Impacts of Multi-GPU MPI Collective Communications on Large FFT Computation

Alan Ayala
Innovative Computing Laboratory
The University of Tennessee
Knoxville, USA

Stanimire Tomov
Innovative Computing Laboratory
The University of Tennessee
Knoxville, USA

Xi Luo
Innovative Computing Laboratory
The University of Tennessee
Knoxville, USA

Hejer Shaiek
Innovative Computing Laboratory
The University of Tennessee
Knoxville, USA

Azzam Haidar*
Nvidia Corporation
California, USA

George Bosilca
Innovative Computing Laboratory
The University of Tennessee
Knoxville, USA

Jack Dongarra
Innovative Computing Laboratory
The University of Tennessee
Knoxville, USA

Abstract—**Most applications targeting exascale, such as those part of the Exascale Computing Project (ECP), are designed for heterogeneous architectures and rely on the Message Passing Interface (MPI) as their underlying parallel programming model. In this paper we analyze the limitations of collective MPI communication for the computation of fast Fourier transforms (FFTs), which are relied on heavily for large-scale particle simulations. We present experiments made at one of the largest heterogeneous platforms, the Summit supercomputer at ORNL. We discuss communication models from state-of-the-art FFT libraries, and propose a new FFT library, named HEFFTE (Highly Efficient FFTs for Exascale), which supports heterogeneous architectures and yields considerable speedups compared with CPU libraries, while maintaining good weak as well as strong scalability.**

Index Terms—**Exascale applications, FFT, scalable, Collective MPI, Heterogeneous systems**

I. Introduction

One of the major challenges supercomputing has been facing over the last decade is to reduce the cost of communication between processors. The fast development of graphics processing units (GPUs) has enabled great speedup on computations locally, and nowadays a great part of the runtime of several applications is consumed in communication [1]. In this paper we are interested in the distributed fast Fourier transform (FFT), which is one of the most important algorithms needed for exascale applications in computational science and engineering. Diverse parallel libraries exist that rely on efficient FFT computations—in particular within the field of particle applications [2]—ranging from molecular dynamics computations to N-Body simulations. Indeed, libraries in this applications area, such as LAMMPS [3] and HACC [4] are targeting exascale computer systems and require highly scalable and robust FFT kernels.

Thus, for all these applications it is critical to have access to a fast and scalable implementation of an FFT algorithm, an implementation that can take advantage of novel hardware components, and maximize these components' benefits for applications. Moreover, as the current trend in the development of HPC platforms is to add an increased number of accelerators per node, creating an imbalance between computation and communication at the node level, it is necessary to investigate how the sheer compute power affects the performance and scalability of FFT algorithms and how to take advantage of this power by using efficient communication platforms.

This paper is organized as follows: in Section II we describe the classical FFT algorithm and the parallel MPI communication model for state-of-the-art libraries; we also introduce a novel library called HEFFTE. Next, Section III presents the communication bottleneck for FFT and approaches to overcome it. Section IV presents an MPI analysis of the routines needed within classical parallel FFT. Section V presents numerical experiments on Summit, and the limitations of hybrid FFT implementations highlighting the case of HEFFTE. Finally, Section VI concludes our paper, giving directions for future work and collaborations.

II. HEFFTE algorithmic design

It is well known that multidimensional FFTs can be performed by a sequence of low-dimensional FFTs (e.g., see [5]). Algorithm 1 presents the classical steps for parallel 3D FFT, which are also displayed schematically in Figure 1. Two main sets of kernels intervene into the 3D FFT computation: those in charge of the three sets of 1D FFTs, and those that take care of reshaping the data at each stage.

In Figure 1 we show the FFT computation steps having a 3D tensor as input data of size $N_0 \times N_1 \times N_2$ distributed among a grid of P processes, $P = P_0 \times P_1 \times P_2$; the algorithm will perform as stated in Algorithm 1. For notation purposes: $\hat{N}_i$ is the output of 1D FFTs of size N_i in direction i. Note that the orange boxes belong to processor 1 during the entire computation. This approach is called pencil-to-pencil, and the computation of a 3D FFT is carried by computing 3 sets of 1D FFTs.

* This work was done while the author was at the University of Tennessee.

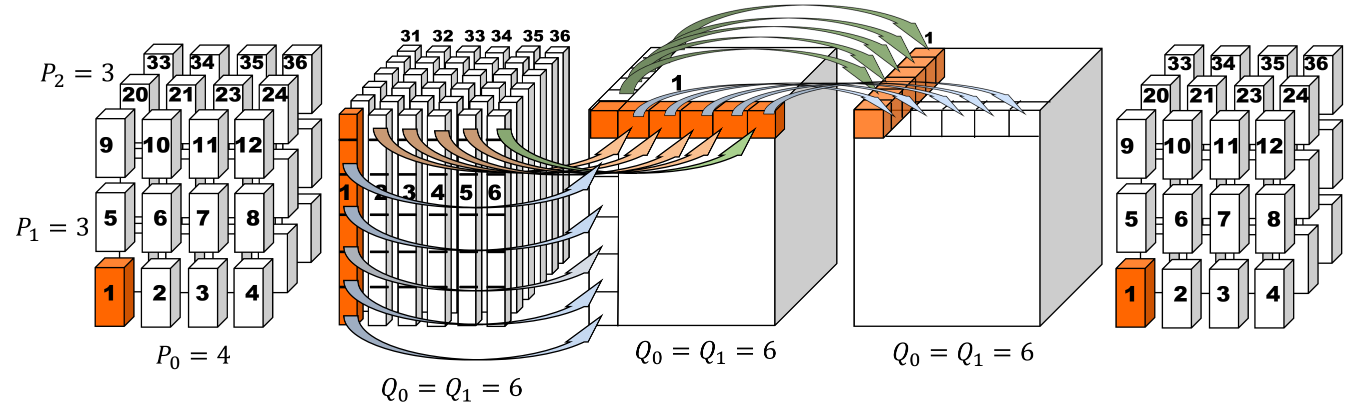

Fig. 1. Phases of 3DFFT computation with pencil decomposition, schematic procedure of Algorithm 1.

Algorithm 1 3D Distributed FFT algorithm based on pencil decomposition

```
Require: Data in Spatial domain: Layout $N_0/P_0 \times N_1/P_1 \times N_2/P_2$
Ensure: Data in Fourier domain: Layout $\widehat{N_0}/P_0 \times \widehat{N_1}/P_1 \times \widehat{N_2}/P_2$

Calculate a 2D grid $Q_0$ and $Q_1$ st $Q_0 \times Q_1 = P_0 \times P_1 \times P_2$
$N_0/P_0 \times N_1/P_1 \times N_2/P_2 \xrightarrow{\text{Reshape}} N_0 \times N_1/Q_0 \times N_2/Q_1$
$N_0 \times N_1/Q_0 \times N_2/Q_1 \xrightarrow{\text{First Dimension 1D FFTs}} \widehat{N_0} \times N_1/Q_0 \times N_2/Q_1$
$\widehat{N_0} \times N_1/Q_0 \times N_2/Q_1 \xrightarrow{\text{Reshape}} \widehat{N_0}/Q_0 \times N_1 \times N_2/Q_1$
$\widehat{N_0}/Q_0 \times N_1 \times N_2/Q_1 \xrightarrow{\text{Second Dimension 1D FFTs}} \widehat{N_0}/Q_0 \times \widehat{N_1} \times N_2/Q_1$
$\widehat{N_0}/Q_0 \times \widehat{N_1} \times N_2/Q_1 \xrightarrow{\text{Reshape}} \widehat{N_0}/Q_0 \times \widehat{N_1}/Q_1 \times N_2$
$\widehat{N_0}/Q_0 \times \widehat{N_1}/Q_1 \times N_2 \xrightarrow{\text{Third Dimension 1D FFTs}} \widehat{N_0}/Q_0 \times \widehat{N_1}/Q_1 \times \widehat{N_2}$
$\widehat{N_0}/Q_0 \times \widehat{N_1}/Q_1 \times \widehat{N_2} \xrightarrow{\text{Reshape}} \widehat{N_0}/P_0 \times \widehat{N_1}/P_1 \times \widehat{N_2}/P_2$
```

As shown in Algorithm 1, the input data is reshaped four times, going successively from the input array to each of the 3 leading dimensions and then to the final output array. Note that that we can reduce the number of reshapes to two, provided we have the input array containing pencils on the first direction ready to compute, and then stopping the reshape phase at the third leading dimension (for comparison, see Figure 1). To perform such reshapes of data across processes we rely on the Message Passing Interface (MPI). HEFFTE supports two options for communications: binary exchange (point-to-point) and all-to-all.

A. Communication model

It is well known that the bottleneck for parallel FFTs is the communication of data between tasks (see e.g., [6]), which significantly affects the scaling at large core/GPU counts. As will be shown in Section V, the communication time dominates the cost of distributed FFTs. For the case of GPU multi-node implementation, we can use fast interconnection such as NVLINK within a node; however, the situation is worse for inter-node communication since the data has to be transferred back and forth to the CPU through PCIe.

The first step to reshape an input array, cf. Algorithm 1, is to perform packing/unpacking phases to make the data contiguous in the memory. These phases generally account for less than 10% of the reshaping time. Next, a communication scheme is performed, and this can be done in two different ways. The first one is the point-to-point (sends/receives) approach where every processor has a loop over all the processors that overlap with it. The second one is the all-to-all approach, where all tasks within a sub-communicator communicate to each other.

For most parallel FFT libraries the communication phase is typically handled by moving data structures on the shape of pencils, bricks, and slabs of data. For each of these options the total amount of data communicated is always the same, so decreasing the number of messages between processors means increasing the size of the messages they send. For instance, if the communication takes brick-shaped data (3D blocks) into pencil-shaped data (1D blocks), and assuming that the data is originally distributed among $P_0 \times P_1 \times P_2$, every row ($P_0$) of processors communicate together to get P_0 sets of complete rows of data. Thus, there will be P_0 messages in that case. By a similar reasoning, we can get the asymptotic number of messages sent by the different communication schemes. The approach brick-shaped data $\Leftrightarrow$ pencil-shaped data requires $O(P^{1/3})$ messages, where P is

Libraries	Point-to-point routines		Collective routines		Process Topology
	Blocking	Non-blocking	Blocking	Non-blocking	
FFTMPI	MPI_Send	MPI_Irecv	MPI_Allttoallv MPI_Allreduce MPI_Barrier	None	MPI_Comm_create MPI_Group
SWFFT	MPI Sendrecv	MPI_Isend MPI_Irecv	MPI_Allreduce MPI_Barrier	None	MPI_Cart_create MPI_Cart_sub

TABLE I
MPI ROUTINES REQUIRED FOR HEFFTE COMMUNICATION FRAMEWORK, INHERITED FROM FFTMPI AND SWFFT LIBRARIES.

the total number of processors. Reshaping pencil-shaped ⇔ pencil-shaped uses $O(P^{1/2})$ messages, brick-shaped ⇔ slab-shaped data requires $O(P^{2/3})$ messages, and slab-shaped ⇔ pencil-shaped data requires $O(P)$ messages.

B. HEFFTE library

HEFFTE is a C++ library for FFT calculation on large heterogeneous systems, and it is part of the Exascale Computing Project (ECP). It is documented in [7]–[10] and aims to become the standard for large FFT computations on the upcoming exascale systems. HEFFTE is open source and will include wrappers for C, Fortran, and Python, as well as wrappers for large-scale applications such as LAMMPS [3] and HACC [11]. HEFFTE supports two options for communications: binary exchange (point-to-point) and all-to-all. For the computation of 1D FFTs, we call an external library—HEFFTE currently supports FFTW3 [12], MKL [13], and CUFFT [14], although other 1D FFT libraries can be supported.

The current version of HEFFTE was built by optimizing the kernels and communication frameworks of two well documented libraries, FFTMPI [15] (built-in on LAMMPS software [3]) and SWFFT [16] (built-in on HACC software [11]). Table I shows the MPI routines FFTMPI and SWFFT call, and hence the ones that HEFFTE needs. From our experiments—see [8, Sec 3.3] [10]—we know that point-to-point communication gives better performance for small CPU/GPU counts, while for large counts it is all-to-all that gives best performance. In the next two sections we study the theoretical bounds of performance as well as its limitations on large heterogeneous supercomputers like Summit.

III. COMMUNICATION BOTTLENECK

As shown in Section V, a parallel GPU implementation speeds up the FFT execution; however, this speedup is limited since the communication takes in general more than 95% of the runtime. To better analyze this limitation, we have developed a performance model to evaluate how well HEFFTE library can perform and to guide upcoming optimization efforts. A challenge in developing high-performance FFTs is that the FFT computation does not have a high computational intensity (defined as the FLOPs/Byte ratio). Indeed, Table II shows the FFT's computational intensity in contrast to the dense matrix-matrix multiplication in double complex arithmetic (ZGEMM).

Knowing the FLOPs/Byte rate one can directly compute a roofline performance model based on the bandwidth rate B (e.g., given in GB/s) that provides the data. For example, the performance P_{ZGEMM} for ZGEMM to read 3 matrices and write back 1 in gigaFLOP/s is bounded by:

$$P_{ZGEMM} \leq min\left\{6900, \frac{0.375\ N}{4}\ B\right\}.$$

Thus, if we have a PCIe connection of 12 GB/s and want to compute the minimal N that reaches asymptotic performance (of 6900 gigaFLOP/s in this case), we solve

$$6900 = 12\ \frac{0.375\ N}{4}, \quad \text{to find} \quad N = 6,133.$$

This means that if we have many matrices of size $N = 6,133$ to multiply, we can pipeline the computation and communication so that communication is totally overlapped with computation (i.e., resulting in an overall top performance of 6900 gigaFLOP/s). This is the basis of blocking in dense linear algebra and finding the smallest blocking size that still gives peak performance.

Similarly, we derive that the performance P_{FFT} for a batch of 1D FFTs is bounded by:

$$P_{FFT} \leq min\left\{600, \frac{0.312\ log_2 N}{2}\ B\right\}.$$

In the previous inequality, the division by 2 is for taking into account the process of reading the vectors and writing them back. Note that for this bound, N would have to be unrealistically large in order to get the performance bounded by 600. In other words, the intensity of the computation does not grow fast enough to apply some blocking techniques like in the dense linear algebra case. In conclusion, performance is always memory bound:

$$P_{FFT} \leq \frac{0.312\ log_2 N}{2}\ B.$$

Thus, nodal performance in an InfiniBand (IB) 100Gb network (12.5 GB/s bandwidth, or B=25 GB/s for the bidirectional communications in FFT) with $N = 2,000$ will be limited by

$$25 * 0.312 * 11/2 = 42.9 \text{ gigaFLOP/s}.$$

Operation	GFlop/s 1 V100 GPU	GFlop/s 6 V100 GPUs	Flops	Bytes	Flops/B
Batch of B 1D FFTs	600	3,600	5 B N $\log_2$ N	16 B N	0.312 $\log_2$ N
ZGEMM	6900	41,400	6 N^3	16 N^2 (*4)	0.375 N (/4)

TABLE II
COMPUTATIONAL INTENSITY IN FLOPS/BYTE FOR 1D FFTS (VS. GEMM) IN DOUBLE COMPLEX ARITHMETIC. LISTED ALSO ARE THE ACHIEVABLE PERFORMANCES FOR THE TWO OPERATIONS IN GIGAFLOP/S ON SINGLE V100 GPU AND A NODE OF 6 V100 GPUS, AS ON THE SUMMIT SUPERCOMPUTER. THE MULTIPLICATION BY 4 AND DIVISION BY 4 FOR GEMM IS TO TAKE INTO ACCOUNT THAT 3 MATRICES ARE READ AND ONE IS WRITTEN BACK TO STORAGE.

Similarly, nodal performance on Summit, featuring nodal bandwidth of 50 GB/s through Dual Rail EDR-IB, will be limited by

$$P_{FFT} \leq 50 \ \frac{0.312 \ log_2 N}{2} = 7.8 \ log_2 N.$$

For example, if $N = 10,000$, nodal performance will be bound by 104 gigaFLOP/s. This means that further optimizations for the nodal FFTs will have limited effect on the overall performance, since 104 gigaFLOP/s will be the maximum that can be extracted from the node (while currently we can achieve much more: 600 gigaFLOP/s from just one GPU). Still, current FFT results on Summit show scalability of $\approx$ 9 gigaFLOP/s per node, which gives a potential speedup of $10\times$ acceleration while still leaving GPU resources for other computations, as typically needed in applications [11].

To reach close to the roofline performance peak for the model presented, FLOPs must be overlapped with the communication. Communication alone usually is the larger fraction of the entire computation, as also illustrated and quantified for some of the runs shown in Section V. Figure 2 shows another example for time needed for communication vs. time for computation. In this example NVLINK marks the time to receive and send the data for the local GPU computation through a 32 GB/s connection. If the receiving and sending of the 1D vectors is pipelined, the NVLINK curve gives the total time; otherwise, the total execution time is about twice as long, at least for the specifics in this illustration.

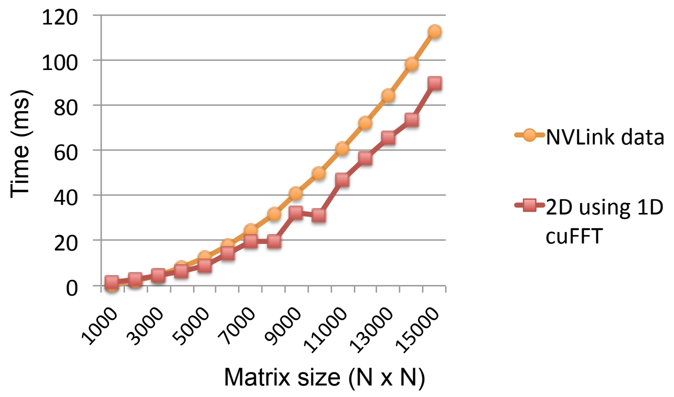

Fig. 2. Time for 2D FFT using 1D cuFFTs on NVIDIA V100 GPU. The NVLINK shows the time to receive and send the data for the computation through a 32 GB/s connection. Computation and communication can be overlapped by pipelining the work on the 1D vectors, in which case the total computation time is given by the NVLINK curve; otherwise is the sum of the two curves (i.e., about two times slower in this case).

TABLE III
SYMBOLS USED IN THE COST MODEL

Symbol	Description
n	Number of Nodes
p	Number of Processes per Node
M	Message size per Processes
B_{inter}	Inter-node P2P bandwidth

IV. MPI COLLECTIVE COMMUNICATION

On a hybrid, multi-GPU and multi-lane computing environment such as Summit, it is important to understand how the communications can be optimized to maximize the bandwidth at each level of the hardware hierarchy. Considering the operation to be performed—alltoall—we have developed a performance model to assess the performance of the entire collective communication on Summit.

Table III shows the symbols used in the performance model. The time for an MPI_Alltoall operation can be divided into two parts: intra-node and inter-node. The upper bound of the time for the entire operation is the sum of the two, when no overlap is possible between the two parts—while optimally scheduling the intra-node communications would hide their cost behind the inter-node communications, so the lower bound is the maximum time of the two parts which equals to the cost of inter-node part in most cases. Also, with an increasing number of nodes, the cost of the inter-node part grows linearly (as each node must send a well-defined amount of data to each other node), while the cost of the intra-node part remains constant and eventually become negligible. Hence, our cost model focuses on the inter-node part and assumes perfect overlap of the two parts. Instead of looking at the alltoall operation from an each involved process perspective, we shift our focus to the external links, and we reason at the level of the nodes. Thus, for each node, the collective transfers $(n-1) \times p^2 \times M$ bytes data to other nodes; and suppose the inter-node P2P bandwidth is B_{inter}, the time of an MPI_Alltoall operations is:

$$T_{alltoall} = T_{inter} = (n-1)p^2 M / B_{inter}. \quad (1)$$

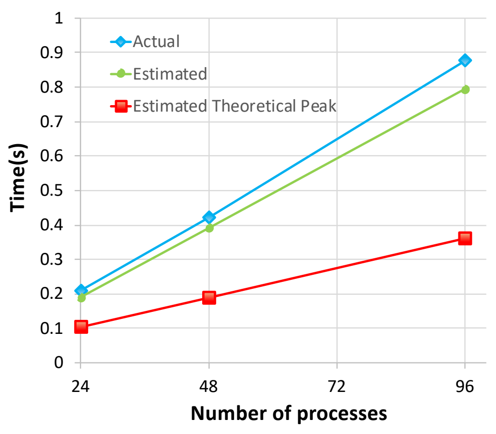

Fig. 3. MPI_Alltoall performance on Summit (16MB).

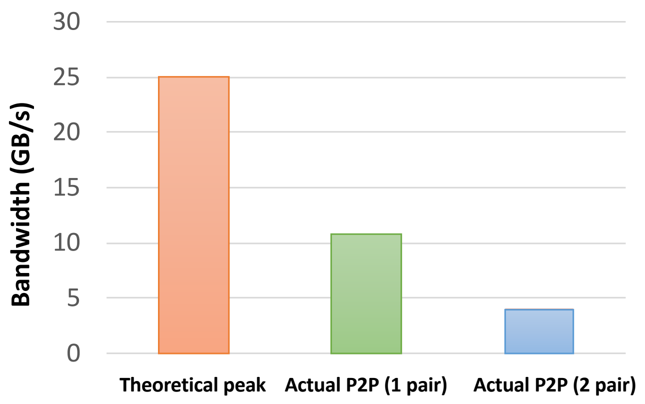

Fig. 4. MPI P2P performance on Summit (16MB).

Figure 3 presents the time of MPI_Alltoall operations with the various number of processes. In this experiment, each node has 6 processes, and each process binds to a GPU and transfers large enough data (16 MB) which is able to reach the peak P2P bandwidth. Based on the Summit architecture configuration, the theoretical inter-node bandwidth is 25GB/s (2 lanes of 12.5GB/s); using this bandwidth in our cost model, the estimated time is shown as the red line in the Figure 3. Meanwhile, the actual performance is the blue line, which is far from the theoretical peak.

Based on our tests, we think the most possible explanation for this performance gap is the low inter-node P2P bandwidth. The green bar on the Figure 4 shows the inter-node P2P bandwidth on Summit using Netpipe [17], which suggests that one pair of P2Ps among two nodes can only utilize half of the theoretical bandwidth. This may be caused by the one pair of P2P being able to use only one lane of the interconnect. In order to attempt to achieve higher bandwidth utilization, we modify the benchmark to allow two pairs of concurrent P2Ps among two nodes and make sure these two P2Ps are on different sockets to avoid the resource competition. The result is shown by the blue bar on Figure 4; with the doubled number of pairs, the bandwidth of each pair decreases to half, which indicates the MPI P2P communications on different sockets interfere with each other, even though in theory they should be independent. In this way, our attempt fails and we cannot reach full bandwidth with the current version of MPI (spectrum-mpi 10.3.0.1) on Summit for P2P communication.

The green line on Figure 3 shows the estimated time of using the actual P2P performance from Netpipe in our cost model. This result is much closer to the actual performance, which suggests the performance gap of MPI_Alltoall operation to the theoretical peak is because of the low P2P performance.

From the previous analysis, we note that multi-rail communication will need to be carefully managed across all processes on the node in order to get any potential benefit. However, the cost of such management might negatively impact the performance, degrading the potential benefit of the multi-rail optimization.

V. Numerical Experiments

In this section we present numerical experiments on Summit. We start by looking at the computational performance comparison between a parallel hybrid implementation (HEFFTE) and a parallel CPU implementation FFTMPI. In Figure 5 we observe that HEFFTE outperforms FFTMPI (which yields similar performance numbers as SWFFT) by a factor of approximately 2× speedup.

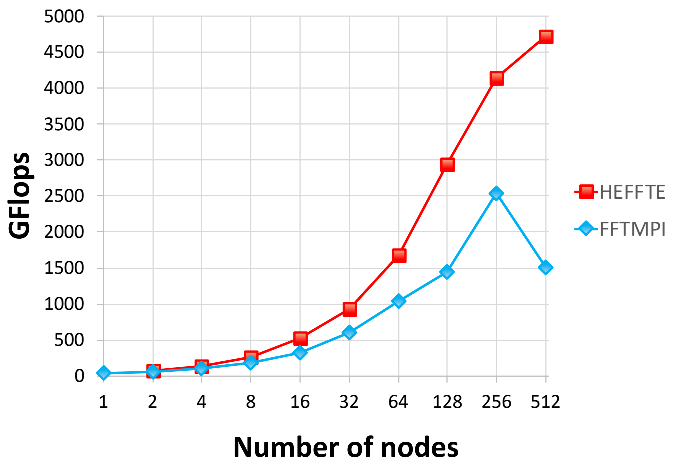

Fig. 5. Strong scalability on 3D FFTs of size 1024^3, for FFTMPI using 40 cores per node (blue) and HEFFTE using 6 V100 GPUs per node (red).

The drop in performance for FFTMPI in Fig. 5 might be explained by the cost of collective communication between over 20,000 processes, which is highly impacted by Summit latency. For HEFFTE, at 512 nodes we have 3072 processes communicating between each other, and intra-node communication is performed on fast NVLINK interconnection. For this problem size, HEFFTE would likely have a performance drop at some large node count.

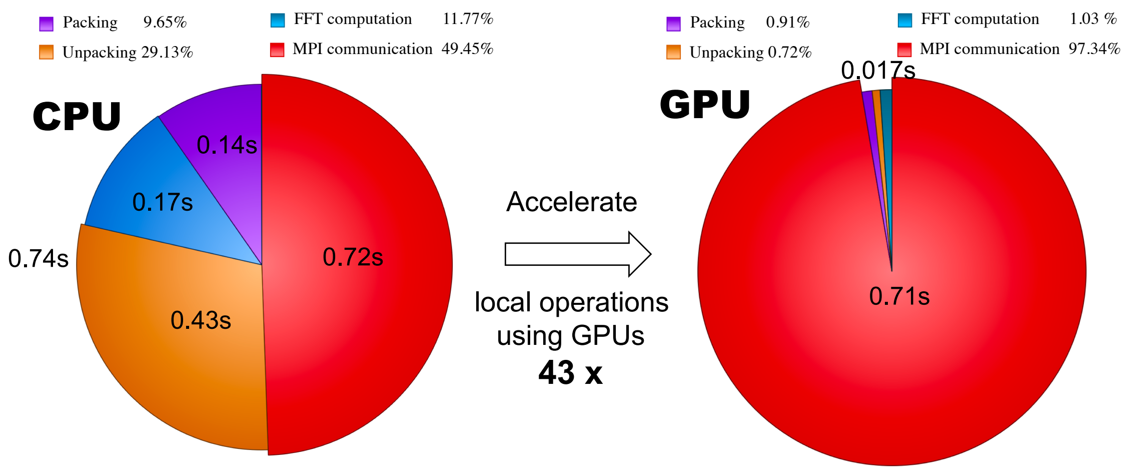

Fig. 6. Profile of a 3D FFT of size 1024^3 on 4 CPU nodes – using 128 MPI processes, i.e., 32 MPIs per node, 16 MPIs per socket (Left) vs. 4 GPU nodes – using 24 MPI processes, i.e., 6 MPIs per node, 3 MPI per socket, 1 GPUs per MPI (Right)

A. *Limitations of HEFFTE library*

The speedups that HEFFTE reports come from the classical approach of state-of-the-art FFT libraries, which consists of reducing the kernels run-time, and overlapping computation and communication. However, there is not much that can be improved on terms of GPU computations, and therefore we need to search for improvements on the communication scheme. To make this point clear, in Figure 6 we present a single experiment where the GPU computation speeds up the CPU computations by 43×. We achieve this speedup by using the 1D FFTs from CUFFT and fast matrix transpositions for the MPI packing and unpacking operations from the MAGMA library [18]. However, over 97% of the run time is now spent only in communication.

Both HEFFTE and FFTMPI get their best performance when using collective communication (particularly all-to-all MPI calls), and these kinds of MPI functions do not use close to the peak of the bandwidth available, and optimizations of such routines are still not available (e.g., on the NVIDIA Collective Communications Library [NCLL] for GPUs efficient communications). Hence, new and better versions of MPI_All2Allv functions are required. To that end, we developed our own routine called Magma_All2Allv (introduced in [8]), which is based on non-blocking data transfer plus the use of CUDA IPC memory handlers. Magma_All2Allv improves collective all-to-all communications on up to 32 nodes—see Figure 7—but starts failing as the node count increases. This phenomenon might be due to the fact that inter-node communication requires data to pass through the host which makes communication slower than direct GPU to GPU transfers, and 32 nodes is, according to our experiments on Summit, the maximum number of nodes to take advantage of these kind of tools.

Fluctuations observed in Fig. 7, may relate to the fact

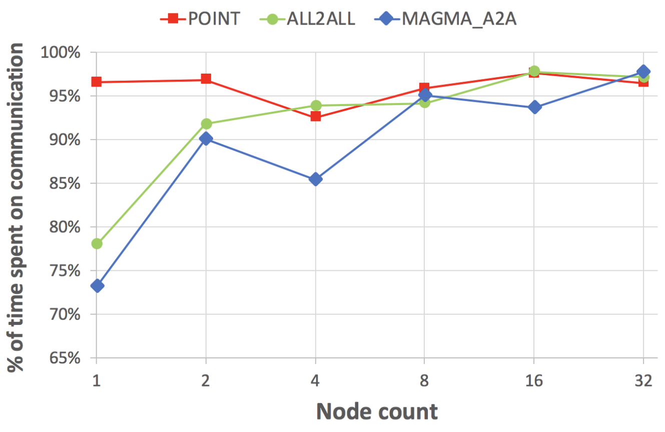

Fig. 7. Comparing the runtime impact of different MPI communication supported on HEFFTE.

that our 3DFFT algorithm (c.f. Alg. 1) works with different processor grids at each stage, and therefore it does require more steps of communication (e.g. initial FFT grid to first direction and/or third direction to final FFT grid, c.f. Fig. 1) for some processor counts, and do not require them for others.

To further improve the performance and gain more speedup, we might want to use single-precision data (which is supported by HEFFTE). However, it would be more efficient to develop communication-avoiding paradigms to achieve further speedups by keeping good scalability. This could be achieved by modifying the algorithms to trade less communication for more computation.

VI. Conclusions

We have presented the limitations of hybrid parallel implementation of FFTs and discussed the communication bottleneck and possible solutions to overcome these difficulties. We presented a detailed analysis of the communication model and how it limits the computation performance, which has then been verified with numerical experiments on a large heterogeneous supercomputer (Summit at Oak Ridge National Laboratory). We reported that communication takes over 95% of the run time for large CPU/GPU counts, and to understand this issue we have analyzed Spectrum-MPI's binary and all-to-all communication frameworks and how they behave on Summit platform. Finally, we have shown that our proposed library, HEFFTE, speeds up almost twice the current state-of-the-art libraries being used by ECP projects. This library has optimized kernels; however, it needs to further optimize the communication framework to achieve close to peak performance.

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] S. J. Plimpton, Ffts for (mostly) particle codes within the doe exascale computing project, 2017.

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

[4] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T. Peterka, J. Insley, D. Daniel, P. Fasel, Z. Lukić, Hacc: Extreme scaling and performance across diverse architectures, Commun. ACM 60 (1) (2016) 97–104. doi:10.1145/3015569.
URL http://doi.acm.org/10.1145/3015569

[15] 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).

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

[6] 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.

[7] 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).

[8] 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).

[9] 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).

[10] 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).

[11] 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).

[12] 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".

[13] Intel, Intel Math Kernel Library, http://software.intel.com/en-us/articles/intel-mkl/.
URL https://software.intel.com/en-us/mkl/features/fft

[14] C. Nvidia, Cufft library (2018).

[16] 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).

[17] Q. O. Snell, A. R. Mikler, J. L. Gustafson, Netpipe: A network protocol independent performance evaluator, in: in IASTED International Conference on Intelligent Information Management and Systems, 1996.

[18] S. Tomov, J. Dongarra, M. Baboulin, Towards dense linear algebra for hybrid gpu accelerated manycore systems, Parellel Comput. Syst. Appl. 36 (5-6) (2010) 232–240, DOI: 10.1016/j.parco.2009.12.005.