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

Samar A. Aseeri
ECRC, KAUST
Thuwal, Saudi Arabia
samar.aseeri@kaust.edu.sa

Anando Gopal Chatterjee
CC, IIT Kanpur
Kanpur, India
anandogc@iitk.ac.in

Mahendra K. Verma
Dept. of Physics, IIT Kanpur
Kanpur, India
mkv@iitk.ac.in

David E. Keyes
ECRC, KAUST
Thuwal, Saudi Arabia
david.keyes@kaust.edu.sa

***Abstract*—The fast Fourier transform (FFT) has applications in almost every frequency related studies, e.g. in image and signal processing, and radio astronomy. It is also used to solve partial differential equations used in fluid flows, density functional theory, many-body theory, and others. Three-dimensional N^3 FFT has large time complexity $O(N^3 \log_2 N)$. Hence, parallel algorithms are made to compute such FFTs. Popular libraries perform slab division or pencil decomposition of N^3 data. None of the existing libraries have achieved perfect inverse scaling of time with $(T^{-1} \approx n)$ cores because FFT requires all-to-all communication and clusters hitherto do not have physical all-to-all connections. With advances in switches and topologies, we now have Dragonfly topology, which physically connects various units in an all-to-all fashion. Thus, we show that if we align the all-to-all communication of FFT with the physical connections of Dragonfly topology we will achieve a better scaling and reduce communication time.**

***Index Terms*—Parallel FFT, Communication time, Efficient job scheduling**

I. Introduction

The fast Fourier transform (FFT) is a key building block for algorithms in computational science and engineering. It has an operational complexity that grows only slightly faster than linear growth with the input data size; but due to its heavy dependency on communication through network, it deviates from linear scaling. It is used in signal processing to encode audio and video data for efficient transmission, storage, and analysis and also to obtain solutions for partial differential equations. Moreover, other algorithms for applications in wave propagation (such as seismic inversion), and diffusion (such as hydrocarbon reservoirs), solid and fluid mechanics, and electromagnetism can be implemented using the FFT. In science and engineering domains, the FFT spectral methods are often found to be the most accurate numerical methods per unit memory; therefore, the scientific and engineering community still rely heavily on the Fourier transform. Thus, it is crucial to ensure the best possible algorithms for both the serial and parallel FFT, which is often the bottleneck of spectral codes. Current algorithmic implementations of the parallel FFT (and similar algorithms) require extensive long-range inter-processor communications, which results in a parallelization bottleneck. Nevertheless, it appears that the FFT will still be needed on exascale computers; therefore, improving the practical performance of FFT software is of interest.

A. FFT Algorithm

Forward Fourier transform is given by the following equation.

$$\hat{f}(\mathbf{k}) = \sum_{k_x,k_y,k_z} f(x,y,z)e^{(-ik_xx)}e^{(-ik_yy)}e^{(-ik_zz)} \quad (1)$$

The beauty of this sum is that k_x, k_y, and k_z can be summed independent of each other. These sums are done at different phases of the process summarized below.

- FFT along Z axis
- Communicate Z pencils to Y pencils.
- FFT along Y axis
- Communicate Y pencils to X pencils.
- FFT along X axis

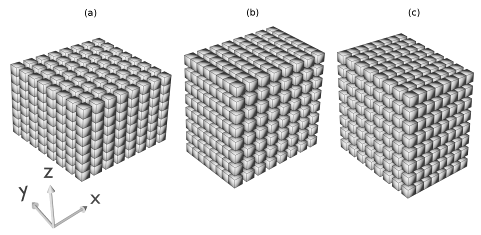


Fig. 1. Three stages of FFT

If we take Figure 1(b), we can see that vertical process communicate to form 1(a) while horizontal process communicate to form 1(c)

B. Background

A wide range of solutions have been proposed to optimize and speed-up parallel FFT algorithmic approaches. The Extreme Computing Research Center (ECRC) at King Abdullah University of Science and Technology (KAUST), in collaboration with the University of Oslo, Norway, has developed a new efficient implementation of the FFT that rides on the back of the little-known features of the message passing interface (MPI) used for communication in essentially all parallel computers, see (1). This feature MPI_ALLTOALLW enables some of the work that is traditionally done in the CPU, namely transposing data between successive stages of the FFT in multiple dimensions, to be done by the network hardware, thus improving FFT efficiency. Moreover, relatively limited expert coding is required to employ this feature, as vendors generally optimally tune the MPI library for their own hardware.

As part of a recent initiative by Samar Aseeri and Benson Muite (2) to incite research topics addressing parallel FFT optimization, a workshop on the parallel fast Fourier transform (PFFT18) was held at HiPC in 2018 (3). The contributed papers addressed the following topics:

- Exploiting high-speed half precision hardware in computing FFT by developing a mixed precision method to preserve accuracy (4).
- Impact of variations between FFTW2, FFTW3, and MKL on modern multicore processors. The authors of this study proposed three solution approaches for the optimization of 2D FFT by removing performance variations (5).
- A new exascale framework FFTX was proposed (6) to maximize performance and productivity when developing FFT-based applications; it is based on SPIRAL which is an FFT library generation autotuning framework (7).

Daisuke Takahashi, an invited speaker at PFFT18 has discussed his efforts on overlapping communication and computation in FFT implementation to obtain better performance (8). He also discussed his recent porting of the parallel 1-D FFT with automatic tuning on a cluster of Intel Xeon Phi processors (9). In the keynote address of the workshop, Mahendra Verma described an FFTK software library (10), which is tested in this current work it was developed to scale efficiently for the TARANG CFD software (11). TARANG is designed to solve turbulence problems and has received Dr. A.P.J. Abdul Kalam Cray HPC award in 2018. Anando Chatterjee, the final invited speaker, discussed aspects of the current work in which the Dragonfly network topology (12) of Shaheen II the Cray XC40 at KAUST (13) were examined together with the FFTK package in many ways to improve performance.

Here, among a number of experiments we mainly present some results of a method by which we can improve the communication time of FFT. We have run these tests on Shaheen which is a Cray-XC40 supercomputer placed in King Abdullah University of Science and Technology, Saudi Arabia. The topological structure of Cray-XC40 is such that many units of this system are connected in all-to-all fashion. We can distribute the MPI processes to a specific such connected nodes and see whether we get any improvement.

For a thorough description of the performance characteristics of FFT (computation, communication and how it is commonly made parallel, etc) and more refer to our authors previous publications (1) and (10).

The Dragonfly network consists of groups of routers connected together using all-to-all links, wherein each group has at least one link directly connected to the other group. Dragonfly interconnection networks on a Cray XC40 was analyzed in (14) to study the performance impact and interconnect behavior for multi-job completion for a real-world application. Another study (15) using Dragonfly on the Edison machine at NERSC (16) explores the effects of job placement and network configuration.

C. *Related Work*

The effects of task placement on Dragonfly networks has been addressed in some studies but to the best of our knowledge none has discussed its impact on the all-to-all FFT algorithm specifically.

Prisacari et al. (17), (18) compares the impact of different job placement policies and routing strategies.

Yang et al. (19) studies the effects of task placement on Dragonfly network and further investigates the contiguous versus non-contiguous allocation effects on applications.

Zhang et al.(20) introduces an allocation policy for dragonfly networks where jobs are spread within the smallest network level such that a given job can fit in at the time of its allocation.

On the topic of rank-reordering, Esposito et al. (21) analyzes the combination of grid topologies and rank reordering for different matrix sizes and number of nodes of a Cray XC system.

II. Motivation

The aim of this paper is to implement some strategies to reduce communication time of the extensively used FFT algorithm. For instance 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.

III. Method

Baseline is to speed up FFTs on Cray XC40 machine by using the full bandwidth offered by the cluster. For testing we used the FFTK parallel library developed by collaborators of this work. Several job placements and reordering cases and other were examined. Before the display of experiments we will define the system, network, library features and the tools considered for this work.

A. *Shaheen Supercomputer*

Shaheen is a Cray XC40 machine it is the 45 fastest supercomputer in the world it consists of 196,608 CPU cores and 790 TB of memory each core running at a clock speed of 2.3 GHz. It manages a speed of about 7 Petaflops/s theoretical peak and 5.5 Petaflops/s of Linpack performance. Its 6174 compute node are contained in 36 water-cooled XC40 cabinets all are connected via the Aries High speed network so-called Dragonfly. Cabinets are distributed into 18 groups of two cabinets each cabinet consists of 3 chassis and each chassis consists of 16 blades. A blade contain tightly integrated 4 nodes each is a dual-socket compute node each socket contains two Intel Haswell processors with 16 cores. Physical structure of Shaheen is schematically visualized in Fig. 2.

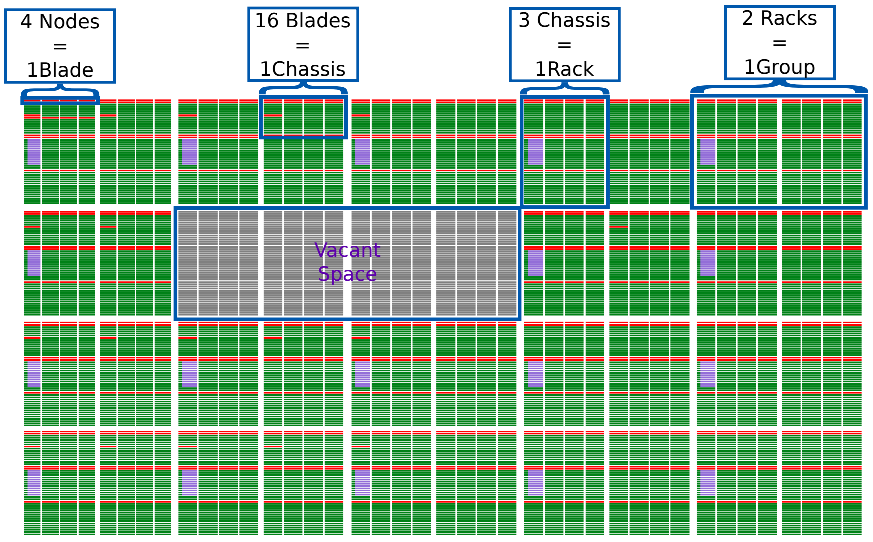


Fig. 2. Schematic representation of Shaheen's physical structure displaying it's `blades`, `chassis`, `racks` and `groups`. Here, the red nodes are service nodes, green are compute nodes, and violet nodes are used for an experiment described in Sec. IV-E

B. Dragonfly Networks

Dragonfly network is a hierarchical topology that mandates all-to-all physical connection at the highest level. Cray-XC40 uses all-to-all at all three levels with blades as units. Topologically, this results in a cube like structure with an all-to-all connection along all three axes. This structure is shown in Fig. 3, to illustrate further, there are 6 chassis of 16 blades within a group, and there are 18 such groups. This makes a $6 \times 16 \times 18$ blade cube that can be visualized in the figure. A very interesting connection pattern exists in this cube. There is all-to-all connection along all three axis shown by thin red, green and blue lines. Point to be noted are

- each chassis contain 4 to 9 service nodes which makes 55 to 60 usable compute nodes per chassis,
- maximum number of directly connected blades as shown in below figure is a slice which contains $16 \times 18 = 288$ blades inclusively containing $288 \times 4 = 1152$ service nodes.

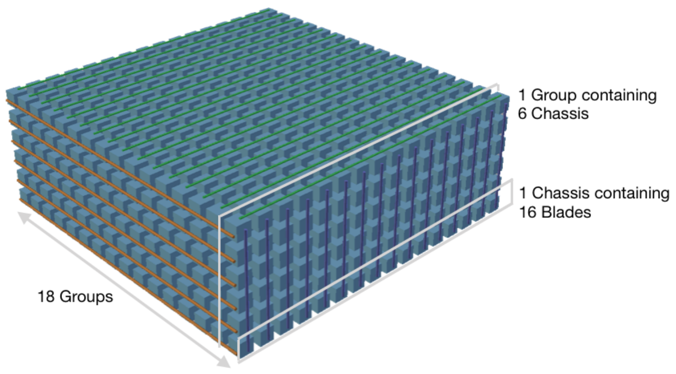


Fig. 3. Schematic representation of Shaheen's topology. Each cube represents a blade.

C. FFTK Library

FFTK or FFT Kampur library was developed by Chatterjee, Verma, and group members of IIT Kanpur. It is designed for three dimensional N^3 large grid FFTs. It scaled up to 196,608 cores on Shaheen II of KAUST for 3072^3 and 6144^3 grid points (10). The pseudospectral fluid solver TARANG uses it. The 2D pencil decomposition is typically used for large core counts. It also works for 2D data by setting N_y=1. Slab FFT can be performed by setting p_{row}=1. Available basis are: FFF, SFF, SSF, SSS, where 'F' signifies exponential Fourier transform along the axis which is used for periodic boundary condition, and 'S' stands for Sin/Cos Fourier transform along the axis which is used for non-periodic boundary condition.

1) Scaling: We scaled FFTK up to all the 6144 nodes using 32 processor per node (ppn) and the data points is fitted with model $T = C \cdot p^{-\gamma}$ (10). This scaling is shown in Figure 4.

In this graph we find that when the communication is taking place within a chassis where all blades are all-to-all connected with each other (Region I, $N \leq 60$) the scaling exponent is $\gamma = 0.84$, whereas beyond the chassis boundary ($N > 64$ to $N \leq 6144$), the scaling exponent is $\gamma = 0.78$ which is less than that of Region I. Moreover we find that there is a sharp change in total time when the communication pattern changes. Thus if we allocate the nodes such that the all-to-all communication takes place over directly connected nodes we should get a better scaling.

2) Further aspects of Scaling and Efficiency: A comparison of FFTK and P3DFFT was conducted on a Blue Gene/P machine in (10) for 8192 nodes with 1ppn and 4ppn and FFTK was equally efficient with P3DFFT. Also, (10) reports scaling of FFTK on Cray XC40 for up to 196608 processors, whereas, P3DFFT library has scaled up to 65536 cores on a Cray XT5 in (22) and so far no other groups have reported scaling studies for FFT on more cores than that of the FFTK.

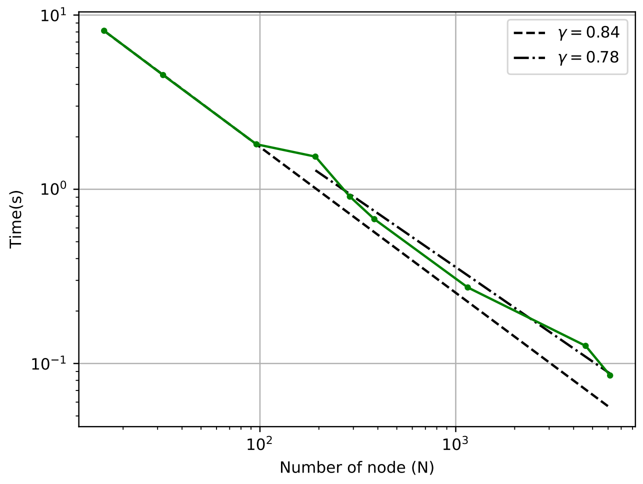


Fig. 4. Scaling of FFTK up to all 6144 nodes for 32ppn.

Moreover, (10) shows scaling results of fluid spectral solvers using FFTK on both Blue Gene/P and Cray XC40. One of the findings states that efficiency on Cray XC40 is lower than that of Blue Gene/P even though Cray XC40 is a more modern system but it appears that efficiency of interconnects has not grown proportionally with that of the processors.

D. NID Marker Tool

To have control on nodes allocation a useful tool called NID Marker (23) was designed for this work specifically for Shaheen II to help us visualize node locations and find node number. Part of the NID Marker is shown in Figure 5. First cabinet of two racks within a group is shown here for brevity. Refer to Figure 2 for a larger view of the tool showing all racks of Shaheen. The Service nodes are colored in red and Compute nodes are in green and the grey ones are vacant space.

IV. Experimental Results

Our concern here is to optimize communication, therefore, we tested five approaches.

- *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.

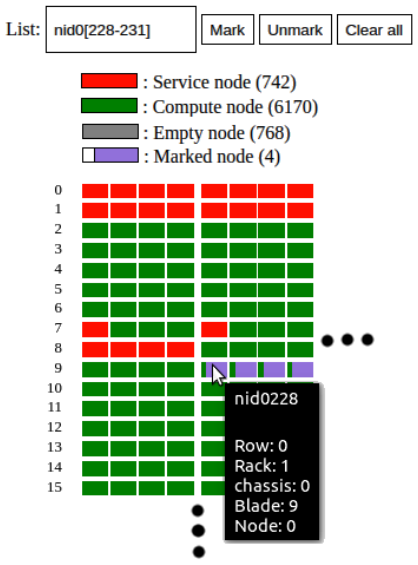


Fig. 5. NID Marker: First cabinet of two racks within a group is shown here. Each rack contains three chassis, and there are 18 groups of two racks.

A. Contiguous vs non-contiguous

SLURM scheduler (24) has a flag that allocated contiguous nodes. This is enabled by '#SBATCH −−contiguous' in the jobscript file.

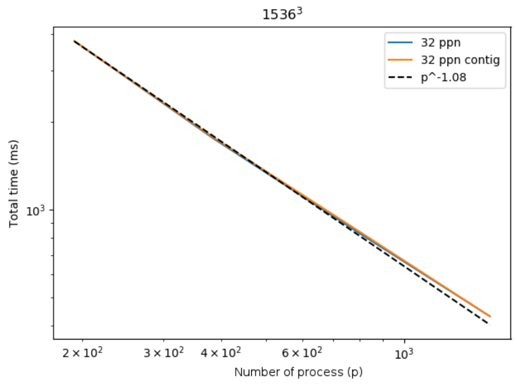
We tested it for two grid sizes 2048^3 and 1536^3 and up to 2048 and 1536 cores respectively. The scaling of the code is modelled by the following equation from (10). In the ideal case, when the number of processors is doubled, the time must come to half, which makes $\gamma = 1$.

$$T = p^{-\gamma} \tag{2}$$

where T is the time taken to perform the pair transforms, p is the the number of processors used. 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. The scaling is shown in Fig. 6

B. 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 (25) to find the sustained bandwidth reached in Alltoall and found that for 1 ppn we are getting around 2GB/s per processor and for 32 ppn, we get bandwidth per processor from 300MB/s at 256 nodes down to 40MB/s at 16384 nodes. By comparing node bandwidth, for



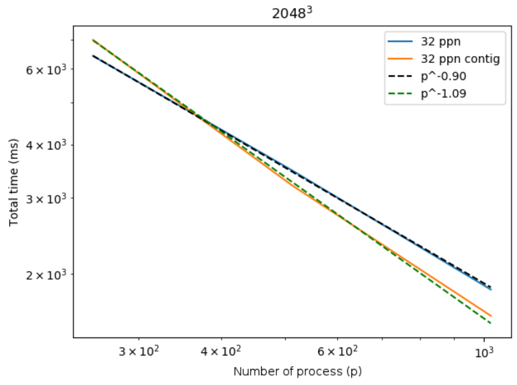


Fig. 6. Scaling of FFTK on grid 1536^3 and 2048^3 respectively showing total time with −−contiguous flag and without it. Here ppn means processor per node and 'contig' refers to contiguous.

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 Fig. 7

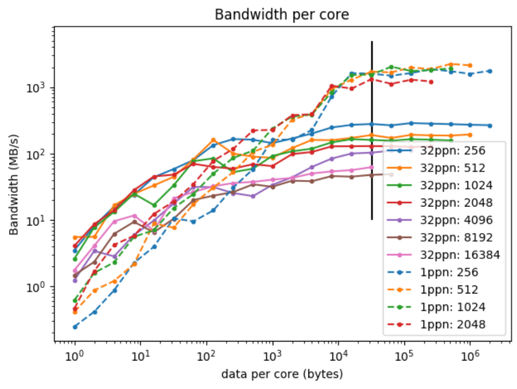


Fig. 7. Benchmarking results using mpiBench. We see that per core bandwidth goes maximum up to 2GB/s for 1ppn and per node bandwidth goes up to 9GB/s for 32ppn.

C. 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 process reside on nearby nodes whereas column process 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 (26), which is a compromise between row-major and column-major ordering. MPI rank ordering is shown for both cases in the following tables.

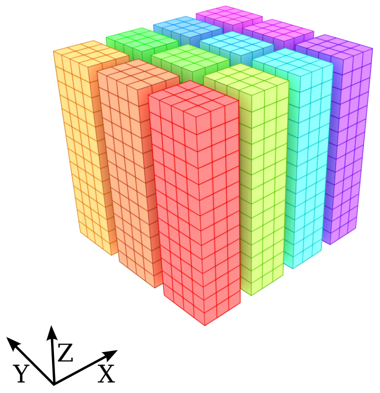


Fig. 8. 2D process grid (pencils) for a 3D data grid

TABLE I
ROW-MAJOR ORDERING

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

TABLE II
MORTON ORDERING

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

TABLE III
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%

In Fig. 9 the X-axis represents number of nodes used. We performed this test with 32ppn. We found that for lower cores, Row-major takes up to 33% less time compared to Morton ordering. The reason is that in row-major ordering, many row-process reside in same node whereas in Morton ordering, neither row-processes nor the column-process reside in the same nodes. Also we found that there is a sudden change in graph pattern beyond the chassis boundary.

D. 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 processors and distributes it. Passing a multidimensional data into MPI_Alltoall is a bit

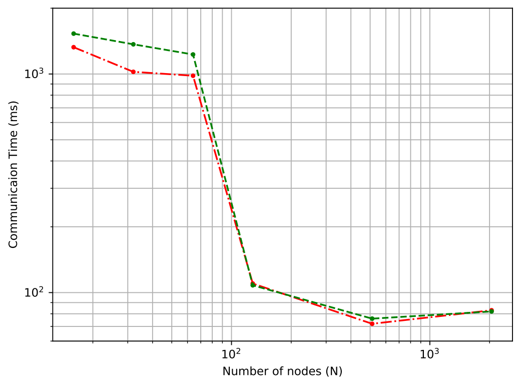


Fig. 9. Scaling of FFTK with Morton-order MPI ranking. Here x-axis represents the number of nodes used with 32ppn.

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.

E. Job Placement

Upon analysis of the topology we see that there is a nice pattern of connection between nodes. Topologically Shaheen looks like a cube , see Fig. 3, where

- each cube represents a blade,
- all blades in a chassis are connected directly by all-to-all connection,
- n^{th} blade of all chassis in a group are connected directly by all-to-all connection,
- n^{th} blade of m^{th} chassis of all groups are connected directly by all-to-all connection.

where, n goes from 1 to 16 and m goes from 1 to 6.

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 (23), described in (Sec. III-D). This takes the node-id list and visualises its location in the cluster.

We studied performance on directly connected nodes, visualized in Fig. 2, across 18 groups. Here, in the resulted plot, Fig. 10, 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 Fig. 10. The numbers of percentage improvement are given in Table. IV. 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.

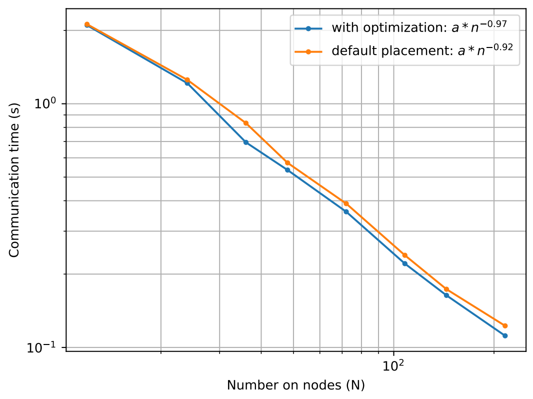


Fig. 10. Scaling of FFTK in 1) directly connected blades, and 2) default allocation by SLURM.

TABLE IV
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%

V. Conclusion and future work

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 (Sec. IV-E). We also found that scaling exponent within a chassis is greater by 7% compared to those beyond that (Sec. III-C). 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 User Group (CUG 2019) (27) has announced three Exascale Cray machines (Shasta) (28) which will most likely use the dragonfly topology or something similar. The present study on job placement was done with 1ppn, we plan to do it with 32ppn using all nodes in the respective connected blades.

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.

References

[1] Dalcin, L., Mortensen, M., Keyes, D. E. (2019). Fast parallel multidimensional FFT using advanced MPI. J. Parallel Distrib. Comput., 128, 137-150 https://www.sciencedirect.com/science/article/pii/S074373151830306X?via%3Dihub

[2] http://www.fft.report

[3] https://hipc.org/pfft/

[4] Anumeena, S., Xiaohe, C., Eduardo D'A., Kwai W., Stanimire T., Optimizing the Fast Fourier Transform Using Mixed Precision on Tensor Core Hardware, 2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW), 3-7, https://ieeexplore.ieee.org/document/8634417

[5] Semyon, K., Ravi Reddy M., Alexey, L., Performance Optimization of Multithreaded 2D FFT on Multicore Processors: Challenges and Solution Approaches, 2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW), 8-17, https://ieeexplore.ieee.org/document/8634318

[6] Franz, F., Daniele, G.S., Anuva, K., Doru, T.M., Tze, M.L., Michael, F., Andrew, C., Peter, M., Van Straalen, B., Phillip C., FFTX and SpectralPack: A First Look, 2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW), 18-27, https://ieeexplore.ieee.org/document/8634111

[7] http://www.spiral.net/hardware/dftgen.html

[8] Daisuke T., Automatic Tuning of Computation-Communication Overlap for Parallel 1-D FFT, Proc. 2016 IEEE 19th International Conference on Computational Science and Engineering (CSE 2016), pp. 253–256 (2016). (short paper)

[9] Daisuke T., An Implementation of Parallel 1-D Real FFT on Intel Xeon Phi Processors, Proc. 17th International Conference on Computational Science and Its Applications (ICCSA 2017), Part I, Lecture Notes in Computer Science, Vol. 10404, pp. 401–410, Springer International Publishing (2017)

[10] Anando G. C., Mahendra K. V., Abhishek K., Ravi S., Bilel H., Rooh K., Scaling of a Fast Fourier Transform and a pseudo-spectral fluid solver up to 196608 cores, J. Parallel Distrib. Comput. 113, 77-91 (March 2018)

[11] http://turbulencehub.org/index.php/codes/tarang/

[12] John K., Wiliam J.D., Steve S., Dennis A., Technology-Driven, Highly-Scalable Dragonfly Topology, 2008 International Symposium on Computer Architecture, 77-88, https://ieeexplore.ieee.org/document/4556717

[13] https://www.hpc.kaust.edu.sa/

[14] Smith, S., Lowenthal, D., Bhatele, A., Thiagarajan, J., Bremer, P., and Livnat, Y., Analyzing inter-job contention in dragonfly networks, 2016. [Online]

[15] Bhatele, A., Jain, N., Livnat, Y., Pascucci, V., and Bremer, P-.T., Evaluating System Parameters on a Dragonfly using Simulation and Visualization. United States: N. p., 2015. Web. doi:10.2172/1241972

[16] https://www.nersc.gov/users/computational-systems/

[17] B. Prisacari, G. Rodriguez, M. Garcia, E. Vallejo, R. Beivide, and C. Minkenberg. Performance implications of remote-only load balancing under adversarial traffic in dragonflies. In Proc. of the 8th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip, INA-OCMC'14, pages 5:15:4, New York, NY, USA, 2014. ACM.

[18] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, Efficient task placement and routing of nearest neighbor exchanges in dragonfly networks, in Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, ser. HPDC '14. ACM, 2014, pp. 129140. [Online]. Available: http://doi.acm.org/10.1145/2600212.2600225

[19] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, Watch out for the bully! Job interference study on dragonfly network, SC, pp. 750-760,2016

[20] Y. Zhang, O. Tuncer, F. Kaplan, K. Olcoz, V. J. Leung, and A. K. Coskun. 2018. Level-Spread: A New Job Allocation Policy for Dragonfly Networks. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 11231132, https://doi.org/10.1109/IPDPS.2018.00121

[21] Aniello Esposito, David Keyes, Hatem Ltaief, and Dalal Sukkari, Performance Impact of Rank Reordering on Advanced Polar Decomposition Algorithms, Cray User Group (CUG), http://hdl.handle.net/0754/628026,2018

[22] D. Pekurovsky, P3DFFT: a framework for parallel computations for Fourier transforms in three dimensions, SIAM J. SCI. COMUT. 34(4)(2012)C192-C209.

[23] http://home.iitk.ac.in/~anandogc/nid_marker.html

[24] Yoo, A.B., Jette, M.A., Grondona, M., SLURM: simple linux utility for resource management, Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSP 2003. LNCS, vol. 2862, pp. 44-60. Springer, Heidelberg (2003). https//doi.org/10.1007/10968987_3

[25] https://github.com/LLNL/mpiBench

[26] J. Thiyagalingam, Is Morton layout competitive for large two-dimensional arrays yet?, Concurr Comput. 18(11)(2006) 1509-1539.

[27] https://cug.org

[28] https://www.cray.com/products/computing/shasta