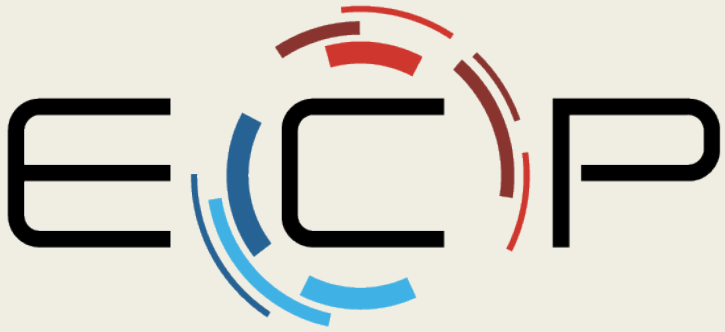

EXASCALE COMPUTING PROJECT

ECP Milestone Report

Design and Implementation for FFT-ECP on Distributed Accelerated Systems

WBS 2.3.3.09, Milestone FFT-ECP ST-MS-10-1410

Stanimire Tomov
Azzam Haidar
Alan Ayala
Daniel Schultz
Jack Dongarra

Innovative Computing Laboratory, University of Tennessee

April 6, 2019

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

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
10-2018	first publication

```
@techreport{thasd2019ECPFFT,
  author={Tomov, Stanimire and Haidar, Azzam and Ayala, Alan and Schultz, Daniel and Dongarra, Jack},
  title={{Design and Implementation for FFT-ECP on \\Distributed Accelerated Systems}},
  institution={Innovative Computing Laboratory, University of Tennessee},
  year={2019},
  month={April},
  type={ECP WBS 2.3.3.09 Milestone Report},
  number={FFT-ECP ST-MS-10-1410},
  note={revision 04-2019}
}
```

Contents

List of Figures

List of Tables

CHAPTER 1

Executive Summary

The goal of this milestone was the design and an initial implementation 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 design and performance of an initial code for 3-D FFTs that we developed for heterogeneous systems with GPUs. We also describe the roadmap for future work based on our observations and analysis. Specifically, this milestone delivered on the following sub-tasks:

- Investigation and development of different communication schemas (e.g., collective `all2all`, group collective). For example, a communication scheme with adaptive local/global `all2all` communication and asynchronous dynamic scheduling with priority communication that exhibits improvement on HPC systems with many GPUs (e.g., nodes on Summit and Sierra);
- A study on the possible use of directed acyclic graphs (DAGs) combined with a task-based runtime system to schedule the MPI communication and to alleviate the cost of the synchronization mechanisms required in the FFT code;
- A study (e.g., model) of the correlation between the data distribution (natural, cyclic, slices, cuboid, and user provided) and the communication cost outlining how each can affect the other in an exascale environment;
- Develop and optimize building block functions needed by the FFT code. This includes the development of a modular framework that could be easily integrated into ECP applications. The kernels to be developed must leverage when possible the Matrix Algebra on GPU and Multicore Architectures (MAGMA) library, MAGMA optimized building blocks, and expertize to extract maximum performance from GPUs [1, 3, 9]. All of these kernels are to be designed with distributed, multi-GPU machines in mind;
- Develop the initial version of the 3-D FFT for heterogeneous, multi-GPU nodes and ECP applications.

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 [7], particularly in Molecular Dynamics (MD) computations, e.g., as in the LAMMPS ECP apps project [5], and Nbody simulations, e.g., as in the HACC ECP apps project [2]. 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 [4], 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` [8] and `FFTMPI` [6] 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 to design and provide initial implementation prototypes covering the `FFTMPI` and `SWFFT` functionalities for 3-D FFTs on GPU-accelerated Exascale platforms.

CHAPTER 3

FFT communication schemas for large-scale GPU-accelerated systems

The performance of large-scale 3-D FFTs is mostly memory bound. Indeed, the roofline performance model for 3-D FFT of size N^3, running on P nodes with bandwidth per node of 50 GB/s, bounds nodal performance in double complex arithmetic to

$$\frac{P}{P-1}50\frac{5}{32}log_2N$$

GFlop/s [10]. Thus, asymptotically, for large P, we can consider that the nodal performance is bound by about

$$7.8\ log_2N \quad \text{GFlop/s}.$$

For example, if $N = 1,024$, we get a limit of 78 GFlop/s per node.

Note that the above bound assumes the theoretical maximum bandwidth possible per node. Figure 3.1 shows the Summit node architecture and connectivity. The maximum bandwidth can be achieved when the node sends and receives data at full speed, which is two times 12.5 GB/s in one direction, i.e., 50 GB/s in bidirectional full bandwidth.

To get as close as possible to the theoretical maximum performance, we design the FFT-ECP 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.

Because of the importance of the FFT communications, we provide a review of the CUDA-aware MPI (Section 3.1) and a study on the current performance of CUDA-aware MPI communications on Summit (in Section 3.2). Section 3.3 describes the communication schemas in FFT-ECP, as motivated by our analysis of the FFT communications and CUDA-aware MPI performance.

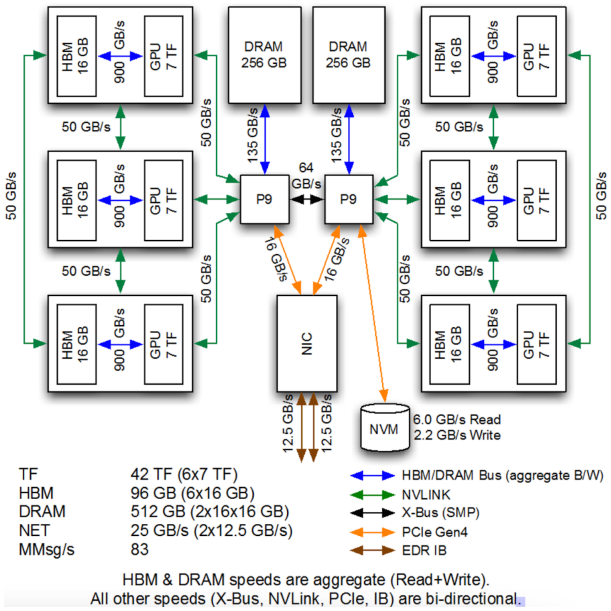

Figure 3.1: Summit node architecture and connectivity.

3.1 CUDA-Aware MPI

CUDA-aware MPI implementations are available through open-source MPIs like OpenMPI and MVAPICH2, as well as commercial versions such MPIs from CRAY, SGI, and IBM. Our current interest is the Summit supercomputer. The MPI on Summit is provided by IBM Spectrum MPI. The Spectrum MPI is based on OpenMPI but also adds some features and improvements, such as advanced affinity features, better performance, etc. Figure 3.2 illustrates the GPUDirect technologies for fast intra- and inter-node GPU communications.

To enable the GPU-aware MPI features, we run with:

```
jsrun <jsrun-options> --smpiargs="-gpu" ./gpu_setter.sh <app>
```

where `gpu_setter.sh` is given by the following script:

```
#! /bin/bash
# gpu_setter.sh
# Rudimentary GPU affinity setter for Summit
# >$ jsrun rs_per_host 1 gpu_per_rs 6 <task/cpu option> ./gpu_setter.sh <your app>

# This script assumes your code does not attempt to set its own
# GPU affinity (e.g. with cudaSetDevice). Using this affinity script
# with a code that does its own internal GPU selection probably won't work!

# Compute device number from OpenMPI local rank environment variable
# Keeping in mind Summit has 6 GPUs per node

mydevice=$((${OMPI_COMM_WORLD_LOCAL_RANK} % 6))
```

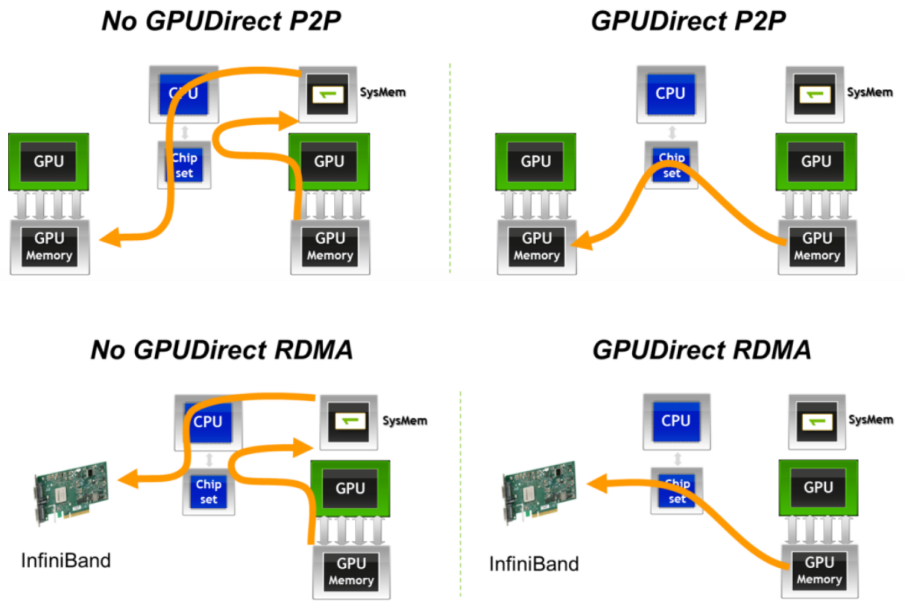

Figure 3.2: GPUDirect technologies in CUDA-aware MPI for fast intra-node P2P GPU communications (Top) and for inter-node Remote Direct Memory Access (RDMA) GPU communications (Bottom).

```
# CUDA_VISIBLE_DEVICES controls both what GPUs are visible to your process
# and the order they appear in. By putting ``mydevice'' first the in list, we
# make sure it shows up as device ``0'' to the process so it's automatically selected.
# The order of the other devices doesn't matter, only that all devices (0-5) are present.

CUDA_VISIBLE_DEVICES=''${mydevice},0,1,2,3,4,5''

# Process with sed to remove the duplicate and reform the list, keeping the order we set

CUDA_VISIBLE_DEVICES=$(sed -r ':a; s/\b([[:alnum:]]+)\b(.*)\b\1\b/\1\2/g; ta;
s/(,,)+/,/g; s/, *$//' <<< $CUDA_VISIBLE_DEVICES)

export CUDA_VISIBLE_DEVICES

# Launch the application we were given
exec ``$@''
```

As mentioned in the script, using the script assumes the code does not attempt to set its own GPU affinity, e.g., with `cudaSetDevice`.

The IBM Spectrum MPI uses the Parallel Active Messaging Interface (PAMI) to provide reliable massaging for point-to-point (P2P) and collectives communications in its MPI implementation. When there are multiple InfiniBand adapters per node, as in the case of the Summit notes (see Figure 3.1), adapter affinity is enabled by default, which in general leads to better performance when CPU/GPU binding is enabled. This hints each MPI rank to use the InfiniBand adapter that is physically closest to the core/GPU where the MPI rank is bound. However, the default PAMI settings for Summmit's multi-host support on POWER9s, might not be optimal for bandwidth sensitive applications, as we have observed for FFT computations. Therefore, in addition to the default, we also use the following settings when looking for

the network affinity that yields optimal node aggregate bandwidth for FFT:

```
# Network_v1
export PAMI_IBV_DEVICE_NAME="mlx5_0:1,mlx5_1:1"
export PAMI_IBV_DEVICE_NAME_1="mlx5_2:1,mlx5_3:1"
export PAMI_ENABLE_STRIPING=1
```

and

```
# Network_v2
export PAMI_IBV_DEVICE_NAME="mlx5_0:1,mlx5_3:1"
export PAMI_IBV_DEVICE_NAME_1="mlx5_3:1,mlx5_0:1"
export PAMI_ENABLE_STRIPING=1
```

Subsequently, in this report, we refer to the first network as `Network_v1` and to second as `Network_v2`. Note that in Summit there are two InfiniBand physical ports and four virtual that we specify above – `mlx5_0` and `mlx5_1` for socket 0, and `mlx5_2` and `mlx5_3` for socket 1.

3.2 Benchmarking CUDA-Aware MPI communications

To guide the tuning and assess the current state and possible bottlenecks of the CUDA-aware MPI implementation on Summit, we developed a number of benchmarks to evaluate the performance of P2P and All2All communications. Figure 3.3 shows bandwidth for P2P GPU communications within a node (Left) and P2P GPU communications on GPUs across Summit nodes (Right).

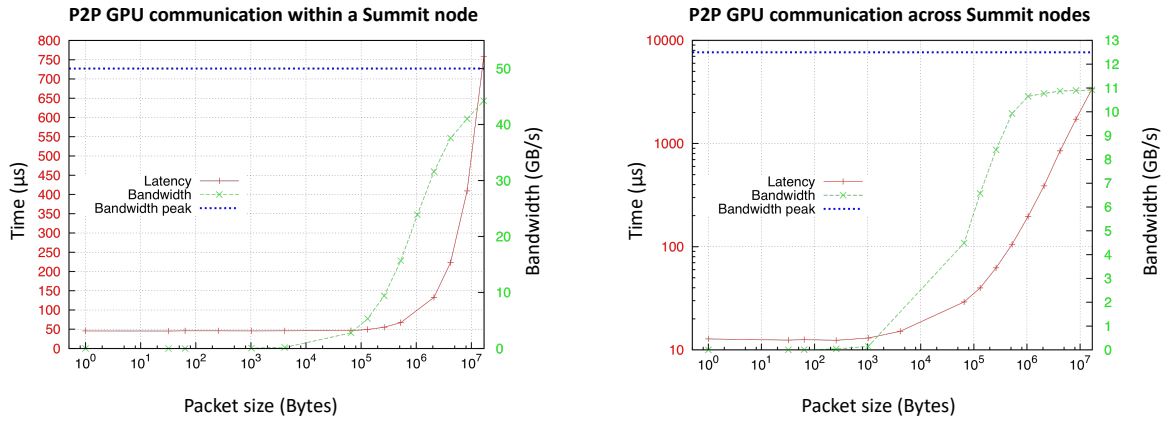

Figure 3.3: P2P communications between GPUs on a node (Left) and P2P communications between GPUs accross Summit nodes (Right).

Figure 3.4 shows bandwidth for All2All GPU communications within a node (Left) and All2All GPU communications on GPUs across Summit nodes (Right). Both benchmarks show runs using default network affinity settings.

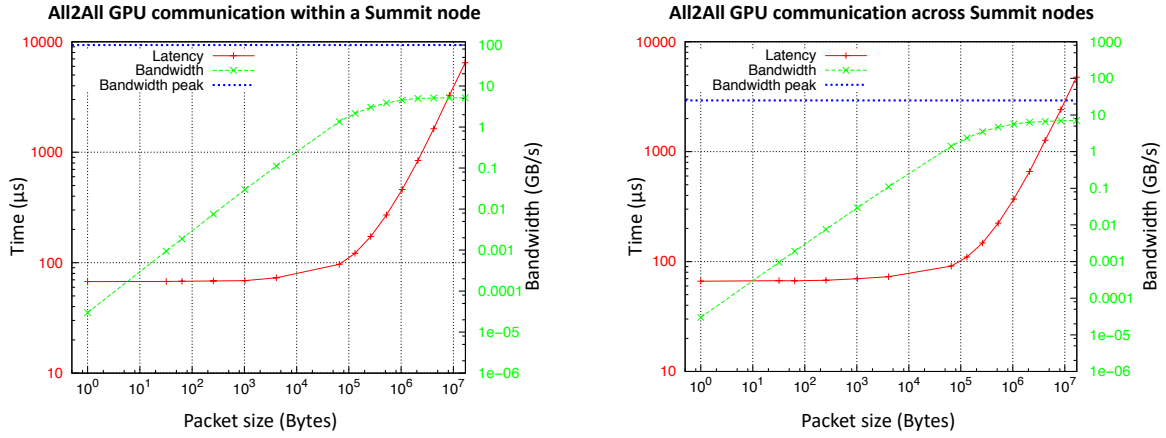

Figure 3.4: All2All communications between GPUs on a node (Left) and All2All communications between GPUs accross Summit nodes (Right).

3.3 Communication schemas in FFT-ECP

There are three main strategies for parallelizing 3-D FFTs, which result in three main communication schemas. Namely, assuming P ranks/processes, the schemas and their corresponding communications are:

- "`Slab`"-decomposed FFTs, where each rank sends/receives data with P ranks;
- "`Pencil`"-decomposed FFTs, where each rank sends/receives data with $\sqrt{P}$ ranks;
- "`Brick`"-decomposed FFTs, where each rank sends/receives data with $\sqrt[3]{P}$ ranks.

Note that the `Slab` communication can be expressed as one large All2All, `Pencil` as $\sqrt{P}$ smaller All2Alls in sub-groups of $\sqrt{P}$ ranks, and `Brick` as $\sqrt[3]{P^2}$ All2Alls in sub-groups of $\sqrt[3]{P}$ ranks.

Furthermore, the All2All communications (in each of the sub-groups) can by themselves be implemented in three different ways. Namely, via P2P communication, All2All, or a combination. All these combinations give viable solutions that can give best performance results based on architecture, number of processors P, and FFT sizes. Therefore, they must be developed and we have been implementing all versions. Each version is further parameterized to prepare the FFT-ECP package for subsequent autotuning that will determine the best parameters based on empirical tuning.

The `Pencil`- and `Brick`-decomposed implementations in FFT-ECP stem from the FFTMPI and SWFFT packages, respectively.

Table 3.1 shows the theoretical maximum bandwidths for the communications in Summit. The numbers are derived from the nodal and network specifications for Summit, as illustrated in Figure 3.1, and represent the upper performance limits that we try to achieve in FFT-ECP. Since performance depends on a number of third party hardware and software technologies, most notably GPUDirect technologies, network interconnect, and the CUDA-aware MPI implementation, our approach is to also develop FFT-ECP versions and tune them based on the performance of the third party capabilities currently available.

The currently achievable bandwidths using the IBM Spectrum MPI are shown on Figures 3.3 and 3.4 for the P2P and All2All communications, respectively. Note that P2P gets relatively close to the peaks

Node count	Network	Point-to-Point	Bidirectional Point-to-Point	All-to-All
1	Default	44.6	84	3
	Network_v1	44.6	84	3
	Network_v2	44.6	84	3
	Theoretical Peak	50	100	100
2	Default	9.7	15.6	4.1
	Network_v1	10.7	17.1	4.2
	Network_v2	15.6	26.2	5.1
	Theoretical Peak	12.5 (x2 possible)	25 (x2 possible)	25 (x2 possible)

Table 3.1: Achieved vs. theoretical bandwidths in GB/s for P2P and All2All communications between two GPUs on the same and on different nodes of Summit.

in Table 3.1. Indeed, P2P achieves 45 GB/s between GPUs on a node vs. 50 GB/s theoretical peak, and 11 GB/s between GPUs across Summit nodes vs. 12.5 GB/s theoretical peak using one EDR InfiniBand switch. However, the performance of the All2All in the Spectrum MPI lags behind the theoretical peaks. All2All implementations must also benefit from duplexing, so their performance is compared to the duplex bandwidths, e.g., Figure 3.4 shows bandwidth performance of below 10 GB/s within a node, while we expect 100 GB/s. This highlights once more the need and importance of developing different versions to exploit and tune for third party capabilities. The results also show a particular need to further improve the CUDA-awareness in MPI All2All routines.

We implemented in FFT-ECP the communication schemas and performance optimization options described. Figure 3.5 illustrates the performance for the P2P, All2All, and combination communication options in the `Pencil`-decomposed 3-D FFT in FFT-ECP. The experiments are for strong scalability on up to 32 Summit nodes ($\times 6$ V100 GPUs per node) on a 1024^3 3-D FFT problem.

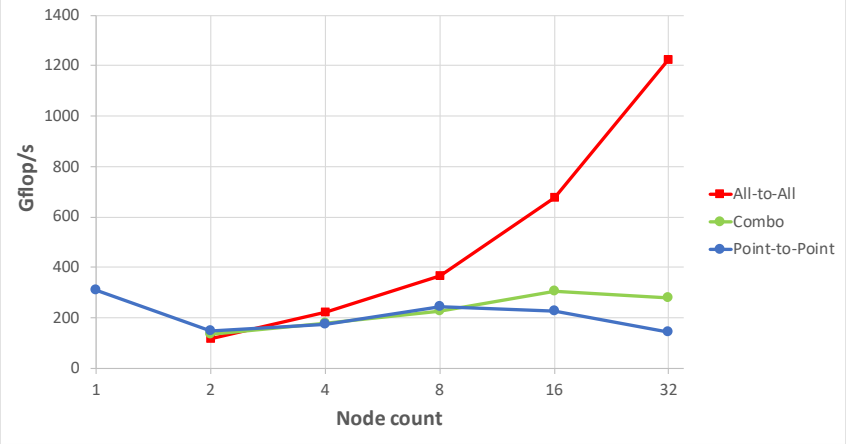

Figure 3.5: Strong scalability performance of FFT-ECP in computing a 3-D FFT on a 1024^3 grid using different communication options (P2P, All2All, and and a P2P-All2All combination) on up to 32 Summit nodes with 6 V100 GPUs per node.

CHAPTER 4

Scheduling computations and communications in FFT

FFT-ECP splits the FFT computation into tasks (either computation or communication tasks) that get scheduled for execution on the underlying hardware. The main tasks are illustrated on Figure 4.1. The Figure also shows the execution trace of a 3-D FFT on 80 MPI processes (in the y-dimension and time is on the x-axes). In general, the goal of the scheduling is to keep the processors busy all the time, e.g., trying to avoid idle time in waiting for data or synchronization.

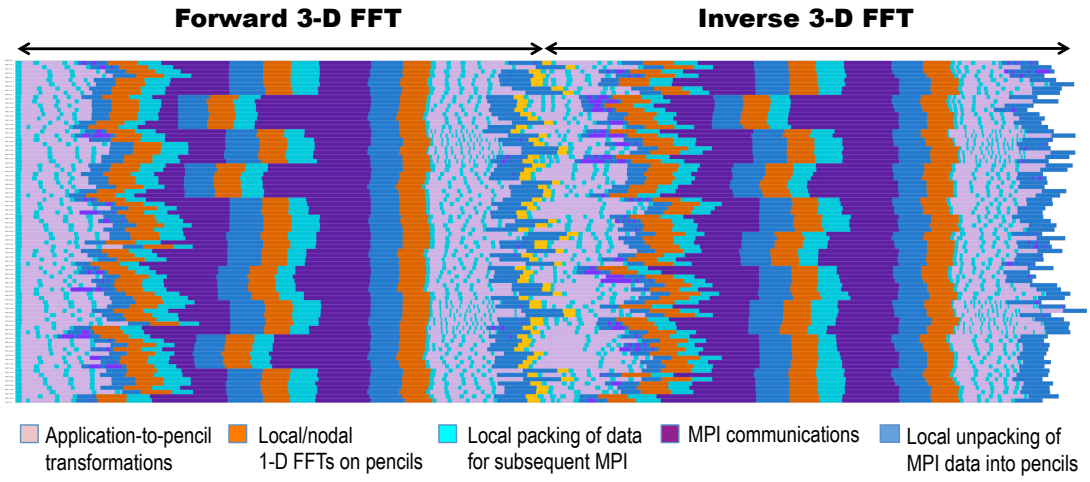

Figure 4.1: A 3-D FFT execution trace with main FFT components. The trace shown is for 80 MPI processes on Intel Xeon E5-2650 v3 cluster on a $1K \times 1K \times 1K$ grid.

As illustrated, there are no void spaces in the FFT-ECP execution and we can avoid synchronizations and idle times. Communications can be triggered asynchronously within a sub-group and entirely in parallel

between subgroups. Currently, the FFT-ECP scheduling is data driven. The GPU kernels' execution is scheduled by CUDA. We also use streams to pipeline kernels, as well as batched kernels and streams to schedule kernels to run efficiently in parallel.

Our analysis and experiments also determined that the granularity of the tasks is critically important for the scheduling to be successful, i.e., leading to high-performance. FFT-ECP controls the tasks' granularity at different levels – from high-level algorithmic decisions, e.g., slab vs. pencil vs. brick algorithms, to parameterizations, different communication mechanisms, blocking, and granularities controlled by third party tools/libraries.

An example for third party library controlling granularities is the MPI used. MPI splits large messages and schedules the delivery, but this can have both negative and positive effects, and therefore users must also be able to control it. Indeed, our analysis and experimentation have determined that default splitting of GPU-to-GPU communications can often lead to suboptimal performance. GPU communications have in general higher (than CPUs) latencies (e.g., see Figures 3.3 and 3.4), and therefore making packages too small can have negative effects. Therefore, to overcome these challenges, FFT-ECP is designed to consider the tasks' granularities as tuning parameters. When third party libraries do not allow enough flexibility to tune, we have also implemented our own MPI versions, e.g., `magma_Alltoall`.

Other MPI functions that we consider for optimizations in FFT-ECP are asynchronous sends and receives. One model is to post a number of asynchronous receives requests, followed by GPU work and synchronous or asynchronous sends, and leave it to MPI to schedule the delivery. Figure 4.2 shows an example where the MPI scheduling can overlap sends and receives, thus benefiting from duplexing the communications that we want to achieve, but there are also many cases where the duplexing is not happening, even when we know that it should be possible.

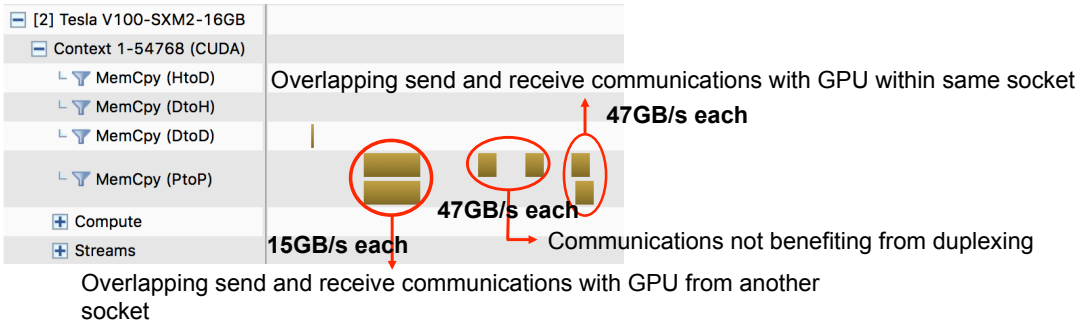

Figure 4.2: MPI scheduling of communications – non-blocking send and receive can be scheduled to overlap at top speed, thus benefiting from duplexing, but there are also cases where the scheduling fails to overlap them.

CHAPTER 5

Memory models and data distributions

FFT-ECP supports four memory models:

CPU allocated memory In this interface input data is on the CPUs' memory and FFT-ECP copies it to GPUs, computes, and brings back the result to CPU memory;

cudaHostRegister-ed memory In this mode CPU allocated memory is registered for GPU use and FFT-ECP does not have to explicitly copy data between CPU and GPU;

Managed memory This is Unified Memory that can be accessed from the CPU and the GPU. We developed it for convenience as both CPU and GPU code can work on it;

GPU allocated memory In this model memory is allocated on the GPU, result is written on the GPU and there are no CPU-GPU data transfers.

The last three memory models work with the same code. Once memory is established on the GPU the latter three versions have almost the same performance. The support for these three different versions is justified by the need to have options and to easily interface with different applications. For example, there are CPU-based computing applications that use CPU FFTs, e.g. FFTW, and the FFT-ECP interface would allow us to just change the calling functions to seamlessly provide GPU acceleration.

FFT-ECP provides memory allocation function that can allocate one of the specified above memories, do proper alignment, if specified, etc. A corresponding memory free routine is also provided.

The data distributions supported are inherited from the FFTMPI and SWFFT packages. Thus, FFT-MPI can be used directly in the ECP HACC and LAMMPS applications, which already use and are integrated with the FFTMPI and SWFFT FFT libraries, respectively.

CHAPTER 6

GPU kernels

We developed highly optimized GPU kernels in CUDA that perform the packing and unpacking routines needed for 3-D FFTs. The 1-D FFTs on the GPUs are computed by cuFFT.

Compared to packing and unpacking on CPUs, the GPU-accelerated code gives about $40\times$ speedup. Here we compare six V100 GPUs on a node vs. all the cores of the two Power9 CPUs on a Summit node. An illustration of this for an FFT on 1024^3 grid, computed on 4 Summit nodes, is given in Table 6.1. The computation is in double complex arithmetic.

Main 3D FFT kernels	Time (ms)		Overall time (ms)	
	GPU	CPU	GPU	CPU
Unpack	2.1	123	286 ms	368 ms
Batched 1D FFTs	1.8	63		
Pack	1.8	30		
MPI All2All	280	152		

Table 6.1: Execution times of 3-D FFTs on four nodes of Summit for N = 1024. Compared are execution times on 24 V100 GPUs vs. 160 Power9 cores using default network with `Pencil`-decomposed 3-D FFT.

The packing and unpacking routines are similar to matrix transpositions that we have also highly optimized and available in the MAGMA library. The unpacking kernel is in general more challenging to develop with coalescent both read and write. As illustrated in the Table, even the CPU is performing it much slower than the packing. On GPUs we developed both kernels to be about the same speed. Note that although the GPU computation, the packing, and the unpacking are all accelerated multiple times, the overall acceleration in this case is less than $2\times$. This is further discussed in the next Chapter.

CHAPTER 7

FFT-ECP prototype for 3-D FFTs on heterogeneous systems

7.1 Overall design and functionalities

As already discussed, this report described a FFT-ECP prototype that we have developed for distributed 3-D FFTs on GPU-accelerated Exascale platforms. The overall design follows the SWFFT and FFTMPI designs, on which FFT-ECP is based. The idea also is to leverage existing FFT capabilities, such as third-party 1-D FFTs from vendors or open-source libraries to quickly provide GPU support in a sustainable 3-D FFT library for Exascale platforms.

While we have ported both SWFFT and FFTMPI to GPUs in different codes, also providing at the same time multiple extensions, versions, and parametrizations, we are designing the final FFT-ECP software product to provide single library that will be flexible, easy to use, and moreover, provide all the functionalities that libraries like SWFFT and FFTMPI provide through a consistent user interface.

The current prototype supports `Pencil`- and `Brick`-decomposed FFTs. We have also multiple other versions for single GPUs, as well as multiple GPUs on a single node. All versions are highly parametrized, supporting different communication strategies based on P2P communications, All2All, or combinations.

Multiple experiments and studies on performance were performed in order to design and tune for performance the FFT-ECP implementations. Some of these were already discussed. Further results and current performance numbers are summarized next.

7.2 Performance results on Summit

As discussed in Chapter 6, our GPU kernels accelerate their CPU counterparts by more than $40\times$ on a Summit node. This is a significant acceleration that makes the FFT computations and local data reshufles

become only a tiny fraction of the overall execution time. Indeed, this is illustrated on Figure 7.2.

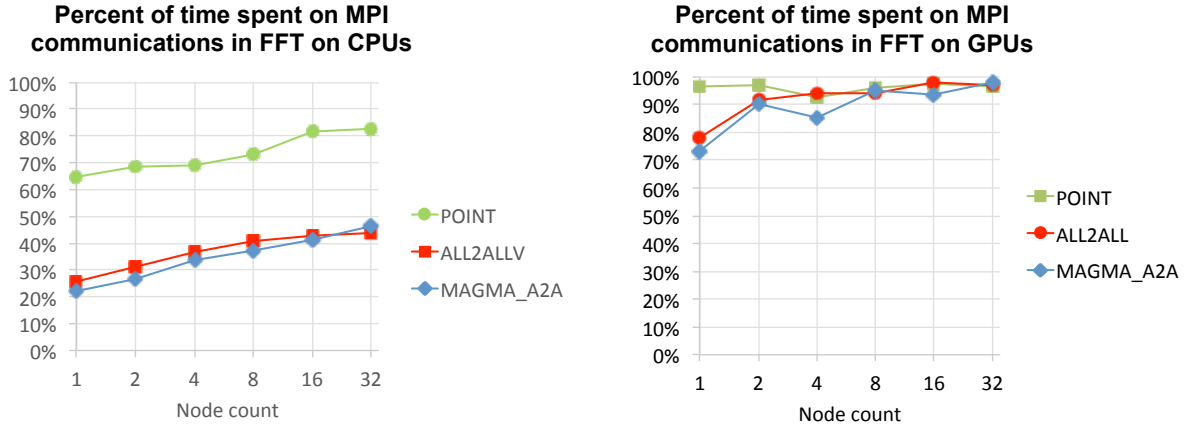

Figure 7.1: Percent of time spent on MPI communications in FFT on CPUs (Left) and GPUs (Right). The FFT computations are for weak scaling going from problem size 720^3 on one node to problem size of 2304^3 on 32 nodes.

Note that for 32 nodes the fast CPUs-based code spends about 50% in MPI communications and 50% in pack, unpack and 1-D FFTs. As the computation for the latter 50% are reduced $40\times$ on GPUs, the GPUs now spent only about 2% in pack, unpack and 1-D FFTs and the rest in MPI communications. Thus, theoretically, this code can be accelerated at most about $2\times$ when using GPUs.

The acceleration that we can achieve using GPUs is illustrated on Figure ??. The results are for strong scaling on 3-D FFTs of size 1024^3 on up to 32 nodes of Summit. Both the CPU and GPU codes are highly optimized and in this case use `Pencil`-decomposed 3-D FFTs based on the FFTMPI FFT library. The CPU code uses 40 cores per node (1 core per MPI process), while the GPU code uses 6 V100 GPUs per node (1 GPU per MPI process).

As discussed above, the GPU code can accelerate the computation at most $2\times$ in this case and indeed, performance is about $2\times$ better. For small number of nodes the GPUs can be significantly faster.

These results summarize a number of optimizations, as discussed in the report. Originally, the GPU version was slower. After a number of optimizations for both the CPU and GPU versions, the results show we reach a theoretically justified peak. Clearly, more acceleration will be possible if the 98% spent in MPI communication can be further accelerated.

The result show that we have very good strong scalability and achieve very good percentage of the theoretical peak performance. Using 32 nodes, the 1024^3 FFT is computed in 0.13 seconds in double complex arithmetic (see Figure 7.3). The communication bandwidth achieved per node is about 25 GB/s. This is the peak if duplexing is not used and have room for $2\times$ further improvements if one can get full benefit from the duplexing.

Figure 7.4 shows the strong scaling performance (in GFlop/s) on 3-D FFTs of size 1024^3 on up to 32 nodes of Summit using `Brick`-decomposed 3-D FFTs based on the SWFFT FFT library. Performance is similar to the `Pencil`-decomposed 3-D FFT in terms of absolute values achieved and scalability. This approach however requires more memory.

Weak scaling, as one can expect from the strong scalability results, are also very good. This is illustrated on Figure 7.5. The Left figure shows the Pencil-decomposed 3-D FFTs, while the Right one shows the `Brick`-decomposed 3-D FFTs.

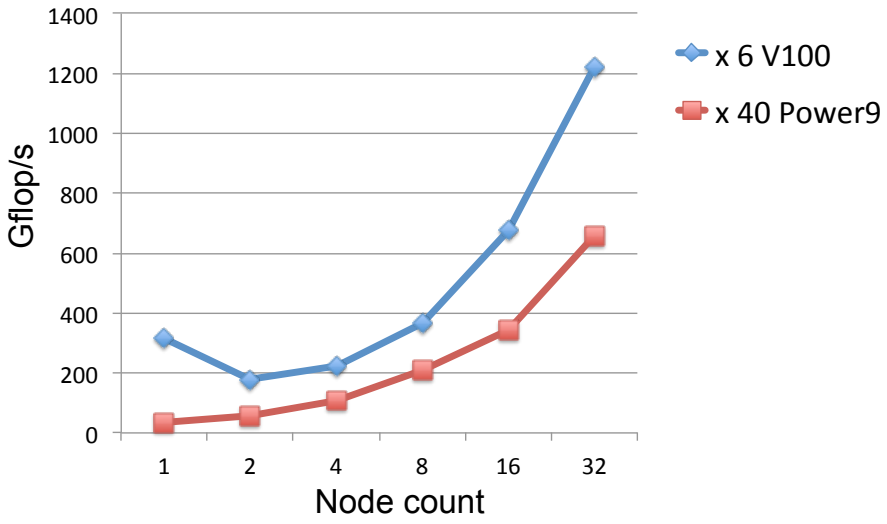

Figure 7.2: Strong scaling performance (in GFlop/s) on 3-D FFTs of size 1024^3 on up to 32 nodes of Summit. Both CPU and GPU codes are highly optimized, using `Pencil`-decomposed 3-D FFTs based on the FFTMPI FFT library. The CPU code uses 40 cores per node (1 core per MPI process), while the GPU code uses 6 V100 GPUs per node (1 GPU per MPI process).

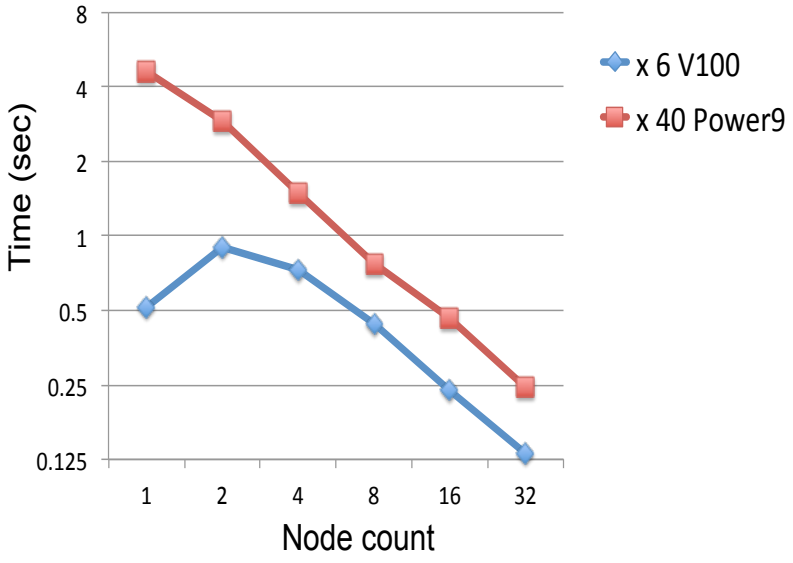

Figure 7.3: Strong scaling computational time (in seconds) on 3-D FFTs of size 1024^3 on up to 32 nodes of Summit. Both CPU and GPU codes are highly optimized, using `Pencil`-decomposed 3-D FFTs based on the FFTMPI FFT library. The CPU code uses 40 cores per node (1 core per MPI process), while the GPU code uses 6 V100 GPUs per node (1 GPU per MPI process).

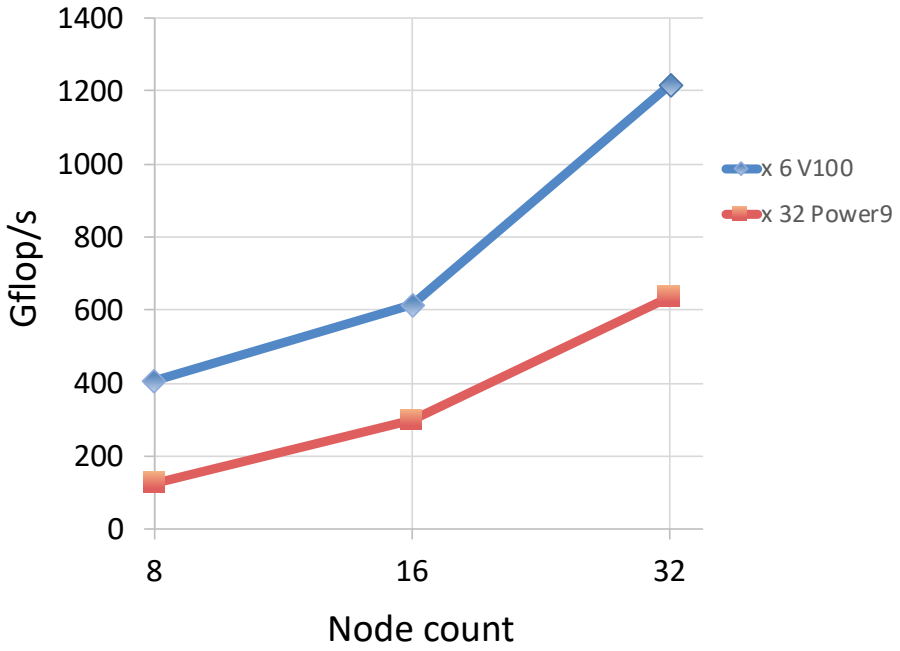

Figure 7.4: Strong scaling performance (in GFlop/s) on 3-D FFTs of size 1024^3 on up to 32 nodes of Summit. Both CPU and GPU codes are highly optimized, using `Brick`-decomposed 3-D FFTs based on the SWFFT FFT library. The CPU code uses 32 cores per node (1 core per MPI process), while the GPU code uses 6 V100 GPUs per node (1 GPU per MPI process).

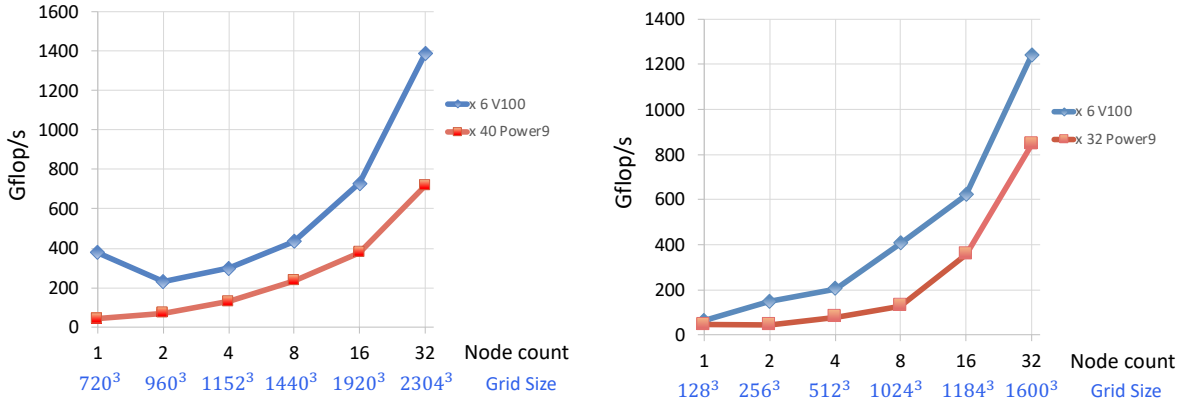

Figure 7.5: Weak scaling performance (in GFlop/s) on 3-D FFTs on up to 32 nodes of Summit. Both CPU and GPU codes are highly optimized, using Pencil-decomposed 3-D FFTs (Left) and `Brick`-decomposed 3-D FFTs (Right). The grid sizes solved are listed on the x-axes, along with the corresponding number of CPU cores or GPUs used for the scaling.

CHAPTER 8

Conclusions and future work directions

In this milestone, we designed and developed an initial implementation of 3-D FFTs in the FFT-ECP project. The target architectures are large-scale distributed GPU-accelerated platforms. We described the design and performance of the code on large-scale heterogeneous systems with GPUs like the Summit supercomputer at ORNL. This milestone also delivered on the following sub-tasks:

- Investigation and development of different communication schemas in FFT-ECP;
- A study of scheduling computations and communications in FFT;
- A study on the memory models and data distributions for FFTs;
- Developed and optimized building blocks and GPU kernels for FFTs;
- Delivered an initial version of the 3-D FFT for heterogeneous, multi-GPU nodes and ECP applications, and provided a study on its performance.

The work also highlighted a number of bottlenecks and opportunities for accelerating the codes. The main bottleneck currently is in the MPI communications, as everything else is highly optimized to the point that it takes only about 2-3% of the total execution time, while the rest is in MPI communications. Thus, further improvement can be achieved by optimizing the communications, e.g., through improvements in MPI and scheduling of tasks. Also, there are versions for particular sizes (and application needs) that communicate less, which are all subject of future developments. These are the main targets of the next milestone – namely, implementation optimization and features phase that aim to provide specific ECP application needs for fast FFTs.

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] Ahmad Abdelfattah, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. Performance, Design, and Autotuning of Batched GEMM for GPUs. In *High Performance Computing - 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings*, pages 21–38, 2016. doi: 10.1007/978-3-319-41321-1_2.

[2] 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.

[3] Azzam Haidar, Tingxing Dong, Piotr Luszczek, Stanimire Tomov, and Jack Dongarra. Batched matrix computations on hardware accelerators based on GPUs. *International Journal of High Performance Computing Applications*, 2015. doi: 10.1177/1094342014567546. URL http://hpc.sagepub.com/content/early/2015/02/06/1094342014567546.abstract.

[4] 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/.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.