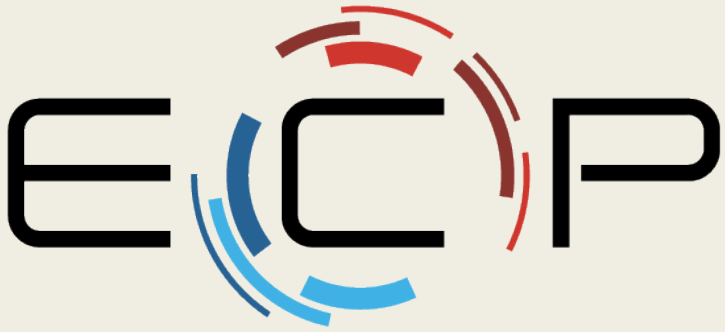

EXASCALE COMPUTING PROJECT

ECP Milestone Report

FFT-ECP Implementation Optimizations and Features Phase

WBS 2.3.3.09, Milestone FFT-ECP ST-MS-10-1440

Stanimire Tomov
Azzam Haidar
Alan Ayala
Hejer Shaiek
Jack Dongarra

Innovative Computing Laboratory, University of Tennessee

October 7, 2019

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

Revision	Notes
09-2019	first publication

```
@techreport{thasd2019ECPFFT,
  author={Tomov, Stanimire and Haidar, Azzam and Ayala, Alan and Shaiek, Hejer and Dongarra, Jack},
  title={{FFT-ECP Implementation Optimizations and Features Phase}},
  institution={Innovative Computing Laboratory, University of Tennessee},
  year={2019},
  month={September},
  type={ECP WBS 2.3.3.09 Milestone Report},
  number={FFT-ECP ST-MS-10-1440},
  note={revision 09-2019}
}
```

Contents

List of Figures

List of Tables

CHAPTER 1

Executive Summary

The goal of this milestone was the imlementation optimizations and features phase of 3-D FFTs in the FFT-ECP project. The target architectures are large-scale distributed GPU-accelerated platforms.

In this milestone we describe the implmentation optimizations, features, and performance of the 3-D FFTs that we developed for heterogeneous systems with GPUs. Specifically, this milestone delivered on the following sub-tasks:

- Extend FFT-ECP to support various precisions, including real, and investigate the feasibility of mixed precision FFT solvers;
- Develop support for flexible data layouts and enable the new library to handle data conversion/-communication on the backend in an optimized and dynamic adaptive fashion based on the communication cost model analyzed in the previous milestone;
- Optimize the distributed 3-D FFT-ECP solver to enable multiple FFTs per MPI process (with accelerators) and multiple GPUs per node.

A main part of this milestone were the performance optimizations and the additions of features that targeted ECP applications need.

The artifacts delivered include the performance optimizations and features added to the solvers, and a tuned FFT-ECP software, freely available on the FFT-ECP's Git repository hosted on Bitbucket, https://bitbucket.org/icl/heffte/. This is the first software release under the FFT-ECP project. Released is a new FFT library, called **heFFTe version 0.1** (Highly Efficient FFTs for Exascale).

See also the FFT-ECP website, http://icl.utk.edu/fft/ for more details on the FFT-ECP project.

CHAPTER 2

Background

The Fast Fourier Transform (FFT) is in the software stack for almost all ECP applications. This includes applications and simulations in molecular dynamics, spectrum estimation, fast convolution and correlation, signal modulation and many wireless multimedia applications.

The distributed 3-D FFT is one of the most important kernels needed for particle applications [6], particularly in Molecular Dynamics (MD) computations, e.g., as in the LAMMPS ECP apps project [3], and Nbody simulations, e.g., as in the HACC ECP apps project [1]. Other ECP particle apps of need for FFTs include ECP ExaAM and WarpX. Looking into the Spack package manager dependencies, we found that 60 scientific software packages depend on and use FFTs.

State-of-the-art FFT libraries like FFTW are no longer actively developed for emerging platforms. To address this deficiency, the FFT-ECP project, initiated as a new ECP effort targeting FFTs [2], aims to provide a sustainable 3-D FFT library for Exascale platforms.

The approach in FFT-ECP includes leveraging existing FFT capabilities, such as third-party 1-D FFTs from vendors or open-source libraries. This is also the approach in the `SWFFT` [7] and `FFTMPI` [5] FFT libraries, which are currently used in the HACC and the LAMMPS ECP apps projects, respectively. `FFTMPI` and `SWFFT` have very good weak and strong scalability on CPU-based systems. The goal of the ECP-FFT project for this period was the imlementation optimizations and features phase of 3-D FFTs. The supported features cover the `FFTMPI` and `SWFFT` functionalities for 3-D FFTs on GPU-accelerated Exascale platforms. The software is released under the **heFFTe version 0.1** library, freely available on the FFT-ECP's Git repository hosted on Bitbucket, https://bitbucket.org/icl/heffte/.

CHAPTER 3

FFT optimizations for GPUs

3.1 Target architecture

We are packaging the work under the FFT-ECP project [8, 12] into a new FFT library, called heFFTe. The heFFTe library aspires to be a new and sustainable high-performance FFT library for exascale platforms that leverages the large investments in FFT software by the broader HPC community. Indeed, there are many FFT libraries. Analysis and performance comparisons of major FFTs on current architectures are available in [11]. The results [11] motivated basing the FFT-ECP heFFTe design on FFTMPI [5], a CPU FFT library developed by Sandia National Laboratory (SNL). In this milestone we describe the latest optimizations and features added in porting the FFT-ECP heFFTe computations to graphics processing units (GPUs), as well as accelerating the MPI communications using GPUDirect technologies. Figure 3.1 illustrates the main target architecture for this period—the Summit supercomputer at Oak Ridge National Laboratory (ORNL). Shown are the node architecture and connectivity. The goal is to develop FFT optimizations and FFT versions that reduce, as well as optimize, communication on all connectivity and memory hierarchy levels:

1. Leverage the GPU's 900 GB/s bandwidth;
2. Reduce global inter-node communications, and
3. Employ GPUDirect technologies as well as MPI optimizations to efficiently communicate both intra-node (through the 50 GB/s NVLinks) and inter-node (through the 2×12.5 GB/s InfiniBand).

We note that Summit has fat nodes – capable of reaching a peak performance of 42 TeraFlop/s in double precision arithmetic – but only 25 GB/s unidirectional bandwith (up to 50 GB/s bidirectional) for internodal communications. This is the main bottleneck for FFTs as FFTs are memory bound – for flops for data of size N are only about $5Nlog_2N$.

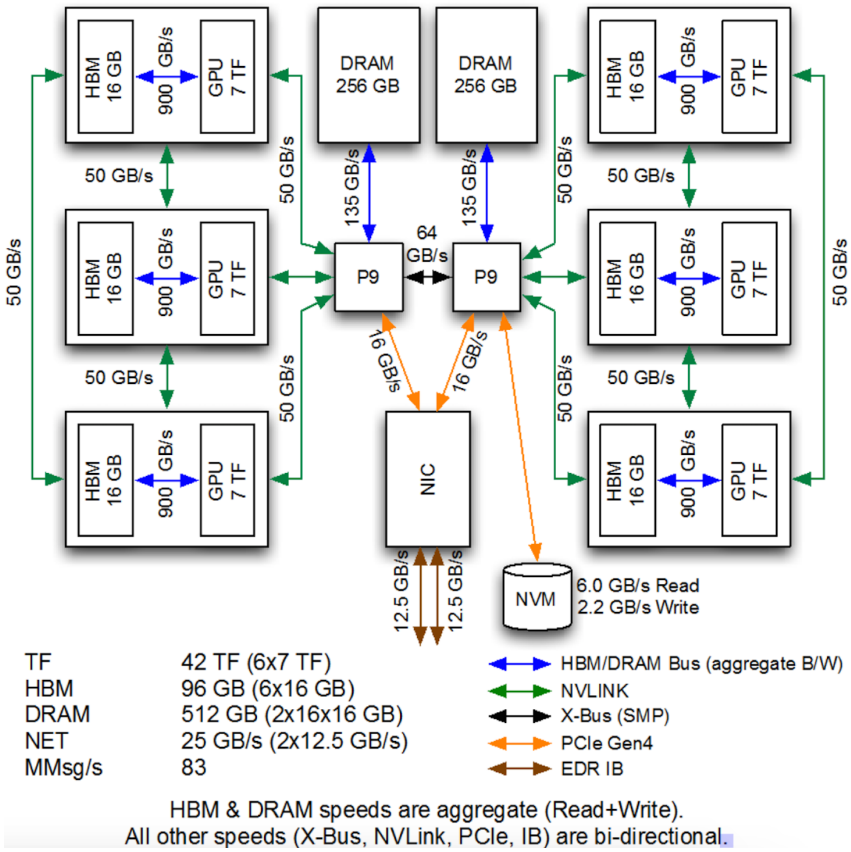

Figure 3.1: Summit node architecture and connectivity.

3.2 Overall design and optimizations

FFT-ECP is based on FFTMPI and follows the same algorithmic patterns (see Figure 3.2). Data transpositions create contiguous vectors (*pencils*) in the x, y, and z directions, and call an external FFT library for the 1-D FFT calculations on the corresponding pencils. One of the most important features of FFTMPI is the flexibility of input and output data layout. That flexibility is kept in FFT-ECP. The algorithm works as follows: given a 3-D tensor $A = \{a_{i,j,k}\}$ distributed on P processors, the first step is to make the data belonging to the first dimension available on the same processor (i.e., $\forall j_0, k_0, \exists$ process P_l s.t. $a_{i,j_0,k_0} \in P_l$ for $\forall i$) so that the computation of the first dimension can start on P_l. After it is finished, data is transposed again and 1-D FFTs are computed along the second dimension, and the same goes for the last one. Then, if necessary, a final step of communication is performed to build the output layout.

To get as close as possible to the theoretical maximum performance, we design the FFT-ECP heFFTe library to be computed entirely on the GPUs and the GPUs to communicate directly to GPUs, i.e., removing any CPU-to-GPU data movements. Thus, we rely on CUDA-aware MPI implementations that avoid staging communications through the CPU host memory. Instead, CUDA-aware MPIs use GPUDirect technologies (for direct P2P communications between GPUs on a node and RDMA for direct communication between GPUs on different nodes, avoiding host memory) to provide high-bandwidth, low-latency communications with NVIDIA GPUs.

The FFT-ECP design also relies on 1-D FFTs from vendors or open source FFT libraries. On NVIDIA GPUs we use the cuFFT library [4].

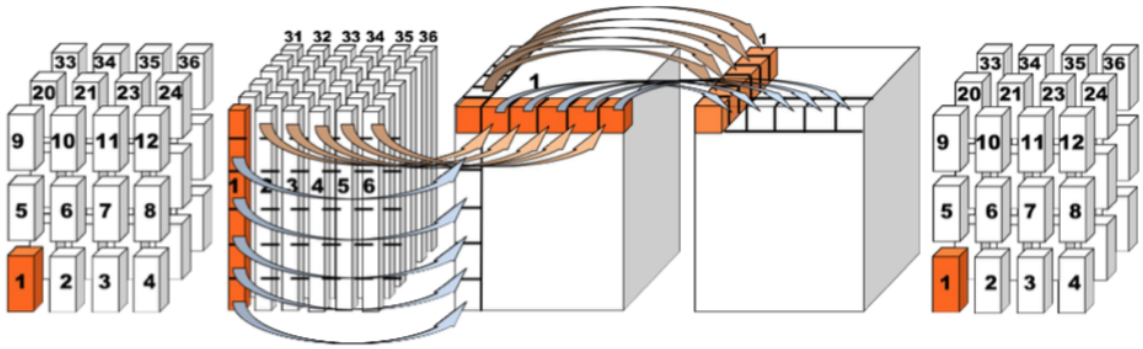

Figure 3.2: FFT-ECP framework design with flexible API

3.3 FFT-ECP performance and limitations

A typical profile of running 3-D FFTs on multi-core CPUs (e.g., Summit) is given on Figure 3.3, Left. The times reported (in seconds) are for double complex arithmetic performing Forward FFT, starting from brick distribution and ending with the same brick distribution over the processes. The local FFT operations (within an MPI process) are memory bound and take about 50% of the time, in this case. Included are local packing, unpacking, and 1-D FFTs (using any CPU FFT library that gives best performance, e.g., Intel MKL FFT, FFTW, or any other). The other 50% are in MPI communications.

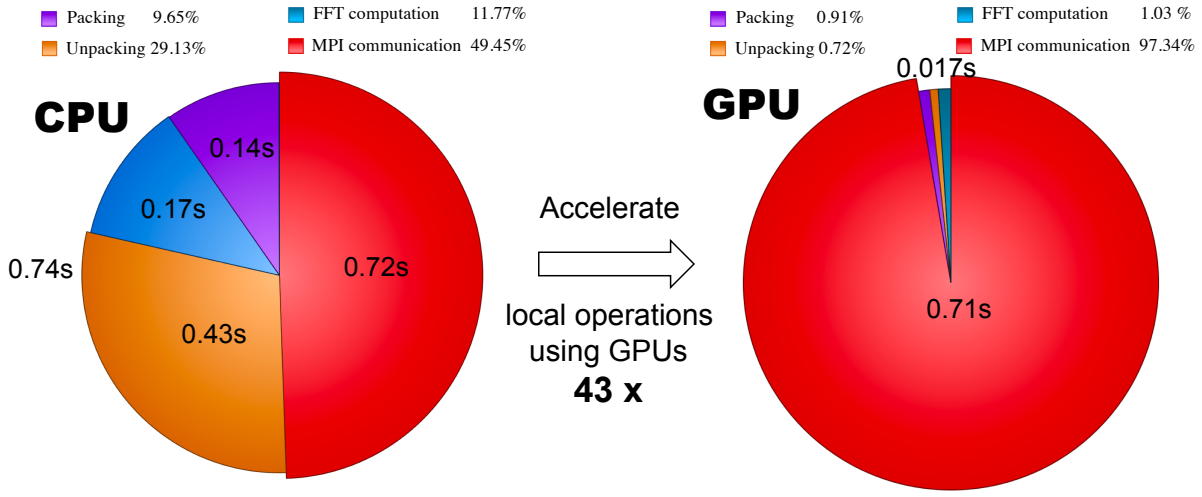

Figure 3.3: Profile of a 3-D FFT of size 1024^3 on 4 CPU nodes—using 128 MPI processes, i.e., 32 MPIs per node, 16 MPIs per socket (Left) vs. 4 GPU nodes—using 24 MPI processes, i.e., 6 MPIs per node, 3 MPI per socket, 1 GPUs per MPI (Right)

Note that the local computations are memory bound and can therefore benefit from the GPUs' high bandwidth (900 GB/s). To leverage this, we ported all operations to the GPU: packing and unpacking using the fast matrix transpositions available in MAGMA [10], and the 1-D FFTs using cuFFT [4] from NVIDIA. This accelerated the local operations $43\times$, in this case. The GPU profile of running the same problem on four nodes with six V100 GPUs each is shown on Figure 3.3, Right. The local 1-D FFTs, packing data, and unpacking now takes only about 2.5% of the total execution time. MPI communications remained about the same, using CUDA-Aware MPI from IBM (Spectrum MPI) doing GPUDirect communication. The

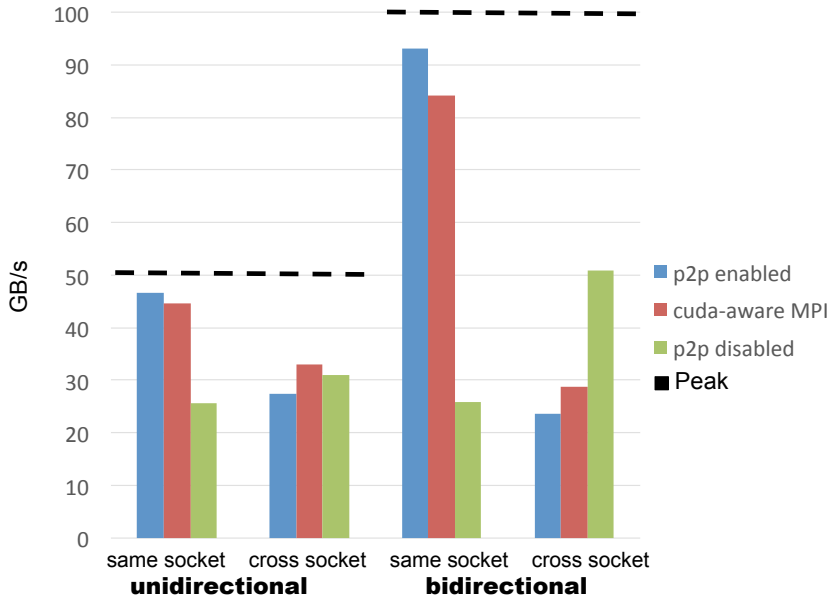

Figure 3.4: Bandwidth Benchmark for different types of p2p with message size = 40MB

results show that the maximum acceleration in cases like this is achieved, and is about $2\times$.

Multiple other versions were also developed to support different features, including user interfaces, e.g., data starting from the CPU memory, GPU memory, different data distributions, etc. In general, any version that involves movement of data to the CPU memory was significantly slower due to the costly data movements through the 50 GB/s NVLinks vs. the 900 GB/s GPU bandwidth.

3.4 Communication reduction in FFTs

The current bottleneck in FFTs are MPI communications. As illustrated on Figure 3.3, Right, MPI communications take more than 97% of the total time.

To optimize communications, we developed benchmarks and tested different technologies for transferring data. Figure 3.4 summarizes the comparison for GPUDirect Peer to Peer (P2P) communications. The comparisons are when a GPU communicates with GPUs on the same socket, cross socket GPUs, unidirectional and bidirectional, and using Nvidia GPUDirect technologies in single process versus CUDA-aware MPI communications from different MPI processes. This helped identify drawbacks, summarized in the conclusions, as some of these communications are far away from the theoretical peaks. Similar benchmarks and studies were performed on All2All communications. The best performing versions were selected to achieve the results in Figure 3.3, Right, and in general for the tuning of the FFT-ECP library.

We achieved the best performance with a combination of P2P Spectrum MPI communications for FFTs on up to four Summit nodes, and Spectrum MPI All2All for more than four Summit nodes. This is illustrated on Figure 5.2. Note that the code also has very good, strong scalability. The computations are in double complex arithmetic and the gigaFLOP/s rate reported assumes $5N^3log_2N^3$ floating-point operations (FLOPs), where N is the size for the N^3 3-D FFT performed. This computation also starts from bricks and ends up with the same brick distribution over the MPI processes (on the GPUs' memories).

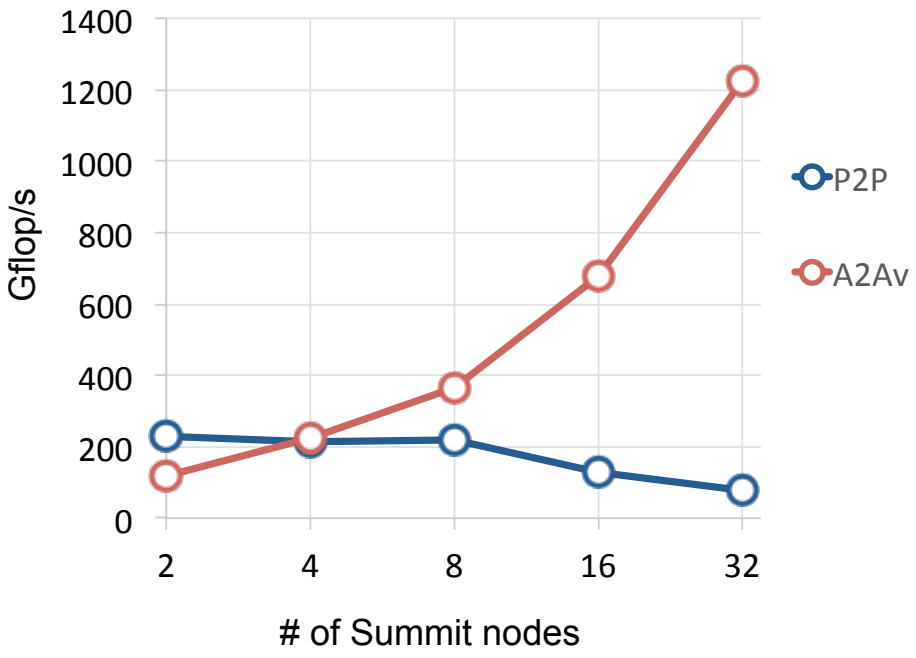

Figure 3.5: Strong scalability on 3D FFTs of size 1024^3, comparing the use of All2All vs. P2P MPI communications using CUDA 9.1 and Spectrum MPI 10.2.

To further reduce communications, we explored algorithmic 3-D FFT variations that can reduce communications. In particular, we looked into ways to reduce the stages of communication. For example, this can be done with so called *slab* decompositions, where a process will get the data needed for two directions of 1-D FFTs, thus avoiding the step of communications between the two. This can save between 25% and 50% of the total time.

This can be further extended, for example, by testing whether all the data fits in the memory of a single GPU, and if that is the case, we can send all the data to a single GPU, do the 3-D transform without any communication and send it back to build the output data layout requested by the the user. Other extensions are to do this agglomeration to a single node, or a subset of the compute resources.

Figure 3.6 illustrates the effect of using slab decompositions to reduce communications, and hence to increase performance. The starting splitting is based on bricks, so the pencil decomposition results in four communication stages: transpose in x, transpose in y, transpose in z, and move back to the original brick decomposition. The slab decomposition on the other hand results in only three stages: move to x-y slabs, transpose in z, and move back to the original brick decomposition. This is a theoretical 25% reduction in communications that results in a corresponding 25% increase in performance. For 8 and 16 nodes, speedup is 35% and 32%, respectively, and goes down as the number of nodes used grows. The effect becomes negative at 128 nodes due to lack of parallelism (at 128 the number of MPIs/GPUs used is 768).

Note the results in Figures 5.2 and 3.6 differ by up to 20% performance degradation when CUDA and MPI were updated on Summit. With the new MPI we also do not see the positive effect of selecting proper GPU network affinities as we used to when using Spectrum MPI 10.2 [12]. GPU affinity was an important tuning parameter.

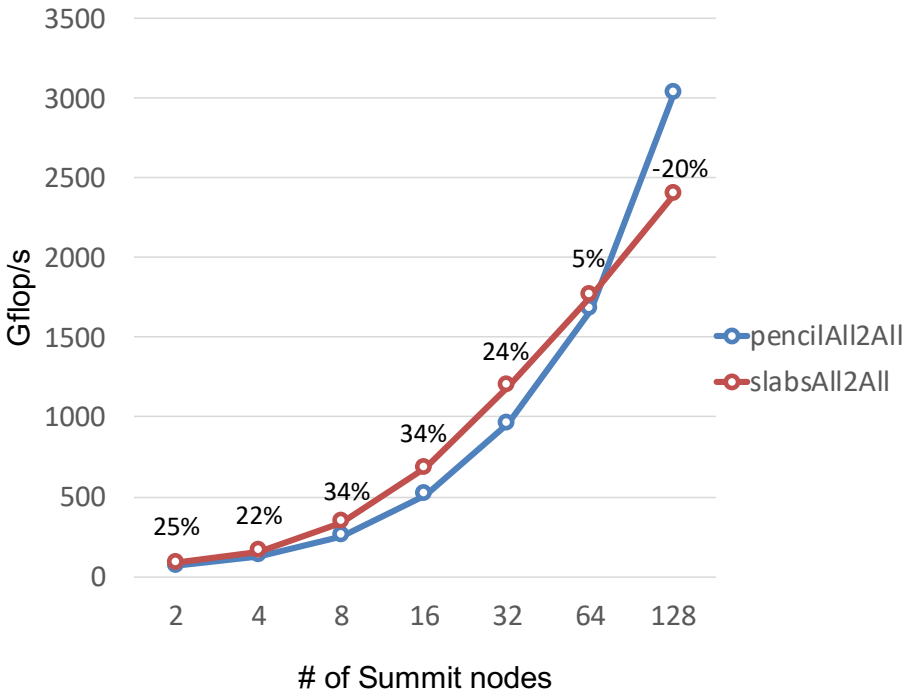

Figure 3.6: Strong scalability on 3D FFTs of size 1024^3, comparing the use of slab vs. pencil decompositions using CUDA 10.1 and Spectrum MPI 10.3

CHAPTER 4

Accelerating ECP applications with FFT-ECP

In this milestone we focused on molecular dynamics applications. We take LAMMPS [3] as a representative applications library, since it is well documented and is part of ECP, to illustrate the possible gains by using the FFT-ECP development.

The KSPACE package from LAMMPS is the one that makes the call for FFT computations. It provides a variety of long-range Coulombic solvers, as well as pair styles which compute the corresponding short-range pairwise Coulombic interactions. Its **kspace_style pppm** command performs 3d FFTs to compute the energy of a molecular system.

Figure 4.1 shows the results of a classic benchmark where LAMMPS calls its built-in FFTMPI library (CPU code), Left, and when it uses our heFFTe library (CPU-GPU code), Right. The results show that FFT is accelerated two times, while the speedup for the entire application can be $1.5\times$.

LAMMPS is used in the EXAALT ECP application project. This is an example of accelerating an ECP application relatively easy thanks to the compatibility of the FFT API used (by design).

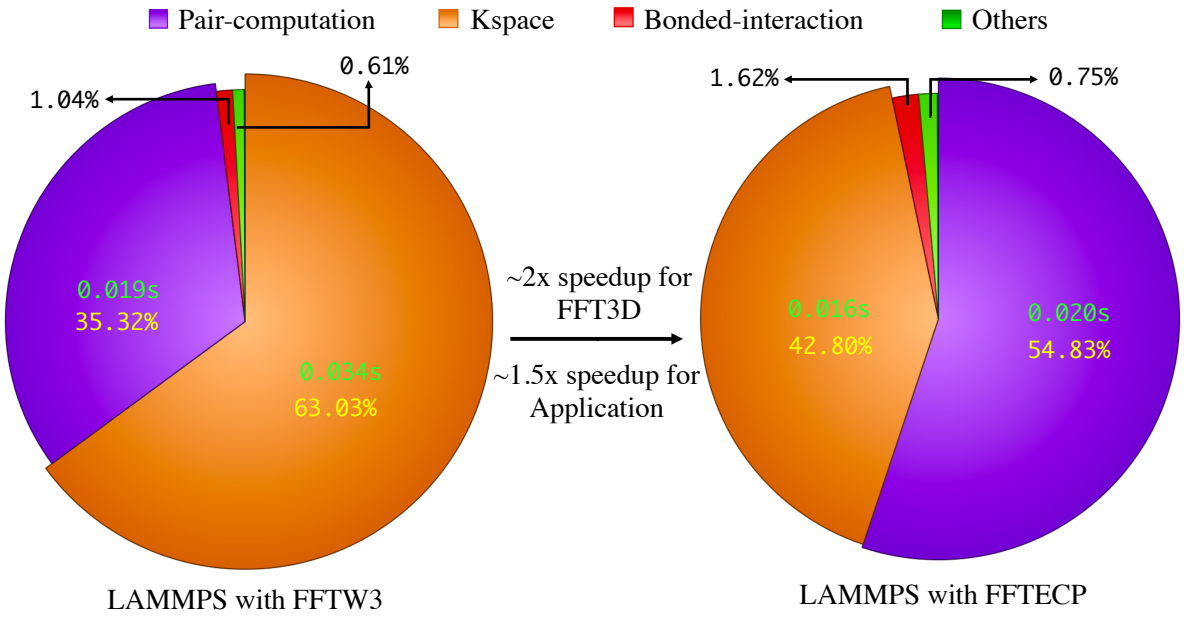

Figure 4.1: Rhodopsin protein benchmark for the LAMMPS package on 2 nodes and 4 MPI ranks per node, on a 128x128x128 FFT grid. On the left we show the time chart by using LAMMPS with FFTMPI, and on the right we observe the gains in performance when LAMMPS uses FFTECP.

CHAPTER 5

The heFFTe version 0.1 release

We released the ECP-FFT FFT library, heFFTe version 0.1. Figure 5.1 shows the heFFTe software stack. As illustrated, heFFTe relies on MPI, OpenMP for multithreading, CUDA and HIP for GPU acceleration,

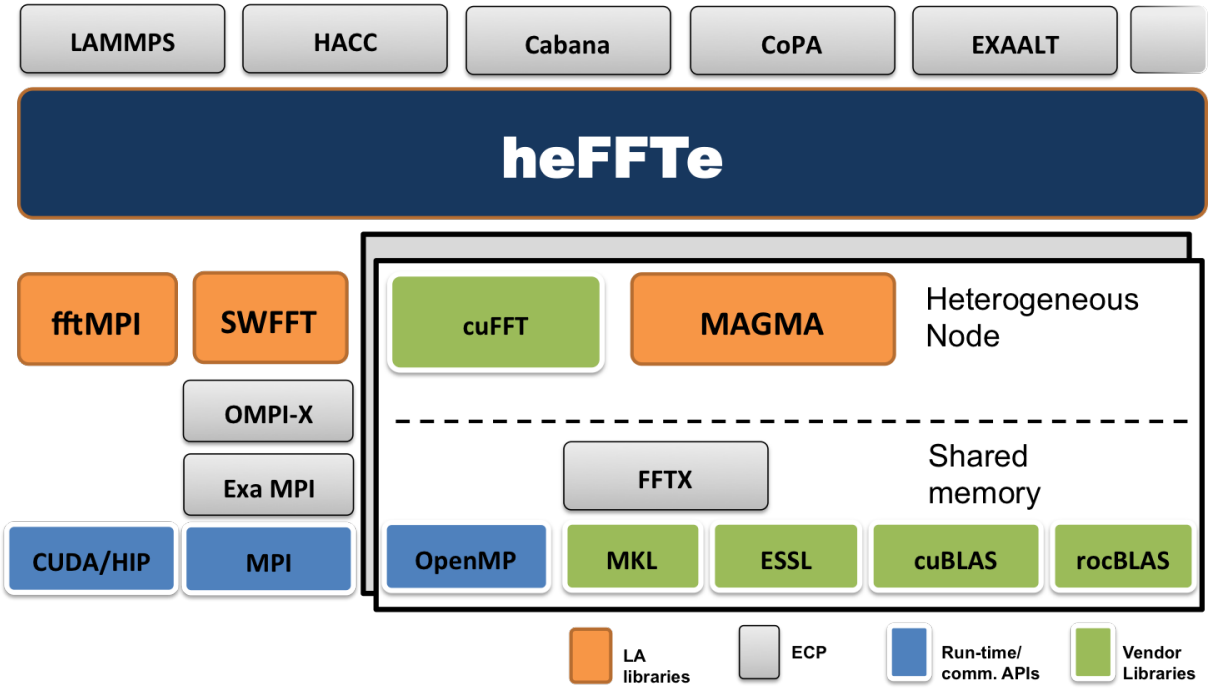

Figure 5.1: heFFTe software stack.

and existing FFT capabilities, including fftMPI, SWFFT, 1-D FFTs from vendors or open sorce efforts, etc. For NVIDIA GPUs, heFFTe uses the CUDA runtime, the cuFFT library, MAGMA and hand-coded CUDA kernels. For AMD, the plan is to use the HIP equivalents, and for Intel we count on the availability of similar solutions. The MPI implementations are expected to be accelerator-aware.

heFFTe supports different precisions and uses MAGMA to accelerate various linear algebra and auxiliary kernels. This includes matrix transpositions, data reshufles for packing/unpacking of data for MPI communications, and casting routines for the development of mixed-precision algorithms. In addition to various precisions, heFFTe supports the following features:

- All options from fftMPI and SWFFT;
- Input data can be bricks or pencils;
- Output data can be bricks or pencils;
- All operations are done on GPUs;
- Communications use CUDA-Aware MPI with GPU-Direct communication technologies.

The current bottlenecks were described in the previous sections. New developments to overcome these bottlenecks include:

- Algorithmic and optimizations work for reduced communication;
- Application-specific FFT optimizations;
- Mixed-precision algorithms;
- Data compression for reduced communications (including lossy).

heFFTe has very good strong scalability, which is essential in order to scale the solution time down of relatively small problems by increasing the amount of computational resources used. We ran scalability tests on up to 512 nodes (3,072 V100 GPUs) on Summit for 3-D FFT problems of size 1024^3. Results are summarized in Figure 5.2. Note that even with that amount of GPUs used heFFTe scales very well, while the corresponding CPU solutions reaches the limits of its strong scalability and even shows some slowdown. The GFlop/s reported are for double complex precision calculations, starting from bricks/cubes data distribution and ending with the original bricks/cubes data distribution. Flops are counted as $15 * 1024^3 * log_2 1024$.

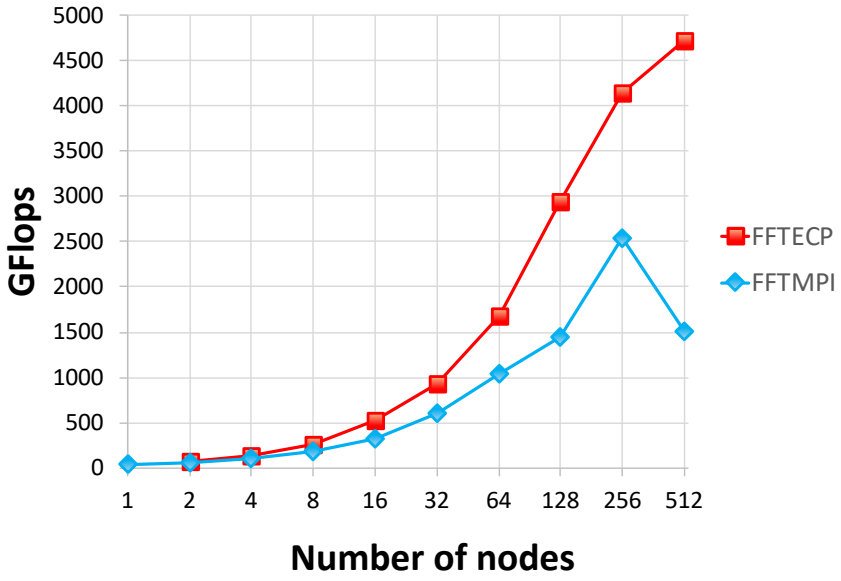

Figure 5.2: Strong scalability and performance comparison of 3-D FFTs of size 1024^3 on up to 512 nodes of Summit supercomputer: FFTMPI using 40 cores per node and heFFTe using 6 V100 GPUs per node.

CHAPTER 6

Conclusions and future work directions

In this milestone, we imlementated optimizations and multiple features of 3-D FFTs in the FFT-ECP project. The target architectures are large-scale distributed GPU-accelerated platforms. We released heFFTe version 0.1 that incorporates the new developments.

The heFFTe library has achieved significant acceleration of 3-D FFTs using GPUs. Results show very good scalability, including strong scalability, due to highly optimized and reduced MPI communications. Locally, on a GPU, we have accelerated the FFT operations about $40\times$ compared to the multi-core CPU counterpart (using V100 GPUs vs. Power9 CPUs on the Summit supercomputer at ORNL). This acceleration uses the high bandwidth that GPUs provide (up to 900 GB/s). At this stage, the FFT computation is dominated (97% and above) by MPI communications, so any further improvement in the overall speed will come from the development of CUDA-aware MPI optimizations.

Figure 3.4 shows that there are areas that need improvement, especially in the cross-socket GPU Direct communications for both uni- and bi-directional communications. Furthermore, there is performance degradation between MPI Spectrum versions 10.2 and 10.3. Performance loss of 20%, as illustrated in Figures 5.2 and 3.6, is significant and needs to be addressed.

Current performance can be best described by the bandwidth achieved through a node. For example, performance on a 1024^3 3-D FFT is about 400 gigaFLOP/s on 8 nodes. One can compute that data is sent from the node at a 21.8 GB/s rate: the formula is Bytes sent ($= 16 \times 1024^3/8$) over the time to send ($= 0.98 \times (5 \times 1.024^3 \times log_2(1024^3)/400)/4$, where 4 represents the four stages used). Note that the node also receives about the same amount of data at the same time, so the bi-directional bandwidth achieved is 43.6 GB/s out of 50 GB/s, which is 87.2% of the roofline peak.

Future work will concentrate on MPI optimizations for strong scaling on many nodes, optimizations for a single node for cross-socket communications, and algorithmic optimizations based on slab partitions, or other reductions of the computational resources used that can lead to reduced communications. More versions and support for different FFT features are being added in FFT-ECP. Application-specific optimizations and use of mixed-precision calculations [9] that can result in additional acceleration due to reduced communications are also of interest.

Acknowledgments

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, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

Bibliography

[1] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Venkatram Vishwanath, Tom Peterka, Joe Insley, David Daniel, Patricia Fasel, and Zarija Lukić. Hacc: Extreme scaling and performance across diverse architectures. *Commun. ACM*, 60(1):97–104, December 2016. ISSN 0001-0782. doi: 10.1145/3015569. URL http://doi.acm.org/10.1145/3015569.

[2] M. Heroux, J. Carter, R. Thakur, J. Vetter, L. McInnes, J. Ahrens, and J. Neely. Ecp software technology capability assessment report. Technical Report ECP-RPT-ST-0001-2018, DOE Exascale Computing Project (ECP), July, 2018. URL https://www.exascaleproject.org/ecp-software-technology-st-capability-assessment-report-car/.

[3] Large-scale Atomic/Molecular Massively Parallel Simulator. Large-scale atomic/molecular massively parallel simulator, 2018. Available at https://lammps.sandia.gov/.

[4] CUDA Nvidia. Cufft library, 2018.

[5] Steven Plimpton, Axel Kohlmeyer, Paul Coffman, and Phil Blood. fftmpi, a library for performing 2d and 3d ffts in parallel. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

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

[7] DF Richards, O Aziz, Jeanine Cook, Hal Finkel, et al. Quantitative performance assessment of proxy apps and parents. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2018.

[8] Hejer Shaiek, Stanimire Tomov, Alan Ayala, Azzam Haidar, and Jack Dongarra. GPUDirect MPI Communications and Optimizations to Accelerate FFTs on Exascale Systems. Extended Abstract icl-ut-19-06, 2019-09 2019. EuroMPI'19 Posters, Zurich, Switzerland.

[9] Anumeena Sorna, Xiaohe Cheng, Eduardo F. D'Azevedo, Kwai Wong, and Stanimire Tomov. Optimizing the fast fourier transform using mixed precision on tensor core hardware. *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, pages 3–7, 2018.

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

[11] Stanimire Tomov, Azzam Haidar, Daniel Schultz, and Jack 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, October 2018. revision 10-2018.

[12] Stanimire Tomov, Azzam Haidar, Alan Ayala, Daniel Schultz, and Jack 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, April 2019. revision 04-2019.