2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-9747-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPSW55747.2022.00072

Performance Analysis of Parallel FFT on Large Multi-GPU Systems

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*—In this paper we present a performance study of multidimensional Fast Fourier Transforms (FFT) with GPU accelerators on modern hybrid architectures, as those expected for upcoming exascale systems. We assess and leverage features from traditional implementations of parallel FFTs and provide an algorithm that encompasses a wide range of their parameters, and adds novel developments such as FFT grid shrinking and batched transforms. Next, we create a bandwidth model to quantify the computational costs and analyze the well-known communication bottleneck for All-to-All and Point-to-Point MPI exchanges. Then, using a tuning methodology, we are able to accelerate the FFT computation and reduce the communication cost, achieving linear scalability on a large-scale system with GPU accelerators. Finally, our performance analysis is extended to show that carefully tuning the algorithm can further accelerate applications heavily relying on FFTs, such is the case of molecular dynamics software. Our experiments were performed on Summit and Spock supercomputers with IBM Power9 cores, over 3000 NVIDIA V-100 GPUs, and AMD MI-100 GPUs.**

***Index Terms*—FFT, Multi-GPU, MPI tuning, Scalability**

I. Introduction

Nowadays, the Fast Fourier Transform (FFT) is widely used in computational sciences and engineering. Back in 1965, the first sequential FFT algorithm was introduced by Cooley and Tukey [1], and its implementation has been evolving to adapt to novel hardware developments. 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 by $\hat{f} = FFT(x)$ and defined as an m-dimensional vector the same size as x by the following computation:

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

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

In terms of arithmetic operations, the summation (1) can be directly computed by a tensor product and this would cost $\mathcal{O}(N \sum_{i=1}^{m} N_i)$. The advantage of the FFT is that this cost can be reduced to $\mathcal{O}(N \log_2 N)$ operations by exploiting the structure of the tensor.

The Cooley-Tukey FFT algorithm is considered one of the top ten algorithms of the 20th century, and several single-device (both vendor and open source) efficient implementations are available. One of the most widely used libraries for this purpose is FFTW [2], 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) [3], ESSL (IBM) [4], rocFFT (AMD) [5] and cuFFT (NVIDIA) [6]. Novel libraries are also being developed to further optimize single-device FFTs, among them: OneAPI for Intel GPUs [7], Vulkan FFT (VkFFT) [8], KFR [9], and FFTX [10], where the latter is part of the ECP software community.

A. State-of-the-art: Parallel FFT Libraries

Parallel versions of the FFT algorithm have been developed for decades in pair with the evolution of distributed computing systems. Parallelism for multi-dimensional FFTs can be straightforwardly achieved by distributing the sums in (1) among processors (for each of the m dimensions). Figure 1 shows three different decomposition sequences used in state-of-the-art parallel libraries to compute a 3-D FFT. Such decompositions easily extend to higher dimensions [11].

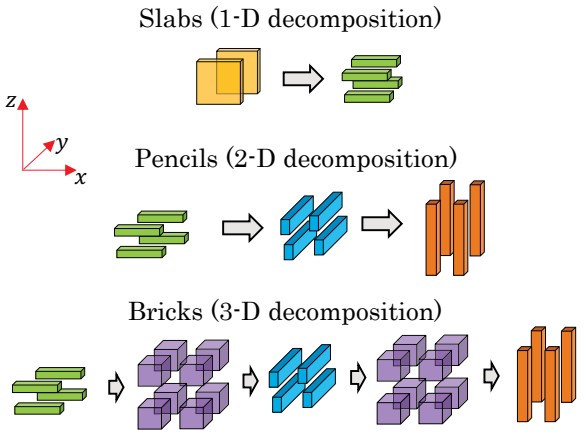

Fig. 1. Algorithmic approaches for parallel 3-D FFT computation.

978-1-6654-9747-3/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPSW55747.2022.00072

As shown in Figure 1, we can implement a parallel 3-D FFT algorithm of size $N_1 \times N_2 \times N_3$ by one of the following methodologies:

- Slabs decomposition: gives the first level of parallelization, where we distribute data among a 1-D grid of processes, each of them is in charge of computing a batch of 2-D FFTs of size $N_1 \times N_3$ and *transferring* data to other processes. Therefore, the algorithm arising from this approach has scalability limitations to up to N_2 processes.

- Pencils decomposition: introduced in [12], allows to reach the next level of parallelization, by distributing data among a 2-D grid of processes, each of them is in charge of computing a batch 1-D FFTs and *transferring* the data to other processes. In this case, we need two communication phases.

- Bricks decomposition: a modified version of the pencils approach, where intermediate communication to a 3-D grid of processes is performed during each communication phase. Therefore, it requires four communication phases. Although this approach seems costly, it can be faster than the previous two in some architectures [13].

The *transfer* phase (also known as remap, reshape or transposition) described for each of the implementations options above, is obtained with a Message Passing Interface (MPI) distribution. Table I shows the available MPI routines in some of the most recent developments on parallel FFT libraries; where the first four libraries support GPU accelerators using MPI GPU-aware features. Refer to [14] for an extended FFT benchmark.

TABLE I
AVAILABLE MPI ROUTINES IN FFT LIBRARIES

Library	**Communication Type**	
	AlltoAll	***Point-to-Point***
AccFFT [15]	MPI_Alltoall	MPI_Isend / MPI_Irecv MPI Sendrecv
FFTE [16]	MPI_Alltoall MPI_Alltoallv	-
fftMPI [17]	MPI_Alltoallv	MPI_Send / MPI_Irecv
heFFTe [18]	MPI_Alltoall MPI_Alltoallv	MPI_Send / MPI_Isend MPI_Irecv
Dalcin et al. [11]	MPI_Alltoallw	-
P3DFFT [19]	MPI_Alltoallv	MPI_Send / MPI_Irecv

Among distributed libraries in the literature, the widely-used FFTW employs a 1-D decomposition approach, which limits its scalability to a small number of nodes. P3DFFT [19] extends FFTW functionalities and supports both 1-D and 2-D decomposition. The PFFT package [20] is also built on top of FFTW and extends the functionalities of the previous two to higher dimensional arrays. On the other hand, libraries 2DECOMP&FFT [21], nb3dFFT [22] and AccFFT [15], showed good scalability but are no longer maintained. Finally, some large scale applications have their own built-in FFT library, such as fftMPI [13] (built-in on LAMMPS [23]) and SWFFT [24] (built-in on HACC [25]). These libraries are widely known in the molecular-dynamics and astrophysics literature. Also, fftMPI and SWFFT are, to our knowledge, the only libraries supporting bricks decomposition.

In this paper, we are interested on CPU-GPU based computing systems, i.e., libraries developed to support GPU accelerators. In this realm, the 1-D decomposition approach introduced in [26] was one of the first codes for large FFT computation on GPUs. Its optimization approach is limited to small number of nodes and focuses on reducing tensor transposition cost by exploiting infiniband-interconnection using the IBverbs library, which makes it not portable. Further improvements to scalability have been presented in FFTE library [16] 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. On the other hand, *heFFTe* [27], is a recent open source FFT library, providing optimizations that achieve linear scalability for large number of cores and GPUs. Finally, in [28] authors introduced a code for multi-GPU FFT computation within the context of turbulence simulations.

B. Contributions of this paper

- We study current parallel FFT implementations and leverage a wide number of tuning parameters to further accelerate state-of-the-art FFT libraries. We provide a novel algorithm with 3-D FFT batched support and grid shrinking, allowing the use of accelerators from three major GPU vendors: AMD, Intel and NVIDIA.
- We develop a bandwidth model, and present MPI tuning techniques to select the best communication scheme.
- We extend the work of [11] to GPU systems, and compare different options of all-to-all communication with SpectrumMPI and MVAPICH.

C. Paper organization

In Section II, we present challenges of distributed FFTs, and describe algorithms and architectures to be used in our experiments. We present the two main MPI-exchange approaches available in state-of-the-art FFT libraries. In Section 3, we present a model to quantify the communication cost for parallel complex-to-complex FFTs, and compare different Point-to-Point and All-to-All communication options. We then show the practical bandwidth achieved amongst different settings. In Section IV we present further experiments on scalability and the effect that MPI GPU-awareness has on parallel FFT computation. Finally, Section V concludes our paper.

Algorithm 1 Parallel 3-D FFT computation on GPUs

Input: 3-D arrays of n_p processors, grids at input and output: P_{in}, P_{out}
If required by user, remap data to a subcommunicator of $l_p < n_p$ processors
Transfer data from P_{in} to a pencil or slab grid, get number of reshapes needed
Set transform type (Forward or Inverse FFT) and number of batches per computation
Define processor grids (MPI groups) for each direction
for $r \leftarrow 1, \cdots, n_{\text{batches}}$ **do**
    for $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
        Compute local 1-D or 2-D FFTs on the GPUs
        Pack data in contiguous memory
        for P on my MPI group **do**
            Transfer computed data to neighbor processors
        end for
        Unpack data in contiguous memory
    end for
end for
Transfer data from the pencil or slab grid to P_{out}

II. Parallel 3-D FFT on Multi-GPU systems

Parallel FFT algorithms rely on single-device libraries for their local 1-D or 2-D computation. Therefore, their main job (and what defines their performance) is how they handle the required global data transfers. The classical approach is to keep arrays of data in *contiguous* memory before and after local FFTs, and use classical MPI_Alltoall or binary (send and receive) communication. Such approaches are known as transpose algorithms [29]. Algorithm 1 shows the steps followed by most state-of-the-art FFT distributions. In italic blue text, we show two novelties added to *heFFTe* library [30] as a software contribution of this paper:

- *FFT grid shrinking*: useful when the input data is distributed among a large number of MPI processes; and, at FFT planning, we identify that the computation can fit in a smaller grid of MPI processes (controlling an amount of memory and resources enough for the computation). Then we remap them pre and post computation. Note that once involving the network communication, then the power of the GPU is limited and the performance is much less than the GPU theoretical peak. Therefore, the smallest the number of processes controlling the computing FFT grid, the faster is the computation.

- *Batched 2-D and 3-D transforms* on distributed systems with GPU accelerators, c.f., Fig. 13, where we show considerable speedups using a batched FFT algorithm.

Another approach consists in transferring data in a non-contiguous fashion, this methodology was recently studied in [11], where authors showed good speed-ups compared to Algorithm 1 from P3DFFT and FFTW. For this, a generalized all-to-all (MPI_Alltoallw) scatter/gather is employed on a predefined sub-array datatype and the **for** loop of Algorithm 1 is replaced by a single call to local FFTs and a direct call to MPI_Alltoallw as shown in Algorithm 2.

Algorithm 2 FFT with non-contiguous data exchange

Create MPI sub-array datatype
for $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
    Compute local 1-D or 2-D FFTs on the GPUs
    for P on my sub-communicator **do**
        Transfer data using MPI_Alltoallw
    end for
end for

The sub-array datatype used in Algorithm 2 is in general discontiguous, and it can be created using MPI_Type_Create_Subarray [31]. This approach is very elegant and can be coded in a few hundred lines [11], while standard FFT libraries require a few thousands; and although it eliminates the need for any local remappings (a.k.a. packing and unpacking), it might not always be beneficial for GPU based libraries, given that these steps account for less than 10% of runtime [15], [18], c.f., Fig. 6 and 7.

In Figure 2, we show an experiment using 24 V-100 GPUs on 4 Summit nodes to compute 4 forward and 4 backward 3-D FFTs, being communication for this problem over 90% of runtime. The x-axis corresponds to each of the calls of the given MPI routine. We observe that Algorithm 2 is not always beneficial for GPU-based implementations, even discounting the roughly 10% of runtime saved from avoiding data packing and unpacking. While the work in [11] for CPU-based FFTs showed performance comparable to FFTW and P3DFFT. The lack of optimizations of MPI_Alltoallw is a critical factor making this approach not as efficient when using GPU arrays. This was expected since MPI_Alltoallw is far less optimized compared to MPI_Alltoall(v). For example, MPICH has four different implementations of MPI_Alltoall, which are selected

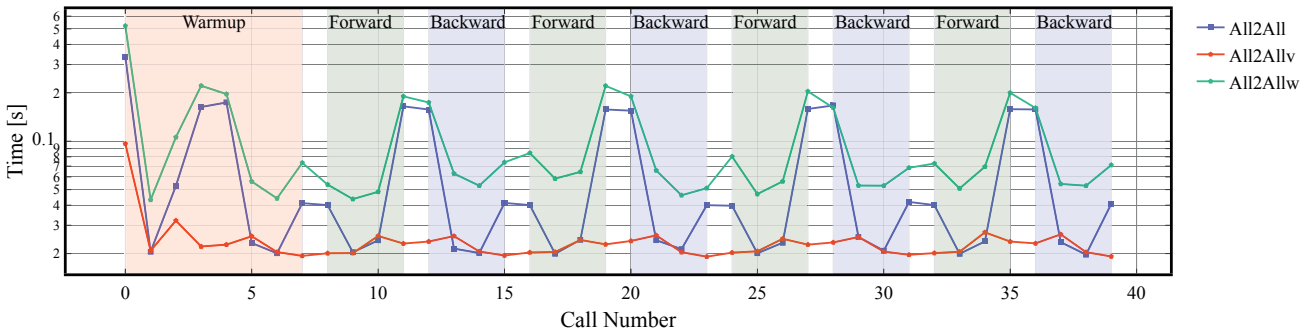

Fig. 2. Comparison of the communication runtime for different GPU-aware All-to-All MPI implementations using *heFFTe* and MPI_AlltoAll(v) (from SpectrumMPI), and MPI_AlltoAllw[†] (from MVAPICH), in the computation a complex-to-complex 3-D FFT of size 512^3. In total, there are 40 calls to MPI.

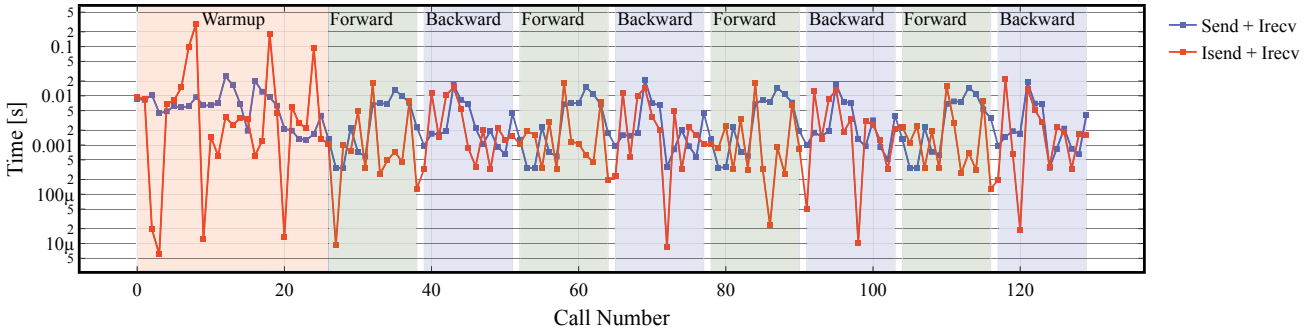

Fig. 3. Comparison of the communication runtime for different GPU-aware Point-to-Point MPI implementations using *heFFTe* and (blocking and non-blocking) SpectrumMPI routines to compute a complex-to-complex 3-D FFT of size 512^3. The x-axis shows the call count for each MPI routine.

according to the array size; while its MPI_Alltoallw, is simply composed of a non-blocking MPI_Isend and MPI_Irecv algorithm for any array size. And for the case of GPU-arrays, we encounter the issue that for some MPI distributions, such as SpectrumMPI, MPI_Alltoallw is not even GPU-aware [32], and the network capabilities, such as NVLINKs, are not efficiently used.

Analogously to the All-to-All case, Fig. 3 shows the communication cost for the computation of a 3-D FFT on 24 V-100 GPUs using MPI (I)send and Irecv routines. We note that there is not much difference when using blocking and non-blocking approaches. Similarly to the AlltoAllw approach of Algorithm 2, in [33], authors used derived data-types to perform transpose-free transforms of 3-D arrays, through customized MPI_Isend and MPI_Irecv routines, where they observed that the transpose-free FFT is, in most cases, just marginally superior than the one with transpose. Finally, some authors have developed methodologies to further accelerate MPI frameworks within distributed FFT by asynchronous communication to overlap communication and computation, c.f., [28], [34], [35].

[†]Note that the latest version of SpectrumMPI (10.4.1-2021) does not provide a GPU-aware MPI_AlltoAllw routine.

A. *Supercomputers for our experiments*

The experimental part of this paper was obtained using Summit supercomputer, which has 4,608 nodes, each consisting of 2 IBM POWER9 CPUs and 6 NVIDIA V-100 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 V-100 and P9 can communicate at a peak of 50 GB/s (100 GB/s bi-directional). Summit nodes are interconnected in a non-blocking fat tree topology via a dual-rail EDR InfiniBand network that provides a practical bandwidth of about 23.5 GB/s. We also used Spock, which is a small system located at the Oak Ridge National Laboratory, and is composed of 36 nodes, with 4 MI-100 AMD GPUs per node. Spock is a precursor of the upcoming Frontier machine, expected to have exascale performance.

B. *Software stack for our experiments*

In Table II, we present the software stack used to obtain the experimental results of this paper. Note that these are some of the latest available versions. For the case of MPI distribution, we use IBM SpectrumMPI, which is the default on Summit supercomputer. However, we also show results with MVAPICH-GDR for analyzing Algorithm 2, since MPI_Alltoallw is not

CUDA-aware in the 10.4 version of SpectrumMPI. Refer to [36] for experiments showing the impact of switching MPI distributions, OpenMPI and MVAPICH, in FFT computation on Summit-like systems. [32].

TABLE II
SOFTWARE VERSIONS USED FOR THE EXPERIMENTS IN THIS PAPER.

Software name	Version
CUDA	11.0.3
CMake	3.20.2
FFTW3	3.3.9
GNU compilers	9.1.0
heFFTe	2.1
MVAPICH-GDR	2.3.6
Spectrum MPI	10.4.1

III. MULTI-GPU PERFORMANCE ANALYSIS

There are several models that have been developed to measure the cost of MPI communication (known to be a bottleneck) of a parallel 3-D FFT of size N using Π processes and n nodes; and since such models are architecture dependent, there is no a single one that can accurately hold for all supercomputers. Amongst the most relevant models:

- In [15], authors propose to use $\mathcal{O}\left(\frac{N}{\sigma(P)}\right)$, where $\sigma(P)$ is the bisection bandwidth of the network.
- In [33], authors use regression to find γ such that the communication cost is $\mathcal{O}(n^{-\gamma})$. This was developed for a Cray XC40 system (Shaheen II).
- In [37], authors propose a theoretical lower-bound for the communication cost on 3-D FFT computations towards exascale computing systems, assuming a 3-D torus topology (which is found in supercomputers targeting exascale such as Fugaku at Riken-Japan). The communication time is given as $\Omega\left(\frac{N}{P^{5/6}\cdot B}\right)$, where B is the network bandwidth.

In this paper, we focus on complex-to-complex FFTs envisaging Summit and Frontier like systems, where intra-node communication is much faster than inter-node communication (due to fast GPU interconnections). Hence, we develop a simple model that can help us estimate the average bandwidth during a 3-D FFT computation on a given number of nodes. For this, we refer the reader to Fig. 1, where for the slabs-decomposition, a single data transfer is required amongst processes distributed on the y axis. Since every process holds N/Π data and communicates $1/\Pi$ of it to its $(\Pi-1)$ neighbors, then the communication costs is[‡](assuming double-complex datatype, i.e., 16 bytes):

$$T_{slabs} = (\Pi - 1)\left(L + \frac{16N}{B\cdot\Pi^2}\right), \tag{2}$$

[‡]Refer to [38] for a similar for 3-D real-to-complex model designed for Intel Xeon Phi Clusters.

where L is the latency, and B the bandwidth (average-wise). Analogously, observing Fig. 1, for the pencil-decomposition, we have that Π is split over a 2-D grid of processes, $\Pi := P \times Q$, with P processes along the x-axis and y processes along the y axis. Then, the communication cost for the two transfers is:

$$T_{pencils} = (P-1)\left(L + \frac{16N}{B\cdot P\cdot\Pi}\right) + (Q-1)\left(L + \frac{16N}{B\cdot Q\cdot\Pi}\right) \tag{3}$$

Once we measure the communication runtime, using equations (2) and (3), we can estimate the average bandwidth as:

$$B_{slabs} = \frac{16N}{\Pi^2\left(\frac{T_{slabs}}{\Pi-1} - L\right)} \tag{4}$$

$$B_{pencils} = \frac{16N\left(\frac{P-1}{P} + \frac{Q-1}{Q}\right)}{\Pi\cdot(T_{pencils} - L\cdot(P+Q-2))} \tag{5}$$

Using equations (4) and (5), in Fig. 4 we present the average bandwidth obtained for a strong scalability experiment from 1 to 128 nodes (6 V-100 GPUs per node, 1 GPU per MPI process). The processor grids employed are showed in Table III. We observe that network saturation causes an exponential decrease in the average bandwidth achieved by each process, for both All-to-All and Point-to-Point approaches, which eventually cause a breakdown of the linear strong-scaling. This phenomenon is common for most state-of-the-art parallel FFT libraries, c.f., [14].

TABLE III
GRID SEQUENCE FOR SCALABILITY EXPERIMENT

# GPUs	Grid sequence[a]
6	(1, 2, 3) (2, 1, 3) (2, 3, 1) (1, 2, 3)
12	(2, 2, 3) (1, 3, 4) (3, 1, 4) (3, 4, 1) (2, 2, 3)
24	(2, 3, 4) (1, 4, 6) (4, 1, 6) (4, 6, 1) (2, 3, 4)
48	(3, 4, 4) (1, 6, 8) (6, 1, 8) (6, 8, 1) (3, 4, 4)
96	(4, 4, 6) (1, 8, 12) (8, 1, 12) (8, 12, 1) (4, 4, 6)
192	(4, 6, 8) (1, 12, 16) (12, 1, 16) (12, 16, 1) (4, 6, 8)
384	(6, 8, 8) (1, 16, 24) (16, 1, 24) (16, 24, 1) (6, 8, 8)
768	(8, 8, 12) (1, 24, 32) (24, 1, 32) (24, 32, 1) (8, 8, 12)
1536	(16, 8, 12) (1, 32, 48) (32, 1, 48) (32, 48, 1) (16, 8, 12)
3072	(16, 12, 16) (1, 48, 64) (48, 1, 64) (48, 64, 1) (16, 12, 16)

[a] The blue grids correspond to input and output 3-D grids, which come before and after, respectively, the FFT grids.

In Table III, the black grids correspond to FFT grids; for example, at 128 nodes, we have $\Pi = 768$, $P = 24$ and $Q = 32$. The blue grids correspond to brick shaped input and output grids, we include these grids in our experiments since in general this is the type of input from real-world

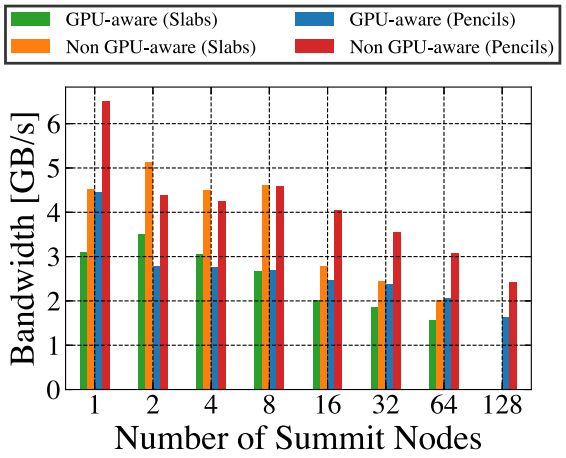

Fig. 4. Average bandwidth per process during the computation of a complex-to-complex 3-D FFT of size 512^3 using *heFFTe* with 6 V-100 GPUs per Summit node and switching the GPU-awareness feature for the MPI communication.

simulations, such as those from molecular dynamics. The 3-D brick shaped grids are usually obtained using a heuristics approach known as minimum-surface splitting, which aims to achieve load-balancing. To our knowledge, the only libraries allowing general input/output grids are fftMPI, *heFFTe* and SWFFT.

IV. Experiments on Scalability and Tuning

For the experiments of this section we employ the software described in Table II, and we report the average runtime of 8 FFTs (4 forward and 4 backward), which are preceded by 2 FFTs to warm up the accelerators.

A. Choosing between Pencils and Slabs decompositions

Choosing between pencils or slabs is critical and can result in a considerable speedup, here is where the model developed in Section III becomes very helpful. We first need to evaluate the network capabilities on the underlying computer and a phase diagram [36] can be created identifying which approach could be the fastest for a given FFT size. In our case, we rely on equations (4) and (5) to decide between pencils and slabs. We set an inter-node bandwidth of 23.5 GB/s and a latency of 1μs, which can be achieved in practice on Summit. This, in our experience, together with a phase diagram, gives the best chance to better predict the fastest algorithmic setting. Another approach that seems to work in practice is the use of regression, c.f., [33], where authors assess the performance of different settings and use regression for prediction. Next, using the grid sizes from Table III, we predict that the slabs decomposition should be faster than the pencil approach when using less than 64 nodes. We experimentally verify this, c.f., Figs. 8 and 9. Next, in Figure 5, we plot a strong scaling curve and label the regions with the fastest setting. Note that the fastest runtimes were achieved using SpectrumMPI with GPU-aware capability.

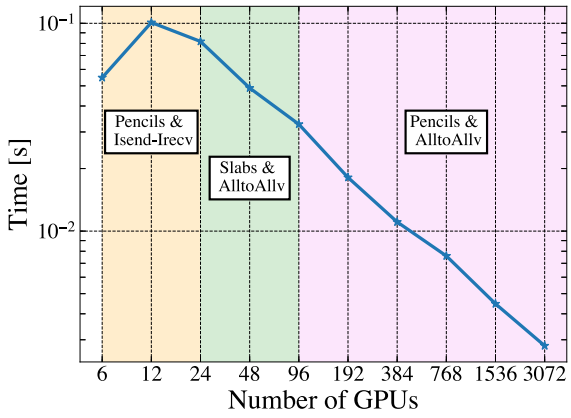

Fig. 5. Best setting regions for the computation of a 3-D complex-to-complex FFT of size 512^3 on increasing number of nodes, using 6 V-100 GPUs per node and 1 MPI per GPU.

B. Choosing between Point-to-Point or All-to-All transfers

In Fig. 6, we show a runtime breakdown for the kernels involved on the computation of a 3-D complex-to-complex FFT of size 512^3 using an *All-to-All* approach on 24 GPUs (4 nodes); we observe that when using MPI_Alltoall (which requires padding) we get higher variability and higher runtime compared to MPI_Alltoallv. Overall, amongst all of our experiments in Summit we observed that, in general, the cost associated with padding overcomes the benefits of using MPI_Alltoall on GPU systems. However, if we take a deeper look into the time for each single call, refer to Fig 2, we observe that for the data transfers associated to the FFT computation (intermediate grids) the difference between MPI_Alltoall and MPI_Alltoallv is negligible; and the large runtime gap arises from the transfers associated to the brick-to-pencils reshape (3D input to pencils) and (pencils to 3D output) which typically require much more padding. Therefore, if the input and output are given in pencils or slabs shape, then MPI_Alltoall might be the best choice, due to the built-in optimizations it has amongst all different MPI distributions. On the other hand, in Fig. 7 we show a runtime breakdown when using a *Point-to-Point* approach instead, with non-blocking (left) and blocking (right) routines. We observe that the communication time (sum of MPI send/recv/waitany) is slightly faster than the All-to-All approach for this number of nodes; however, the total runtime for the 3-D FFT is pretty much the same ($\approx$0.09s) for both approaches and for larger number of nodes the All-to-All turns out to be the fastest option (see Figs. 8 and 9); this is agreement to what has been reported on state-of-the-art GPU-based libraries [14], and a known phenomenon in the FFT community [15], [33], [39].

Next, note that in Figs. 6 and 7 the batch of 1-D FFTs computed with cuFFT have different timings when using contiguous and non-contiguous (strided) data. In Fig. 10, we take a deeper look into each of the calls to cuFFT and observe a spike when the FFT input is strided, and the

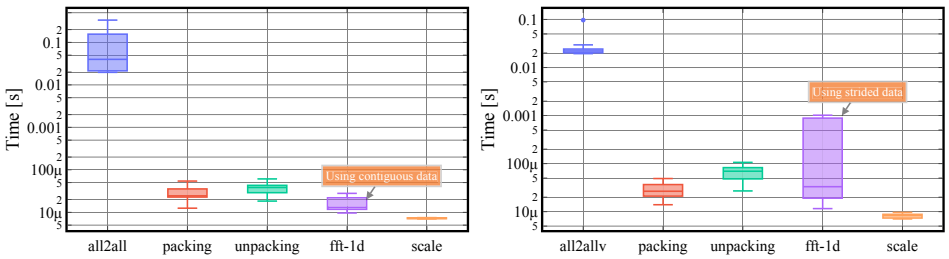

Fig. 6. Runtime breakdown for a 3-D FFT of size 512^3 using 24 V-100 GPUs with All-to-All communication and pencils approach. On the left, we use MPI_Alltoall for the communication and CUFFT to compute the batch of 1-D FFTs using contiguous data (transposed approach). On the right, we use MPI_Alltoallv and non-contiguous (strided) data.

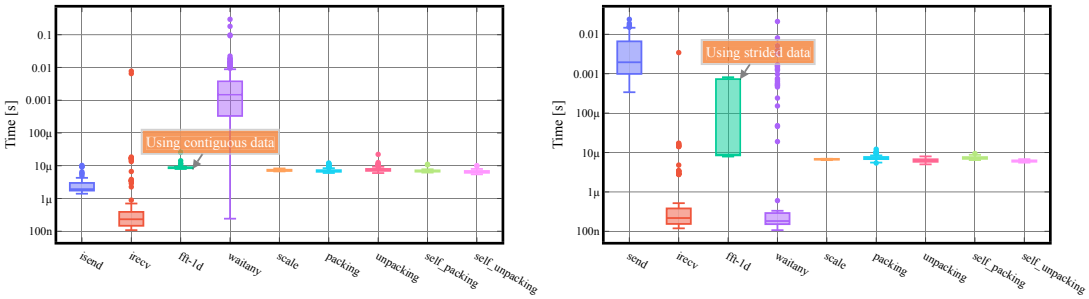

Fig. 7. Runtime breakdown for a 3-D FFT of size 512^3 using 24 V-100 GPUs with Point-to-Point communication and pencils approach. On the left, we use MPI_Isend and MPI_Irecv for the communication and CUFFT to compute the batch of 1-D FFTs using contiguous data (transposed approach). On the right, we use MPI_Send and MPI_Irecv on non-contiguous (strided) data.

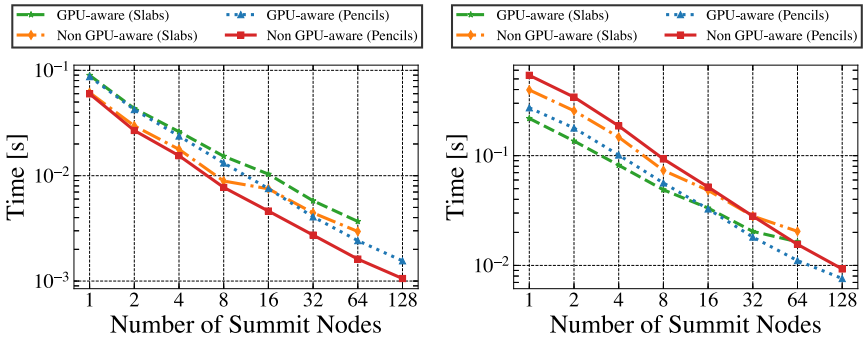

Fig. 8. Comparison of the performance of All-to-All MPI communication with and without GPU-aware MPI in the computation of a 3-D FFT of size 512^3 using 6 V-100 GPUs per node, and 1 MPI per GPU. On the left we show the communication cost, and on the right the total time.

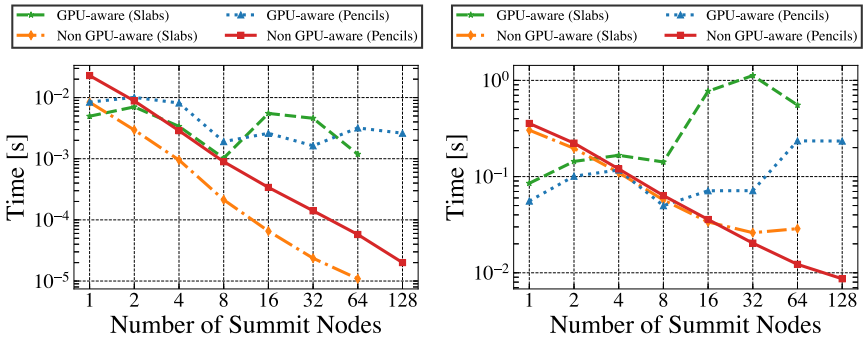

Fig. 9. Comparison of the performance of Point-to-Point MPI communication with and without GPU-aware MPI in the computation of a 3-D FFT of size 512^3 using 6 V-100 GPUs per node, and 1 MPI per GPU. On the left we show the communication cost, and on the right the total time.

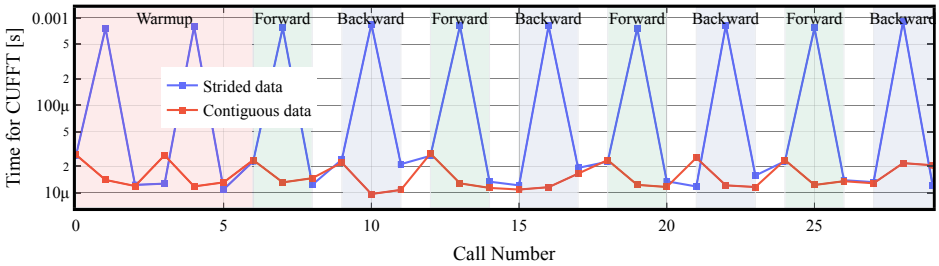

Fig. 10. Time for the computation of a batch of 1-D complex-to-complex FFTs of size 512 using CUFFT within a 3-D FFT computation.

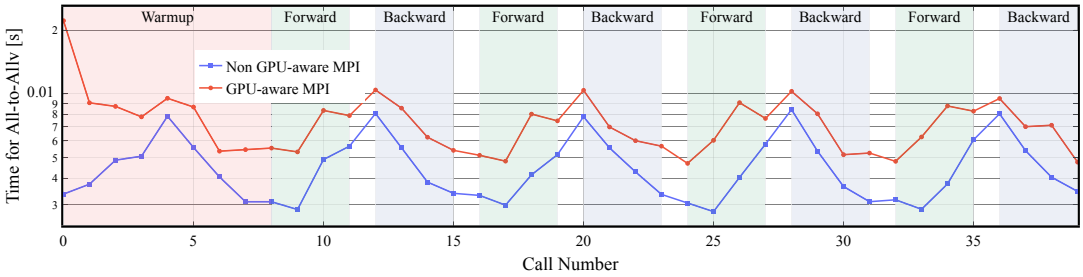

Fig. 11. Comparison of MPI_Alltoallv performance with and without GPU-aware MPI.

difference is considerable. Indeed, this also happens when using FFTW and rocFFT. Therefore, one may opt to always use contiguous input for the 1-D or 2-D local FFTs; however, this increases the packing/unpacking costs which can be orders of magnitude larger than the cuFFT cost ($\approx 15\mu$s). Amongst all of our experiments, the strided data version with AlltoAllv communication gave the best runtime for large number of nodes (≥ 64) with GPU accelerators.

C. Scalability and the Effect of GPU-aware MPI

Parallel FFT libraries with GPU support such as *heFFTe* [18] and AccFFT [15] have shown good linear scaling for large number of GPUs ($\approx 6,000$). And to extend this scaling to exascale systems, we may need a combination of Point-to-Point and All-to-All approaches (fftMPI already provides this feature for CPU arrays). In this context, Figs. 8 and 9 show that for up to 768 GPUs, All-to-All approaches scale quite well, while the Point-to-Point approaches fail when using GPU-aware MPI. If the GPU awareness is disabled, they keep scaling; however, in this case the data movement is performed as follows: device $\rightarrow$ host $\rightarrow$ host $\rightarrow$ device. In some cases, this is beneficial; and indeed, for small number of nodes the Point-to-Point approach is the fastest. However, for large number of nodes (which interests us for scalability purposes) disabling the GPU-aware feature can increase the communication cost in $\approx$30%, c.f., Fig. 11, where we show a runtime comparison at 16 nodes when switching between these options. The flag `-no-gpu-aware` from *heFFTe* helps to easily perform such experiments and the results are consistent for the different node counts studied in this paper.

D. FFT Optimization Effect on Real-world Simulations

Several large-scale simulation software rely on the computation of 3-D FFTs for essential tasks, amongst the most widely-used:

- LAMMPS [23] uses 3-D real and complex transforms for its KSPACE package, which computes long-range Coulombic interactions.
- HACC [25] relies on 3-D FFTs for N-Body simulations in astrophysics applications.
- WarpX [40] uses 3-D FFTs for energy computation on particle simulations. This software, in particular, uses MPI_Alltoallw with derived data types for global redistributions, and as shown in Section II, it can highly benefit from MPI GPU-aware optimizations.
- Within the machine learning community, there is an increasing interest in algorithms for fast, scalable and portable real and complex FFTs, for applications such as pattern recognition [41].

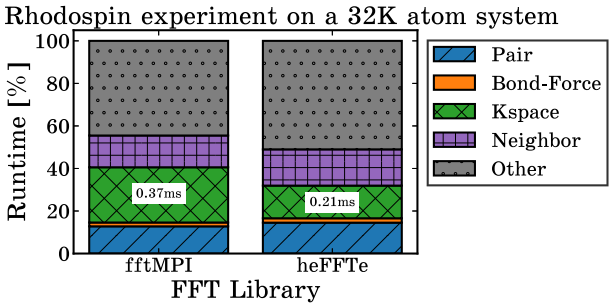

Fig. 12. Breakdown for the LAMMPS Rhodopsin experiment. Using 32 nodes, 6 V-100 GPUs per node, and 1 MPI per GPU.

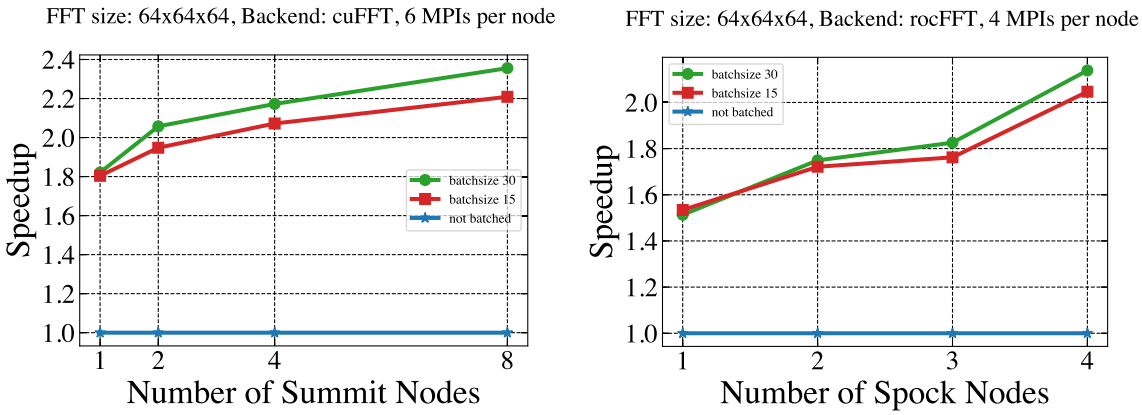

Fig. 13. Batched computation of a 3-D FFT of size 64^3 on NVIDIA (left) and AMD (right) GPUs, setting 1 MPI per GPU. We observe speedups of over $2\times$ with respect to the not batched version. At publication date, authors were not allowed to use more than four Spock nodes (being a prototype supercomputer), this is the reason why the plot was not further scaled.

In Figure 12, we show the runtime breakdown for a standard LAMMPS benchmark, using 32 nodes and a fixed 512^3 FFT grid. We observe that the runtime for the KSPACE computation is reduced around 40% when switching from its default fftMPI (with pencils approach) to *heFFTe*, for which we select the best parameter settings guided by Fig. 5. For fftMPI, we used its cuFFT enabled version, recently added to LAMMPS library [42].

Applications listed above often require not one but many transforms per iteration, and typically the size of the FFTs are small. Therefore, we have implemented Algorithm 1 in *heFFTe* [30] to support batched 2-D and 3-D transforms. In Fig. 13, we take a 3-D FFT of size 64^3 (found in several application benchmarks) and show over $2\times$ speedups, when comparing the cost of a single 3-D transform within a batch, to an isolated not batched computation. These speedups come from the overlap of communication and computation. The more transforms per MPI unit generates more overlap with network exchanges. If needed, we can also use the grid shrinking feature from Algorithm 1 to use less processors and increase the number of flops. When the problem size increases, e.g., 512^3 (as in previous experiments), the advantage of batching in GPU-based systems is considerably reduced since computation cost becomes negligible compared to the communication cost. Finally, the *heFFTe* batched implementation of this paper is, to our knowledge, the only one that supports AMD, Intel[§] and NVIDIA GPUs, and the speedups obtained from its usage can be extremely helpful for FFTs on higher dimensions and to ensure scalability on the upcoming exascale supercomputers.

[§]In this work, we were not able to include results from Intel GPUs due to permission and supercomputer access limitations at publication date.

V. Conclusions

In this paper we studied the performance of distributed FFTs on systems with GPU accelerators. We introduced a new portable algorithm that pushes the boundaries of the existing FFT software ecosystem, covering a wide range of methodologies from state-of-the-art libraries, and adding novel features such as FFT grid shrinking and batched 2-D and 3-D transforms. Where the latter showed speedups over $2\times$ on systems with AMD and NVIDIA GPU accelerators. We compared the pencils and slabs decompositions, the most common ones for parallel FFTs, and measured their theoretical performance by developing a bandwidth model, which can be used for selecting the fastest decomposition on a given hardware. We showed that a careful tuning of the algorithm yields to linear scalability and presented a case study using 3072 GPUs. We also studied when an Alltoall, Alltoallv or AlltoAllw approach could be beneficial according to network capabilities; as well as the cases where a Point-to-Point exchange would be the best choice. Finally, we showed how application software targeting exascale can benefit from the contributions of this paper.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[2] M. Frigo and S. G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[3] "Intel Math Kernel Library." [Online]. Available: https://software.intel.com/mkl/features/fft

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

[5] "rocFFT library," 2021. [Online]. Available: https://github.com/ROCmSoftwarePlatform/rocFFT

[6] "cuFFT library," 2021, Available at http://docs.nvidia.com/cuda/cufft.

[7] Intel, "Intel OneAPI Library." [Online]. Available: https://software.intel.com/oneapi/fft.html

[8] "Vulkan FFT library," 2021. [Online]. Available: https://github.com/DTolm/VkFFT

[9] "KFR library," 2021. [Online]. Available: https://github.com/kfrlib/kfr

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

[11] L. Dalcin, M. Mortensen, and D. E. Keyes, "Fast parallel multidimensional FFT using advanced MPI," *Journal of Parallel and Distributed Computing*, vol. 128, pp. 137–150, 2019.

[12] C. H. Q. Ding, R. D. Ferraro, and D. B. Gennery, "A Portable 3D FFT Package for Distributed-Memory Parallel Architectures," in *PPSC*, 1995.

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

[14] A. Ayala, S. Tomov, P. Luszczek, G. Ragghianti, S. Cayrols, and J. Dongarra, "Interim Report on Benchmarking FFT Libraries on High Performance Systems," University of Tennessee, ICL Tech Report ICL-UT-21-03, 2021-07 2021.

[15] A. Gholami, J. Hill, D. Malhotra, and G. Biros, "AccFFT: A library for distributed-memory FFT on CPU and GPU architectures," *CoRR*, vol. abs/1506.07933, 2015.

[16] D. Takahashi, "FFTE: A fast Fourier transform package," *http://www.ffte.jp/*, 2005.

[17] "fftMPI: Parallel 2D and 3D complex FFTs," 2021, Available at https://fftmpi.sandia.gov.

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

[19] D. Pekurovsky, "P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C192–C209, 2012.

[20] M. Pippig, "PFFT: An extension of FFTW to massively parallel architectures," *SIAM J. Sci. Comput.*, vol. 35, 2013.

[21] N. Li and S. Laizet, "2DECOMP&FFT - A Highly Scalable 2D Decomposition Library and FFT Interface," *Cray User Group 2010 conference*, 2010.

[22] J. H. Göbbert, H. Iliev, C. Ansorge, and H. Pitsch, "Overlapping of Communication and Computation in nb3dfft for 3D Fast Fourier Transformations," in *High-Performance Scientific Computing*, E. Di Napoli, M.-A. Hermanns, H. Iliev, A. Lintermann, and A. Peyser, Eds. Cham: Springer International Publishing, 2017, pp. 151–159.

[23] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. Michael Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - A flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications*, p. 108171, 2021.

[24] D. Richards, O. Aziz, J. Cook, H. Finkel *et al.*, "Quantitative Performance Assessment of Proxy Apps and Parents," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2018.

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

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

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

[28] K. Ravikumar, D. Appelhans, and P. K. Yeung, "GPU Acceleration of Extreme Scale Pseudo-Spectral Simulations of Turbulence Using Asynchronism," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019.

[29] I. T. Foster and P. H. Worley, "Parallel Algorithms for the Spectral Transform Method," *SIAM J. Sci. Comput.*, vol. 18, no. 3, p. 806–837, 1997.

[30] "heFFTe library," 2021, Available at https://bitbucket.org/icl/heffte.

[31] T. Hoefler and S. A. Gottlieb, "Parallel Zero-Copy Algorithms for Fast Fourier Transform and Conjugate Gradient Using MPI Datatypes," in *EuroMPI*, 2010.

[32] "Release notes on IBM SpectrumMPI 10.4," 2021, Available at https://www.ibm.com/docs/en/smpi/10.4?topic=release-notes.

[33] A. G. Chatterjee, M. K. Verma, A. Kumar, R. Samtaney, B. Hadri, and R. Khurram, "Scaling of a Fast Fourier Transform and a pseudo-spectral fluid solver up to 196608 cores," *J. Parallel Distributed Comput.*, vol. 113, pp. 77–91, 2018.

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

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

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

[37] K. Czechowski, C. McClanahan, C. Battaglino, K. Iyer, P.-K. Yeung, and R. Vuduc, "On the communication complexity of 3D FFTs and its implications for exascale," *Proceedings of the International Conference on Supercomputing*, 2012.

[38] 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.

[39] A. Ayala, S. Tomov, M. Stoyanov, A. Haidar, and J. Dongarra, "Accelerating Multi-Process Communication for Parallel 3-D FFT," in *2021 Workshop on Exascale MPI (ExaMPI)*, 2021, pp. 46–53.

[40] D. Grote, J.-L. Vay, A. Friedman, and S. Lund, "Warp Software," 2021. [Online]. Available: https://sites.google.com/a/lbl.gov/warp/home

[41] N. Monnier, D. Ghali, and S. X. Liu, "FFT and machine learning application on major chord recognition," in *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2021, pp. 426–429.

[42] "LAMMPS library," 2021. [Online]. Available: https://github.com/lammps/lammps