Progress in a computation overlap framework in P3DFFT++, a portable open source library for Fast Fourier Transforms

Dmitry Pekurovsky

San Diego Supercomputer Center
University of California, San Diego
La Jolla, California, USA
dpekurovsky@uscd.edu

ABSTRACT

P3DFFT++ is a highly adaptable open source framework for multidimensional Fast Fourier Transforms and related spectral algorithms. This class of algorithms covers a large number of computational domains. Spectral algorithms are quite difficult to scale on large HPC systems. Hiding communication latency, at least partially, is one way to reduce time to solution, and is one of the goals of this work. In addition, this work aims to implement the algorithms so they can run on heterogeneous platforms with GPUs, and do so in a highly portable, user-friendly manner. In this context we implement and evaluate overlapping computation with network communication, in a way suitable for future work on overlapping data transfer to/from GPU. Eventually the framework will have a unified API for running on CPUs and GPUs. Our benchmarking results demonstrate potential of communication overlap to improve performance by up to 22%.

1 Introduction

Spectral Transforms is a type of algorithms commonly used in a variety of areas of computational science and engineering to simulate a wide range of phenomena. Examples include Direct Numerical Simulations of turbulence, simulations of the ocean and atmosphere, astrophysics, acoustics, seismic simulations, material science, medical imaging and molecular dynamics. This class of algorithms includes, but is not limited to, Fast Fourier Transforms (FFTs). Due to their ubiquitous nature, they are found in many third-party packages, such as numerical libraries.

Spectral transforms are challenging to implement efficiently on large-scale High Performance Computing (HPC) systems. The challenge of extracting a good performance out of FFTs at large scale, for example, is well-studied and has to do with dependence on the system's bisection bandwidth, as well as on-node memory bandwidth [1-3]. They are already a major bottleneck for many important applications, responsible for a large number of compute cycles. Performance challenges are likely to become exacerbated as we approach the Exascale.

This paper describes continuing work on P3DFFT++, an open source general purpose library for Spectral Transforms in 3 and eventually 4 dimensions [4], following earlier work on P3DFFT, a library for scalable Fourier Transforms [5]. It aims to provide an efficient, scalable solution for Spectral Transforms beyond FFT, in an easy-to-use, portable fashion. Our previous year paper [6] described the motivation and design of the library, including the basic software framework and the extremely flexible data structure that can support a wide range of usage scenarios. In this paper we continue this work to focus on performance aspects, in particular the overlap of communication with computation through the use of nonblocking communication. We describe several ways that this can be achieved in an algorithm such as 3D spectral transform, and how it is implemented in P3DFFT++. We present preliminary performance results from our ongoing efforts, which demonstrate some promise of our approach. The hope is that eventually success in overlapping communication with computation will lead to significant performance advantage of our implementation, compared to other packages that do not employ an overlap. This will lead to a more efficient library package that will translate to significant decrease in throughput time for the end user, as well as saved cycles on high-end HPC platforms.

2 Previous work and problem definition

In this work we refer to spectral-like transforms when we mention any multi-dimensional transform algorithm on a structured grid that has the following properties:

1. It can be reduced to a sequence of 1D transforms for an entire array, one for each dimension, independent of other dimensions (e.g. 1D FFT).
2. Each such 1D transform is compute and memory-bandwidth intensive. In terms of data decomposition, it is best to have all data in that dimension to reside locally in memory for each

core/task. This allows us to avoid exchanging data within each 1D transform, which would be extremely expensive.

In order to have such local data access, there needs to be a reshuffling of the data between the 1D stages. This is illustrated in Fig. 1, where a typical 3D FFT algorithm with 2D processor decomposition is broken down into steps. (Note that 2D decomposition is critical for larger scale use of spectral transforms. 1D decomposition does not scale beyond N cores/tasks, where N is the linear grid size of the 3D array N^3. 1D decomposition is a limitation of several earlier libraries, such as FFTW.) Therefore a 3D transform comprises of three 1D transforms, interspersed with two all-to-all exchanges in subcommunicator groups. This logic is followed in P3DFFT, an earlier incarnation of P3DFFT++, and the current version of P3DFFT++. 2D decomposition, in principle, allows the algorithm to scale up to N^2 cores/tasks.

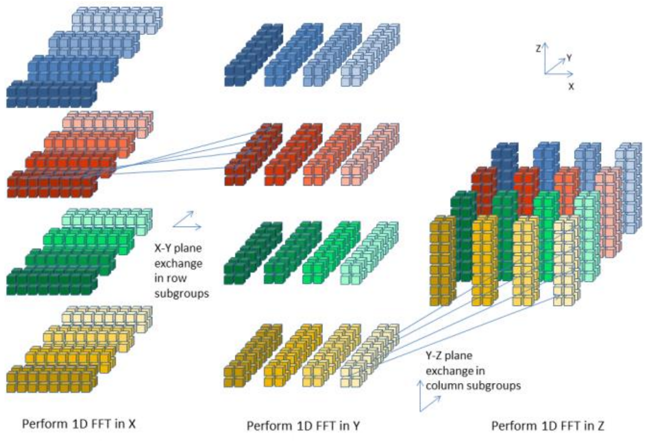
Z
Y
X
X-Y plane exchange in row subgroups
Y-Z plane exchange in column subgroups
Perform 1D FFT in X
Perform 1D FFT in Y
Perform 1D FFT in Z

Figure 1: 3D FFT implementation with 2D decomposition typically involves a sequence of 1D transforms in X, Y and Z dimensions, interspersed with two sub-communicator all-to-all exchanges.

While P3DFFT++ follows the same logic, it allows for a much wider set of data structures and transform types than in P3DFFT. It is written in C++/MPI, with Fortran and C interfaces. It utilizes established 1D FFT implementations such as FFTW, ESSL or cuFFT [43]. The software framework is object-oriented and can be easily expanded. There is an early CUDA/GPU version, in addition to the CPU version, which is more established. The library implements real-to-complex and complex-to-complex FFTs, as well as sine/cosine transforms. It is extensible for other transform types, for example wavelets and high-order finite difference schemes. It features a very general data layout, which should be sufficient to map to the layout of almost any user program. Also, quite importantly for mixed precision applications, it works with both single and double precision data. The package includes example programs in all three languages (C/C++/Fortran), as well as full documentation. P3DFFT++ has been found to scale well up to 32k cores and beyond, provided adequate hardware support (see Fig. 2). An earlier generation of the library, P3DFFT, was found to scale up to 512k cores [5], and the same scaling is expected for P3DFFT++.

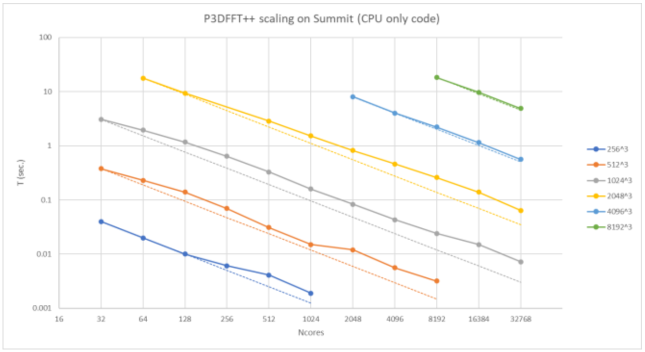

Figure 2: Strong performance scaling of P3DFFT++ for a sample problem of complex 3D FFT, using Summit supercomputer at ORNL.

A number of other 3D FFT/spectral transforms packages have been published in the last decade, confirming a great interest in this class of algorithms. They include both open source, third-party libraries as well as parts of open and proprietary codes. While most impementations share the basic structure of 3D algorithm shown in Fig. 1, there are many variations in terms of types of algorithms covered, data structures assumed, and performance features incorporated. It would go beyond the scope of this paper to do a thorough review of existing packages, so we will mention only a few libraries that are most recent and noteworthy in our opinion.

heFFTe [7] is a new powerful library from UTK group. It came about as a part of DOE's Exascale Computing Project. It is written using modern C++11 features. It provides both CPU and GPU implementation. It provides an option for 3D decomposition and implements real-to-complex and complex-to-complex transforms. This library does not seem to support other spectral transforms, and, like most libraries, is designed for a fixed data layout.

FFTE [8] is another implementation for both CPU and GPU. It has been in continuous development for a number of years. Its latest version provides support for NVIDIA GPUs only through PGI compiler. In addition, it has limited features – for example, only grids of certain sizes and MPI rank counts are supported.

AccFFT [9] is a library implementing real-to-complex 3DFFT on NVIDIA GPUs. Its last release was in 2014, and it is no longer supported.

SWFFT [10] is a stand-alone 3D FFT CPU library derived from Hardware/Hybrid Accelerated Cosmology Code (HACC) from Argonne [11]. It is one of the few libraries supporting 3D decomposition. It seems to support only complex FFTs. Also, it seems to have restrictions in sizes of transforms and decompositions.

FLUPS [12] is a library for solving Poisson unbounded problems using spectral methods. It provides sine/cosine transforms, in addition to FFTs, in combinations for three dimensions. This makes it (along with P3DFFT++) more general than other libraries. It has hybrid MPI/OpenMP implementation and is capable of using nonblocking communication. It lacks advanced features, such as general data layout. It also does not have a GPU implementation.

NB3DFFT [13] is one of very few libraries implementing overlap of communication with computation through nonblocking communication. It was last released in 2015 and is limited in the features it supports.

This short overview, hopefully, gives the reader an idea of the space of features spanned by modern spectral transforms libraries. While no single library can be expected to have the best solution for any problem and platform, P3DFFT++ aims to maximize the extent of both portability, usability and performance. The latter aspect is an extremely important one for practical use on modern systems, as mentioned previously, and will be the subject of the remainder of this paper.

3 Nonblocking Communication

The idea of overlapping communication with computation is not new, however it is not often found in spectral transforms implementations. One reason for this may have to do with not having an obvious way to stagger the stages of the algorithm to overlap communicating with computational stages. After all, the stages of the algorithm depend on each other and are supposed to be done in sequence.

We see a couple of ways to solve this problem. One situation when this becomes possible is when the original problem has several independent variables to be transformed (for example, velocity components). In such case the algorithm execution for these variables can be staggered, so that communication for one variables is happening while another variable goes through a computation phase (see Fig. 3).

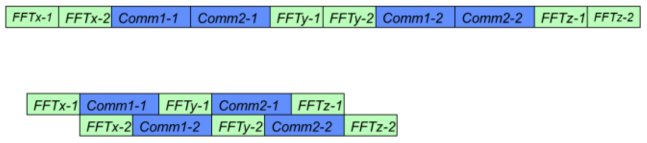

Figure 3. A diagram showing sequential (above) and staggered/overlapped (below) processing in 3D FFT algorithm of two independent variables. Computational stages are shown in green and are labeled FFT, the dimension of transform (x, y or z) and the variable ID. Communication stages are labeled Comm, with one number identifying the variable and the second number the communication phase (1 or 2). Parallel execution of computation and communication is shown as adjacent rows, and results in shorter overall execution time.

Another way to get traction with the overlap idea is to divide the local grids into a number of segments, or chunks, and stagger the stages of the algorithm over these chunks so as to overlap communication of one chunk with computation of the next one. The splitting of local arrays translates into smaller message sizes, and thus can potentially reduce the efficiency of communication. This is undesirable and less effective than expressing the overlap through multiple variables, however this may be the only way to introduce overlap in case when there is only one variable to be transformed.

While work is underway to implement multi-variable transforms in P3DFFT++, we can refer the reader to our past work on P3DFFT [14], where we have shown that the benefit of overlap through nonblocking communication in some cases can be as high as 27% overall performance improvement (see Fig. 4). The remainder of this section deals with single-variable overlapped algorithm through splitting the buffers.

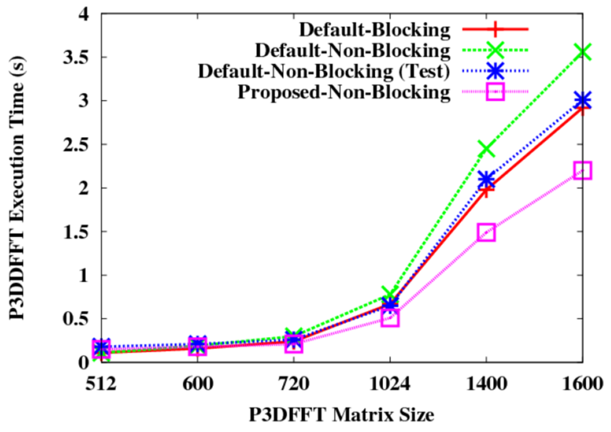

Figure 4. Results from previous work [14] comparing performance from blocking vs. nonblocking (overlapped) implementation of P3DFFT. Time of solution is plotted on the vertical axis (less is better).

3.1 Single-variable overlap

As mentioned above, in order to implement overlap in a single-variable algorithm, it is necessary to split the local buffers. For simplicity, we split them into N_c equal parts. The best value for Nc depends on a number of factors, arising from hardware and system software characteristics of the system the application is running on, as well as problem size, data type etc

The algorithm then becomes a modification of Fig. 1, where computation of one segment is overlapped with communication for the next segment (in a manner analogous to Fig. 3, but instead of independent variable we are overlapping segments of the same variable). As for the communication mechanism, the most natural choice is to use nonblocking collectives, namely MPI_Ialltoallv. For comparison, we have also implemented an alternative mechanism, using pairwise exchanges with MPI_Isend and MPI_Irecv nonblocking calls.

3.2 Discussion

In Fig. 5 we show comparison between these implementations on Summit platform, for a range of different N_c values. We see that the pairwise method performs better in all cases. This result is clearly specific to the given platform and MPI vendor, as well as possibly the problem considered. Thus we are leaving both collective and pairwise communication options in P3DFFT++ for the user to choose.

We also note that there is a sweet spot in performance as a function of number of segments N_c. The same was observed for other grid sizes and core counts. As N_c increases, so does the degree of overlap $(N_c-1)/N_c$. However, it also decreases the message size, which leads to poor bandwidth utilization, and this starts to affect performance negatively at the higher end of N_c range.

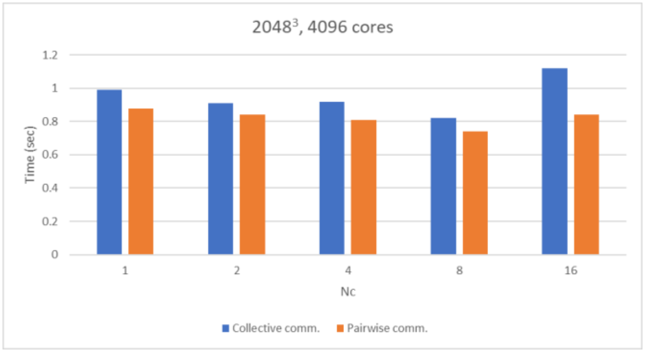

Figure 5. Comparison of collective communication versus pairwise exchange mechanisms in the all-to-all exchange in nonblocking communication version of P3DFFT++. Results were obtained on Summit at ORNL, using complex 3D FFT of size 2048^3, on 4096 cores. Time of solution is plotted on the vertical axis (less is better).

We compare performance of the overlap version with the default (blocking) version in Figs. 6 and 7. We plot the data from the blocked version, and the overlapped version (pairwise exchange) with N_c=1, as well as the best N_c in terms of solution time. The N_c=1 case implies no overlap is taking place, however it is still interesting to compare against the default case, which uses MPI_Alltoallv.

In many cases we observe improvement with the overlap version, showing up to 22% better performance than the default version. The magnitude of the difference is highest at low core counts, which can be explained by decreasing message size as the core count increases. In some cases the N_c=1 runs produce better results than N_c>1 runs. This is likely explained by smaller message sizes, leading to poorer bandwidth utilization and thus cancelling the benefits of the overlap. It is also noteworthy that N_c=1 runs are typically better than the default version, even though no overlap is taking place. This is likely a reflection of peculiarities of the MPI implementation, with pairwise exchanges providing better performance than MPI_Alltoallv.

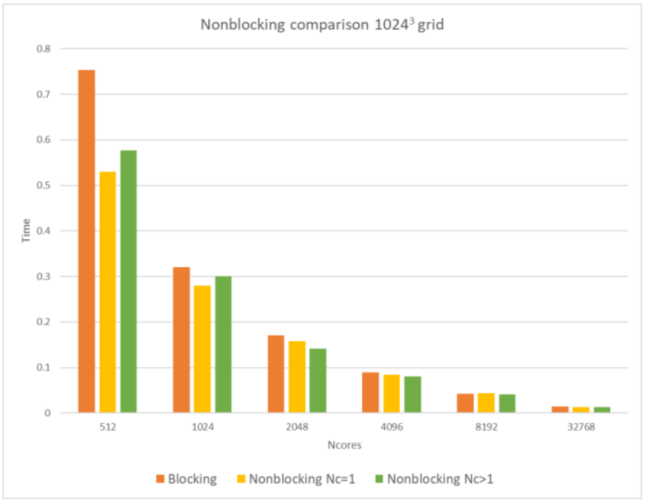

Figure 6. Performance comparison between blocking, nonblocking pairwise version without overlap, and nonblocking version with overlap (best N_c), as a function of number of cores. Time of solution is plotted on the vertical axis (less is better). These results were obtained on Summit supercomputer at ORNL, with complex 3D FFT benchmark test on 1024^3 grid.

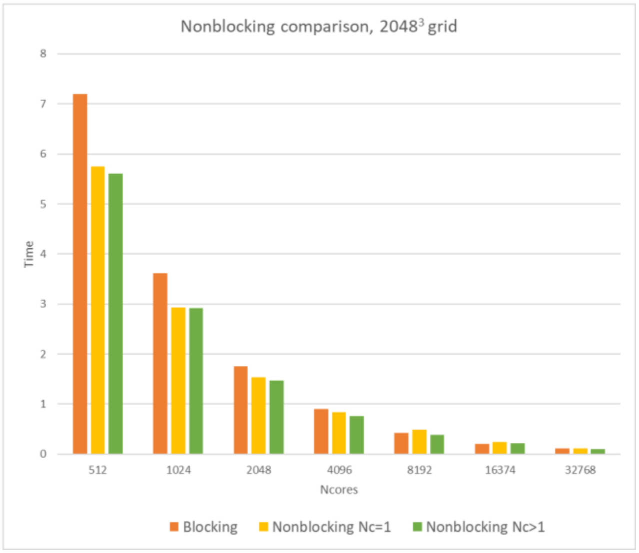

Figure 7. Performance comparison between blocking, nonblocking pairwise version without overlap, and nonblocking version with overlap (best N_c), as a function of number of cores. Time of solution is plotted on the vertical axis (less is better). These results were obtained on Summit supercomputer at ORNL, with complex 3D FFT benchmark test on 2048^3 grid.

Although these conclusions may be specific for the system and problem studied, they indicate potential of the overlap approach, as well as the benefit of having a pairwise exchange version, in addition to a collective communication version. The best setting for the N_c parameter could be determined by trial and error for each system/problem combination.

Having the software framework in place for single-variable overlap gives us the possibility to consider further improvements in the algorithm, for example, a threaded implementation, where MPI is combined with OpenMP threads, and each thread is responsible for a segment of the local array.

Finally, this framework will also be used for overlapping GPU-CPU traffic with GPU computation, in an ongoing effort to optimize the GPU version.

4 Conclusions

We have discussed ongoing work on P3DFFT++ open source library for spectral transforms. After reviewing past work and defining the problem, we have focused on different ways of implementing overlap of communication with computation. We have described single-variable overlap implementation and investigated dependence of performance on the number of segments and the mechanism of nonblocking communication. We have shown promising results from benchmarking runs, where we see up to 22% improvement in performance resulting from overlap with pairwise exchange communication in complex 3D FFT. These results apply, without loss of generality, to other types of spectral transforms besides FFT.

These developments will be part of a public release of P3DFFT++ in the near future. In the future we will also implement multi-variable transforms, where the ability to overlap larger buffers is likely to lead to better performance through bandwidth utilization. We will also study threaded overlap. This work also lays the ground for overlapping GPU traffic with computation, which is an important part of extracting higher performance from GPUs, and will be part of continued work on the package in the near future. Every effort is made to make the package portable and easy to use, while giving the user maximal controls affecting potential performance of the library.

ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This work was supported by U.S. NSF grant OAC-1835885.

REFERENCES

[1] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P.K. Yeung and R. Vuduc, “On the communication complexity of 3D FFTs and its implications for exascale”, Proceedings of the 26th ACM international conference on Supercomputing, pp. 205-214, June 2012.

[2] C. McClanahan, K. Czechowski, C. Battaglino, K. Iyer, P.K. Yeung and R. Vuduc, “Prospects for scalable 3D FFTs on heterogeneous exascale systems”, ACMIEEE conference on supercomputing, SC. 2011

[3] H. Gahvari, and W. Gropp, “An introductory exascale feasibility study for FFTs and multiautotune”, IEEE International Symposium on Parallel & Distributed Processing (IPDPS), pp.1-9, April 2010.

[4] D. Pekurovsky, P3DFFT++ Home page, http://www.p3dfft.net

[5] D. Pekurovsky, “P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions”, SIAM Journal on Scientific Computing, 34(4), pp.C192-C209, 2012.

[6] D. Pekurovsky “A Portable Framework for Multudimensional Spectral-like Transforms At Scale”. Proceedings of the 2021 Improving Scientific Software Conference (No. NCAR/TN567+PROC). 2021 August 12. DOI: 10.26024/p6mv-en77

[7] A. Ayala, S. Tomov, A. Haidar, and J. Dongarra, (2020). “heFFTe: Highly Efficient FFT for Exascale,” International Conference on Computational Science (ICCS 2020), Amsterdam, Netherlands, June 2020. DOI: 10.1007/978-3-030-50371-0_19

[8] FFTE website, http://www.ffte.jp

[9] A. Gholami, J. Hill, D. Malhotra, and G. Biros (2015). AccFFT: A library for distributed-memory FFT on CPU and GPU architectures, http://arxiv.org/abs/1506.07933

[10] SWFFT website, https://git.cels.anl.gov/hacc/SWFFT

[11] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T.J. Peterka, J. Insley, D. Daniel, P. Fasel, and Z. Lukic’ (2017). HACC: Extreme Scaling and Performance Across Diverse Architectures, Comm. ACM (Research Highlight) 60, 97. DOI:10.1145/3015569.

[12] D.G. Caprace, T. Gillis, and P. Chatelai (2021). FLUPS: A Fourier-based library of unbounded Poisson solvers. SIAM Journal on Scientific Computing, 43(1), C31-C60.

[13] J.H. Göbbert, H. Iliev, C. Ansorge and H. Pitsch (2016) “Overlapping of Communication and Computation in nb3dfft for 3D Fast Fourier Transformations”, In Jülich Aachen Research Alliance (JARA) High-Performance Computing Symposium, pp. 151-159, Springer, Cham., September 2016

[14] K. Kandalla, H. Subramoni, K. Tomko, D. Pekurovsky, D. Panda (2013) “A Novel Functional Partitioning Approach to Design High-Performance MPI-3 Non-blocking Alltoallv Collective on Multi-core Systems.” 42nd International Conference on Parallel Processing (ICPP), 2013