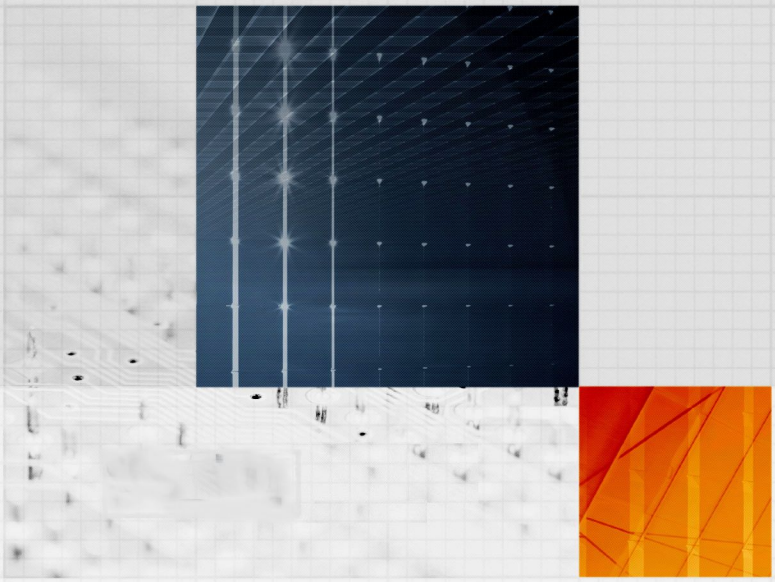

Interim Report on Benchmarking FFT Libraries on High Performance Systems

Alan Ayala
Stanimire Tomov
Piotr Luszczek
Sébastien Cayrols
Gerald Ragghianti
Jack Dongarra

ICL Technical Report ICL-UT-21-03

Knoxville, July 28, 2021

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Abstract

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. FFTs are also heavily used in ECP applications, such as EXAALT, Copa, ExaSky-HACC, ExaWind, WarpX, and many others. As these applications' accuracy and speed depend on the performance of the FFTs, we designed an FFT benchmark to measure performance and scalability of currently available FFT packages and present the results from a pre-Exascale platform. Our benchmarking also stresses the overall capacity of system interconnect; thus, it may be considered as an indicator of the bisection bandwidth, communication contention noise, and the software overheads in MPI collectives that are of interest to many other ECP applications and libraries.

This FFT benchmarking project aims to show the strengths and weaknesses of multiple FFT libraries and to indicate what can be done to improve their performance. In particular, we believe that the benchmarking results could help design and implement a fast and robust FFT library for 2D and 3D inputs, while targeting large-scale heterogeneous systems with multicore processors and hardware accelerators that are a co-designed in tandem with ECP applications. Our work involves studying and analyzing state-of-the-art FFT software both from vendors and available as open-source codes to better understand their performance.

Contents

List of Figures

List of Tables

Acknowledgment

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 tested platforms, in support of the nation's exascale computing imperative.

Chapter 1

Background

The Fast Fourier Transform (FFT) is considered one of the top 10 algorithms of the 20th century [1] and plays a key role within applications in a variety of fields ranging from electronics to molecular dynamics. In essence, the FFT of x, an m-dimensional vector of size $N \equiv N_1 \times N_2 \times \cdots \times N_m$, is denoted $y = FFT(x)$ and defined as an m-dimensional vector the same size as x by the following equations:

$$y(k_1, k_2, \ldots, k_m) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)} \tag{1.1}$$

for $0 \leq k_i \leq N_i - 1,\ i = 1, \ldots, m$, where $\tilde{x} = x(n_1, \ldots, n_m)$.

From Eq. 1.1, we see that the FFT could be directly computed by a tensor product; however, this would cost $O(N \sum_{i=1}^{m} N_i)$. The advantage of the FFT is that the cost can be reduced to $O(N \log_2 N)$ operations by exploiting the structure of the tensor.

The parallel FFT is implemented by a sequence of 1D or 2D FFTs [2], which are computed using efficient numerical libraries that are optimized for intra-node use. These include open source FFTW [3] and vendor-supplied cuFFT [4], rocFFT [5], MKL [6], etc. Fig. 1.1 shows the simplified steps required to perform a 3D FFT that is typically used in molecular dynamics applications [7, 8]. For some applications, the input data has a shape that is ready to perform one-dimensional FFTs on *pencils* or two-dimensional FFTs on *slabs*, and these do not require initial or final reshape operations. It was shown [9] that shaving one reshape step can reduce the runtime by about 25%, since asymptotically the multi-dimensional FFT's runtime is dominated by the number of data reshape operations.

Communication bottlenecks in parallel algorithms—including in FFT—have been worsening in the past decade, since the number of hardware threads in current and upcoming multicore processors continues to grow dramatically with improvements in bandwidth and

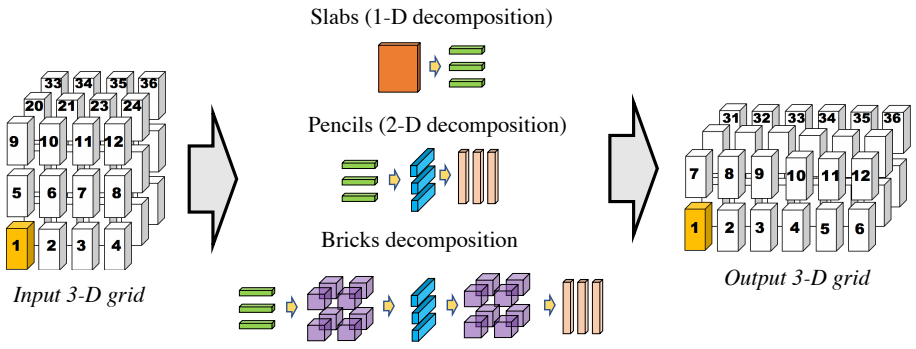

Figure 1.1: The sequence of steps for computing the 3D FFT for different decomposition of the input tensor. The *pencil* decomposition (green bars throughout) is the default option in most libraries. Using *slabs* (red box at the top) saves an extra step of data reshape. The *bricks* decomposition (purple translucent cubes) is supported by some libraries and could be beneficial on some platforms.

reduced latency [10]. For instance, the Summit supercomputer uses powerful nodes with two IBM POWER9 processors and six NVIDIA Volta V100 GPUs, which are capable of reaching 42 Tflop/s cumulatively in double precision. However, the interconnect between the nodes is supported by a bandwidth of only 25 GB/s: a ratio of 2000 operations per byte transferred. Another supercomputer from the top of the TOP500 list, the Sunway TaihuLight, has SW26010 processors with 260 cores, and 1 execution thread per core, with a unidirectional bandwidth of 8 GB/s between nodes and 1 microsecond of latency [11].

Consequently, the increasing gap between compute and communication capabilities in current supercomputers makes many algorithms communication-bound. When this happens, algorithm performance drastically decreases compared to the machine's peak capabilities. Therefore, it is critical to develop algorithms capable of dealing with such a drastic communication-computation imbalance, as well as create an ecosystem of integrated tuning techniques for improved communication. Such approaches are crucial in general and are paramount for the FFTs, where communication could take more than 95% of their total running time on GPU-accelerated machines [9, 12].

Upcoming Exascale systems are expected to continue with hybrid CPU-GPU design with increased stress on GPU usage. A major issue with parallel FFT implementations for such systems is that they quickly become communication-bound due to the excess of compute capacity concentrated in the GPU hardware accelerators. Indeed, theoretical analysis of hybrid Exascale systems shows that the FFT computation itself would take only a small fraction of the total runtime, while the communication between the distributed nodes would be the main bottleneck where most of the runtime is spent [13].

Several authors reported the impact of multi-process communication on distributed FFT performance [12, 14, 15, 16] using both binary and collective Message Passing Interface (MPI) communication schemes that are used in majority of modern numerical libraries. At the im-

Table 1.1: Single-device and shared-memory FFT libraries.

Library Name	Programming Language	License	Developer	GPU Support	2D&3D Support	Strided data
cuFFT	C	NVIDIA ®	NVIDIA	yes	yes	yes
ESSL	C++	IBM ®	IBM	no	yes	yes
FFTE	Fortran	Permissive	U. Tsukuba/RIKEN	yes	yes	yes
FFTPACK	Fortran	BSD-3-Clause	NCAR	no	no	no
FFTS	C	MIT	U. Waikato	no	no	no
FFTW	C	GPL-2.0	MIT	no	yes	yes
FFTX	C	BSD-3-Clause	LBNL/Sandia	yes	yes	yes
KFR	C++	GPL-2.0	KFR	no	no	yes
KISS	C++	BSD-3-Clause	Sandia	no	yes	yes
oneMKL	C	Intel® SSL	Intel	yes	yes	yes
rocM	C++	MIT	AMD	yes	yes	yes
VkFFT	C++	MPL-2.0	D. Tolmachev	yes	yes	yes

plementation level, current efforts among the hybrid CPU-GPU FFT libraries rely on default routines like CUDA-aware MPI, and benchmarks of these libraries have shown far-from-peak performance along with the topology issues that hinder scalability [17]. Specialized MPI implementations for accelerators—like the NCCL library from NVIDIA that currently provides some collective routines [18]—are still at the early stages and continue to evolve. This motivated the development of specialized communication frameworks and custom routines for faster data exchange for parallel FFTs [12, 19, 20].

1.1 Single-device and shared-memory FFT libraries

Within a single node, efficient FFT implementations are available to compute multidimensional FFTs. One of the most widely used libraries is FFTW [3], which has been tuned to perform well on a number of CPU architectures (no GPU support). Vendor libraries for this purpose have also been highly optimized, such is the case with MKL (Intel) [6] and ESSL (IBM) [21] for CPUs, and rocFFT (AMD) [5] and cuFFT (NVIDIA) [4] for GPUs. Recent alternatives include Intel's FFT kernels within the OneAPI library, Vulkan FFT (VkFFT) and KFR. Within the ECP community, there is an effort for the optimization of single-node FFT computations, using FFTX [22] and Spiral [23]. Most of the previous Single-device libraries have been extended to distributed memory versions—some by the original developers and others by different authors.

Tuning has been successfully performed for single-core and shared-memory implementations (e.g., FFTW [3] allows the `FFT_MEASURE_FLAG` option to be passed to the plan creation function to select the best available configuration of parameters and resources). Similarly, the vendor libraries mentioned above also offer tuning tailored to their specific systems. More recent developments are looking for intra-node optimization, such is the case for the FFTX Exascale project [22] and Spiral [23]. Table 1.1 shows the main single-device FFT backends from vendors and open-source. Single-device libraries are very important for paral-

lel FFTs, since the latter rely on the former for the computation of the local low-dimensional FFTs: either 1D or 2D.

1.2 Distributed FFT libraries

In Table 1.2 we present a list of currently available state-of-the-art distributed FFT libraries and their main features.

1.2.1 CPU-based libraries

Many 3-D parallel libraries started as extensions of the CPU-based single-node libraries listed in the previous section. For instance, FFTW is a CPU-only library that supports MPI via slab decomposition; however, it has limited scalability and is limited to a small number of nodes, given the small size of the 3-D FFT. Below, we briefly describe some of these libraries.

Table 1.2: Feature comparison of state-of-the-art distributed FFT libraries.

Library Name	Programming Language	License	Developer	CPU Backend	GPU Backend	Real-to Complex	Layout Slab/Brick
AccFFT	C++	GPL-2.0	GA Tech.	FFTW	cuFFT	yes	no / no
2DECOMP &FFT	Fortran	NAG-2011	NAG	FFTW ESSL		yes	yes/no
Cluster FFT	Fortran	Intel® SSL	Intel	MKL	oneMKL	no	no / no
CRAFFT	Fortran	check with developer	Cray	FFTW ACML SPIRAL		yes	no / no
FFTE	Fortran	Permissive	U. Tsukuba RIKEN	FFTE	cuFFT	yes	yes/no
FFTMPI	C++	BSD-3-Clause	Sandia	FFTW KISS MKL		no	no / yes
FFTW	C	GPL-2.0	MIT	FFTW		yes	no / no
heFFTe	C++	BSD-3-Clause	UTK	FFTW MKL	cuFFT rocM oneMKL	yes	yes / yes
nb3dFFT	Fortran	GPL-2.0	RTWH Aachen	FFTW ESSL		yes	no/no
P3DFFT++	C++	BSD-3-Clause	UCSD	FFTW ESSL		yes	no/no
SpFFT	C++	BSD-3-Clause	ETH Zürich	FFTW	cuFFT rocM	yes	yes/no
SWFFT	C++	BSD-3-Clause	Argonne Natl. Lab	FFTW		no	no/yes

2Decomp&FFT

2Decomp&FFTis written in Fortran by N. Li and S. Laizet [24] in the context of the Open Petascale Libraries. The execution relies on a 2D pencil decomposition of the data, and then makes a heavy use of MPI_Alltoall(v) routines. 2Decomp&FFTsupports different backends such as ACML, FFTE, FFTW, and MKL.

The latest version of 2Decomp&FFT, 1.5, was release in 2012. Minor modifications are required in order to be able to compile it on today's systems. The code offers an auto-tuning routine that aims to find the most efficient processor grid. In its last release, authors mentioned a prototype development of a GPU version by performing single-device FFTs via cuFFT.

FFTMPI

This library was introduced in [8] for efficiently computing three dimensional FFTs, encountered in the simulation of long-range interactions within the LAMMPS [7] library, and currently within the EXALLT-ECP project.

FFTW

FFTW was one of the first distributed MPI implementations of parallel FFT computations [3]. Authors performed this using slab decompositions. This, however, limits its scalability to a small number of nodes.

nb3dFFT

This library was introduced by Göbbert et al. [25]. The latest version is from 2015 and only supports real-to-complex transforms. These capabilities make it a good candidate for applications in digital signals and pattern recognition.

P3DFFT Family of Libraries

Parallel Three-Dimensional Fast Fourier Transforms, or P3DFFT [26], implement three dimensional variants of FFT. The libraries, written mainly by Dmitry Pekurovsky at the San Diego Supercomputer Center (SDSC) at the University of California San Diego (UCSD), use 2D, or pencil, decomposition [27] which overcomes an important limitation to scalability inherent in the FFT implementations that use 1D, also known as slab, decomposition. For this kind of decomposition, the number of individual cores that run P3DFFT can be as large as n^2, where n is the linear problem size. The authors claim this decomposition approach can scale up to $2^{19} = 524,288$ cores.

P3DFFT is written in Fortran 90 and optimized primarily for parallel FFT performance. It uses MPI for inter-processor communication in a distributed memory setting. P3DFFT version 2.7.5 and above allow the user to enable multithreading as an option in order to leverage the CPU-GPU model of parallelism with the combination of MPI and OpenMP. For cross-language interoperability, P3DFFT features a C/C++ interface with examples in both Fortran and C available with the source code distribution. An Autoconf-generated configuration script provides the installation options for the user. The P3DFFT package requires a single-device FFT library, and the code includes API calls to FFTW and IBM's ESSL.

P3DFFT++ is the next generation implementation of P3DFFT, and its versioning starts with 3.0. This new library extends the interface of the original P3DFFT to allow a wider range of usage scenarios. New choices for defining custom data layouts beyond the originally predefined 2D pencil blocks are provided to the user. As the name suggests, P3DFFT++ is written in C++ and it includes both C and Fortran interface bindings. For distributed memory parallelism it uses MPI. Table 1.3 compares the functionality of P3DFFT version 2.7.6 with P3DFFT++ version 3.1.1. One important difference between these two versions is the lack of complex-to-complex transform in the early P3DFFT versions. Therefore, in the experiments for complex transforms presented in §2.4, we only use P3DFFT++. In order to support non-power-of-two number of MPI ranks, we used the Git commit `2f2d70d` that fixes a limitation in the released version 3.1.1 of P3DFFT++.

Table 1.3: Comparison of functionality of P3DFFT v.2.7.6 with P3DFFT++ v.3.1.1.

Feature	P3DFFT 2.x	P3DFFT++
Support for real-to-complex and complex-to-real FFT	Yes	Yes
Support for complex-to-complex FFT	No	Yes
Sine and cosine transforms	1D only	Yes
Pruned transforms	Yes	No
In-place and out-of-place	Yes	Yes
Multiple grids	No	Yes
Hybrid MPI/OpenMP	Yes	No

SWFFT

This library was first introduced [28] as part of the cosmology project HACC [29]. SWFFT aims to provide efficient and scalable parallel FFTs for inputs distributed between MPI ranks on a 3D Cartesian communicator, and it uses the pencil-decomposition approach.

SWFFT does not work for arbitrary grid sizes and number of MPI processors, however. Still, SWFFT is one of the very few libraries that supports brick decomposition, see Figure 1.1.

1.2.2 CPU-GPU libraries

There are only a few CPU-GPU FFT implementations today. The developments are generally based on efforts to minimize the communication impact. For instance, the slab approach [30] is one of the first heterogeneous codes for large FFT computation on GPUs. Its optimizations and tuning techniques focus on reducing tensor transposition cost by exploiting the Infiniband interconnection using the IBverbs library, which limits its portability to Infiniband-based systems. Further improvements to scalability have been presented in the FFTE library [31], which supports pencil decomposition and includes several optimizations, although with limited features and limited communication improvements. FFTE relies on the commercial PGI compiler, which may also limit its usage.

AccFFT

Introduced in [14], this library was developed in an effort to overlap computation and collective communication by reducing the Peripheral Component Interconnect Express (PCIe) overhead, getting considerable speedups and good scalability for large, real-to-complex transforms using NVIDIA K20 GPUs.

CRAFFT

The CRay Adaptive FFT (CRAFFT) [32] is a proprietary interface to FFT functionality in Cray Scientific Libraries suite available since at least 2008 as part of Cray's Programming Environment on the supported hardware starting with the XT and continuing on the XC series of supercomputing machines [33]. This library supports serial and parallel, single-and double-precision, Fortran and C routines that compute the discrete Fourier transform in one, two, or three dimensions. Similarly to heFFTe, CRAFFT does support many backends and allows automatic and dynamic selection of the fastest FFT kernel.

FFTE

The Fastest Fourier Transform from the East (FFTE) was first introduced in [31], and it is one of the very few CPU-GPU libraries under continuous development [16]. Its latest version, 7.0, includes a wide range of optimizations and provides GPU support only via PGI compiler.

One issue with FFTE is its lack of support for arbitrary grid sizes and number of MPI processors. This restriction requires that the number of MPI processes evenly divide each dimension of the computation grids.

HeFFTe

Among all libraries studied in this report, heFFTe supports the most single-device backends and enables parallel FFT computations on AMD, Intel, and NVIDIA GPUs. The heFFTe library was first introduced in recent years [34, 35, 9] and is an open-source implementation that is a part of the Exascale Computing Project. The heFFTe library aims to be highly scalable and has shown linear scaling for large node-counts [9]. As shown in Table 1.2, heFFTe includes several features missing by other state-of-the-art libraries.

Chapter 2

Experimental Results

In this chapter, we summarize the results obtained using the testing programs supplied by the eight parallel FFT libraries described in the previous chapter.

2.1 Benchmark setup

Table 2.1 shows the versions and the testing programs supplied by the libraries used for the experiments in this report. For each of these libraries, we used the parallel FFT testing code on either CPUs, GPUs, or both as they were provided in the released version or from the repository for the respective libraries. This testing strategy uses library-specific testers, which are likely to favorably expose performance benefits of the particular implementation. At the same time, it creates a potential for unfair comparison due to the different timing regimes of varying approaches to providing unified time measurement across nodes, such as taking average, median, or maximum of timer readings between parallel nodes. Separately built executables for testing may link-in slightly different dependent modules despite our best efforts to limit the build environment to the same minimum set of available settings shown in Table 2.2. The estimation of computed error would likely be different across the tester codes resulting in inconsistent error reporting and correctness validation. We foresee addressing some of these issues in our planned development of a unified testing harness as part of the future work detailed in Chapter 4.

The performance experiments were performed using the Summit supercomputer at Oak Ridge National Laboratory. This is a hybrid supercomputer, featuring both CPUs and GPUs, and was ranked number two on the TOP500 lists since June 2020 until of June 2021 as of this writing. The machine is ideal to achieve our FFT benchmarking goals, and we follow with the description of the experimental setup.

Table 2.1: FFT Libraries for benchmarking tests. All libraries were tested using their released versions with the exception of P3DFFT++ that used Git commit 2f2d70d to fix issues related to the non-power-of-2 MPI process counts.

Library Name	Version	CPU Test	GPU Test
AccFFT	2.0	no	yes
2Decomp&FFT	1.5.847	yes	no
FFTE	7.0	no	yes
FFTW	3.3.8	yes	no
FFTMPI	1.0	yes	no
heFFTe	2.0	yes	yes
SWFFT	1.0	yes	no
P3DFFT	2.7.9	yes	no
P3DFFT++	2f2d70d	yes	no

Experiments with the Cray/HPE CRAFFT and Intel Cluster FFT libraries were not performed for this report since their portability is limited to specific hardware platforms with vendor-specific software stacks. The former (Cray Adaptive FFT routines) is included with the Cray/HPE LibSci library of scientific software for their XT, XE, and XK systems. The latter is included with the Intel oneAPI Math Kernel Library and is available only for the Intel 64 and Intel Many Integrated Core architectures. In the case of SpFFT, we do not provide experiments since this library targets sparse data, and it would not be possible to compare its performance with respect to the other libraries. For building the libraries from Table 2.1, we used the compiler and dependencies described in Table 2.2.

Table 2.2: Software versions used for the experiments in this report.

Software Module	Version Used in Tests
CUDA	10.1.243
FFTW	3.3.8
GNU compilers	6.4.0
Spectrum MPI	10.3.1.2-20200121
CMake	3.11.3

2.2 Computational resources

Our experiments were performed using up to 1,024 Summit nodes, out of a total of 4,608. As shown in Figure 2.1, each node consists of two sockets, each composed of a 22-core IBM POWER9 CPU and 3 NVIDIA Volta V100 GPUs. The 6 GPU accelerators provide a theoretical double-precision capability of approximately 46.8 TFlop/s.

Within the same socket, computing units have access to NVIDIA NVLink interconnects with a theoretical bandwidth of 50 GB/s (100 GB/s bi-directional). Inter-node connection is limited to 25 GB/s (in each direction).

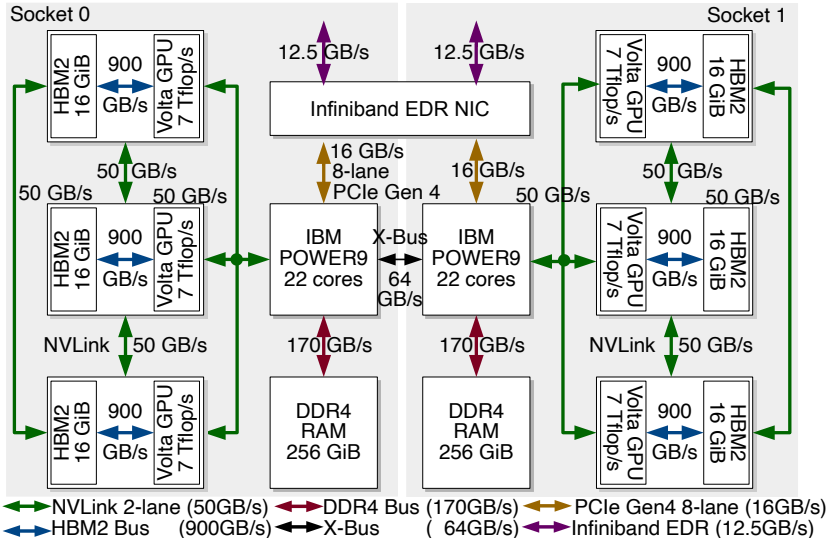

Figure 2.1: Architecture of Summit nodes: computing units and network connections.

2.3 Experimental setup

We perform a strong scalability experiment to benchmark the eight libraries described in Table 2.1. For this, we consider:

- A 3-D transform of size 1024^3, using double-precision complex random input.
- Unless otherwise specified, the input and output are in pencil shape. Schematically, this means the input and output are those enclosed with the dashed rectangle in Figure 2.2.

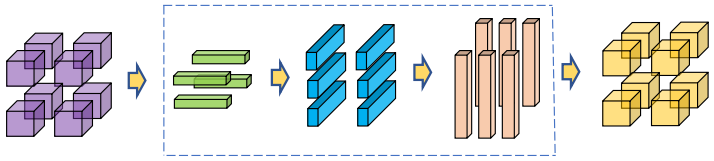

Figure 2.2: Phases for the computation of a 3-D FFT with pencil decomposition. From brick-shaped input to brick-shaped output.

- For all libraries we use the pencil decomposition, with the exception of FFTW, which internally uses the slab decomposition, see Figure 1.1.
- We report the average of ten consecutive executions of forward and backward transforms.
- We employ the test drivers provided by the library developers, see Table 2.2, with the exception of FFTW, for which we implemented a tester. If tuning is set by default, we disable this option for a fair comparison among all libraries.
- If no time is shown for a given node-count, it means the library had a runtime issue, which is related to insufficient memory to handle the input data in most cases.
- For CPU-based libraries, we use 40 MPI processes per node, 1 per IBM POWER9 core, 20 per socket, see Figure 2.1. Since FFTE and SWFFT only allow a number of processes divisible by the FFT size, we also add an experiment using 32 MPI processes per node.
- For GPU-based libraries, we use 6 MPI processes per node, 1 per NVIDIA V100 GPU, 3 per socket, see Figure 2.1. Since FFTE only allows a number of processes divisible by the FFT size, then we also add an experiment using 4 MPI processes per node, 2 per socket.

2.4 CPU-based libraries

For our first experiment, we compare six CPU libraries going from pencil-shaped input to pencil-shaped output. Figure 2.3 shows the strong scalability results using 40 MPI processes per node.

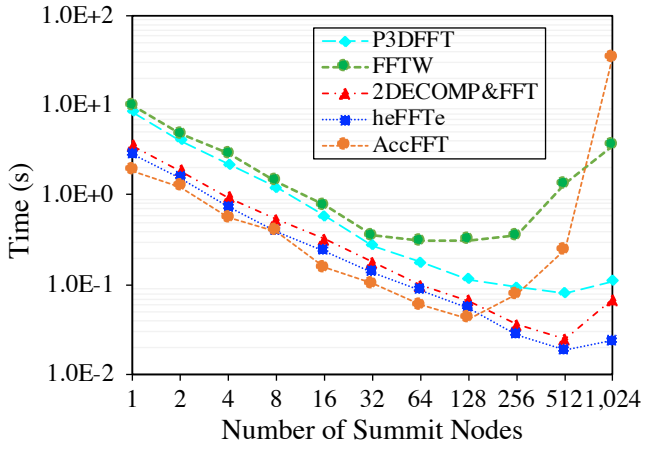

Figure 2.3: Strong scalability for a 3-D FFT of size 1024^3 using 40 MPI processes per node, 1 per IBM POWER9 core, 20 per socket. Using 2 reshapes (transpositions) per FFT direction.

Next in Figure 2.4, we compare heFFTe and FFTMPI using 4 reshapes, see Figure 2.2. These libraries, and SWFFT, are the only ones that have this option enabled by default on their test drivers. Refer to Figure 2.6 for a comparison of these three libraries.

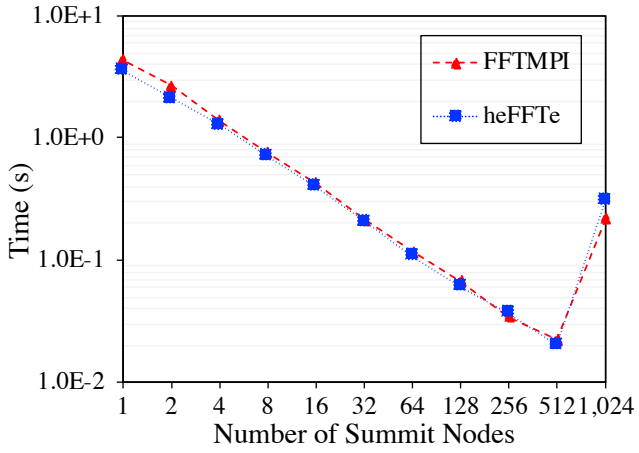

Figure 2.4: Strong scalability for a 3-D FFT of size 1024^3 using 40 MPI processes per node, 1 per IBM POWER9 core, 20 per socket. Using 4 reshapes (transpositions) per FFT direction.

Figures 2.3 and 2.4 show that some libraries halt linear scaling when using a large number of cores. This may be due to implementation issues or communication management.

Next, Figures 2.5 and 2.6 show results using only 32 cores by node, since libraries such as FFTE and SWFFT are size-constrained, meaning that they only work for certain pairs of FFT-size and number of processes.

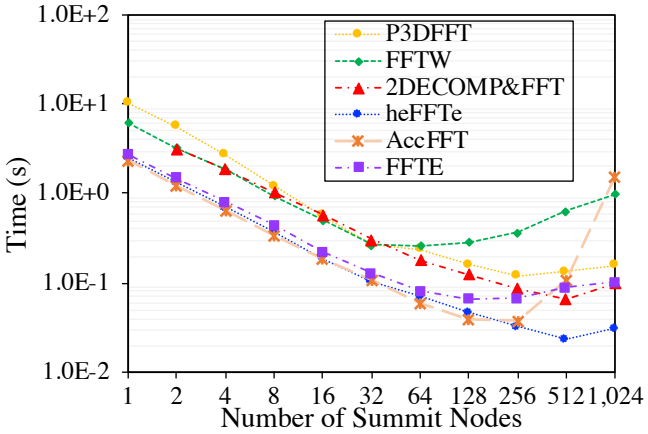

Figure 2.5: Strong scalability for a 3-D FFT of size 1024^3 using 32 MPI processes per node, 1 per IBM POWER9 core, 16 per socket. Using 2 reshapes (transpositions) per FFT direction.

In Figure 2.6, we use brick-shaped input and output for analyzing FFTMPI, heFFTe, and SWFFT libraries, using 4 tensor transpositions per FFT, see Figure 2.2.

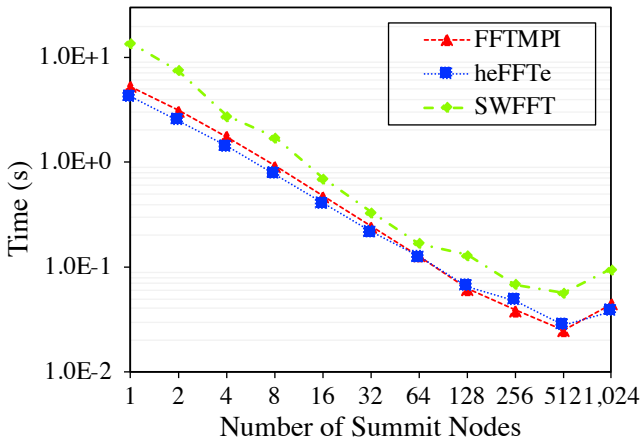

Figure 2.6: Strong scalability for a 3-D FFT of size 1024^3 using 32 MPI processes per node, 1 per IBM POWER9 core, 16 per socket. Using 4 reshapes (transpositions) per FFT direction.

2.5 GPU-based libraries

AccFFT, FFTE, and heFFTe are among the very few libraries that have GPU support. In Figures 2.7 and 2.8, we show a comparison between these three, using cuFFT as their 1-D backend. For heFFTe, we show two communication options: Point-to-Point (p2p) and All-to-All (a2a). We observe that tuning communication can have a considerable impact on performance. In Figure 2.7, we used only 4 out of the 6 available GPUs per node, due to FFTE restrictions. Some of the libraries were unable to perform the transform at the lower range of node counts due their required working buffer allocation and the limited size of the GPUs' main memory.

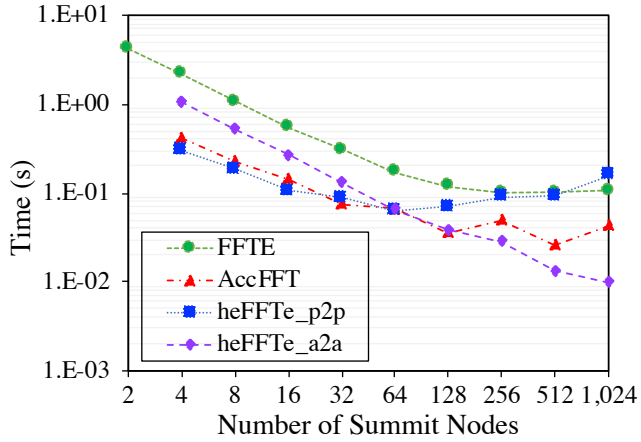

Figure 2.7: Strong scalability for GPU libraries using 4 NVIDIA V100 GPUs per node, 2 per socket.

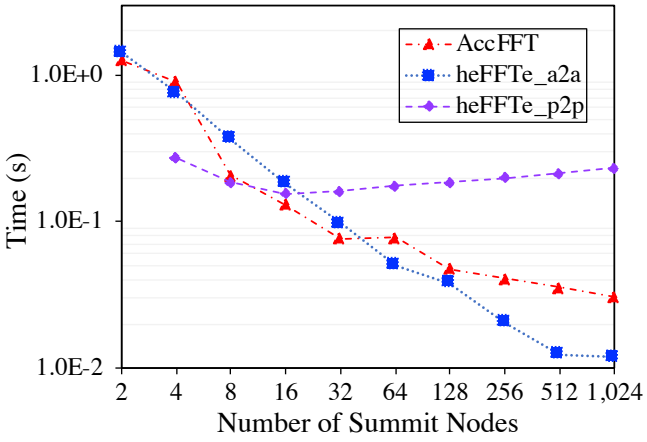

Figure 2.8: Strong scalability for GPU libraries using 6 NVIDIA V100 GPUs per node, 3 per socket.

Chapter 3

Profiling and Tuning Considerations

3.1 Tuning FFT Libraries for Improved Performance

Most of the FFT libraries tested in this report are configurable. They accept a number of parameters that change their behavior to better fit the application needs and adapt to the underlying platform, including the hardware as well as the software stack. As an example use of this interface, we show the effect of the two-dimensional process grid shape $p_x \times p_y$ on the performance of the P3DFFT++ library in Figure 3.1. For easier analysis, we divided the figure into the top part with the number of rows in the MPI process grid that are a power-of-two: $p_x = 2^\ell$ for $\ell \in \{1, 2, \ldots, 13\}$; and the bottom part with non-power-of-2 row counts: $p_x = 5 \times 2^\ell$ for $\ell \in \{1, 2, \ldots, 13\}$, respectively. The figure clearly indicates the need to select the appropriate process grid to ensure scalability. Specifically, the grids that do not scale are either short-and-wide or tall-and-narrow. At the same time, the grid shapes that scale well resemble a square to the extent possible according to the prime factors of the total number of the MPI ranks: $p_x/p_y \approx 1$. This informed our experimental runs, which attempted to use the optimal settings of each library to the best of our knowledge.

3.2 Communication bottleneck

All distributed libraries discussed in Chapter 1 have to deal with the communication-bound nature of the FFT parallel algorithm. To efficiently tackle the communication cost when scaling multidimensional FFT operations, sophisticated technologies were brought forward from different domains to optimize the costly all-to-all communication. Leveraging specific characteristics of network offloading is a non-blocking all-to-all scheme [27]. As a response to the increase in system hierarchy and complexity, [36] proposed a hierarchical all-to-all

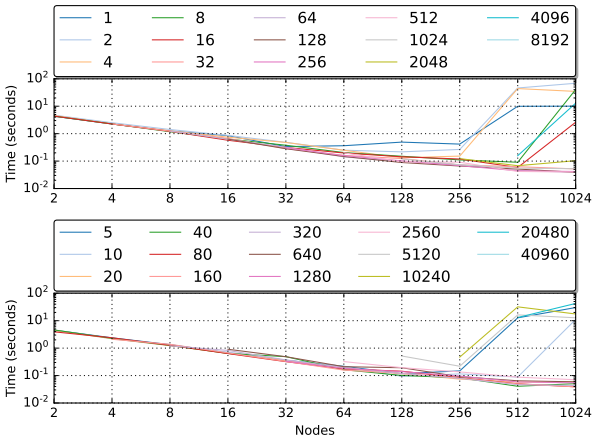

Figure 3.1: Timing of 3D FFT from P3DFFT++ for different number of Summit nodes with different shapes of two-dimensional process grids: process rows as power of 2: 2^{ℓ} of row MPI ranks (top) and power of 2 multiple of 5: $5 \times 2^{\ell}$ of row MPI ranks (bottom).

approach composed of multiple local Gather and Scatter operations. In a similar context, event-driven techniques were utilized to implement a collective framework to morph collective schemes among hierarchical, heterogeneous systems [37].

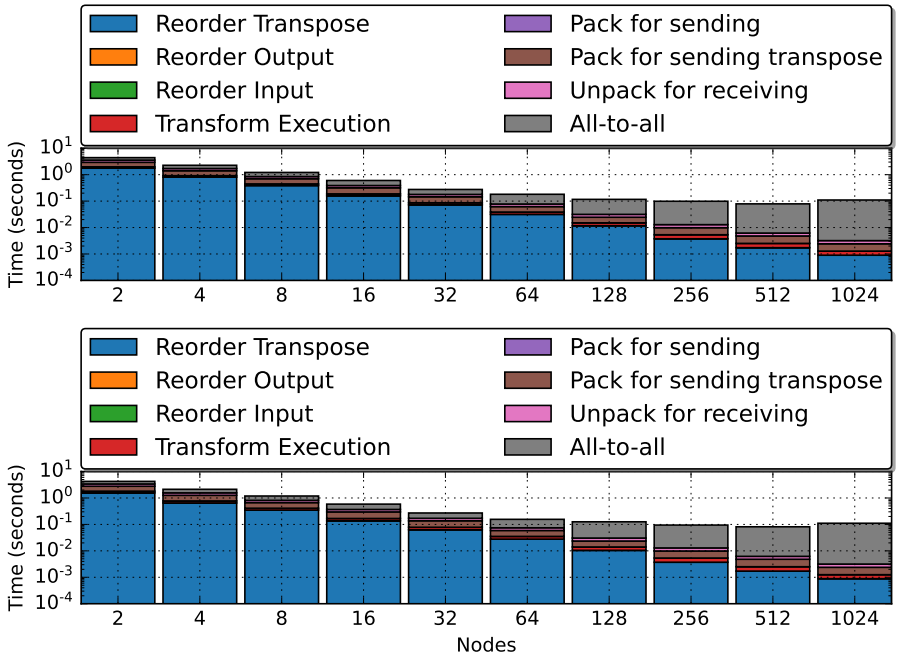

Figure 3.2: Breakdown of running time of 3D FFT from P3DFFT++ for different number of Summit nodes for in-situ (top) and ex-situ (bottom) transform.

On the other hand, to address the increased availability of accelerator units, collective operations specifically designed for interconnection between GPUs (NVLink), like NCCL [18], and re-routing strategies [38] have emerged with promises to maximize bandwidth usage. A node-wise visible communication pattern was investigated to optimize All-Gather and All-Reduce [39] collective operations by leveraging notified communication within a shared window. In [20], an application-targeting parallel FFT algorithm with GPU support was developed, they employed a hierarchical communication framework leveraging the power of MPI and OpenMPI for fast data movement and showed specific optimizations on Summit.

We can understand the bottlenecks by experimentally tracing the specific library kernels during the FFT execution and identifying the most time consuming portion. Unsurprisingly, the libraries described in this report spend an increasing portion of their runtime in data communication as the number of computing elements increases. In Figure 3.2, we present the execution profile of running the P3DFFT++ library on Summit for in-place (top) and out-of-place (bottom) transforms. This figure, clearly shows how communication time increasingly dominates when increasing the node-count.

3.3 The Effects of MPI Communication Bottlenecks

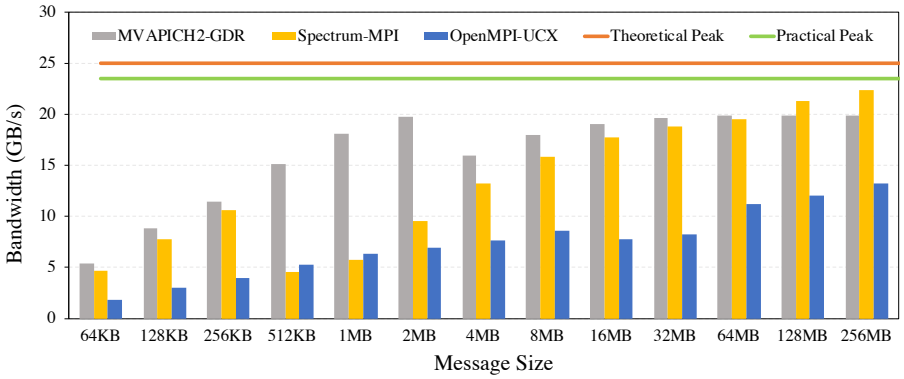

Figure 3.3: Comparison of achievable bandwidth from two-node exchange via MPI_Send, using of MVAPICH, SpectrumMPI and OpenMPI-UCX on Summit.

In order to substantively mitigate the communication bottlenecks observed in Section 3.2, we performed additional experiments that show that further tuning of MPI parameters and topology settings can further help to get faster completion of the communication tasks, especially at the small scale runs and limited number of resources.

For instance, in Figure 3.3, we show a comparison of the bandwidth available through a number of MPI libraries on Summit. Our experiments reveal that in practical setting one can achieve as much as 23.5 GB/s throughput for inter-node communication, out of the 25 GB/s theoretically available from the InfiniBand Dual Port EDR interconnect. We can also

see that by using MVAPICH we can get faster exchange rates than with Spectrum MPI for small message sizes.

In general, a combination of parameters' tuning, see Section 3.1, and network settings' tuning improves the performance of the FFT libraries studied in this report.

Chapter 4

Conclusions

In this report, we benchmark a number of parallel FFT libraries on CPUs and GPUs, and used them to analyze the performance of several FFT libraries on a large scale system with hybrid CPU-GPU hardware. We present a comparison of the experimental results from single-device and from distributed memory implementations that are available in the tested state-of-the-art FFT libraries. Furthermore, we evaluated performance for the strong scaling regime of the computations of the complex-to-complex 3D transform of size 1024^3 on up to 1,024 Summit nodes. These runs used either CPU-only setting with up to 40,960 IBM POWER9 physical cores or mixed CPU-GPU setting with up to 6,144 NVIDIA Volta V100 GPUs.

For this benchmarking report, we used the testing driver implementations for 3D complex transforms provided by the respective libraries with the exception of FFTW, for which we had to create a custom test driver to complete our set of results. These benchmarking test drivers allowed each library to choose its optimal settings, such as the best processor grids that were available given the number of computational units such as CPUs or GPUs. They also allow the library to optimally set the affinity given the arrangement of computing elements within the processor sockets, NUMA islands, and nodes. We used the average runtimes of ten consecutive executions which allowed us to use them for comparisons, performance analysis, and extraction of various conclusions. The following observations stand out:

Scaling on CPU-based vs. GPU-based architectures. The strong scaling efficiency[1] is about the same for CPU-based and GPU-based nodes using the same interconnect network, i.e., CPUs vs. GPUs on Summit. For the FFT benchmark on Summit the efficiency is about 30% for up to 512 nodes.

[1] We use the standard definition of efficiency as the fastest time across all libraries on one node over N times the best time for N nodes: $e = \frac{t_1}{N \times t_N} \times 100\%$.

Timing on CPU-based vs. GPU-based architectures. The GPUs' high compute performance and memory bandwidth tend to make the local FFT computations and data reshuffles insignificant (only about 5% or less of the total time) compared to the total execution time (95% or more time is spent in MPI communication). This makes the GPU runs on Summit about 2× faster than the CPU runs for the same number of nodes used.

Strong scaling limitations. For the $1,024^3$ FFT results on Summit, the strong scaling limitations begin to affect the timings for 1,024 node runs. All libraries cease to scale at this point with some experience scaling issues even earlier. For CPU-only runs, the best execution time gets down to about 0.02 seconds and, for GPU-only runs, to about 0.01 seconds.

FFT library features. Many of the multidimensional FFT libraries are specialized and developed *ad hoc* for particular applications. This makes it challenging to create a uniform and fair benchmark for all of them. For instance, it was necessary to increase the number of tests to cover all libraries. Nevertheless, not all node configurations were possible to test and there are some missing points in the scalability graphs due to runtime errors arising from algorithmic limitations. The heFFTe library tends to provide the most complete support for the benchmarked features, stemming partially from its design goals to recognize and implement as many as possible features needed for different ECP applications in a single library.

Ranking of the FFT libraries. The strong scaling behavior is similar for all libraries in general. For CPU-based runs, FFTW ceases to strong-scale the earliest past 32 nodes, followed by AccFFT (past 128 nodes; although performance-wise AccFFT has reached best timing among the other libraries by this point), followed by P3DFFT, 2Decomp&FFT, SWFFT, heFFTe, and FFTMPI. For GPU-based runs the scaling order is FFTE, AccFFT, and heFFTe. Performance-wise, for CPU-based runs there are two groups – the best performing ones are AccFFT, heFFTe, FFTMPI, FFTE, SWFFT, and 2Decomp&FFT. The second group is 2 to 3× slower with P3DFFT and FFTW. For GPU-based runs, the best time is achieved by heFFTe, followed closely by AccFFT, and FFTE is about 2× slower. In general, we find the performance differences to increase in the strong scaling limit (1,024 nodes).

Furthermore, we observed that there are tuning parameters that can significantly influence the scaling of the tested libraries. In particular, the shape of the two-dimensional decomposition of the MPI ranks in the distributed setting is a very important tuning parameter for performance and scalability. For example, scalability may be seriously harmed if the non-square process grids are used.

We have also presented a discussion on how the well-known communication bottleneck on distributed 3D-FFT codes affects scalability. Execution profile analysis shows that most of the state-of-the-art libraries have optimized local computations in such a way that the overall performance is now strongly linked to the performance of communication. For communication the libraries rely on routines such as MPI_Alltoallv, that may not be always well optimized to take advantage of the network architecture. We also evaluated the effect of tuning

the grid of processors and the effect of the MPI library selection, showing that performance can be improved.

From our study and experiments of over a dozen FFT libraries, we comprehended the diversity of parameters, their issues on scaling their performance, and potential improvements that can accelerate the FFT computation and, in consequence, benefit applications. For instance, some libraries such as FFTE and SWFFT, have limitations on the FFT size that they can manage. Others, like FFTMPI, do not support real-to-complex transforms. This highlights the critical need to have a harness software that can help compare libraries not only from the runtime perspective, but also from accuracy, energy consumption, and scalability perspectives for different data structures and precisions. The development of such a harness will also impact current FFT efforts, by providing potential improvements from a deep analysis of the runtime breakdown, overlap of communication and computation, bottlenecks, and feedback on what other libraries do on their implementations.

The current work will be extended to the creation of a benchmark harness that defines more rigorously the benchmark, simplifies, and automates the process of running, collecting, and analyzing results. Reports will be created for upcoming pre- and exascale machines and the results compared and analyzed.

Appendix

Software projects referenced in the report:

CUDA Compute Unified Device Architecture; a parallel computing platform produced by NVIDIA Inc.

EXAALT Exascale Atomistic capability for Accuracy, Length, and Time; a molecular dynamics simulation platform

ExaSky-HACC Software extending the HACC cosmological simulation code to run on Exascale computing systems

ExaWind A predictive simulation of wind farm energy production physics

HACC Hardware/Hybrid Accelerated Cosmology Code

heFFTe Highly Efficient FFTs for Exascale

WarpX Application for Exascale Modeling of Advanced Particle Accelerators

Terminology employed in this report:

Co-design A form of participatory design which seeks to actively involve all stakeholders to collaborate on a solution (e.g., application/library development teams, end-users, and hardware vendors)

CoPA Co-design center for Particle Applications

ECP The US Department of Energy's Exascale Computing Project

FFT The Fast Fourier Transform is an important algorithm widely used in supercomputing applications

InfiniBand A high-performance computer networking standard widely used in supercomputers

MPI Message Passing Interface; a message passing standard implemented by libraries for process communication across parallel computer systems

Node The component of a distributed memory parallel computer system which hosts a number of CPUs and GPUs with a shared memory

Pencil/Slab/Brick decomposition A method of distributing data in a multi-dimensional FFT algorithm for the purposes of parallelizing the computation across multiple compute nodes

PGI compiler An optimized software compiler suite published by the Portland Group Inc. and now owned by NVIDIA Inc

Tuning The process of experimentally optimizing the parameters of an algorithm for a specific hardware architecture

Bibliography

[1] J. Dongarra and F. Sullivan. Guest editors introduction to the top 10 algorithms. *Computing in Science Engineering*, 2(1):22–23, 2000.

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

[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] cuFFT library, 2018.

[5] rocFFT library, 2021.

[6] Intel. Intel Math Kernel Library. http://software.intel.com/en-us/articles/intel-mkl/.

[7] Large-scale atomic/molecular massively parallel simulator, 2018. Available at https://lammps.sandia.gov/.

[8] 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.

[9] Alan Ayala, S. Tomov, A. Haidar, and Jack Dongarra. heFFTe: Highly Efficient FFT for Exascale. In *ICCS 2020. Lecture Notes in Computer Science*, 2020.

[10] J. Demmel. Communication-avoiding algorithms for linear algebra and beyond. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013.

[11] J. Dongarra. Report on the Sunway TaihuLight System. Technical report, 2016.

[12] Alan Ayala, Xi Luo, S. Tomov, H. Shaiek, A. Haidar, G. Bosilca, and Jack Dongarra. Impacts of Multi-GPU MPI Collective Communications on Large FFT Computation. In *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*, 2019.

[13] Kenneth Czechowski, Chris McClanahan, Casey Battaglino, Kartik Iyer, P.-K Yeung, and Richard Vuduc. On the communication complexity of 3D FFTs and its implications for exascale. 06 2012.

[14] 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.

[15] parallel 2d and 3d complex ffts, 2018. Available at http://www.cs.sandia.gov/~sjplimp/download.html.

[16] D. Takahashi. Implementation of Parallel 3-D Real FFT with 2-D decomposition on Intel Xeon Phi Clusters,. In *13th International conference on parallel processing and applied mathematics.*, 2019.

[17] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Sameer Kumar, Ewing Lusk, Rajeev Thakur, and Jesper Larsson Träff. MPI on a Million Processors. In Matti Ropo, Jan Westerholm, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 20–30, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[18] NVIDIA. NCCL library, 2019.

[19] Lisandro Dalcin, Mikael Mortensen, and David E. Keyes. Fast parallel multidimensional FFT using advanced MPI. *Journal of Parallel and Distributed Computing*, 128:137–150, 2019.

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

[21] Salvatore Filippone. The ibm parallel engineering and scientific subroutine library. In Jack Dongarra, Kaj Madsen, and Jerzy Waśniewski, editors, *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science*, pages 199–206, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[22] Franz Franchetti, Daniele Spampinato, Anuva Kulkarni, Doru Thom Popovici, Tze Meng Low, Michael Franusich, Andrew Canning, Peter McCorquodale, Brian Van Straalen, and Phillip Colella. FFTX and SpectralPack: A First Look. *IEEE International Conference on High Performance Computing, Data, and Analytics*, 2018.

[23] 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.

[24] N. Li and S. Laizet. 2decomp&fft - a highly scalable 2d decomposition library and fft interface. 2010.

[25] 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.

[26] D. Pekurovsky. P3dfft: A framework for parallel computations of fourier transforms in three dimensions. *SIAM Journal on Scientific Computing*, 34(4):C192–C209, 2012.

[27] Krishna Kandalla, Hari Subramoni, Karen Tomko, Dmitry Pekurovsky, Sayantan Sur, and Dhabaleswar K Panda. High-performance and scalable non-blocking all-to-all with collective offload on infiniband clusters: a study with parallel 3d fft. *Computer Science-Research and Development*, 26(3-4):237, 2011.

[28] 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.

[29] J.D. 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.

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

[31] Daisuke Takahashi. FFTE: A fast Fourier transform package. *http://www.ffte.jp/*, 2005.

[32] Jonathan Bentz. FFT libraries on Cray XT: CRay Adaptive FFT (CRAFFT). In *Cray User Group Meeting CUG2008*, Helsinki, Finland, May 5-8 2008.

[33] Scott Parker, Vitali Morozov, Sudheer Chunduri, Kevin Harms, Chris Knight, and Kalyan Kumaran. Early Evaluation of the Cray XC40 Xeon Phi System Theta at Argonne. *Cray User Group 2017 proceedings*, 2017.

[34] 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.

[35] Stanimire Tomov, Azzam Haidar, Alan Ayala, Hejer Shaiek, and Jack Dongarra. FFT-ECP Implementation Optimizations and Features Phase. Technical Report ICL-UT-19-12, 2019-10 2019.

[36] Jesper Larsson Träff and Antoine Rougier. Mpi collectives and datatypes for hierarchical all-to-all communication. In *Proceedings of the 21st European MPI Users' Group Meeting*, pages 27–32, 2014.

[37] Xi Luo, Wei Wu, George Bosilca, Thananon Patinyasakdikul, Linnan Wang, and Jack Dongarra. Adapt: An event-based adaptive collective communication framework. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 118–130, 2018.

[38] Kiran Ranganath, AmirAli Abdolrashidi, Shuaiwen Leon Song, and Daniel Wong. Speeding up collective communications through inter-gpu re-routing. *IEEE Computer Architecture Letters*, 18(2):128–131, 2019.

[39] Muhammed Abdullah Al Ahad, Christian Simmendinger, Roman Iakymchuk, Erwin Laure, and Stefano Markidis. Efficient algorithms for collective operations with notified communication in shared windows. In *2018 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM)*, pages 1–10. IEEE, 2018.