Scalability Issues in FFT Computation

Alan Ayala[1(✉)], Stanimire Tomov[1], Miroslav Stoyanov[2], and Jack Dongarra[1,2,3]

[1] Innovative Computing Laboratory - UT, Knoxville, TN 37996, USA
aayala@icl.utk.edu
[2] Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA
[3] University of Manchester, Manchester M13 9PL, UK

Abstract. The fast Fourier transform (FFT), is one the most important tools in mathematics, and it is widely required by several applications of science and engineering. State-of-the-art parallel implementations of the FFT algorithm, based on Cooley-Tukey developments, are known to be communication-bound, which causes critical issues when scaling the computational and architectural capabilities. In this paper, we study the main performance bottleneck of FFT computations on hybrid CPU and GPU systems at large-scale. We provide numerical simulations and potential acceleration techniques that can be easily integrated into FFT distributed libraries. We present different experiments on performance scalability and runtime analysis on the world's most powerful supercomputers today: Summit, using up to 6,144 NVIDIA V100 GPUs, and Fugaku, using more than one million Fujitsu A64FX cores.

Keywords: Scalability · Parallel FFT · Hybrid systems

1 Introduction

The fast Fourier transform (FFT) is a key mathematical tool and widely used in a variety of fields in science and engineering. In essence, the FFT of x, an m-dimensional vector of size $N := N_1 \times N_2 \times \cdots \times N_m$ is defined by $y := FFT(x)$, which is obtained as follows,

$$\tilde{y} := \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)}, \tag{1}$$

where $\tilde{y} = y(k_1, k_2, \ldots, k_m)$, and $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

From Eq. 1, we see that the FFT could be directly computed by a tensor product. However, this would cost $\mathcal{O}(N \sum_{i=1}^{m} N_i)$, while the advantage of the FFT is that the cost can be reduced to $\mathcal{O}(N \log_2 N)$ operations.

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations (the Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem.

V. Malyshkin (Ed.): PaCT 2021, LNCS 12942, pp. 279–287, 2021.
https://doi.org/10.1007/978-3-030-86359-3_21

The parallel FFT is implemented by a sequence of 1-D or 2-D FFTs, see e.g., [13], which are computed using efficient intra-node optimized libraries, such as FFTW [11] and CUFFT [1]. Figure 1 shows the steps to perform a 3-D FFT, typical in molecular dynamics, c.f., [14,17]. For some applications the input data has a shape ready to perform one-dimensional (*pencils*) or two-dimensional (*slabs*) FFTs and no initial nor final reshaping is needed. In [5], authors showed that saving one reshape step can accelerate the runtime around 25%, since, asymptotically, the multi-dimensional FFT runtime is dominated by the number of data-reshapes.

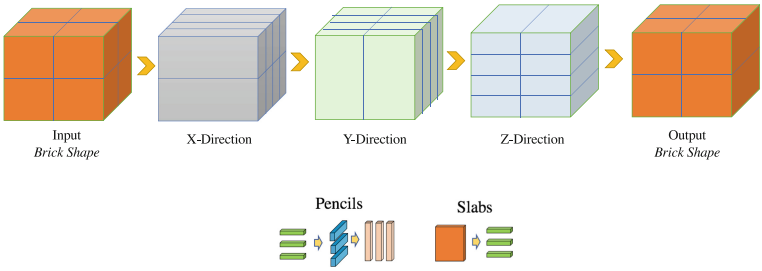

Fig. 1. Sequence for the computation of 3-D FFTs. If slab decomposition is possible, then an extra reshape step is saved.

In the current state-of-the-art, many authors have reported the impact of multi-process communication on distributed FFT performance [4,12,16,18], using both binary and collective MPI communication schemes that are available in current libraries. In this paper, we study these performance impacts from a numerical perspective, with a focus on architecture and algorithmic tuning for better performance. We analyze the effects of the communication bottleneck on scalability and provide techniques to maintain linear scaling. In Sect. 2, we make evident how FFT computation halts scaling even using latest efforts on MPI communication and their ability to perform CUDA-aware communication and specialized MPI for accelerators such as the NCLL library from NVIDIA [2]. This is critical for upcoming exascale system with millions of cores [6]. When addressing how network topology issues break scalability, we also provide techniques to prevent them.

Finally, the FFT is a key component for applications ranging from electronics to molecular dynamics. It is used at small and large scale; as within software targeting exascale (e.g., LAMMPS [14] and HACC [10]) and those from the machine learning community [15]. Such applications are being prepared for very large computing systems, with hybrid components and complex topologies. Therefore, it is critical to ensure parallel FFT scalability at large-scale.

2 Parallel FFT Performance Bottleneck

A major issue with distributed hybrid FFTs is that, due to the sheer compute capabilities of today's supercomputers, the algorithm quickly becomes communication bound. Such type algorithms, where already studied and authors in [8] warned of their effect on upcoming large-scale clusters. In [7], authors performed an extensive theoretical analysis on hybrid systems targeting exascale and realized that the FFT computation itself would take only a small fraction of the total run time, while the communication between processors would be the bottleneck where most of the run-time is spent. Nowadays, computing systems have greatly increased their computation power but their communication features have not been increased in the same proportion. For example, Summit supercomputer uses powerful nodes with two IBM POWER9 processors and six Nvidia V100 GPUs capable of reaching 42,000 GFlop/s in double precision, but the interconnect between the nodes is supported by a bandwidth of just 25 GB/s. Another supercomputer, the Sunway TaihuLight, has SW26010 processors with 260 cores, and 1 execution thread per core, with a unidirectional bandwidth of 8 GB/s between nodes and 1 μ of latency [9]. It therefore becomes critical to develop techniques and methodologies that help us of dealing with limited communication capabilities, together with an ecosystem of integrated tuning techniques for better communication frameworks. Such approaches are crucial in general and are paramount for the FFT, where communication can take more than 95% of total run-time on the latest GPU-accelerated supercomputers [4,5].

2.1 Scalability Issues

The recent developments of parallel FFT libraries capable of handling CPU and GPU components at the same time, has allowed considerable speedups in computation. However, this is highly limited by the communication bottleneck which has a considerable impact even for small-scale problems (due to latency issues) [5]. The bottleneck behaves different for every architecture and no general conclusion can be given on optimization criteria. For instance, experiments from [4,12,18] show that MPI All-to-All communication, was, in general, the best behaving methodology for data exchange in Summit-like architectures; however, as it can be seen in Fig. 2, in some systems, such as Fugaku, at large-scale, All-to-All (A2A) communication drastically fails to scale. An alternative for this case is to switch to binary MPI communication (P2P) which helps to keep a linear scaling. Note, however, that for a given problem size, if it is too small compared with the number of resources, then the scalability will also start to break, due to increased latency, see for example the P2P curve for the 256^3 problem. The experiment was performed using *heFFTe* library [3].

2.2 Peak Performance Model

When making a software contribution on parallel implementation, it is important to see how well the performance approaches to the machine theoretical peak.

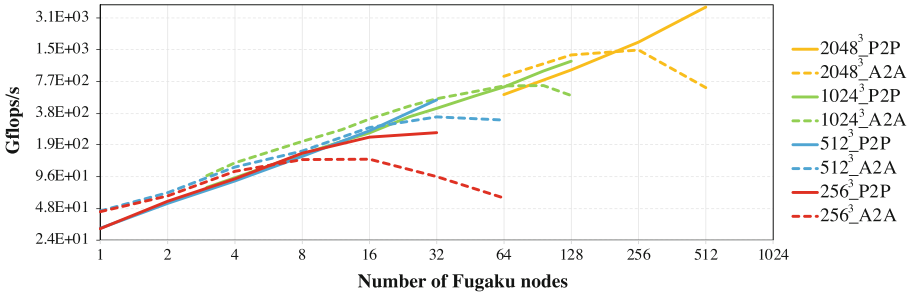

Fig. 2. Strong scaling of a 3-D FFTs. Using 48 A64FX cores per node. Comparing the scalability of A2A and P2P approaches using double-complex precision data.

Since the bandwidth injection between a single node is, in general, very high compared to inter-node injection. We developed a mathematical model for the theoretical performance peak on a supercomputer, c.f., [5, Sec. 3], given as:

$$\Phi := \frac{5P\log(N)B}{\alpha r}(GFlops/s), \tag{2}$$

where, the parameters are explained in Table 1.

Table 1. Parameters for communication model

Symbol	Description
N	Size of FFT
P	Number of nodes
r	Number of reshapes (tensor transpose, c.f., Fig. 1)
α	Size of datatype (Bytes)
B	Theoretical inter-node bandwidth (GB/s)

In Fig. 3, we show the roofline model for *heFFTe* v.2.0 on Summit and Fugaku, which have, respectively, 25 and 40.8 GB/s of inter-node theoretical bandwidth injection.

2.3 Choosing the Fastest FFT Parallel Algorithm

In Fig. 1 we see that there exists different ways to implement the parallel FFT, and it also depends on the user's data arrangement at input and output. For the sake of simplicity, let us consider a 3-D FFT, where the possible reshape combinations are (*B: Bricks, P: Pencils, S: Slabs*):

– **Pencils:** $B2P \rightarrow P2P \rightarrow P2P \rightarrow P2B$; this approach is the one available in libraries such as AccFFT [12] and FFTMPI [17].

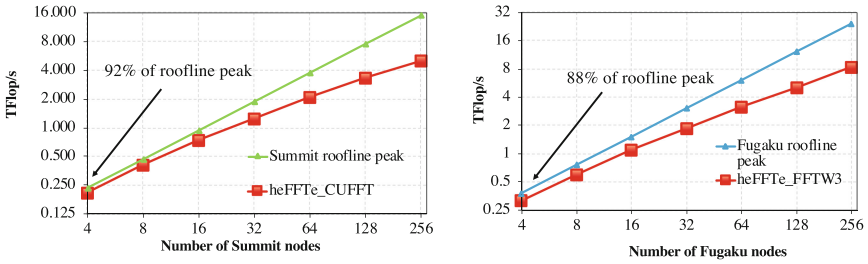

Fig. 3. Roof-line performance model—heFFTe performance on a 3-D FFT of size 1024^3 using 6 MPI/node, 1 GPU-Volta 100 per MPI for Summit, and 48 A64FX per node on Fugaku.

– **Slabs:** $B2P \rightarrow P2S \rightarrow S2B$; this approach uses a combination of pencils and slabs, and it is included in *heFFTe* library [3].

The choice of a given reshape sequence will depend on the type of architecture. Note that, for example, the number of messages for a P2P reshape is of the order of $P^{2/3}$, where P is the number of processors involved in the communication, c.f., Fig. 1. Hence, assuming 3-D double-complex data—and using Eq. 2 and the asymptotic number of messages sent by each of the reshape types, with $B = 25\,\text{GB/s}$ and $L = 1\,\mu\text{s}$—Fig. 4 is a phase diagram for Summit, which allows to choose the theoretical fastest decomposition to use. This offline pre-processing tuning strategy can help users to identify which 3-D decomposition to use for the FFT parallel algorithm. The proposed methodology can easily be extended to other supercomputers and higher dimension transforms.

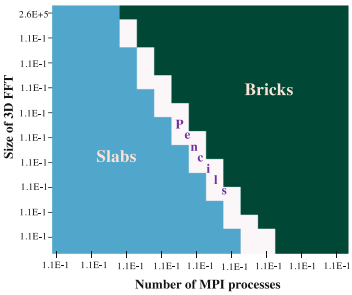

Fig. 4. Selection of the best reshape approach based on the 3-D FFT size and the number of resources.

3 Experimental Results

In this section, we present numerical experiments to support our analysis from previous sections. Since this paper targets large-scale computation, our results

were obtained using the two world's most powerful supercomputers today, with the following architectures:

- *Summit* at ORNL - USA, having 4,608 nodes, each consisting of 2 IBM POWER9 CPUs and 6 NVIDIA V100 GPUs. These 6 GPU accelerators provide a theoretical double-precision capability of approximately 40 TFLOP/s. Within the same node, processors have two NVIDIA NVLink interconnects, each having a peak bandwidth of 25 GB/s (in each direction), hence V100 and P900 can communicate at a peak of 50 GB/s (100 GB/s bi-directional).
- *Fugaku* at RIKEN - Japan, currently at testing stage, and has 158,976 nodes, each consisting of Fujitsu A64FX CPU. We use the maximum amount of number of resources currently allowed with 48 cores per node.

Experiments on this paper where performed using a state-of-the-art library for parallel FFTs: *heFFTe* version 2.0 [3], which reportedly provides considerable speedups with respect to its peers [4]. If not stated otherwise, our results display average values of 10 experiments (5 forward and 5 backward 3D-FFT computations) using double-complex precision random data and 4 data reshapes per direction (Input $\rightarrow$ X $\rightarrow$ Y $\rightarrow$ Z $\rightarrow$ Output).

3.1 Strong and Weak Scalability

Several authors have shown that parallel FFT runtime on large problems are highly due to MPI communication, which asymptotically takes more than 95% of runtime on hybrid systems, c.f., [4,5,12]. Hence, it is critical to select the fastest MPI (binary or collective) communication for the data exchanges required by parallel FFT distributions. In Fig. 5, we present weak and strong scalability on up to one million A64FX cores on Fugaku, this experiment clearly shows the effect on scalability of the selection of the Point-to-Point (MP2P) and All-to-All (A2A) communication frameworks, and its relationship with the number of resources. When dealing with hybrid systems, such as Summit supercomputer, the percentage of time spend on communication exploits, making the performance scaling highly dependent on the underlying MPI library, we explore the MPI selection in next subsection.

The strong scalability plot from Fig. 5, sheds light on how P2P communication is faster for large number of resources, and we verified this for medium sized allocation and employed the P2P approach for our largest experiments on the weak scalability plot. Figure 6 shows a weak scaling using AlltoAll communication on Summit, and using SpectrumMPI 10.3 with data striping enabled, we can get good linear scaling and this is faster than the P2P approach, which is the opposite situation compared to Fugaku.

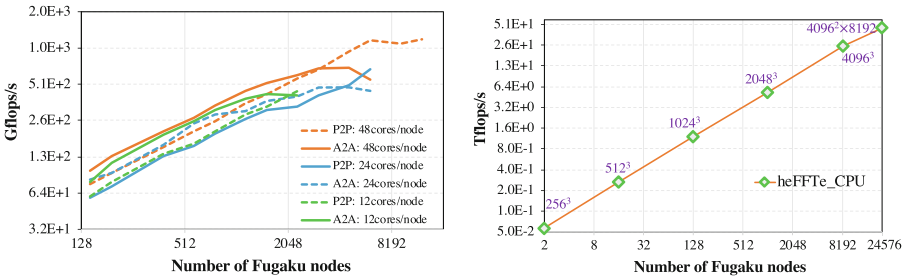

Fig. 5. *Left:* Comparison of strong scaling for a 3-D FFT of size 1024^3, using different node count. *Right* Weak scalability for different 3-D FFT sizes. For both experiments we use *heFFTe* with FFTW backend and 48 MPI processes (1 MPI processes per A64FX core) per node.

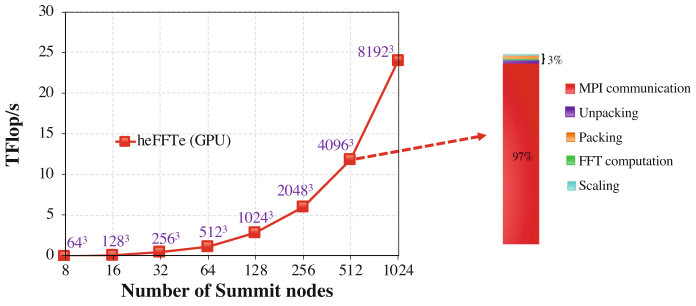

Fig. 6. Weak scalability for different 3-D FFT sizes on a hybrid architecture (Summit). Using NVIDIA CUFFT backend and 6 MPI processes (1 MPI processes per Volta 100 GPU) per node.

3.2 MPI Selection for Further Acceleration

In Sect. 3, we showed how the right reshaping algorithm can provide speedups of over 25% compared to default implementations. Next, assuming that the algorithm is fixed and properly chosen, we observed that to achieve linear scalability, it is very important to figure out how to optimally use the computational resources and architecture tools from manufactures to manually tune the port inter-connections to achieve maximum bandwidth injection. Therefore, let us analyze the parallel computing technologies in both, Summit and Fugaku, supercomputers:

– Fugaku uses a TofuD network topology, with three different types of options: *torus, mesh, noncont*. For our experiments we used MPIFCC provided with the Fujitsu compiler, and we enabled auto-parallelization using the *Kparallel* flag. We observe that the *torus* and *noncont* networks provided the best injection bandwidth. In theory, using the 6 available TofuD ports we can get a total of 40.8 GB/s theoretical bandwidth injection.

– Summit inter-node connections are not as fast as the NVLINKS available intra-nodes, and they are arranged on a non-blocking fat tree topology with dual-rail EDR InfiniBand network that provides a theoretical bandwidth of 25 GB/s. For our experiments we use IBM SpectrumMPI, which is optimized for this architecture.

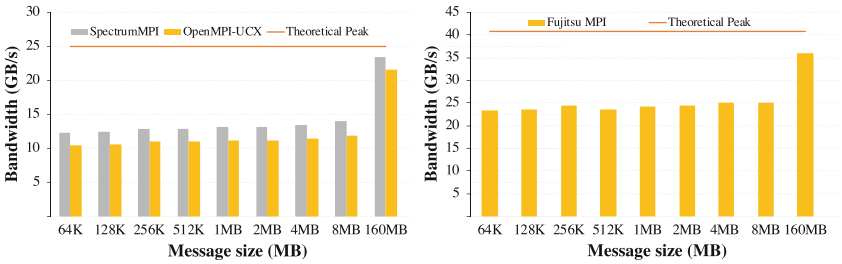

Fig. 7. Comparison of bandwidth injection obtained for different MPI implementations on Fugaku and Summit.

Information about the interconnections have to be obtained in advance and can be integrated to state-of-the-art libraries for auto-tuning, this feature is not, currently, supported by libraries covered in Sect. 1. Next, for a given FFT computation we can find the message size that will be transferred between nodes and Fig. 7 shows which MPI implementation offers the best bandwidth injection. For instance, for a $256 \times 256 \times 256$ double-complex (16 Bytes) precision FFT on 128 nodes, each processor communicates around 2 MB of data.

4 Conclusion

In this paper, we studied performance and scalability limitations of large-scale FFT computation on state-of-the-art CPU and GPU distributed systems. We provided methodologies to further accelerate parallel FFT by targeting software improvements on critical algorithm bottlenecks and making them aware of the underlying architecture. Our numerical studies and bounds on performance scalability can be generalized to all type of architectures (e.g., those from grid computing) and can be employed to make performance predictions. We finally presented experiments on today's top supercomputers, showing how carefully chosen system-aware parameters and algorithms can lead to very good linear strong and weak scalability.

References

1. cuFFT library (2018). http://docs.nvidia.com/cuda/cufft
2. NCLL library (2019). https://github.com/NVIDIA/nccl

3. heFFTe library (2020). https://bitbucket.org/icl/heffte
4. Ayala, A., et al.: Impacts of Multi-GPU MPI collective communications on large FFT computation. In: 2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI) (2019)
5. Ayala, A., Tomov, S., Haidar, A., Dongarra, J.: *heFFTe*: highly efficient FFT for exascale. In: Krzhizhanovskaya, V.V., et al. (eds.) ICCS 2020. LNCS, vol. 12137, pp. 262–275. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50371-0_19
6. Balaji, P., et al.: MPI on a million processors. In: Ropo, M., Westerholm, J., Dongarra, J. (eds.) EuroPVM/MPI 2009. LNCS, vol. 5759, pp. 20–30. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03770-2_9
7. Czechowski, K., McClanahan, C., Battaglino, C., Iyer, K., Yeung, P.K., Vuduc, R.: On the communication complexity of 3D FFTs and its implications for exascale (2012). https://doi.org/10.1145/2304576.2304604
8. Demmel, J.: Communication-avoiding algorithms for linear algebra and beyond. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (2013)
9. Dongarra, J.: Report on the sunway TaihuLight system. Technical report (2016)
10. Emberson, J., Frontiere, N., Habib, S., Heitmann, K., Pope, A., Rangel, E.: Arrival of first summit nodes: HACC testing on phase I system. Technical report, MS ECP-ADSE01-40/ExaSky, Exascale Computing Project (ECP) (2018)
11. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proc. IEEE **93**(2), 216–231 (2005). Special issue on "Program Generation, Optimization, and Platform Adaptation'
12. Gholami, A., Hill, J., Malhotra, D., Biros, G.: AccFFT: a library for distributed-memory FFT on CPU and GPU architectures. CoRR abs/1506.07933 (2015)
13. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Accuracy and Stability of Numerical Algorithms, 2nd edn. Addison Wesley, Boston (2003)
14. Large-scale atomic/molecular massively parallel simulator (2018). https://lammps.sandia.gov/
15. Lin, S., Liu, N., Nazemi, M., Li, H., Ding, C., Wang, Y., Pedram, M.: FFT-based deep learning deployment in embedded systems. In: 2018 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1045–1050 (2018)
16. Parallel 2d and 3d complex FFTs (2018). http://www.cs.sandia.gov/~sjplimp/download.html
17. Plimpton, S., Kohlmeyer, A., Coffman, P., Blood, P.: fftMPI, a library for performing 2d and 3d FFTs in parallel. Technical report, Sandia National Lab. (SNL-NM), Albuquerque, NM, USA (2018)
18. Takahashi, D.: Implementation of parallel 3-D real FFT with 2-D decomposition on Intel Xeon Phi Clusters. In: 13th International Conference on Parallel Processing and Applied Mathematics (2019)