Accelerating Multi-Process Communication for Parallel 3-D FFT

Alan Ayala[1], Stan Tomov[1], Miroslav Stoyanov[2], Azzam Haidar[3], and Jack Dongarra[1,2,4]

[1]University of Tennessee, Knoxville, TN, USA
[2]Oak Ridge National Laboratory, Oak Ridge, TN, USA
[3]Nvidia Corporation, Santa Clara, CA, USA
[4]University of Manchester, Manchester, UK

***Abstract*—Today's largest and most powerful supercomputers in the world are built on heterogeneous platforms; and using the combined power of multi-core CPUs and GPUs, has had a great impact accelerating large-scale applications. However, on these architectures, parallel algorithms, such as the Fast Fourier Transform (FFT), encounter that inter-processor communication become a bottleneck and limits their scalability. In this paper, we present techniques for speeding up multi-process communication cost during the computation of FFTs, considering hybrid network connections as those expected on upcoming exascale machines. Among our techniques, we present algorithmic tuning, making use of phase diagrams; parametric tuning, using different FFT settings; and MPI distribution tuning based on FFT size and computational resources available. We present several experiments obtained on Summit supercomputer at Oak Ridge National Laboratory, using up to 40,960 IBM Power9 cores and 6,144 NVIDIA V-100 GPUs.**

***Index Terms*—Exascale FFT, Hybrid systems, Scalability, MPI tuning**

I. Introduction

During the last two decades, major improvements in the development of supercomputers allowed to build faster and more efficient software for problems in science and engineering, which constantly grow in scale. One of the major challenges that these developments have had, consists on minimizing the cost of communication between processors, which depends on the bandwidth and latency of the system, features that have not been growing at the same rate as the arithmetic computational power. Thus, communication is known to be the bottleneck for important algorithms in computer science, and efficient algorithms need to be developed [1]. One of the most widely used algorithms for numerical applications is the Fast Fourier Transform (FFT), which was considered as one of the top 10 algorithms of the 20th century [2].

In essence, the FFT of x, an m-dimensional vector of size $N \equiv N_1 \times N_2 \times \cdots \times N_m$, is denoted $y = FFT(x)$ and defined as an m-dimensional vector the same size as x by the following equations:

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

where $\hat{y} \equiv y(k_1, k_2, \ldots, k_m)$ for $0 \leq k_i \leq N_i - 1, \quad i = 1, \ldots, m$, where $\bar{x} = x(n_1, \ldots, n_m)$.

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

There exist single-device efficient implementations to compute multidimensional FFTs. In this paper we focus on computing 3-D FFTs. One of the most widely used libraries for this purpose is FFTW [3], which has been tuned to optimally perform in several architectures. Vendor libraries for this purpose have also been highly optimized, such is the case of MKL (Intel) [4], ESSL (IBM) [5], rocFFT (AMD) [6] and CUFFT (NVIDIA) [7]. Novel libraries are also being developed to further optimize single-device FFTs, among them: OneAPI for Intel GPUs [8], Vulkan FFT (VkFFT) [9], KFR [10], and FFTX [11], where the latter is part of the ECP software community.

In the last decade, several distributed CPU-based FFT implementations have been developed. Algorithm 1 presents the classical parallel FFT algorithm with different parameter options. Figure 1 shows three different decomposition approaches used in FFT algorithms. Among the libraries, the widely-used FFTW employs a 1-D decomposition approach, which limits its scalability to a small number of nodes. P3DFFT [12] extends FFTW functionalities and supports both 1-D and 2-D decompositions. Libraries 2DECOMP&FFT [13], nb3dFFT [14] and AccFFT [15], showed good scalability but are no longer maintained. Finally, large scale applications have built their own FFT library, such as fftMPI [16] (built-in on LAMMPS [17]) and SWFFT [18] (built-in on HACC [19]). These libraries are widely known in the molecular-dynamics and astrophysics literature.

In the realm of hybrid architectures, the impact of the communication bottleneck needs to be efficiently managed to properly target exascale systems [20], [21]; which makes hybrid-distributed FFT a challenging task. Few state-of-the-art

2021 Workshop on Exascale MPI (ExaMPI) | 978-1-6654-1108-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/EXAMPI54564.2021.00011

978-1-6654-1108-0/21/$31.00 ©2021 IEEE
DOI 10.1109/ExaMPI54564.2021.00011

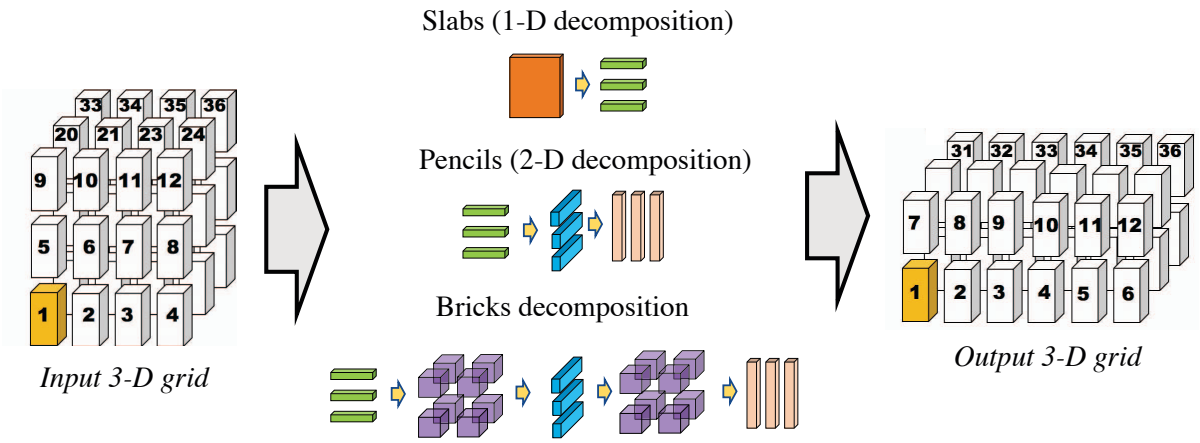

Fig. 1. The sequence of steps for computing the 3D FFT for different decomposition of the input tensor. The *pencil* decomposition (green bars throughout) is the default option in most libraries. Using *slabs* (orange box at the top) saves an extra step of data reshape. The *bricks* decomposition (purple translucent cubes) is supported by some libraries and could be beneficial on some platforms.

libraries exist for CPU-GPU architectures, and they have based their developments on minimizing the communication effect. For instance, the 1-D decomposition approach introduced in [22] is one of the first heterogeneous codes for large FFT computation on GPUs. Its optimization approach is limited to small number of nodes and focus on reducing tensor transposition cost by exploiting infiniband-interconnection using the IBverbs library, which makes it not portable. Further improvements to scalablity have been presented in FFTE library [23] which supports pencil and slab decompositions and includes several optimizations, although with limited features and limited improvements on communication. Also, FFTE relies on the commercial PGI compiler, which may limit its usage. In [15], authors developed AccFFT, a library that seeks to overlap computation and blocking collective communication by reducing the PCIe overhead, they provide good (sublinear) scalability results for large real-to-complex transforms using NVIDIA K20 GPUs. Finally, *heFFTe* [24]–[28], is a recent open source FFT library which targets exascale. It is open source and its optimizations yield to linear scalability for large node-count.

The standard algorithm, known as *pencil* decomposition, consists of a sequence of 1D FFTs for each of the m directions, c.f., [29]. A variant of this is the *slab* decomposition, which uses 2D FFTs, c.f., Figure 1. Alg. 1 shows the standard approach to compute a distributed FFT.

A. Related work

In [30], authors introduced a communication framework for accelerating the data-exchanges on CPU-based parallel FFTs, their approach has the potential to further speedup the libraries listed above.

Algorithm 1 m-dimensional FFT computation

1: **Input:** n-dimensional array, processor grids: P_{in}, P_{out}
2: Tune parameters based on array size and P_{in}
3: Define processor grids (MPI groups) for each direction
Call: Load balancing algorithm to distribute data.
4: Define best permutation order of computation
5: **for** $k \leftarrow 1$ to m **do**
6: Compute local 1-D FFTs
Call: *Single-device library, e.g., FFTW, CUFFT.*
7: Pack data
8: **for** P on my MPI group **do**
9: Exchange (Send/receive) local box with neighbors
Call: *Point-to-Point or Collective MPI*
10: **end for**
11: Unpack data
12: **end for**
13: Reshape data on P_{out} grid

B. Contributions

- We push the boundaries of the existing FFT software stack by introducing novel implementation improvements on the communication bottleneck with efficient MPI tuning.
- Our improvements yield to speedups on the computation of parallel 3-D FFT. Our implementation within *heFFTe* library achieves linear scalability and close to the peak performance.
- We introduce communication tuning parameters for different FFT algorithmic approaches and exchanges needed within the transposition kernels of state-of-the-art FFT libraries.

TABLE I
MPI ROUTINES USED FOR TENSOR TRANSPOSITION WITHIN DISTRIBUTED FFT LIBRARIES.

Library Name	Language	Developer	Point-to-Point exchange		Collective exchange		Process Topology
			Blocking	Non-Blocking	Blocking	Non-Blocking	
AccFFT	C++	GA Tech.	MPI_Sendrecv	MPI_Isend MPI_Irecv	MPI_Alltoallv MPI_Bcast	-	MPI_Cart_create
2DECOMP&FFT	Fortran	NAG	MPI_Send	MPI_Irecv	MPI_Alltoall MPI_Alltoallv	-	MPI_Cart_create MPI_Cart_sub
FFTE	Fortran	U. Tsukuba Riken	-	MPI_Isend MPI_Irecv	MPI_Alltoallv MPI_Alltoallw	MPI_Ialltoallv	-
FFTMPI	C++	Sandia	MPI_Send	MPI_Irecv	MPI_Allreduce MPI_Alltoallv	-	MPI_Group MPI_Comm_create
FFTW	C	MIT	-	-	MPI_Alltoallv	-	MPI_Group MPI_Comm_create
heFFTe	C++	UTK	MPI_Send MPI_Recv	MPI_Isend MPI_Irecv	MPI_Allreduce MPI_Alltoallv	heFFTe_Alltoallv	MPI_Group MPI_Comm_create
nb3dFFT	Fortran	RTWH Aachen	MPI_Send MPI_Recv	-	MPI_Allreduce MPI_Alltoallv	-	MPI_Group
P3DFFT	C++	UCSD	-	-	MPI_Alltoallv	-	MPI_Cart_create MPI_Cart_sub
SpFFT	C++	ETH Zürich	MPI_Send	MPI_Irecv	MPI_Allreduce MPI_Alltoallv	-	MPI_Group MPI_Comm_create
SWFFT	C++	Argonne	MPI_Send	MPI_Irecv	MPI_Allreduce MPI_Alltoallv	-	MPI_Cart_create MPI_Cart_sub

C. Paper organization

In Section II, we provide an extensive study of the communication bottleneck for computing a parallel FFT using MPI. We analyze the different MPI routines being used on a dozen of well-known libraries. We provide an offline tuning technique to accelerate the communication by using a phase diagram that can be built for CPU and CPU-GPU hybrid systems. Next, Section III presents several experiments on Summit supercomputer using up to 40,960 IBM Power9 cores and 6,144 NVIDIA V-100 GPUs. We show how the communication impact the performance of parallel FFT, and how our developments can help to mitigate its effects on scalability. Then, we present a roofline performance analysis. Finally, Section IV concludes our paper.

II. ACCELERATING MULTI-PROCESS COMMUNICATION

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. Authors in [20] 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 processes would be the bottleneck where most of the run time is spent. However, they did not provide an efficient library for such computation.

We refer to hybrid systems as those consisting of heterogeneous components, or multiple sub-systems (as in the case of grid computing). In this paper, we focus on hybrid platforms that use graphics processing units (GPUs), which can highly accelerate the execution of arithmetic operations; however, when a large amount of GPU-GPU communication is required, this could greatly affect performance gains of speeding up local computations. Algorithms that get affected by this issue are known as communication bounded algorithms, among them: the Fast Fourier Transform (FFT).

TABLE II
NUMBER OF MESSAGES REQUIRED TO BE SENT PER PROCESS DURING A 3-D RESHAPE USING A $P^{1/3} \times P^{1/3} \times P^{1/3}$ PROCESSOR GRID.

Reshape Type	# Messages
Brick $\Longleftrightarrow$ Pencil	$P^{1/3}$
Brick $\Longleftrightarrow$ Slab	$P^{2/3}$
Pencil $\Longleftrightarrow$ Pencil	$P^{2/3}$
Pencil $\Longleftrightarrow$ Slab	$P^{2/3}$
Slab $\Longleftrightarrow$ Slab	P

A. Quantifying the FFT communication bottleneck

Typical input data is given by the user in *brick* shape, which is defined over an 3-D data grid., c.f., Figure 1. To obtain the FFT transform in the x direction, data communication is required. For both possible final distributions (*pencils* or *slabs*), the number of messages is the same, as shown in Table II, where an input grid of size $P^{1/3} \times P^{1/3} \times P^{1/3}$ is assumed, P being the total number of processes. The number of messages is accounted for in every MPI group, which are defined from overlapping process topologies at consecutive reshape stages (i.e., an MPI group contains processes that need to exchange data before the 1-D [or 2-D] FFT computation on the next direction). Figure 2 shows the exponential growth on a number of messages for a 3-D FFT using *heFFTe* with a CPU and GPU backend, where all intra-node resources on Summit are used (i.e., the network was saturated). In [31], authors studied multi-GPU communication for FFT computation.

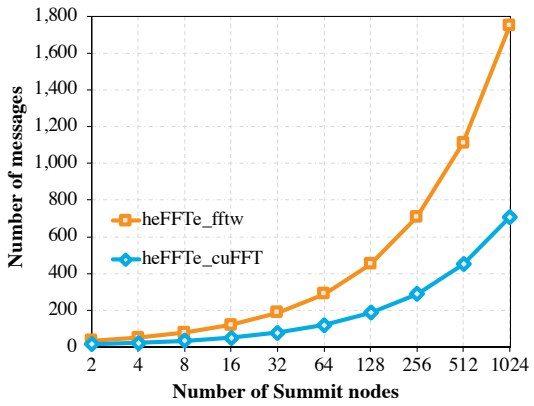

Fig. 2. Number of messages sent for a double-complex precision 3-D FFT of size 1024^3. Using a CPU and GPU backend, 6 V100 GPUs per node for cuFFT, and 40 IBM POWER9 CPUs per node for FFTW.

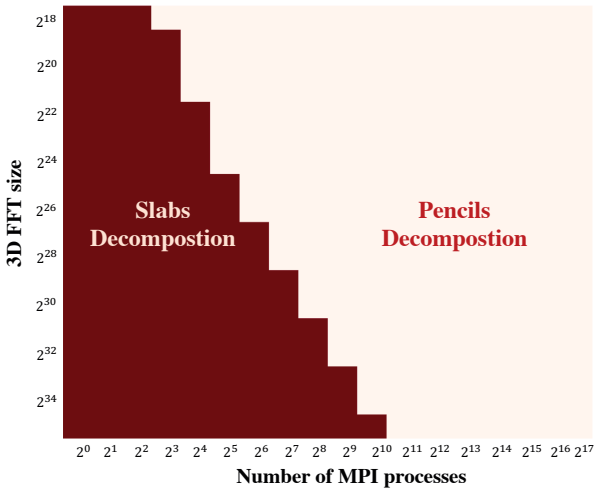

Fig. 3. Selection of the best reshape approach based on the 3-D FFT size and the number of resources.

Next, one can analyze which reshape procedure would be the most beneficial for computing a full FFT. For the sake of simplicity, one can consider a 3-D FFT, where the possible reshape combinations are *(B:brick, P:pencil, S:slab)*:

- **Pencils**: $B2P \rightarrow P2P \rightarrow P2P \rightarrow P2B$; this approach is the one available in the AccFFT, and fftMPI libraries.
- **Slabs**: $B2P \rightarrow P2S \rightarrow S2B$; this approach uses a combination of pencils and slabs, and it is included in *heFFTe* and FFTE libraries.

From Figure 1, we can clearly see that the slab decomposition reduces the number of data reshapes. However, its global advantage does not always hold, see Figure 9. In theory, for a distributed 3-D FFT of size K^3, the slab decomposition can scale up to K processes. For the experiments of this paper, the MPI communication cost can be quantified as follows:

$$T_c = n_{B2P}T_{B2P}+n_{P2P}T_{P2P}+n_{P2S}T_{P2S}+n_{S2B}T_{S2B}, \quad (2)$$

Assuming 3-D, double-complex data—and using Equation 2 and Table II with a $25GB/s$ theoretical bandwidth and $1\mu s$ latency— Figure 3 is a phase diagram for Summit, which allows one to choose the fastest theoretical decomposition to use. Note that formula (2) can be extended to support higher dimensions transforms.

In Figure 9, the tuning methodology is experimentally verified for multi-precision complex 3D-FFT of size 1024^3 on the Summit supercomputer. The advantage of slab decomposition breaks out around 384 MPI processes, and it actually starts to break the linear scaling a little earlier. This is consistent with the phase diagram, and it becomes more accurate when the real-world, observed bandwidth and latency are used at that scale. This communication model can also be extended to other architectures.

B. Communication and Roofline models

Few communication models are available for studying the bottleneck of hybrid parallel FFTs [15], [20], [32]. Those models target the throughput from each component or processor involved in the communication. As another approach, below we present a rather simple model that focuses on inter-node communication, which, according to our experience, better describes the global bottleneck for CPU-GPU based systems. This is because intra-node fast interconnection is usually available in modern systems, e.g., via NVLINK, while inter-node bandwidth is much slower. Also, within a single node, highly optimized local FFT computations can be performed, e.g, using CUFFTXt [7] or FFTX [11]. And then, all the bottleneck comes from inter-node communication. This model is also suitable for cases where there is not much information on subcomponents of the hybrid system, which can be the case of grid computer systems.

In [33], authors developed a communication model for estimating the time, in seconds, necessary to perform a single reshape within the computation of a 3-D FFT on Summit supercomputer, given as:

$$\Psi_{\text{Summit}} := 1.953P\log(N), \quad (3)$$

where P is the total number of processes and N is the FFT size. Fig. 10, shows *heFFTe*'s performance for a typical FFT of size $N = 1024^3$, and compares it to the roofline peak for increasing number of nodes, getting about to 90% close to peak value.

C. Accelerating MPI communication via tuning

Once a communication framework is established, c.f., Section II-A, we only rely on the MPI capabilities. FFT

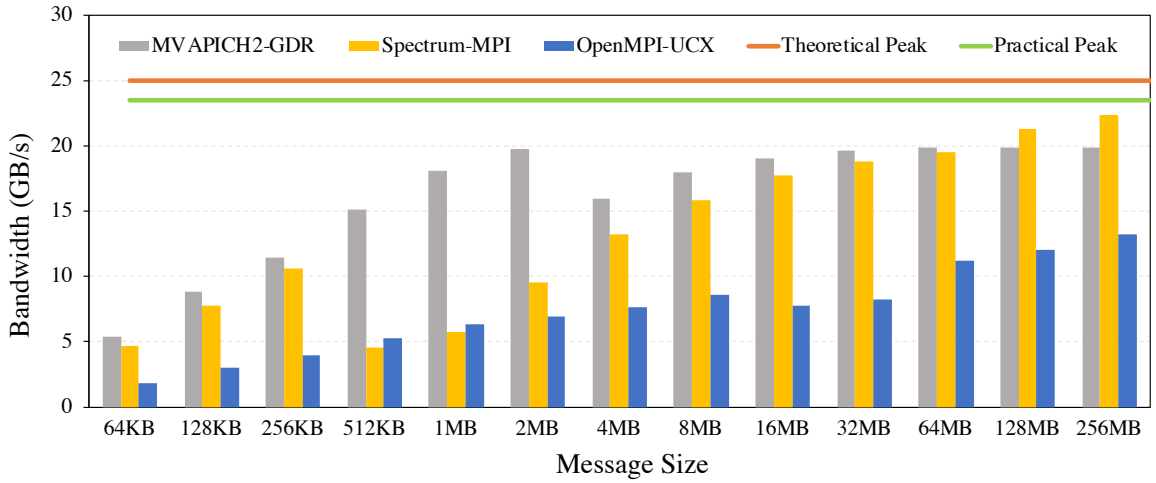

Fig. 4. Strong scalability on 3D FFTs of size 1024^3, using 24 MPI processes (1 MPI per Power9 core) per node (blue), and 24 MPI processes (4 MPI per GPU-V100) per node (red).

libraries support different types of communication as shown in Table I, and supercomputers usually offer multiple MPI libraries; e.g. OpenMPI, MVAPICH and SpectrumMPI.

The choice of MPI transfer type, either point-to-point or all-to-all, is critical for performance and scalability. Making the right choice depends on:

- The FFT size and algorithmic choice (pencil/slabs)
- Type and quantity of computing resources
- Underlying network topology
- Available MPI distribution

In Fig. 4, we compare OpenMPI with UCX support, MVAPICH2-GDR, and the default MPI distribution in Summit (IBM's Spectrum MPI). The latter does not perform very well for small data volume, which corresponds to small FFT sizes, typically found on applications such as LAMMPS [17]. The difference in performance can be considerable; and therefore, adding the choice of the MPI distribution as a tuning feature would potentially yield to greater speedups.

Recently, vendors providing GPU accelerators are developing frameworks for faster data transfer between nodes; among them, libraries such as NCCL , RCCL, OpenSHMEM and NVSHMEM. Currently, there is ongoing work in *heFFTe* for an early adoption of these tools and linking them to the MPI framework of applications. Also, a collaboration with OMPI-X (ECP project) has led to a novel all-to-all one-sided communication routine targeting parallel FFT and different MPI distributions [34], [35].

D. Software improvements

The novel software developments that we propose in this paper, have been integrated into *heFFTe v.2.1* [24], and they are based on advanced C++11 features. We created a common API that allows for a higher-level logic of FFT and reshape operations to be expressed independent from the CPU or GPU backends, while explicit instantiation and inlining is used to remove any unnecessary overhead. The API allows for an easy selection of tuning parameters (currently supporting over a dozen of options). For implementing the largest time-consuming kernel, i.e., the reshape kernel (transposition), we developed *reshape3d_alltoallv* which exploits features, as CUDA-Aware MPI, to allow pipelining with kernels such as 1-D FFTs and packing/unpacking that are computed on GPUs. The data and the reshape operations are wrapped in RAII containers using std::unique_ptr() and custom CUDA vectors that mimic the move and copy semantics of std::vector. In addition to the usual RAII benefits, the approach allows to express any arbitrary combinations of data transformations based on either one-dimensional pencils or two-dimensional slabs.

Tuning the communication as described above is the most critical part for efficient FFTs; to this end, when using point-to-point MPI, we use an heuristic tool called *minimum surface partition*, which helps to create intermediate processor grids to handle the transposition in a load-balanced manner. When collective communication is necessary, we include a novel *heFFTe* feature to allow either MPI_Alltoallv or MPI_AlltoAll, where for the latter an efficient padding is implemented and can potentially overcome the former. The novel software developments described in this paper are highly templated to support a wide range of options and parametric tuning. This helps to easily tune and choose between 6 backends for 1-D and 2-D FFTs, data types, multi-precision, MPI communication type, intermediate data decomposition and reshaping options. This is all done while preserving correctness and allowing in-place and out-of-place transforms.

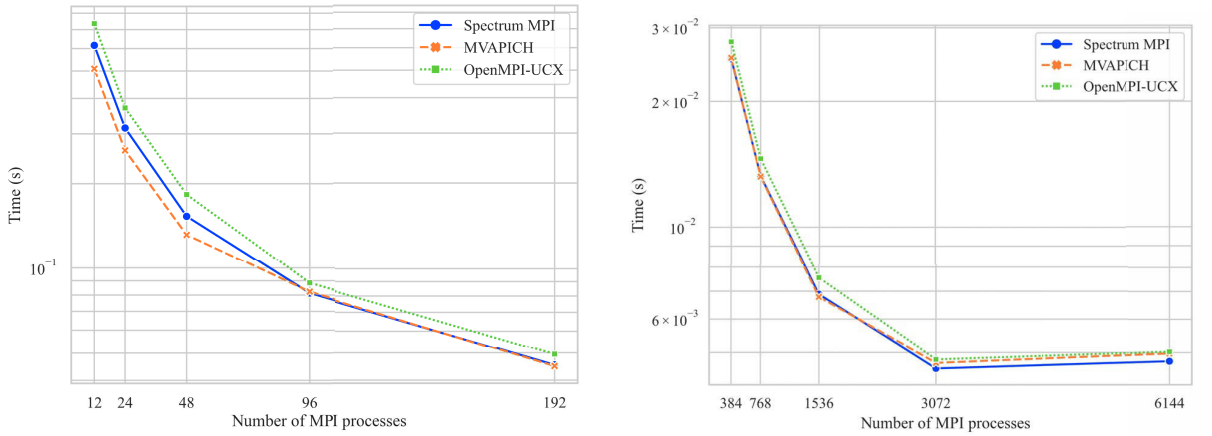

Fig. 5. Comparison of MPI runtime for different MPI distributions, on a strong scaling experiment using a 3-D FFT of size 1024^3, with 6 MPI processes per node, and 1 MPI per V100 GPU.

III. Experimental results

In this section, we present different experiments to show how efficiently tuning the FFT parameters and the communication framework yield to very good performance on one of the world's largest supercomputer.

A. Hardware platform

Our experiments were obtained using Summit supercomputer, which has 4,608 nodes, each composed by 2 IBM Power9 CPUs and 6 Nvidia V100 GPUs, as shown in Fig. 6. These 6 GPU accelerators provide a theoretical double-precision capability of approximately 40 TFlop/s. Within the same node, processors have two NVIDIA's NVLink interconnections, 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 (100GB/s bi-directional). Summit nodes are interconnected in a Non-blocking Fat Tree topology, via a dual-rail EDR InfiniBand network which provides a node injection bandwidth of 25 GB/s.

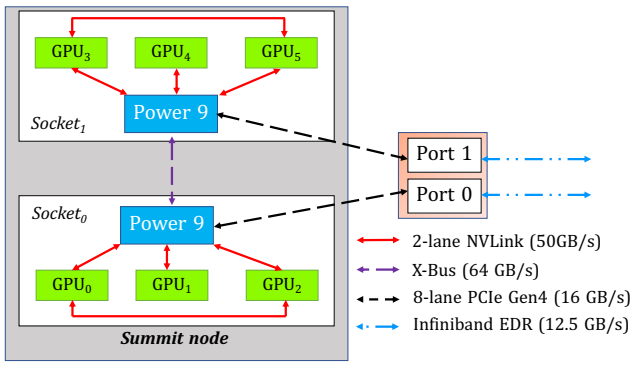

Fig. 6. Architecture of a single Summit node: computing units and network connections.

B. Experimental setup

All experiments we present where obtained as an average of 10 runs, after a warm-up call. Input data was randomly generated using a fixed seed, it is double-precision complex and its size is 1024^3, unless otherwise specified.

C. MPI tuning to improve FFT performance and scaling

As our first experiment, we compare the runtime for strong scaling of a 3-D FFT using different MPI distributions (OpenMPI, MVAPICH and Spectrum MPI). In Figure 5, we observe that MVAPICH with GDR support performs better than the other two, for up to 192 GPUs (64 nodes); while for a greater number of resources, the difference is not considerable, and the default MPI in Summit (IBM's Spectrum) tends to be faster.

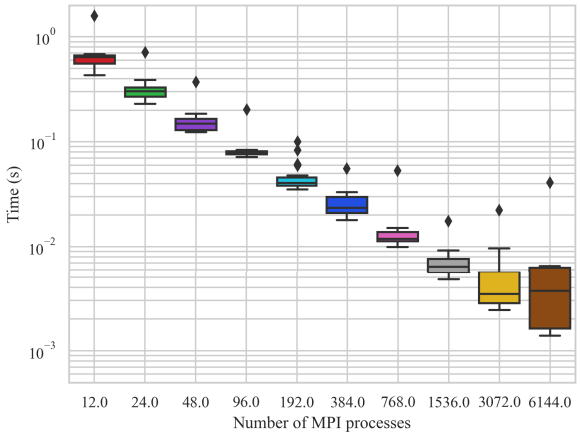

Fig. 7. Variability on the MPI cost for a scalability experiment using a 3-D FFT of size 1024^3, using 6 MPI processes per node, and 1 MPI per V100 GPU. The horizontal lines of each box correspond to the minimum, first quartile, median, third quartile, and maximum.

In the strong scalability experiment presented in Figure 5, we observe that, at 512 nodes (3,072 GPUs), the algorithm stops linearly scaling. This is due to the impact of latency. Since 6,144 processes are communicating, Figure 7 helps to understand this issue, here we saturate the nodes using all GPU's, which causes the number of messages sent to increase, c.f., Figure 2, and the breakdown of MPI time shown proves that small size message become difficult to be managed by the MPI distribution due to the system throughput limitations.

Next, in Figure 8, we present a study case where it is clear that choosing between point-to-point or collective MPI communication can be critical towards achieving linear scaling (which is of utmost important for upcoming exascale software). We compare two state-of-the-art libraries that are currently under development, where FFTE uses MPI_AlltoAllv, and for *heFFTe* we leverage both, non-blocking MPI send/receive and All-to-All. The optimizations within SpectrumMPI can explain the All-to-All exchange overcoming the binary (or point-to-point) approach. Refer to [36] for further performance comparison among different state-of-the-art libraries.

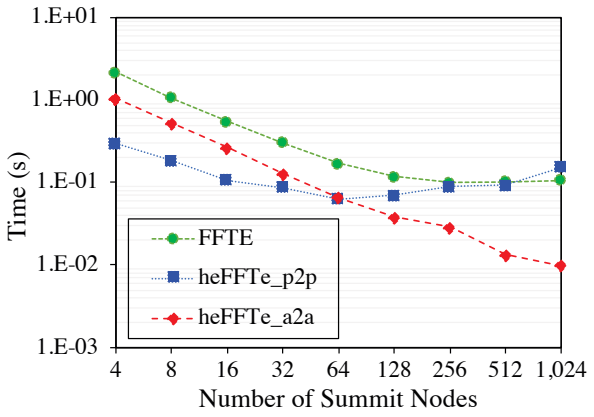

Fig. 8. Strong scalability for GPU libraries using 4 NVIDIA V100 GPUs per node, 2 per socket. Using *heFFTe 2.1* and FFTE 7.0 on a 3-D FFT of size 1024^3. Both libraries use cuFFT as 1-D backend.

D. Scalability

In this experiment, we use single and double precision data and the pencil and slab decomposition approaches, c.f. Section II, which commonly available in classical FFT libraries. In Fig. 9, we observe that both of these algorithmic choices scale very well, and tuning between them can be achieved by using the phase diagram introduced in Section II-A. For this experiment we choose the best configuration obtained from offline tuning in *heFFTe*.

E. Roofline peak performance analysis

In Fig. 10, we numerically analyze how we approach the roofline peak performance as described in Section II-B. We observe that by appropriately choosing the transform size and

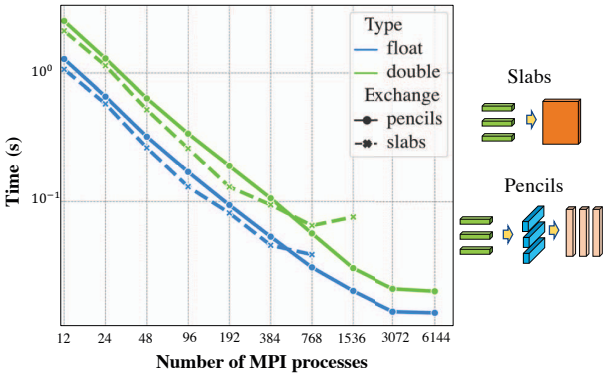

Fig. 9. Comparison of pencil and slab decompositions for strong scaling of a 3-D FFT of size 1024^3. Using *heFFTe* with cuFFT backend, 6 MPI processes (1 MPI processes per GPU-V100) per node, and complex random data.

the number of nodes, we can get closer to the proposed peak. The idea of this plot is to show user how a correlation can be established between these two parameters to ensure that resources are efficiently used. This is important for several applications, where the FFT is only a small portion of their total runtime. And they require some GPU resources to simultaneously run other tasks, such as visualization or profiling.

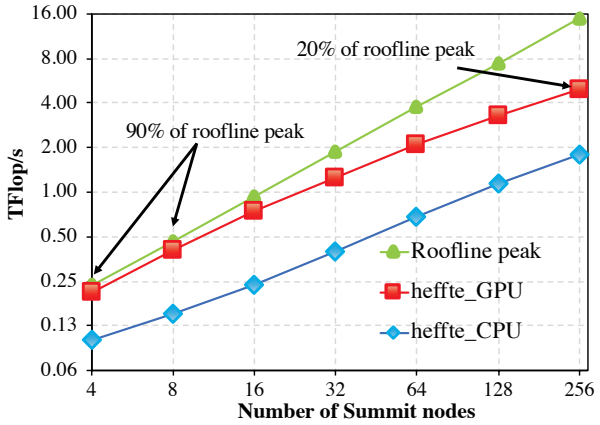

Fig. 10. Roof-line performance from Eq. 3 and *heFFTe* performance on a 3D FFT of size 1024^3; using 40 MPI processes, 1MPI/core, per node (blue), and 6 MPI/node, 1MPI/1GPU-Volta100, per node (red).

Note that for 256 nodes (1536 V100 GPUs) we are only achieving 20% of the peak. This is because we are using too much resources for this problem size. Increasing the FFT size would yield to performance closer to the peak (as shown for this case at 4 and 8 nodes). Version 2.1 of *heFFTe* provides tools for offline tuning which helps to efficiently select the number of computing resources for a given FFT size.

IV. Conclusions

This paper provides novel methodologies to further accelerate FFT computation on CPU and GPU distributed systems by tuning the multi-process communication. We

introduced techniques and software improvements that push the boundaries of state-of-the-art FFT libraries. Our analysis of the communication bottleneck via different MPI parameters and distributions, shed light on potential improvements that can be achieved by an efficient tuning procedure which we implemented on top of *heFFTe* library.

Next, we evaluated the impact of the communication bottleneck on FFT scalability. We analyzed different FFT algorithmic approaches (such as pencil and slab decomposition), and point-to-point and collective MPI communication. From our analysis, we saw that linear scalability is achievable, and we can reach very close to the experimental roofline peak on Summit supercomputer. Future work is expected on creating a mixed-precision optimization, and the acceleration of symmetrical, non-uniform and sparse FFTs.

ACKNOWLEDGMENT

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.

REFERENCES

[1] J. Demmel, Communication-avoiding algorithms for linear algebra and beyond, in: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, 2013.

[2] J. Dongarra, F. Sullivan, Guest editors introduction to the top 10 algorithms, Computing in Science Engineering 2 (1) (2000) 22–23.

[3] M. Frigo, S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231, special issue on "Program Generation, Optimization, and Platform Adaptation".

[4] Intel, Intel Math Kernel Library.
URL https://software.intel.com/mkl/features/fft

[5] S. Filippone, The ibm parallel engineering and scientific subroutine library, in: J. Dongarra, K. Madsen, J. Waśniewski (Eds.), Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 199–206.

[6] rocFFT library (2021).
URL https://github.com/ROCmSoftwarePlatform/rocFFT

[7] CUFFT library (2021).
URL http://docs.nvidia.com/cuda/cufft

[8] Intel, Intel OneAPI Library.
URL https://software.intel.com/oneapi/fft.html

[9] Vulkan FFT library (2021).
URL https://github.com/DTolm/VkFFT

[10] KFR library (2021).
URL https://github.com/kfrlib/kfr

[11] F. Franchetti, D. Spampinato, A. Kulkarni, D. T. Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. Van Straalen, P. Colella, FFTX and SpectralPack: A First Look, IEEE International Conference on High Performance Computing, Data, and Analytics (2018).

[12] D. Pekurovsky, P3dfft: A framework for parallel computations of fourier transforms in three dimensions, SIAM Journal on Scientific Computing 34 (4) (2012) C192–C209. arXiv:https://doi.org/10.1137/11082748X.

[13] N. Li, S. Laizet, 2DECOMP&FFT - a highly scalable 2d decomposition library and fft interface, 2010.

[14] J. H. Göbbert, H. Iliev, C. Ansorge, H. Pitsch, Overlapping of communication and computation in nb3dfft for 3d fast fourier transformations, in: E. Di Napoli, M.-A. Hermanns, H. Iliev, A. Lintermann, A. Peyser (Eds.), High-Performance Scientific Computing, Springer International Publishing, Cham, 2017, pp. 151–159.

[15] A. Gholami, J. Hill, D. Malhotra, G. Biros, Accfft: A library for distributed-memory FFT on CPU and GPU architectures, CoRR abs/1506.07933 (2015). arXiv:1506.07933.

[16] S. Plimpton, A. Kohlmeyer, P. Coffman, P. Blood, fftMPI, a library for performing 2d and 3d FFTs in parallel, Tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States) (2018).

[17] Large-scale atomic/molecular massively parallel simulator, available at https://lammps.sandia.gov/ (2018).

[18] D. Richards, O. Aziz, J. Cook, H. Finkel, et al., Quantitative performance assessment of proxy apps and parents, Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States) (2018).

[19] J. Emberson, N. Frontiere, S. Habib, K. Heitmann, A. Pope, E. Rangel, Arrival of First Summit Nodes: HACC Testing on Phase I System, Tech. Rep. MS ECP-ADSE01-40/ExaSky, Exascale Computing Project (ECP) (2018).

[20] K. Czechowski, C. McClanahan, C. Battaglino, K. Iyer, P.-K. Yeung, R. Vuduc, On the communication complexity of 3D FFTs and its implications for exascale, 2012. doi:10.1145/2304576.2304604.

[21] M. Lee, N. Malaya, R. D. Moser, Petascale direct numerical simulation of turbulent channel flow on up to 786k cores, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013.

[22] A. Nukada, K. Sato, S. Matsuoka, Scalable multi-GPU 3-D FFT for TSUBAME 2.0 supercomputer, High Performance Computing, Networking, Storage and Analysis (2012).

[23] D. Takahashi, FFTE: A fast Fourier transform package, http://www.ffte.jp/ (2005).

[24] heFFTe library (2021).
URL https://bitbucket.org/icl/heffte

[25] S. Tomov, A. Haidar, D. Schultz, J. Dongarra, Evaluation and Design of FFT for Distributed Accelerated Systems, ECP WBS 2.3.3.09 Milestone Report FFT-ECP ST-MS-10-1216, Innovative Computing Laboratory, University of Tennessee, revision 10-2018 (October 2018).

[26] S. Tomov, A. Haidar, A. Ayala, D. Schultz, J. Dongarra, Design and Implementation for FFT-ECP on Distributed Accelerated Systems, ECP WBS 2.3.3.09 Milestone Report FFT-ECP ST-MS-10-1410, Innovative Computing Laboratory, University of Tennessee, revision 04-2019 (April 2019).

[27] S. Tomov, A. Haidar, A. Ayala, H. Shaiek, J. Dongarra, FFT-ECP Implementation Optimizations and Features Phase, Tech. Rep. ICL-UT-19-12 (2019-10 2019).

[28] H. Shaiek, S. Tomov, A. Ayala, A. Haidar, J. Dongarra, GPUDirect MPI Communications and Optimizations to Accelerate FFTs on Exascale Systems, Extended Abstract icl-ut-19-06 (2019-09 2019).

[29] A. Grama, A. Gupta, G. Karypis, V. Kumar, Accuracy and stability of numerical algorithms, Addison Wesley, second ed., 2003.

[30] L. Dalcin, M. Mortensen, D. E. Keyes, Fast parallel multidimensional FFT using advanced MPI, Journal of Parallel and Distributed Computing 128 (2019) 137–150.

[31] A. Ayala, X. Luo, S. Tomov, H. Shaiek, A. Haidar, G. Bosilca, J. Dongarra, Impacts of Multi-GPU MPI Collective Communications on Large FFT Computation, in: 2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI), 2019.

[32] D. Takahashi, 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.

[33] A. Ayala, S. Tomov, A. Haidar, J. Dongarra, heFFTe: Highly Efficient FFT for Exascale, in: ICCS 2020. Lecture Notes in Computer Science, 2020.

[34] A. Ayala, et al., Accelerating FFT towards Exascale Computing, NVDIA GPU Technology Conference Posters, Santa Clara - CA, USA (April 2021).

[35] A. Ayala, S. Tomov, M. Stoyanov, J. Dongarra, Scalability issues in FFT computation, in: V. Malyshkin (Ed.), Parallel Computing Technologies, Springer International Publishing, Cham, 2021, pp. 279–287.

[36] A. Ayala, S. Tomov, P. Luszczek, G. Ragghianti, S. Cayrols, J. Dongarra, Interim report on benchmarking FFT libraries on high performance systems, ICL Tech Report ICL-UT-21-03, University of Tennessee (2021-07 2021).