A scheduling policy to improve 10% of communication time in parallel FFT

MS260

Next Generation FFT Algorithms in Theory and Practice: Parallel Implementations and Applications

Samar Aseeri, PhD

SIAM CSE'21

QUICK BACKGROUND

Definition of Fourier Transform

In 1807 Jean Fourier invented a technique to solve the heat diffusion equation in a conducting plate with arbitrary forcing. This technique was the Fourier Transform.

$$x \to k : f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx} F(k) dk, \qquad k \to x : F(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x) dx$$

Cooley and Tukey, 1965

- Cooley and Tukey published a paper on the Fast Fourier Transform an O(n log n) method for the calculation of the Discrete Fourier Transform which is an approximation of the Fourier Transform which originally is derived from the Fourier Series.
- The FFT algorithm was developed by Gauss in 1805 but this was not recognized until recent time.

Why is it Important?

- Time complexity of FFT according to DFT reduces from $O(n^2)\ to\ O(n\ logn)$.
- It is an accurate and low computational cost algorithm.
- It solves multiscale problems.
- Derivatives are simply calculated: $\widehat{f^{(n)}} = (ik)^n \hat{f}$

Limitations

The main performance bottleneck of parallel 3D FFTs is the communication. Once 3D data is distributed over MPI processes, all-to-all communications are unavoidable.

INTRODUCTION

- Aim is to speed up FFTs on Cray XC40 machine by using the full bandwidth offered by the cluster
- For this, we implement some strategies to reduce communication time of the extensively used FFT algorithm
- For testing we used the FFTK parallel library developed by collaborators of this work
- We try to obtain an optimal performance for the FFT by leveraging the all-to-all topology of Dragonfly networks besides the implementation of other techniques.
- Several job placements and reordering cases were examined and some findings will be highlighted here

FFTK LIBRARY

PARALLEL Libraries

Examples of FFT Parallel Libraries:

FFTW, FFTE, FFTK, 2decomp&fft, P3DFFT, PFFT, OpenFFT, AccFFT, GPU FFT and Xion phi FFT, Hybrid FFT

DEVELOPERS

- Chatterjee, Verma, and group members of IIT Kanpur
- It scaled up to 196608 cores of Shaheen II of KAUST
- Fluid solver TARANG uses it
- 2D pencil decomposition is typically used for large core counts

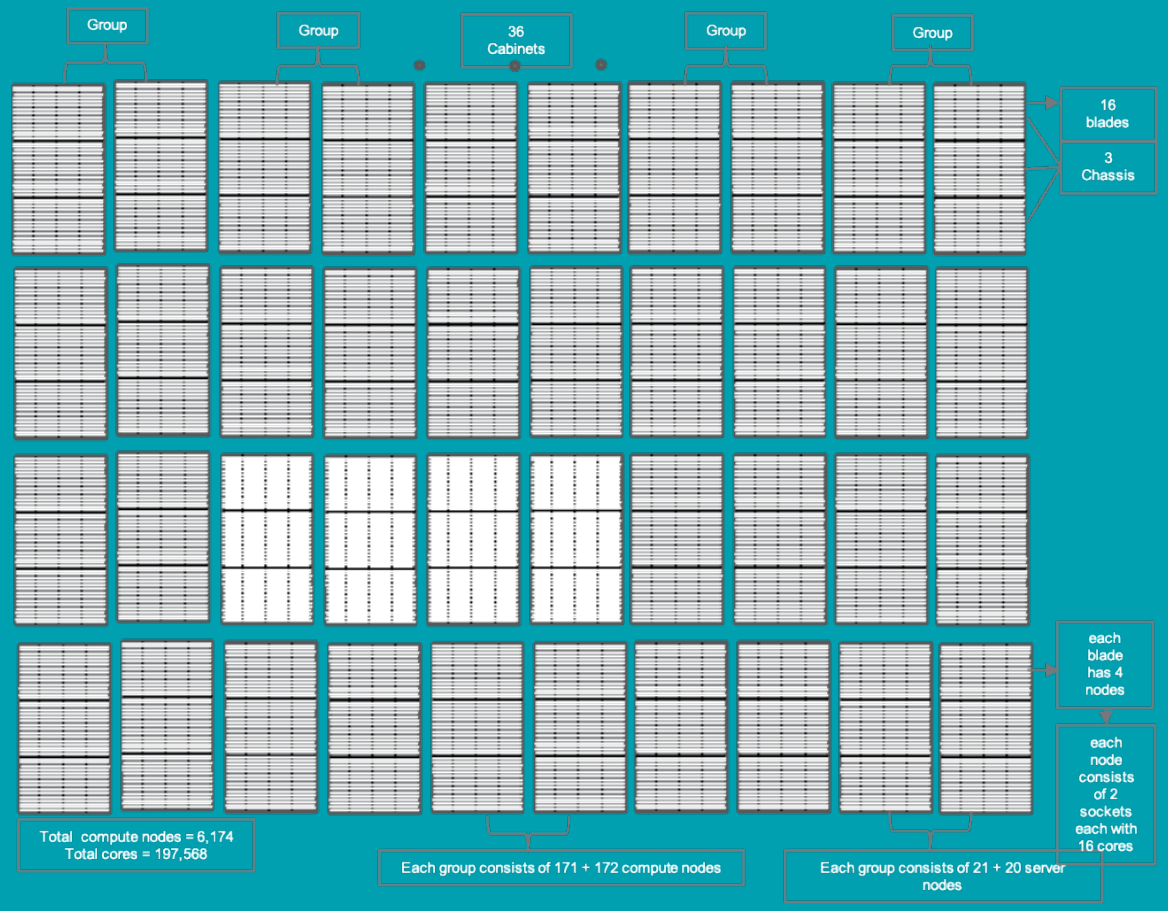
ALGORITHM

- The forward transform is given by: $\hat{f} = \sum_{k_x k_y, k_z} f(x,y,z) e^{-ik_x x}\, e^{-ik_y y} e^{-ik_z z}$
- Algorithm
  - FFT along Z axis
  - Communicate Z pencils to Y pencils.
  - FFT along Y axis
  - Communicate Y pencils to X pencils.
  - FFT along X axis

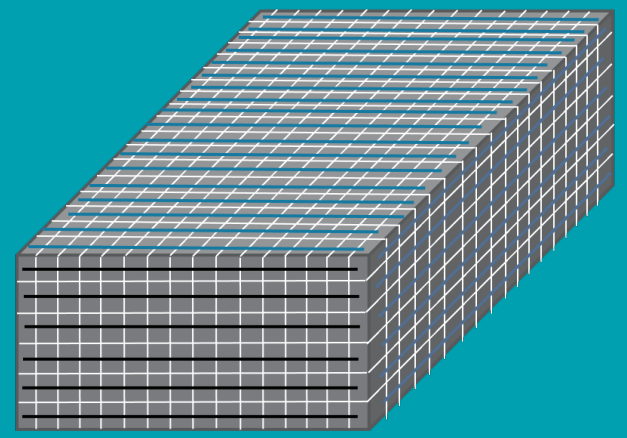
FEATURES

- Works also for 2D data by setting $N_y = 1$
- Slab FFT can be performed by setting $p_{row} = 1$
- Available basis: FFF, SFF, SSF, SSS and ChFF

SHAHEEN TOPOLOGY



- It is a Cray XC40 and it is the 52 fastest supercomputer in the world
- It consists of about 200000 CPU cores
- It manages a speed of about 7 Petaflops/s theoretical peak and 5.5 Petaflops/s of Linpack performance
- It uses Dragonfly Topology

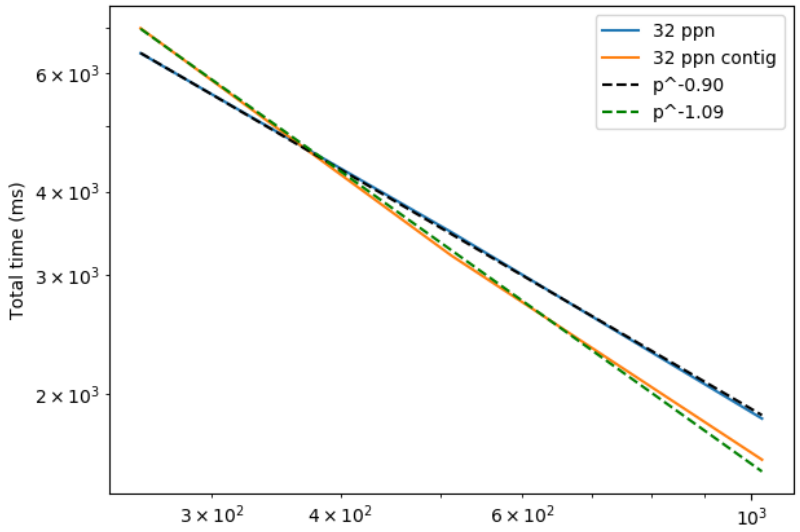


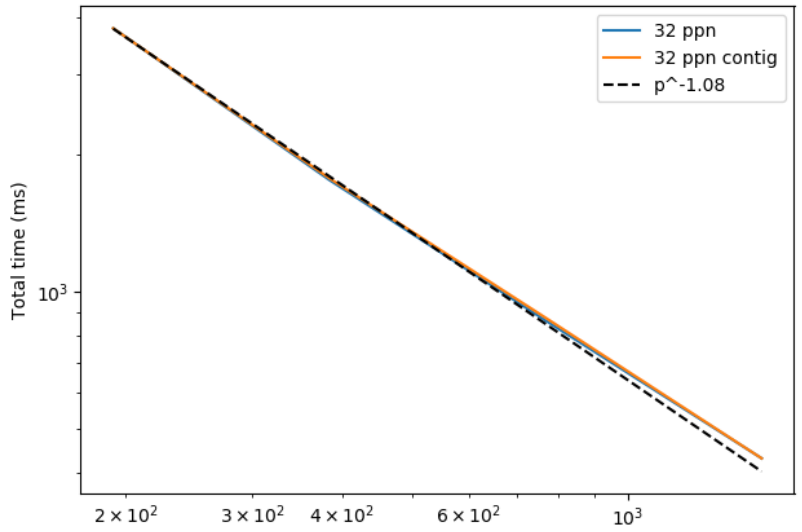
EXPERIMENTS

- Contiguous vs non-contiguous: In this approach we tested whether the '––contiguous' flag, which allocates job on contiguous nodes, has any effect on total time. We found that for certain grid sizes we get better strong scaling with this flag.
- Bandwidth test: Here we find what is bandwidth per wire, we are actually getting, compared to the documented bandwidth.
- Morton Order: This is a job-placement scheme which is a compromise between row-major and column-major placement.
- MPI_Vectors vs Local rearrangement: Passing 3D data to Alltoall required some tricks. Either we can use MPI_Vectors or rearrange the data ourselves. We compared time for both cases.
- Job Placement: In this approach we place the job on specific nodes that have physical all-to-all connection. Using this approach we were able to reduce the communication time by 10% compared to regular scheduling.

Contiguous vs non-contiguous

- SLURM scheduler has a flag that allocated contiguous nodes.
- This is enabled by '#SBATCH −−contiguous' in the jobscript file.
- We tested it for two grid sizes 2048^3 and 1536^3 and up to 2048 and 1536 cores respectively.
- The scaling of the code is modelled by the equation $T = p^{-\gamma}$, where T is the time taken to perform the transforms, p is the the number of processors used.
- In the ideal case, when the number of processors is doubled, the time must come to half, which makes $\gamma = 1$.
- For the performed grid sizes, we see that for 2048^3, non contiguous gives $\gamma = 0.9$ and contiguous gives $\gamma = 1$. But for 1536^3, we get $\gamma = 1$ for contiguous as well as non-contiguous cases.

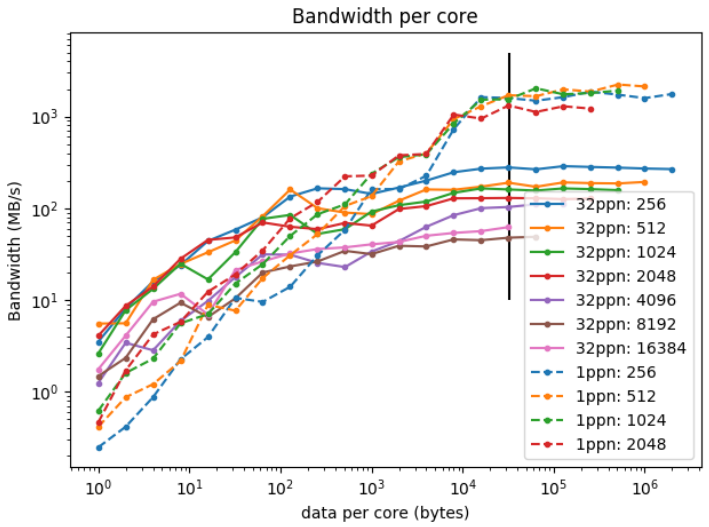




Scaling of FFTK on grid 2048^3 and 1536^3 respectively showing total time with −−contiguous flag and without it. Here ppn means processor per node and 'contig' refers to contiguous.

Bandwidth test

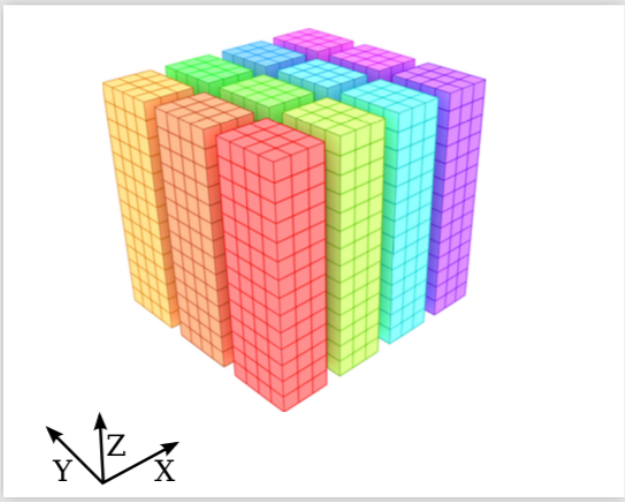
- It is documented that network wires of Shaheen II have a bandwidth of 14 GB/s and ideally per-node-bandwidth should match this value.
- We used mpiBench to find the sustained bandwidth reached in Alltoall and found that for 1 ppn we are getting around 2GB/s per process and for 32 ppn, we get bandwidth per process from 300MB/s at 256 nodes down to 40MB/s at 16384 nodes.
- By comparing node bandwidth, for 1ppn we get 2GB/s and for 32ppn we get 9GB/s at 8 nodes to 1GB/s at 512 nodes. The benchmark is shown in the Figure.

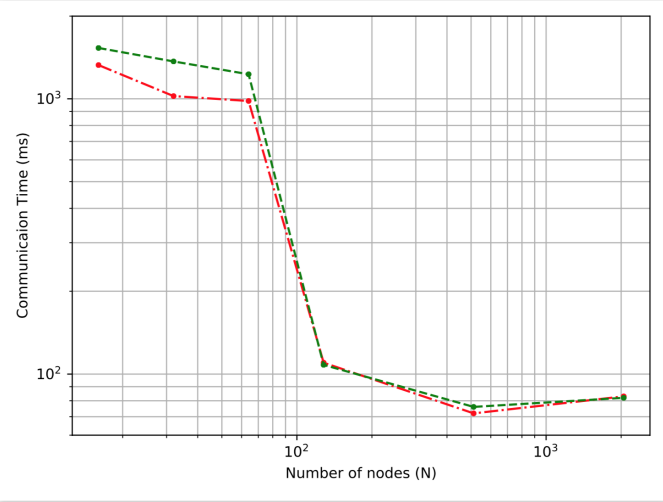


Benchmarking results using mpiBench. We see that per core bandwidth goes maximum up to 2GB/s for 1ppn and per node bandwidth goes up to 9GB/s for 32ppn.

Morton Order

- FFT when done in pencil decomposition, the 3D grid looks like a 2D array of pencils from the top, these pencils interact either row-wise or column-wise during different phases of communication.
- In row-major MPI ranking, the row processes reside on nearby nodes whereas column processes are well separated in the cluster.
- From Experiment A, we have seen that communication in nearby nodes is faster compared to discontinuous nodes.
- So we decided to use Morton order, which is a compromise between row-major and column-major ordering.
- MPI rank ordering is shown for both cases in the following tables.





Scaling of FFTK with Morton-order MPI ranking. Here x-axis represents the number of nodes used with 32ppn.

TABLE 1 Row-major ordering

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

TABLE 2 Morton ordering

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

TABLE 3 Percentage improvement

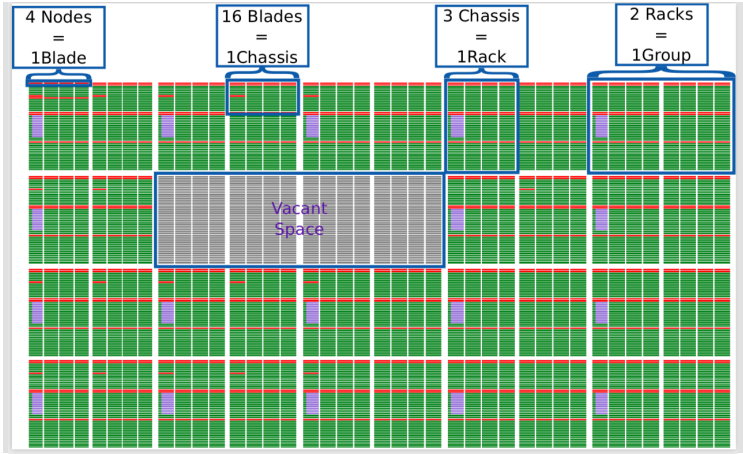
p	Row-major	Morton	Percentage
512	1326	1530	-15.4%
1024	1023	1367	-33.6%
2048	982	1228	-25.1%
4096	110	108	1.8%
16384	72	76	-5.6%
65536	83	82	1.2%

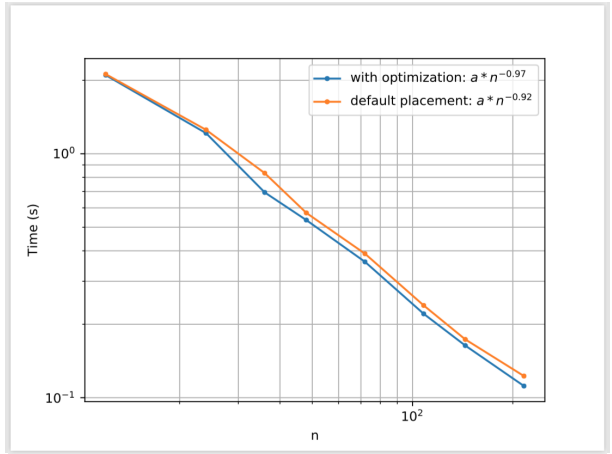
MPI_Vectors vs Local rearrangement

- MPI_Alltoall is a crucial component of FFT. When we pass data to this function, it assumes that the data is 1D. It divides the 1D data evenly among all processes and distributes it.
- Passing a multidimensional data into MPI_Alltoall is a bit tricky. Either we can use MPI_Vector or rearrange the array ourselves.
- We have tried both cases up to 65535 processors and found that both of them take the same time.

Job Placement

- Due to many levels of physical all-to-all connections we decided to explore whether we can use such physical connections to improve MPI_Alltoall performance.
- To do this experiment we needed to produce a map between sequential node numbers and it's location in the cluster.
- For this we created a web based Nid-marker. This tool takes the node-id list and visualises its location in the cluster.
- We studied performance on directly connected nodes, across 18 groups.
- Here, in the resulted plot, the blue line represents communication time when placed on directly connected nodes and the orange line represents communication time when placed on default allocated nodes.
- Here we save around 100 milliseconds (ms) per transform at 36 nodes and around 10ms at 216 nodes saving around 10% of the total communication time.
- Also, we get a better scaling when jobs are placed on directly connected nodes as shown in the plot.
- Numbers of percentage improvement are given in Table. 4.
- The present study was performed with 1ppn and used only one node of connected blades.
- We will study the same with 32ppn with all 4 nodes in the respective blades. Then we can go up to 27,648 cores.





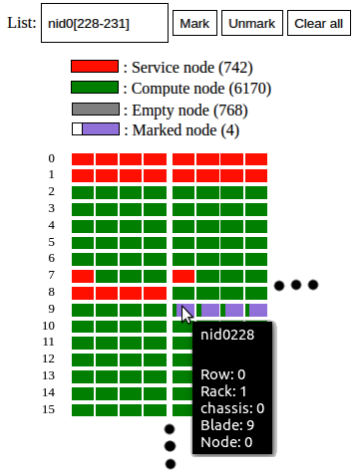


TABLE 4 Percentage improvement

p	Optimized (ms)	Default (ms)	Percentage improvement
12	2100	2119	0.9%
24	1214	1253	3.1%
36	694	833	16.6%
48	535	572	6.6%
72	360	389	7.4%
108	221	239	7.6%
144	164	173	5.6%
216	112	123	9.0%

CONCLUSION

- In this paper we have evaluated various factors that may impact the communication time of parallel Fourier Transform.
- We found that the impact of job placement improves the performance of parallel Fourier Transform, tested on FFTK library, on a Dragonfly network cluster, Shaheen II.
- The evaluations, so far, demonstrate that when the jobs are placed on directly connected blades, we get 10% reduction in communication time.
- We also found that scaling exponent within a chassis is greater by 7% compared to those beyond that.
- This study is also applicable to other supercomputers based on this topology

  and almost all other parallel FFT library as they use MPI_Alltoall for communication. Moreover,

  the Cray Exascale machines (Shasta) announced

  at the CUG will most likely use the dragonfly topology or something similar.

Collaborators

Dr. Anando Chatterjee

Prof. Mahendra Verma

Prof. David Keyes

IITK

KAUST

2020 CUG Cray User Group

https://cug.org/proceedings/cug2020_proceedings/includes/files/spec108s1.pdf

ACKNOWLEDGMENT

The authors thank the members of the Supercomputer Laboratory at King Abdullah University for providing the necessary resources and guidance. This research was supported by the Extreme Computing Research Center (ECRC) at KAUST and by the Simulation and Modelling Laboratory at IIT Kanpur.

Contacts

4700 KAUST Thuwal 23955-6900 JEDDAH, KSA

@ samar.aseeri@kaust.edu.sa

@samar_hpc & @Saudis_in_HPC

http://www.fft.report/