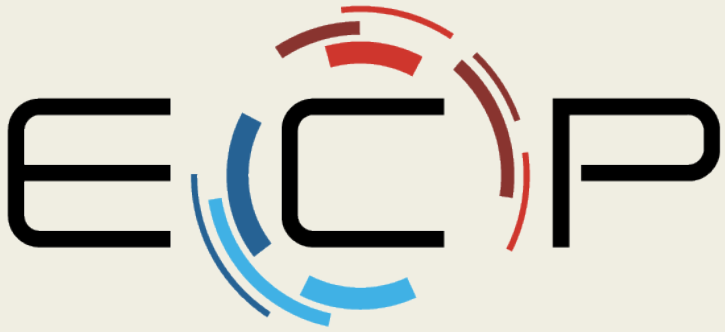

EXASCALE COMPUTING PROJECT

ECP Milestone Report

Evaluation and Design of FFT for Distributed Accelerated Systems

WBS 2.3.3.09, Milestone FFT-ECP ST-MS-10-1216

Stanimire Tomov
Azzam Haidar
Daniel Schultz
Jack Dongarra

Innovative Computing Laboratory, University of Tennessee

October 5, 2018

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{thsd2018ECPFFT,
  author={Tomov, Stanimire and Haidar, Azzam and Schultz, Daniel and Dongarra, Jack},
  title={{Evaluation and Design of FFT for Distributed Accelerated Systems}},
  institution={Innovative Computing Laboratory, University of Tennessee},
  year={2018},
  month={October},
  type={ECP WBS 2.3.3.09 Milestone Report},
  number={FFT-ECP ST-MS-10-1216},
  note={revision 10-2018}
}
```

Contents

List of Figures

List of Tables

CHAPTER 1

Executive Summary

The goal of this milestone was the evaluation and design phase of FFT targeting distributed accelerated systems. In this milestone we describe the current performance of FFT libraries and propose a design framework for the FFT-ECP project. Specifically, this milestone delivered on the following sub-tasks:

- Evaluation and benchmarking of current/existing FFT libraries from open-source developers and vendors;
- Evaluation and benchmarking of the FFT code used in other ECP applications, including: LAMMPS and HACC;
- Study the interoperability between current vendor FFT libraries and the existing FFT library used in ECP applications, particularly for use in heterogeneous nodes with many accelerators;
- Propose a framework design for FFT-ECP and investigation for possible integration and/or use of vendor- developed or open-source FFT codes with our 2-D and 3-D FFT-ECP framework that emphasizes multi-GPU nodes;
- Analysis of the communication/computation cost and memory overhead for different FFT variants and provide a study of the behavior on current and future architectures with distributed and heterogeneous multi-GPU nodes.

CHAPTER 2

Background

The Fast Fourier Transform (FFT) is used in many applications such as molecular dynamics, spectrum estimation, fast convolution and correlation, signal modulation and many wireless multimedia applications. The distributed 3D FFT is one of the most important kernels involved in Molecular Dynamics (MD) computations and its performance can affect MD scalability at large scale. MD requires to solve 3D FFTs of medium size ($10^6 - 10^8$ points). The performance of the first principles calculations strongly depends on the performance of the FFT solver that performs many FFTs of size $\approx 10^7$ points in a calculation that we call batched FFT. Moreover, many Poisson PDE type of equations arising from many engineering areas such as PLASMA simulation, density field, etc., need to solve FFT of size above 10^9. On the DOE side, we found that more than dozen of ECP applications use FFT in their codes.

However, while needed, state-of-the-art FFT libraries like FFTW are no longer actively developed for emerging platforms. To address this deficiency, the ECP ST has initiated two new FFT efforts [8] – the FFTX that explores the development of a new FFT software stack, e.g., capable of replacing FFTW, and the FFT-ECP, subject of this project, that aims to provide a sustainable 3D FFT library built from existing components. The latter approach has been used to construct *ad hoc* FFTs in applications, e.g., relying on third-party 1D FFTs from vendors or open-source libraries. Our goal is to:

- Collect existing FFT capabilities from ECP applications (LAMMPS/`FFTMPI` and HACC/`SWFFT`);
- Explore also vendor FFT capabilities to build 3D FFT libraries using node kernels;
- Asses current deficiencies and how to address them, while leveraging the existing FFT capabilities to build a sustainable FFT library for Exascale platforms.

The FFT-ECP goals and approach present a high value proposition – the effort will not only provide a new and sustainable high-performance FFT library for Exascale platforms, but also will leverage the large investments in FFT software by the broader HPC community. Moreover, we will propose and develop a flexible API to simplify the use of FFT-ECP in applications.

CHAPTER 3

Evaluation and Benchmarking of State-of-the-art FFT Libraries

The current state-of-the-art of FFT libraries does not seem to be scalable to large number of heterogeneous nodes or even to one multi-GPU node with high-end GPUs such as the Nvidia V100 GPU. Furthermore, these libraries require large size FFTs in order to deliver acceptable performance on one GPU. Efforts to simply enhance classical and existing FFT packages with optimization tools and techniques such as autotuning and code generation have not been able so far to provide the efficient, high-performance FFT library needed and capable of harnessing the power of supercomputers with heterogeneous GPU-accelerated nodes. Therefore, in this milestone, we first studied the existing FFT capabilities from vendors and open-source libraries, the potential of using them as building blocks, and assessed their performance gaps in order to develop a GPU-based distributed 3D FFT library that can deliver high performance on current and future machines.

3.1 FFTs on single node multicore processors

There are many FFT CPU-based nodal libraries, and we evaluated a number of them. These FFT libraries come from open source efforts or vendors, and include libraries like AccFFT [4], Cray FFT, FFTW [3], FFTE [19], Intel MKL [9], IBM, nb3DFFT [5], Parallel FFT, P3DFFT [15], PsFFT, and Spiral [17].

FFTW is widely recognized as one of the most popular and efficient CPU-based FFT libraries. Therefore, for the FFT-ECP developments we concentrate on the FFTW library, as well as on vendor implementations, as they provide in general the best performance. This was also confirmed by the benchmarks that we created and ran in order to analyze these libraries. For current multicore nodes, current performance levels are illustrated in Figure 3.1. Shown is the performance in GFlop/s for 3D FFTs from the MKL FFT and the FFTW libraries on two 10-core Intel Xeon E5-2650 v3 processors (Haswell).

To compute the Gflop/s rate, here and everywhere below, we assume that the flops for a 3D FFT over

$N \times N \times N$ data points are $5N^3 log_2 N^3$.

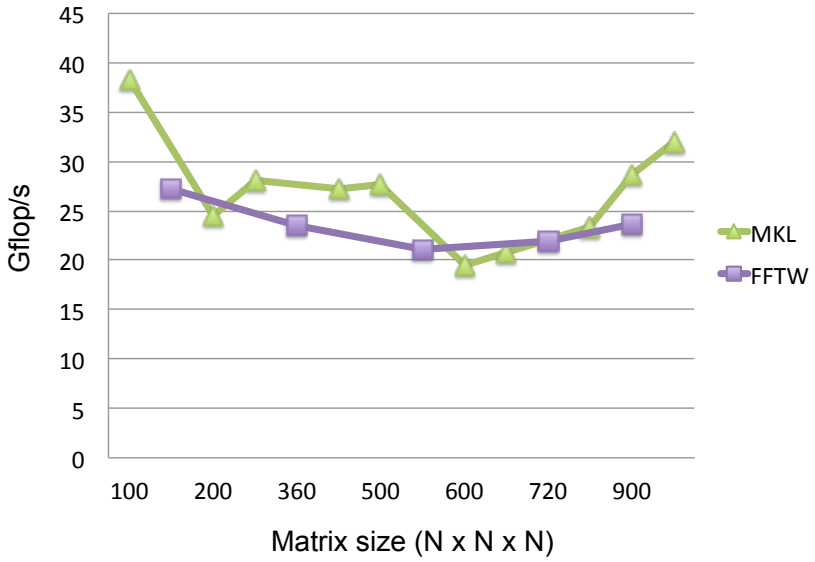

Figure 3.1: Performance of 3D FFTs in MKL and FFTW in double complex arithmetic on two 10-core Intel Xeon E5-2650 v3 processors (Haswell). Performance numbers assume $5N^3 log_2 N^3$ flops for a 3D FFT on a $N \times N \times N$ tensor.

We note that this is the best performance obtained among the different libraries. In this case MKL is slightly faster and performance may oscillate based on the different sizes tested. FFTW is very competitive and gives somehow smoother performance across the different sizes. Both libraries have a performance of about 25 GFlop/s (in double complex arithmetic) on average, with MKL reaching up to 38 GFlop/s and FFTW up to 27 GFlop/s.

The `FFTMPI` [16] and `SWFFT` [18] FFT libraries are of particular interest because FFT-ECP will follow their approach. The libraries are further discussed in Section 4. Here we only mention that these libraries use the nodal 1D FFT capabilities of libraries like MKL and FFTW to represent multidimensional FFTs. For example, a 3D FFT will be a batch of 1D FFTs in the x direction, followed by batches of 1D FFTs in the y, and z directions, where the order of the directions does not matter. We can run these libraries on a single node and results of doing this are illustrated in Figure 3.2. Shown is the performance of `FFTMPI` for 3D FFT using MKL and FFTW3, respectively.

We note that although `FFTMPI` is based on the 1D FFTs from MKL and FFTW, the nodal performance of `FFTMPI` is lower than the standalone MKL or FFTW from Figure 3.1. Namely, performance here is about 14 GFlop/s vs. 25 GFlop/s in the standalone libraries case. While some slowdown is expected as the `FFTMPI` design and optimizations target different problems (than running on a single node), the experiment still point out that although the libraries use the same building blocks, further optimizations matter and can have significant impact on performance.

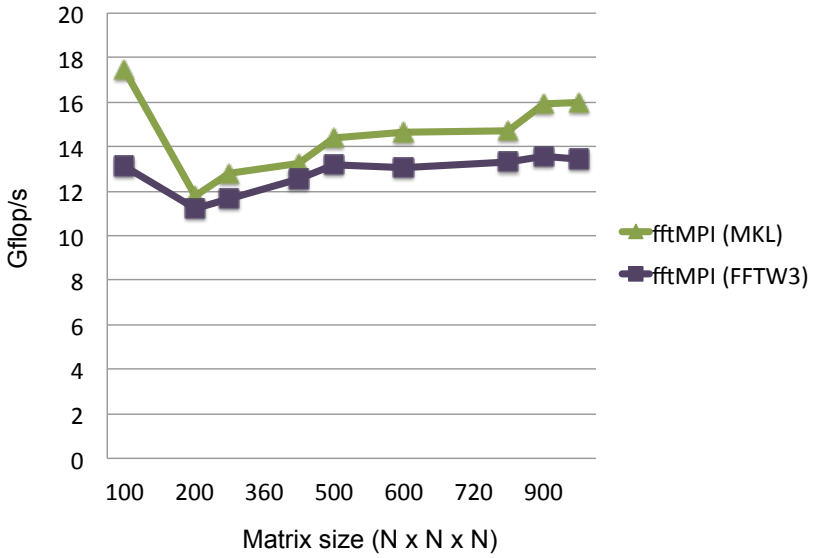

Figure 3.2: Performance of 3D FFTs in `FFTMPI` in double complex arithmetic on two 10-core Intel Xeon E5-2650 v3 processors (Haswell). Shown is performance of 3D FFT from `FFTMPI` using 1D FFTs from MKL vs. FFTW.

3.2 FFT building blocks for GPUs

The best performing FFT library for GPUs is Nvidia's cuFFT [14] library. The performance of cuFFT on a V100 GPU is illustrated in Figure 3.3 – on the Left is performance of batched 1D FFTs that is the main building block in our approach, and on the Right is a 3D FFT.

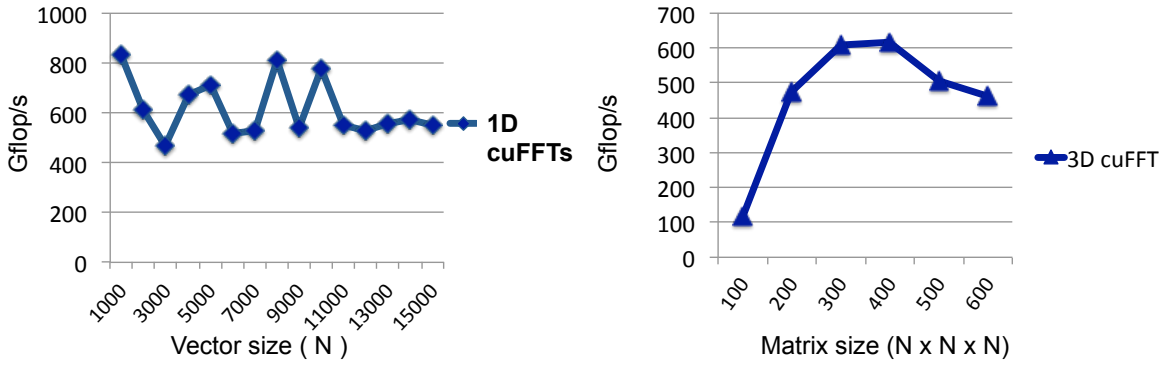

Figure 3.3: Performance of NVIDIA cuFFT in double complex arithmetic on V100 GPU. Shown is performance of a batch of 1,000 1D FFTs (Left) and 3D FFT (Right).

Note that the performance is significantly higher (about $20\times$) than what we get on multicore CPUs, where the GPU and the two CPUs have about the same power draw. The Summit supercomputer at ORNL features six V100 GPUs per node, thus leaving the possibility that a library could potentially extract about 3,000 GFlop/s per node in doing FFTs (vs. the ≈ 25 GFlop/s extracted from just a multicore node). However, our analysis and performance modeling shows that the scalability performance per node is much lower (which we discuss in detail later).

To further evaluate the FFT-ECP approach on single GPU, we developed a number of 2D and 3D FFT versions made out of 1D FFT building blocks (from cuFFT). The pefromance is illustrated on Figure 3.4.

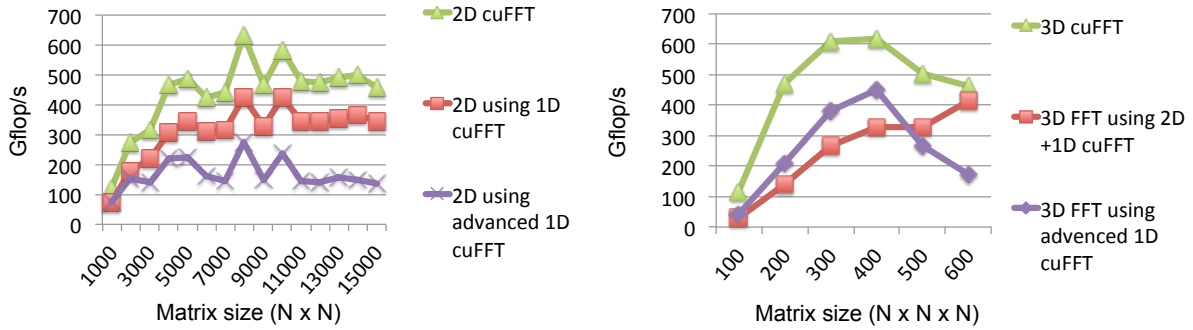

Figure 3.4: Performance of NVIDIA 2D cuFFT vs. 2D FFTs composed of 1D FFT building blocks from cuFFT (Left) and NVIDIA 3D cuFFT vs. 3D FFTs composed of 1D FFT building blocks from cuFFT (Right).

Similarly to the CPU results for MKL/FFTW vs. `FFTMPI` using MKL/FFTW (in Figure 3.1 vs. Figure 3.2), the vendor tuning for the entire computation (a 3D FFT) is currently faster than our quickly assembled codes. Still, they are very close in performance, indicating that further optimizations and tuning can lead to top performance (matching NVIDIA's FFT), and that we can very efficiently combine current building blocks in constructing multidimensional FFTs.

The red curves in Figure 3.4 use matrix transposition kernels from MAGMA [20, 21] to align the data so that the 1D FFTs are always performed on data with stride one (entries that are consecutive in memory). The purple curve indicates a version that uses the `cufftPlanMany` plan interface from cuFFT. This option allows one to specify proper strides for the elements in the 1D FFTs, and thus to avoid the use of explicit transposition kernels. While this is easier to code (when available; as in cuFFT), we see that performance is not always better.

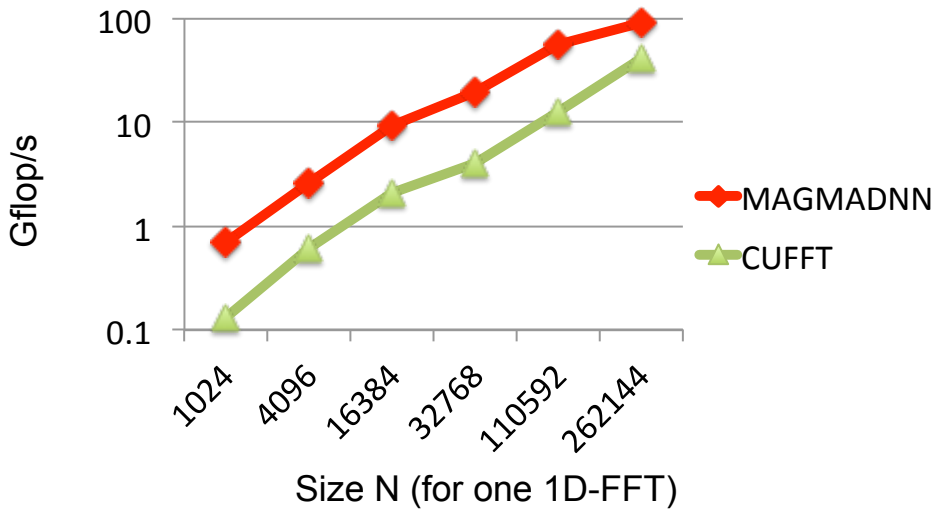

Figure 3.5: Performance of a single 1D MAGMA FFT vs. cuFFT for small size vectors on an NVIDIA V100 GPU using larger size radix and ZGEMM.

Finally, although the focus of the FFT-ECP is not on nodal performance, we explored the possibility of accelerating 1D FFTs by stopping the recursion in Cooley-Tukey type of FFT algorithms to a larger radix, e.g., 32, and perform the 1D FFTs on vectors of size 32 using GEMM. We did an implementation in MAGMADNN [13] and Figure 3.5 shows that acceleration vs. cuFFT is possible on GPUs (up to certain small sizes) using this type of acceleration. This optimization technique can have implications and use for tuning the strong scalability of FFTs.

3.3 FFTs for distributed memory systems

Well optimized FFT implementations are known to have excellent weak scalability (i.e., perfect, until the cross-sectional bandwidth of the nodes involved scales linearly) and good strong scalability on distributed multicore systems. We developed benchmarks and shell scripts that run them for various sizes, number of nodes, configurations options, etc., collect the output and extract performance numbers to be easily plotted and analyzed. We plan to use these developments in automated tuning infrastructure tools to be developed for the FFT-ECP optimization and tuning.

Figure 3.6 shows the scalability of 3D FFTs from FFTW on a small cluster using up to 180 MPI ranks on 9 nodes with 2×10 cores Intel Xeon E5-2650 v3 processors (Haswell) connected with Infiniband (100G EDR MSB7700/U1 switch).

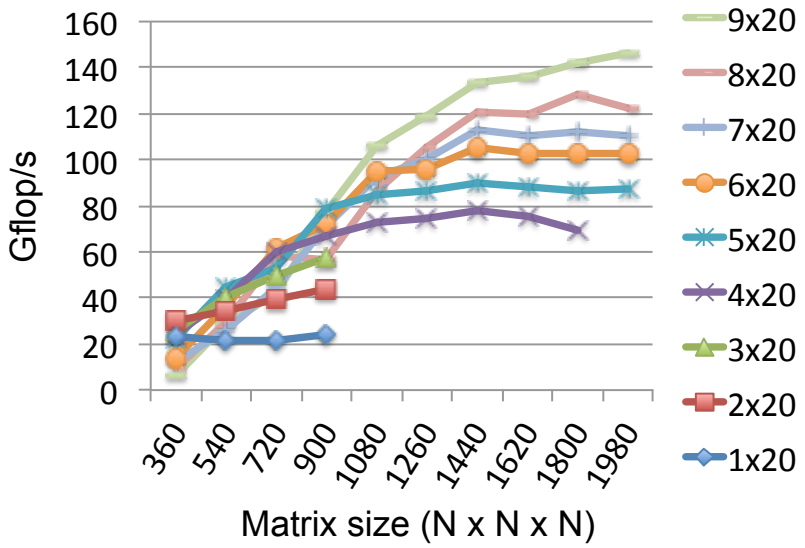

Figure 3.6: Scalability of 3D FFTW on up to 180 MPI ranks (on 9 nodes with 2×10 cores Intel Intel Xeon E5-2650 v3 processors (Haswell) in an Infiniband cluster (100G EDR MSB7700/U1 switch).

We developed a performance model to guide our future developments and optimizations and show (later in our report) that performance for the FFT is memory bound. In particular, while the FFT has weak scaling, we show that based on the interconnect bandwidth, the performance that can be extracted per node is limited. For example, for a 3D FFT with Infiniband (12.5 GB/s; bi-directional throughput 25 GB/s), the performance that can be extracted from a node will be limited to

$$25 * \frac{5}{32} * log_2 N \text{ Gflop/s},$$

which is about 52 Gflop/s for N = 10,000 (and should weak-scale with the number of nodes used). This is important because up coming systems feature multiple GPUs per node and although a node, e.g., on the

Summit supercomputer, can reach theoretically up to 3000 GFlop/s in doing local FFTs, only up to 104 GFlop/s will be used in the scalability (as bandwidth on Summit is Dual Rail EDR-IB – up to 50 GB/s bi-directional throughput).

Thus, obtaining high nodal performance would have very limited effect on the overall performance. Still, the current FFT results on Summit that use CPUs only show a scalability of 9 Gflop/s per node [2]. Therefore, the use of GPUs would be able to accelerate it multiple times, while still leaving plenty of compute power for simultaneous calculations other than the FFT.

Because of the importance of the nodal FFT performance for the overall scalability, as described above, we show the average per node performance on our Intel cluster. Figure 3.7, Left, shows that the weak scalability per node of MKL can get to about 26 GFlop/s. The achievable (measured) bandwidth here is 21.6 GB/s, and therefore, by the above formula, a limit for the performance per node is 37 Gflop/s.

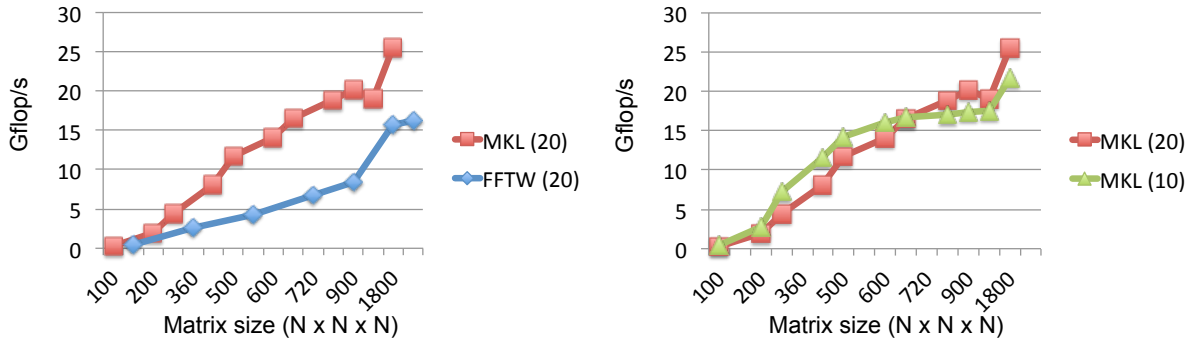

Figure 3.7: Scalability of FFTW and MKL 3D FFT on a cluster with two 10-core Intel Xeon E5-2650 v3 processor (Haswell) nodes connected with Infiniband (100G EDR MSB7700/U1 switch). The performance shown is in Gflop/s per node (Left). On the Right is MKL performance per node when using 10 vs. 20 cores on the node.

Figure 3.7, Right shows that one may not need the full power of the node to get to full speed for a 3D FFT. Indeed, comparing the nodal average performance of MKL 3D FFT when using 10 vs. 20 of the node's cores, illustrates that the performance remains about the same. Similarly, to supercomputers with GPUs, we expect that not all GPUs will be needed to achieve top performance on 3D FFTs.

CHAPTER 4

FFT Libraries in ECP Applications

Today's machines have very complex memory hierarchies and thus data movement, data layout translation, and communication should be the main focus of any distributed FFT library that aims to improve the performance of any ECP application that relies on FFT. Vendors and optimized open-source libraries provide very well optimized and tuned FFT routines for a single node or a single GPU (as shown in Section 3). Therefore, although any effort on optimizing FFT kernels will be helpful, we prefer to target an approach that uses the available existing kernels and adapts them to build a more general and scalable 3D FFT library.

ECP applications that require FFT-based solvers might suffer from the lack of fast and scalable 3D FFT routines for distributed-heterogeneous parallel systems as the ones projected for the upcoming exascale computing systems. Also, ECP applications may not be able to use existing FFT libraries without application-specific adjustments and tuning. For that, one of the main key to succeed with such project, is not only to study and analyze the current existing FFT libraries but also to study ECP application needs, study their FFT implementation and provide them with a suitable modular high-performance implementation that is flexible and easy to use and integrate in their framework. Therefore, we initiated discussions with ECP applications, and in particular LAMMPS and HACC, got their current FFT implementations and proceeded with a detailed analysis of each of its function calls. In this Chapter we provide a summary of our investigation and the lessons learned. The libraries that we analyzed are the `FFTMPI` and `SWFFT` that are used by the LAMMPS and HACC projects, respectively.

The main motivation and concept behind the development of both libraries is to provide flexibility and control in the grid sizes, the number of processors, and in the data layout that real-life applications need. Both libraries are based on the same concept and provide very similar flexibility needed by the corresponding applications. Both take data in pencil or cubic distribution, then reshuffle the data to perform the FFT and can deliver back the results either pencil or cubic distributions. Communication can be chosen to be point to point or global. Thus, the code takes the input data and then first forms the pencil data decomposition, after which a one-dimensional FFT is taken along the long dimension of the pencil (see Figure 5.2). For convenience, the same distribution algorithm is employed to carry out the remaining two transforms by redistributing the domain into pencils along those respective dimensions.

At the end, redistribution is carried out to provide output as specified.

4.1 The FFTMPI Library Used in LAMMPS

FFTMPI is a set of routines to perform 2D and 3D complex-to-complex Fast Fourier Transforms (FFTs) efficiently on parallel computers. FFTMPI has been mainly used by the LAMMPS ECP project [11]. The routines in FFTMPI are designed for distributed-memory parallel machines and use MPI as their message-passing protocol. Actually, the routines only perform the data movement tasks necessary to compute multi-dimensional FFTs in parallel; the transforms themselves are computed by on-processor 1D FFT routines provided by the numerical libraries of the machine vendor or the freely available FFTW package. The data remapping routines can also be called directly by the user (independent of the FFTs) to change the layout of the application's 2D or 3D arrays across the processors available.

4.2 The SWFFT Library Used in HACC

SWFFT is the distributed 3D Fast Fourier transform (FFT) used in the HACC [6, 7] code and now available as a separate library. HACC uses a high-order spectral algorithm to solve the Poisson equation resulting from a density field sourced by a large number of tracer particles. Because HACC often runs at the memory limits of the target system, reducing memory overhead is a high priority for this ECP application, as is good performance, and, obviously, scalability. SWFFT was designed to carry out large-scale FFTs over a large number of MPI ranks. There are basically two standard strategies for parallelizing FFTs. Slab-decomposed parallel FFTs which do not scalable to very large number of MPI ranks, and data partitioning across a two-dimensional subgrid ("pencil" decomposition), which are better suited for a large number of MPI ranks. SWFFT uses the later. SWFFT assumes that the data is available in a 3-D "cuboid" domain decomposition and then uses a pencil-decomposition in order to carry out successive 1-D FFTs, interleaved with global communication. The implementation has acceptable performance, low memory overhead, interleaves communication and computation, and avoids potential communication deadlocks.

4.3 Performance and Analysis

We benchmarked both FFT libraries on distributed CPUs system, where each node consists of two socket Intel Xeon E5-2650 v3 processors (Haswell). The goal was to study the weak and strong scaling and provide an analysis of the current performance (e.g., if it is good enough, or if we can improve it), as well as assess what are the expectations for the FFT-ECP library that we will be developing. We note that both FFTMPI and SWFFT are CPU libraries and thus do not take advantage of GPUs. However, our proposed FFT-ECP is going to be distributed CPUs and multi-GPU library.

We illustrate the scalability performance results when varying the grid size and the number of processors in Figure 4.1 and Figure 4.2 for the FFTMPI library on 2D and 3D grids, respectively and on Figure 4.3 for the SWFFT library for 3D grid. We note that SWFFT does not have a 2D FFT and only uses FFTW3 as backend for the 1D FFT computations.

From a first quick look into the scalability figures (Figure 4.1, Figure 4.2 and Figure 4.3), we can see that both libraries scale well on distributed CPU machines. The FFT algorithm is memory bound by itself and also requires data communication that makes it also network bound.

A strong scalability analysis can be viewed as a vertical line on a particular size where we can see vertically the speedup that is obtained when increasing the number of processors. For example, on Figure 4.2, the

right graph, if we look vertically on a grid of size $1K \times 1K \times 1K$, we can find that using 40 MPI process, the FFT run at 28 Gflop/s and when using 80 MPI process, it got up to 52 Gflop/s and then it can reach 100 Gflop/s when using 160 MPI process. We can say that we obtained a nice good strong scalability.

However, based on these observations, on our experience developing HPC libraries and on our analysis of the FFT algorithm on distributed memory machines, we can say that these current implementations are bound by data movement (e.g., data copy and communication), and that the computation consists of a small portion of the total time compared to the data movement.

Also, based on our theoretical model presented in Section 5.2, we can conclude that the obtained performance are far from the theoretical upper bound which indirectly mean there might be room for improvement on distributed memory CPU-based systems.

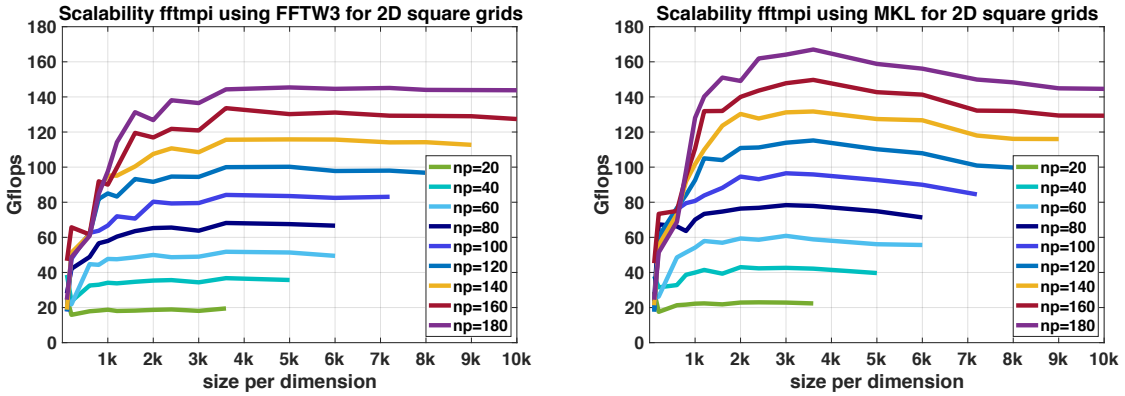

Figure 4.1: Scalability of the FFTMPI implementation for 2D square grid when using different number of processors.

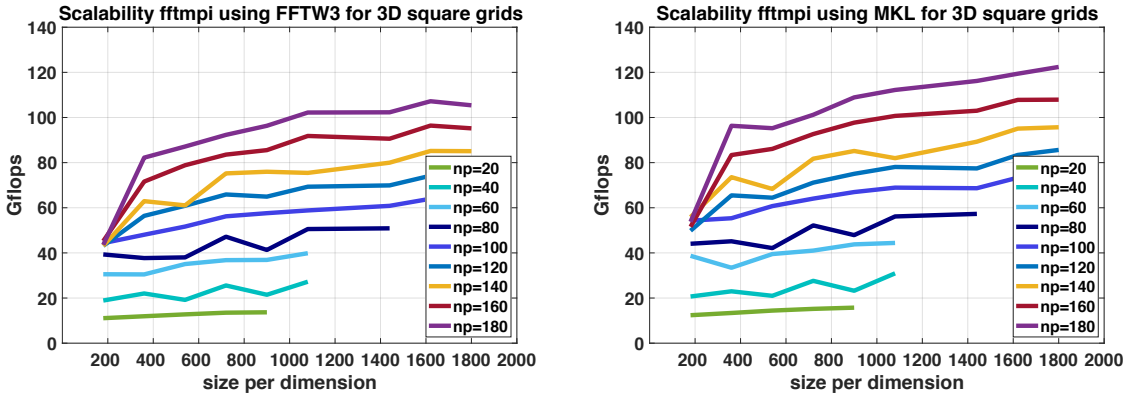

Figure 4.2: Scalability of the FFTMPI implementation for 3D square grid when using different number of processors.

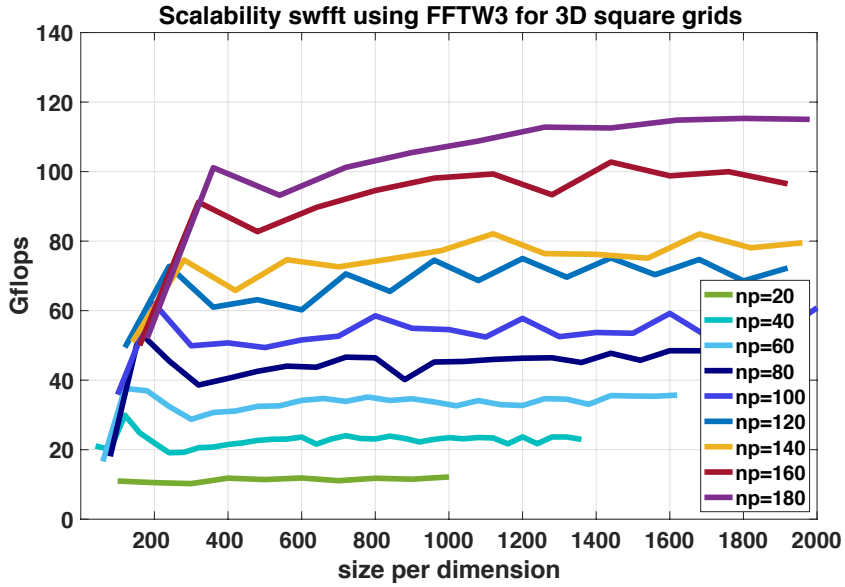

Figure 4.3: Scalability of the SWFFT implementation for 3D square grid when using different number of processors.

Table 4.1 illustrates the weak and strong scalability that SWFFT had obtained on Summit using only its CPUs in runs by [2]. Summit has two IBM Power9 sockets (each socket having 22 cores with 4 hwthreads/core) per node. Note that, if we take the strong scalability 3D FFT problem for N=576, and we run it on one NVIDIA V100 GPU, we can achieve about 853 Gflop/s, which is the equivalent to the 128 CPU nodes, which also means 5,632 Power9 cores.

		weak scalability				strong scalability, N=576	
Nodes		N	GFlop/s	GFlop/s / node		GFlop/s	GFlop/s / node
2		576	17	8.5		16.8	8.4
4		720	35.6	8.9		32.8	8.2
8		936	71.2	8.9		71.2	8.6
16		1152	126.4	7.9		152	9.5
32		1440	268.8	8.4		288	9.0
64		1872	512	8.0		555	8.5
128		2304	1062	8.3		960	7.5
V100						853	

Table 4.1: Weak and strong scalability of SWFFT on Summit using only its IBM CPU Power9 processors (two per node); runs by [2]. Note that in the context of GPUs, one NVIDIA V100 GPU can solve the strong scalability 3D FFT problem for N=576 at 853 GFlop/s, which is the equivalent of 128 CPU nodes.

In order to verify our analysis, we decided to dive into the FFT implementation and get a detailed analysis of its computational and communication portions. For that we instrumented the code to generate performance traces and we present in Figure 4.4 and Figure 4.5 these traces of the execution time for the FFTMPI library. We show the timing of each function call on each MPI process for a run on 80 MPI processes for a 2D FFT of size $10K \times 10K$ in the 2D-trace-run on Figure 4.4 and for a grid of size $1K \times 1K \times 1K$ for the 3D-trace-run on Figure 4.5. As it can be seen from the figures, a large portion

of the time is spent in the data movement (meaning local data copy within the same MPI process and MPI communication between the MPI ranks – these are the blue and cyan colors, respectively), while a small portion is spent in the 1D FFT computation (orange and yellow portion). This was expected from our theoretical analysis and model of the scalability, as previously discussed and presented with further details in Section 5.2.

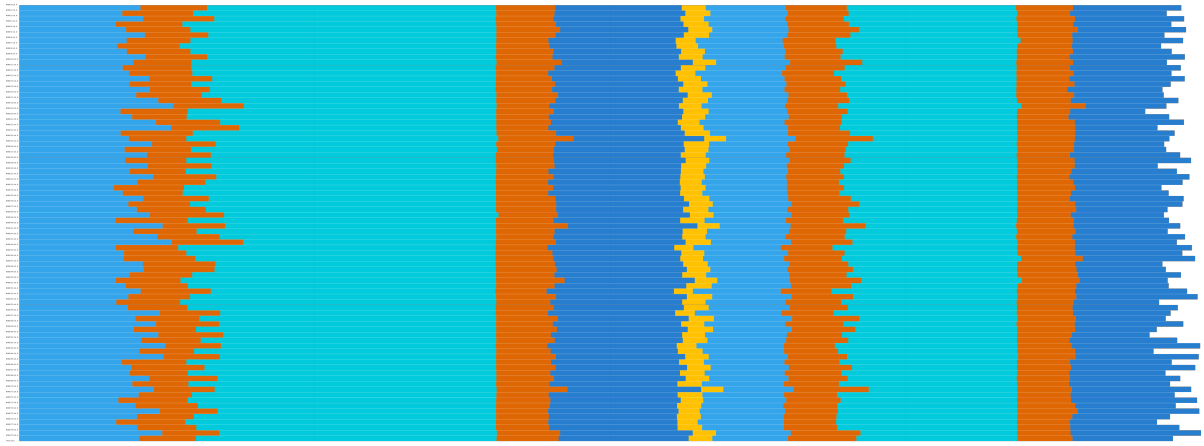

Figure 4.4: Trace of 2D `FFTMPI` using MKL on 80 MPI process (Intel Xeon E5-2650 v3 processors Haswell) for a 2D FFT on a $10K \times 10K$ grid.

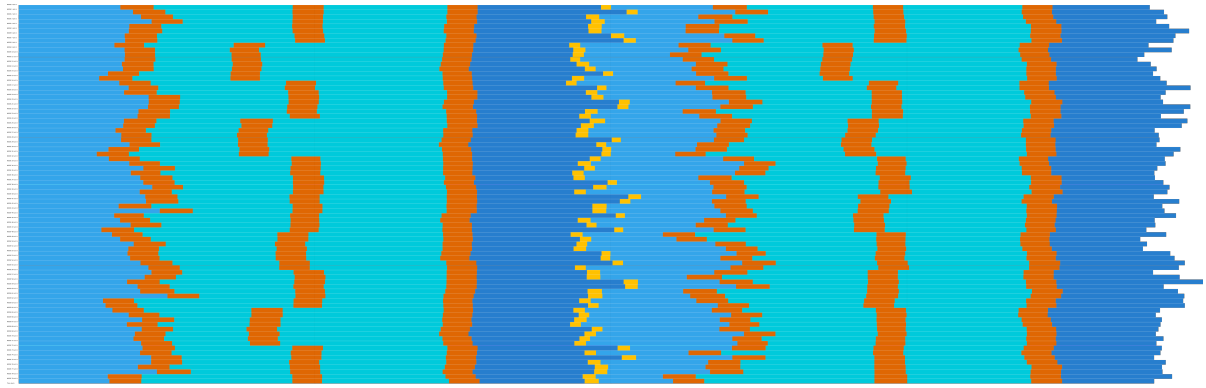

Figure 4.5: Trace of 3D `FFTMPI` using MKL on 80 MPI process (Intel Xeon E5-2650 v3 processors Haswell) for a 3D FFT on a $1K \times 1K \times 1K$ grid.

In order to understand more deeply the issue, we decided to study the cost of each function call, thus we instrumented the lowest level of the code and we generated the traces of the lowest function calls for the same test case as of Figure 4.4 and Figure 4.5. The detailed traces are presented in Figure 4.6 and Figure 4.7. Each function is represented by a color, but we omit the detail of each function as this will be considered low technical study. However, in the new trace we add the capability to differentiate between the local data copy and the MPI communication function calls. For that, we show in bluish color the functions related to local data copy, while we illustrate in purplish the MPI communication functions. This observation correlates perfectly with our analysis and with our FFT-ECP project design for a distributed FFT library where we highlight the fact that most of the effort need to be focused into the design phase such as to optimize the data movement and try to overlap the data copy with the data communication, which we believe will be the bottleneck of any distributed memory FFT library.

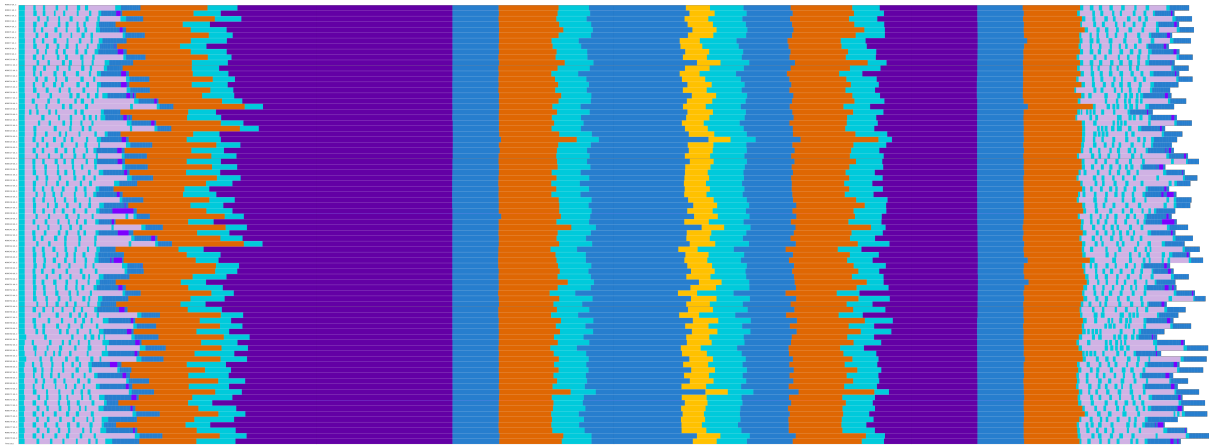

Figure 4.6: Trace of 2D FFTMPI using MKL on 80 MPI process (Intel Xeon E5-2650 v3 processors Haswell) for a 2D FFT on a $10K \times 10K$ grid.

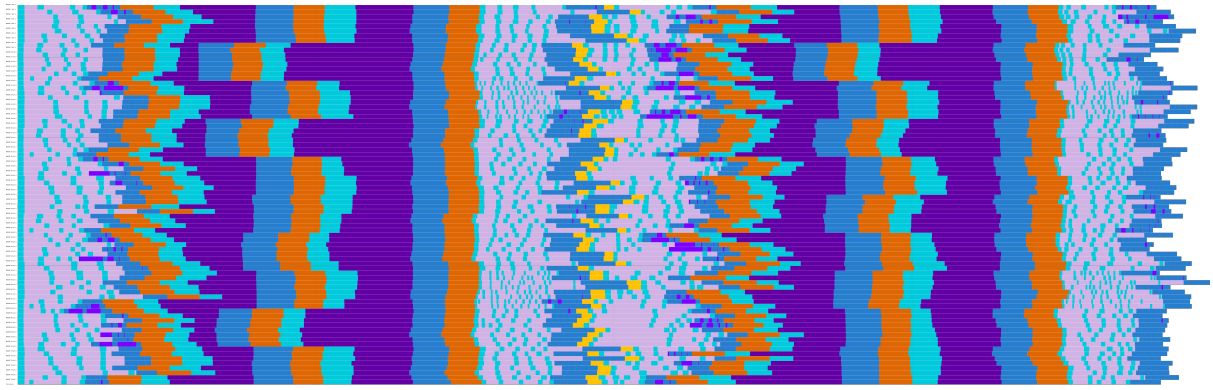

Figure 4.7: Trace of 3D FFTMPI using MKL on 80 MPI process (Intel Xeon E5-2650 v3 processors Haswell) for a 3D FFT on a $1K \times 1K \times 1K$ grid.

As a consequence, our FFT-ECP implementation design will be based on the theoretical model described below, on this analysis, and on the benchmarking of MPI communication and memory bandwidth. We will be developing different technique of communications (e.g., point to point, global, global within a sub communicator, asynchronous task based communications) The goal is to propose a design that can overlap and hide the data copy and communication as well as computation while reaching the maximal possible bandwidth which indirectly means reaching as high as possible to the theoretical performance upper bound. Note that the flexibility in data input and data output is something highly desired by the ECP apps and for that we are taking this flexibility into consideration for our FFT-ECP design.

CHAPTER 5

Framework Design for ECP-FFT

As discussed so far, the overall objective of the FFT-ECP project is to design and implement a fast and robust 2-D and 3-D FFT library that targets large-scale heterogeneous systems with multi-core processors and hardware accelerators and to do so as a co-design activity with other ECP application developers.

Studying existing vendor and open source optimized FFT's libraries, the FFT libraries that current ECP applications use, as well as the ECP application needs in term of data input/output, FFT's sizes, grid flexibility and code portability helped us understand current performance and bottlenecks, as well as current supported hardware and how these libraries are interoperable with each other. It also helped us on the design process and led us draw the path of our development, as outlined in this section.

5.1 Interoperability of Vendor FFTs and FFTs in ECP Applications

Generally speaking, vendor libraries (like MKL FFT and NVIDIA cuFFT) and open source libraries (like FFTW, FFTE) can interoperate with FFTs in ECP applications. However, there are challenges that have brought the need to develop specialized FFTs libraries, like the `FFTMPI` and `SWFFT` in LAMMPS and HACC, respectively. Some of these challanges are summarized and listed as follows:

- Open source libraries and vendor libraries have been mostly focused on nodal performance or small clusters, and thus lacking in scalability on large-scale supercomputers;
- There is no support for GPUs in the state-of-the-art libraries for distributed systems, and thus no portability/support for the up coming Exascale platforms;
- Application-specific input and output, e.g., arbitrary initial decompositions, is not supported directly through standard FFT APIs;
- `FFTMPI` and `SWFFT` were specifically designed to target weak and strong scalability on CPU based

systems, to handle large and small problems, application-specific input and output, and reduced memory requirements.

While there are challenges, as given above, both `FFTMPI` and `SWFFT` manage to use FFTW or MKL for their 1D FFTs. This is encouraging for the interoperability and especially for adding support for GPUs, because cuFFT provides FFTW3 interfaces to the cuFFT library (see Section 7 in [14]).

Our next step (milestone) is the design and implementation phase of the FFT-ECP. Our first step in that direction, which we are close to complete, is to use the FFTW3 interface of cuFFT to easily provide GPU support for `FFTMPI` and `SWFFT`.

5.2 Performance Analysis and Model

We developed a performance model to evaluate how well FFTs perform and to guide our up-coming development and optimization efforts. A challenge in developing high-performance FFTs is that the FFT computation does not have high computational intensity (that we define as the Flops/Byte ratio). Indeed, Table 5.1 shows the FFT's computational intensity in contrast to the dense matrix-matrix multiplication in double complex arithmetic (ZGEMM).

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 5.1: 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 Gflop/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.

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

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

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

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

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 GFlop/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:

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

Here we divide by 2 to take into account of once reading the vectors and writing the results back. Note that here N will be unrealistically high in order to make performance bounded by 600. In other words, the intensity of the computation does not grow fast enough in order to apply some blocking techniques like in the dense linear algebra case. In conclusion, performance is always memory bound:

$$P_{FFT} \leq GB/s \ \frac{0.312 \ log_2 N}{2}.$$

Thus, nodal performance in an Infinibend 100Gb cluster (12.5 GB/s bandwidth, or 25 GB/s for the bidirectional communications in FFT) with $N = 2,000$ (as in Figure 3.6) will be limited by

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

Similarly, nodal performance in 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 performacne will be bound by 104 Gflop/s, as also mentioned in Section 3.3. This means that further optimizations for the nodal FFTs will have limited effect on the overall performance, since 104 GFlop/s will be the maximum that can be extracted from the node (while currently we can achieve much more – 600 Gflop/s from just one GPU). Still, current FFT results on Summit show scalability of ≈ 9 Gflop/s per node, does leaving a potential for $10\times$ acceleration while still leaving GPU resources for other computations, as typically needed in applications [2].

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 4. Figure 5.1 shows another example for time needed for communication vs. time for computation. In this example NVLINK marks the time

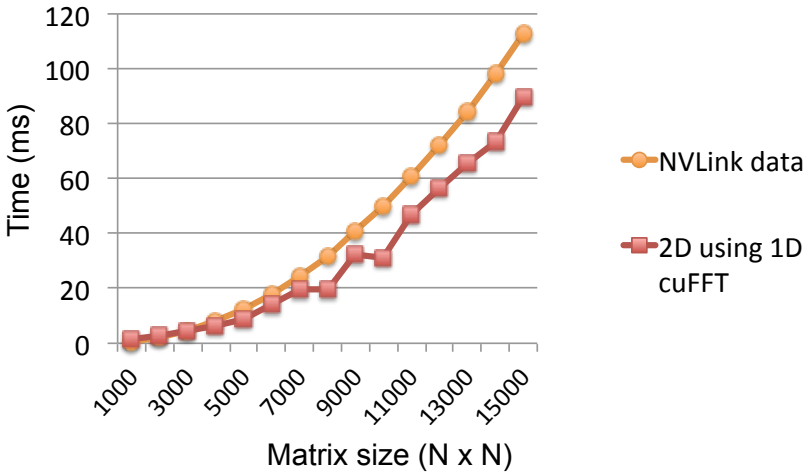

Figure 5.1: Time for 2D FFT using 1D cuFFTs on NVIDIA V100 GPU. The NVLINK shows the time to receive and send the data for the computation through a 32 GB/s connection. Computation and communication can be overlapped by pipelining the work on the 1D vectors, in which case the total computation time is given by the NVLINK curve, otherwise is the sum of the two curves (i.e., about twice slower in this case).

to receive and send the data for the local GPU computation through 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 ilustration.

5.3 Framework Design Highlight

The main components needed for the FFT-ECP design framework are illustrated in Figure 5.2. The first and last step address the need for flexible FFT API to take application specific input and output (bricks/pencils), including arbitrary initial decompositions. The approach that we will persue for this step is to start from the current `FFTMPI` and `SWFFT` implementations and to extend them by providing efficient GPU support for their main communication primitives. This includes loading from and storing

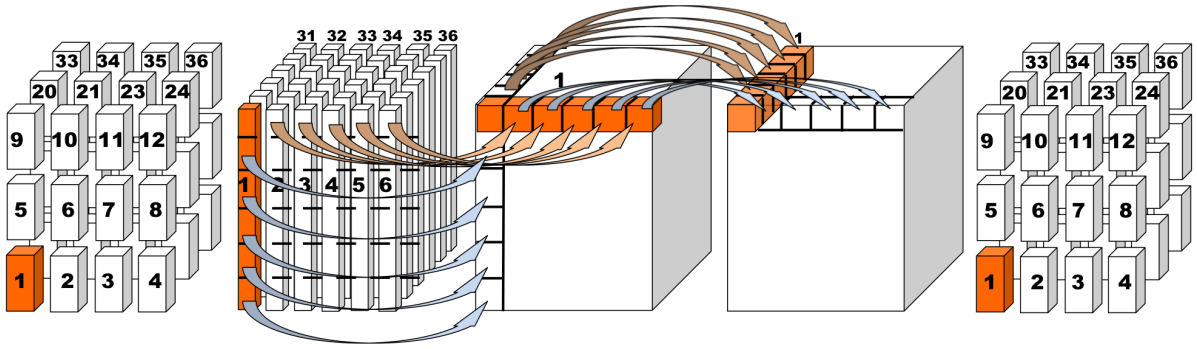

Figure 5.2: An overall 3D FFT computational pipeline: 1) Need flexible FFT API to take application specific input and output (bricks/pencils/etc., shown on the left and on right); 2) Need efficient packing/unpacking (on a node) and MPI communication routines (shown in the middle); 3) Need efficient 1D (or 2D in some cases) FFTs on the node (shown in the middle).

to arbitrary tiling of a 3D domain, data transformations from brick-to-pencil, pencil-to-brick, and pencil-to-pencil. Local packing and unpacking kernels would be accelerated leveraging GPUs' high bendwidth and efficient GPU transposition kernels that are already available in MAGMA [20].

Most of the effort for the next phase need to be focused on optimizing the data movement and try to overlap the data copy with the data communication, which we believe will be the bottleneck of any distributed memory FFT library, as illustrated in our performance analysis and model. While computation will be much smaller fraction of the entire execution and we can do it very efficiently on GPUs, as illustrated, we will explore ways to pipeline communications and computations in order to largely overlap all computations with the communications.

Finally, providing all versions along with their parameterizations and different optimization techniques, as highlighted throughout the report, will inevitably create a tuning challenge. We have extensive expertize and well proven track record in the development and use of autotuning techniques for important GPU kernels [1, 10, 12]. The FFT-ECP software will be linked to our autotuning tools, which combined with our kernels designs and use of various state-of-the-art building blocks will provide performance portability, software interoperability, and sustainability.

CHAPTER 6

Conclusions

In this milestone, we developed benchmarks and software tools to assess current FFT capabilities and gaps, needed by FFT users in many FFT application areas outside ECP. We also evaluated the current performance of FFT libraries and proposed a design framework for the FFT-ECP project. Specifically, this milestone delivered on the following sub-tasks:

- Evaluation and benchmarking of current/existing FFT libraries from open-source developers and vendors;
- Evaluation and benchmarking of the FFT code used in other ECP applications, including: LAMMPS and HACC;
- Study the interoperability between current vendor FFT libraries and the existing FFT library used in ECP applications, particularly for use in heterogeneous nodes with many accelerators;
- Propose a framework design for FFT-ECP and investigation for possible integration and/or use of vendor- developed or open-source FFT codes with our 2-D and 3-D FFT-ECP framework that emphasizes multi-GPU nodes;
- Analysis of the communication/computation cost and memory overhead for different FFT variants and provide a study of the behavior on current and future architectures with distributed and heterogeneous multi-GPU nodes.

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

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

[4] Amir Gholami, Judith Hill, Dhairya Malhotra, and George Biros. Accfft: A library for distributed-memory FFT on CPU and GPU architectures. *CoRR*, abs/1506.07933, 2015. URL http://arxiv.org/abs/1506.07933.

[5] Jens Henrik Göbbert, Hristo Iliev, Cedrick Ansorge, and Heinz Pitsch. Overlapping of communication and computation in nb3dfft for 3d fast fourier transformations. In Edoardo Di Napoli, Marc-André Hermanns, Hristo Iliev, Andreas Lintermann, and Alexander Peyser, editors, *High-Performance Scientific Computing*, pages 151–159, Cham, 2017. Springer International Publishing. ISBN 978-3-319-53862-4.

[6] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, and Katrin Heitmann. Hacc: Extreme scaling and performance across diverse architectures. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 6:1–6:10, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2378-9. doi: 10.1145/2503210.2504566. URL http://doi.acm.org/10.1145/2503210.2504566.

[7] 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.

[8] 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/.

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

[10] Jakub Kurzak, Stanimire Tomov, and Jack Dongarra. Autotuning GEMM kernels for the Fermi GPU. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2045–2057, November 2012.

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

[12] Y. Li, J. Dongarra, and S. Tomov. A note on auto-tuning GEMM for GPUs. In *Proceedings of the 2009 International Conference on Computational Science, ICCS'09*, Baton Roube, LA, May 25-27 2009. Springer.

[13] Lucien Ng, Kwai Wong, Azzam Haidar, Stanimire Tomov, and Jack Dongarra. Magmadnn ff high-performance data analytics for manycore gpus and cpus, December 2017. URL http://icl.cs.utk.edu/magma/software/. Magma-DNN, 2017 Summer Research Experiences for Undergraduate (REU), Knoxville, TN.

[14] CUDA Nvidia. Cufft library, 2018.

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

[16] 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.

[17] Thom Popovici, Tze-Meng Low, and Franz Franchetti. Large bandwidth-efficient FFTs on multicore and multi-socket systems. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018.

[18] 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.

[19] Daisuke Takahashi. Ffte: A fast fourier transform package. *http://www. ffte. jp/*, 2005.

[20] 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.

[21] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra. Dense linear algebra solvers for multicore with GPU accelerators. In *Proc. of the IEEE IPDPS'10*, pages 1–8, Atlanta, GA, April 19-23 2010. IEEE Computer Society. DOI: 10.1109/IPDPSW.2010.5470941.