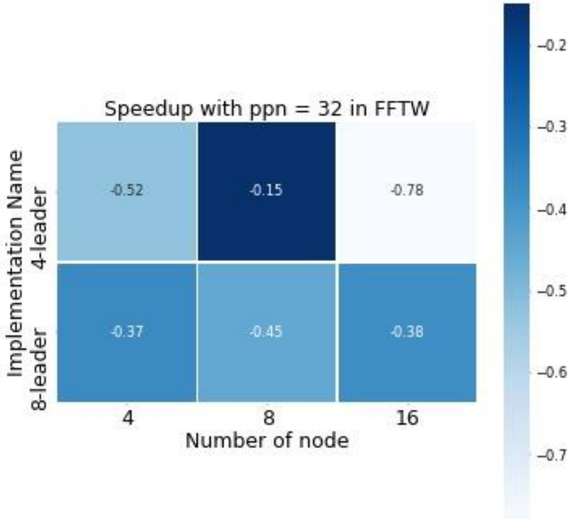
Result of multiple node-level communicator on PARAM(FFTW)





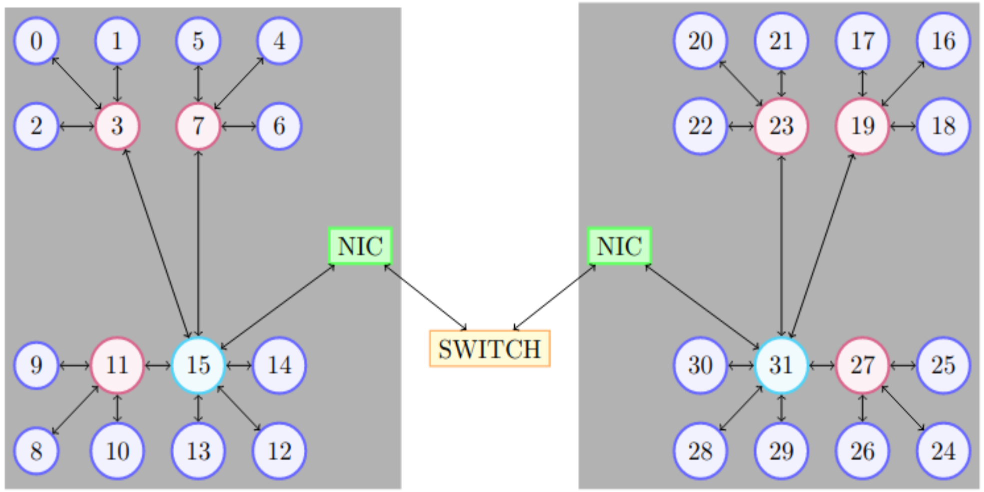
Result of multiple node-level communicator on PARAM(FFTW)

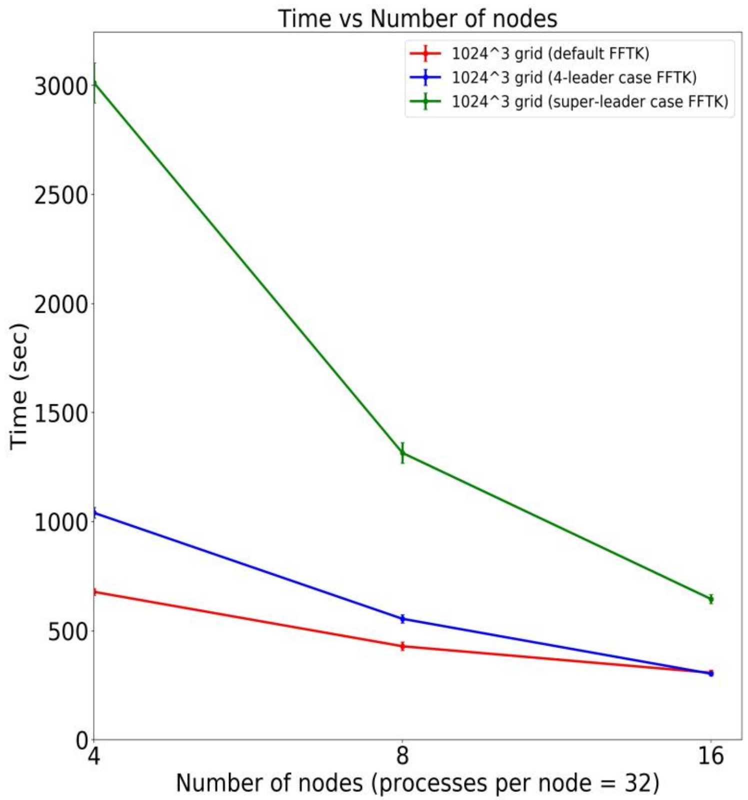




Super-Leader

- With multiple leaders per node, every leader process handles inter-node communication.
- Multiple leaders handling communication can lead to congestion at NIC.
- In this implementation, we ensure that only one process(super-leader) handles the inter-node communication.

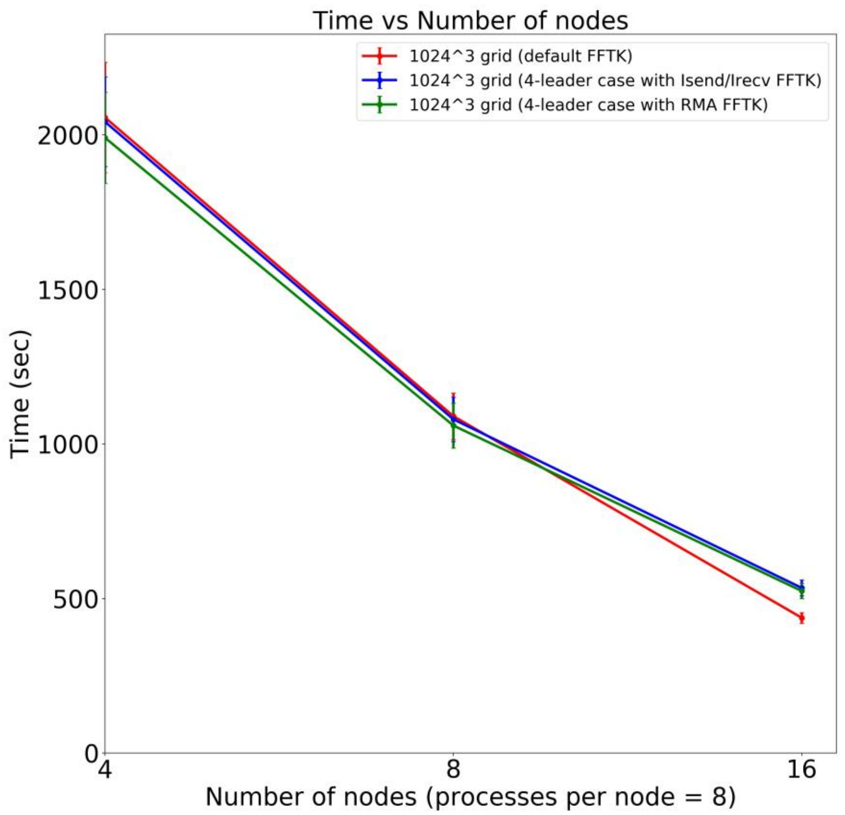


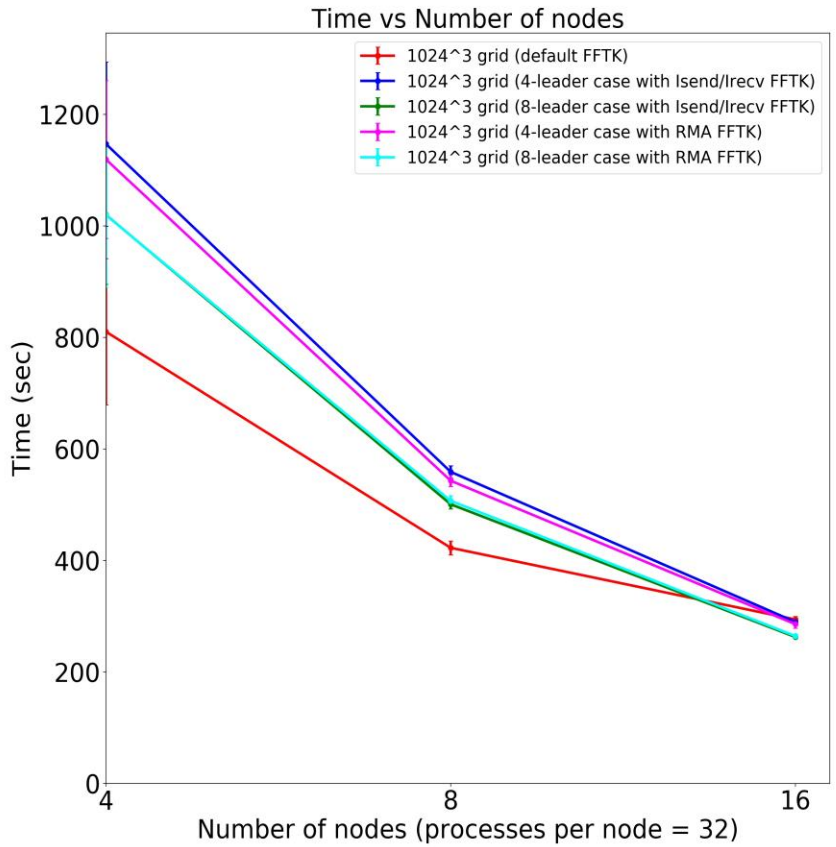


Use of one-sided communication

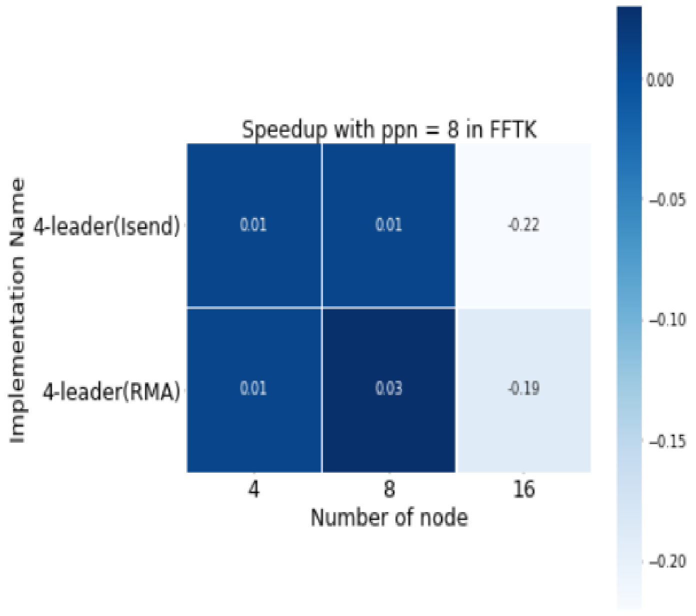
- We used MPI_Isend/MPI_Irecv in all the previous implementations. For large message size, it requires a handshake which increases the latency.
- One-sided communication does not require a handshake.
- We used MPI_Put/MPI_Get in our implementation.
- For synchronization purposes, MPI_Win_fence is used.
- One-sided communication is implemented for multiple leaders per node.

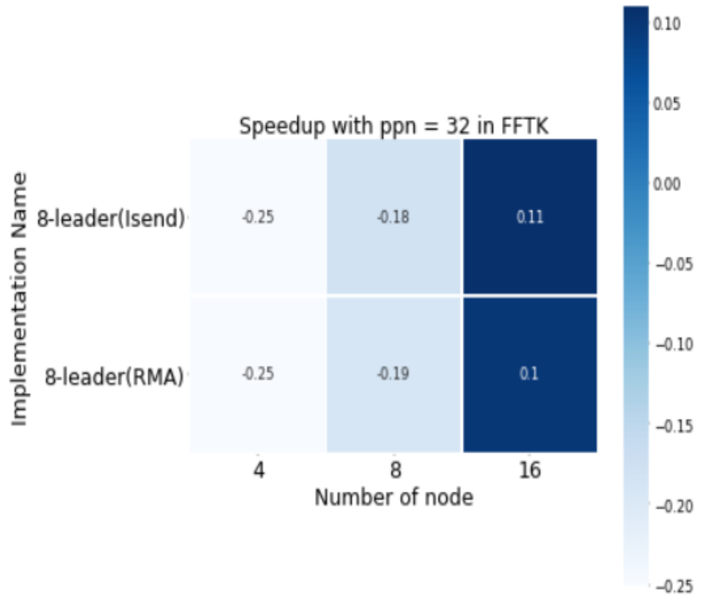
Result of one-sided communication on PARAM (FFTK)



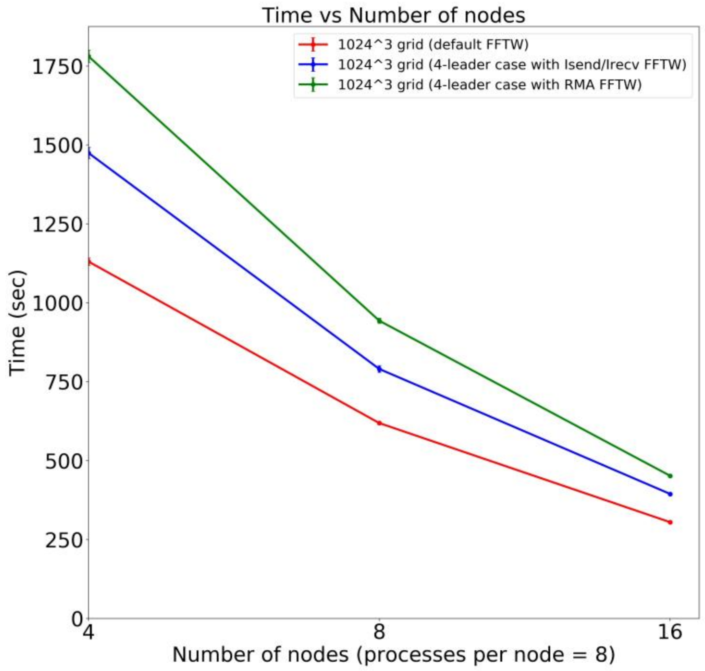


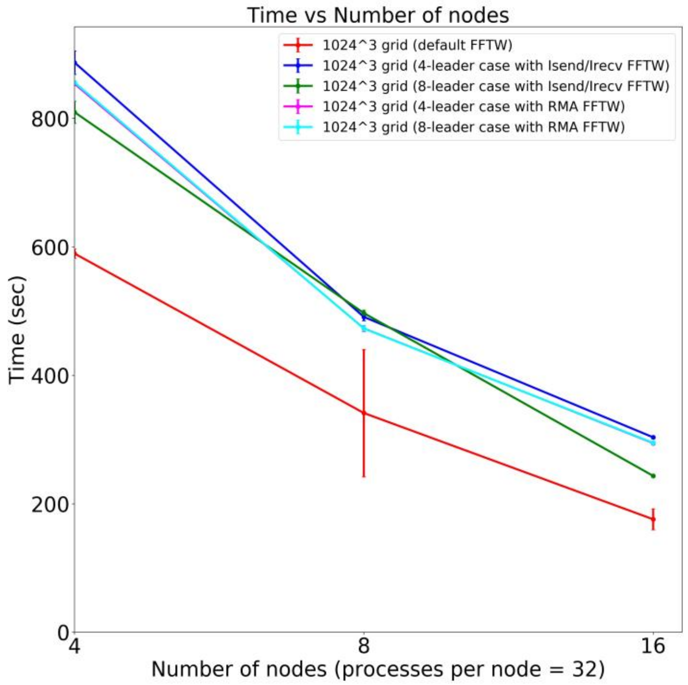
Result of multiple node-level communicator on PARAM(FFTK)



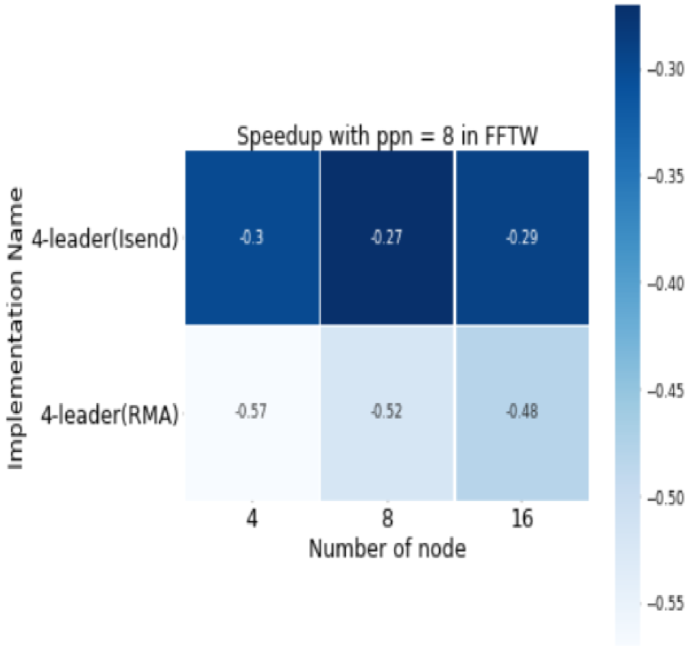


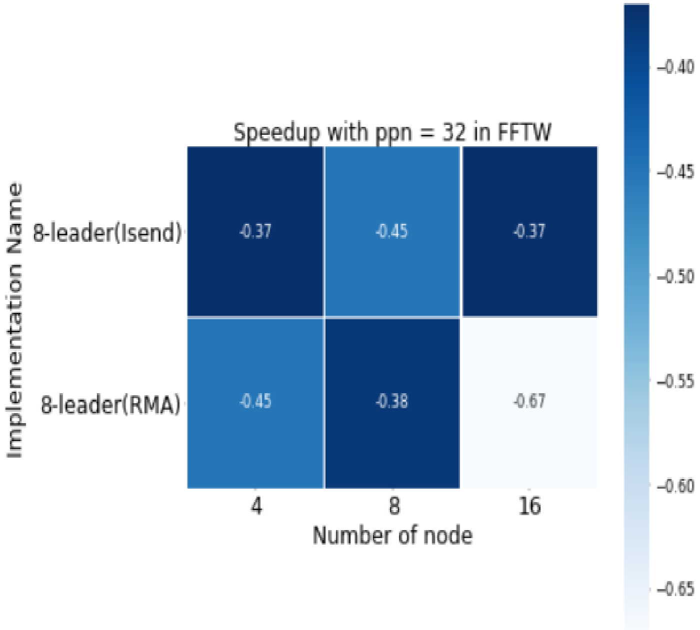
Result of one-sided communication on PARAM (FFTW)





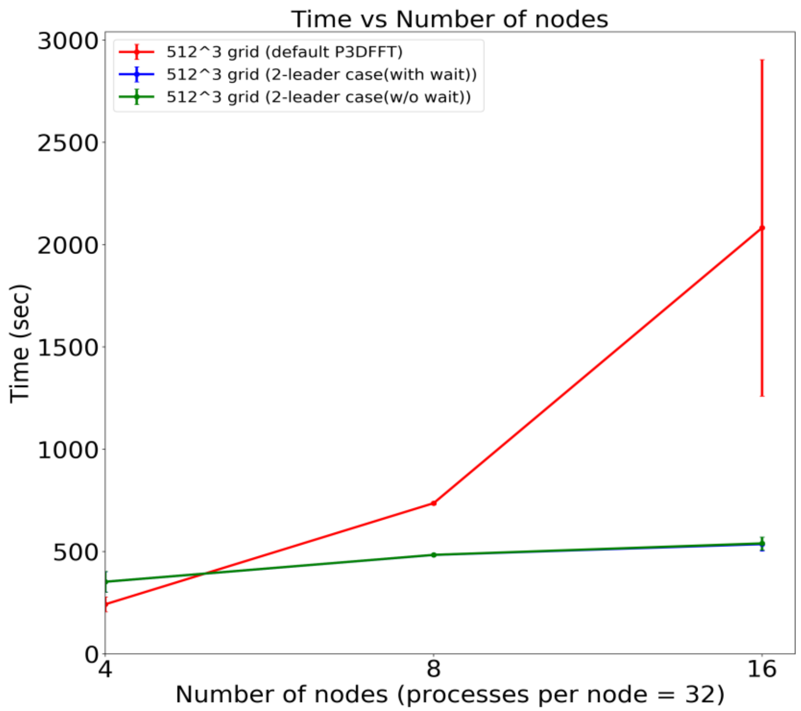
Result of multiple node-level communicator on PARAM(FFTW)





P3DFFT

- Found that P3DFFT uses MPI_Alltoallv() for communication.
- Overload the MPI_Alltoallv() function to call our MPI_Alltoall() function when data size is evenly distributed among the processes.
- Reduce the overall time taken by our code compared to the P3DFFT default time for 32ppn and higher nodes 8 and 16.



Thanking You!