

## P3DFFT: A FRAMEWORK FOR PARALLEL COMPUTATIONS OF FOURIER TRANSFORMS IN THREE DIMENSIONS\*

DMITRY PEKUROVSKY†

**Abstract.** Fourier and related transforms are a family of algorithms widely employed in diverse areas of computational science, notoriously difficult to scale on high-performance parallel computers with a large number of processing elements (cores). This paper introduces a popular software package called P3DFFT which implements fast Fourier transforms (FFTs) in three dimensions in a highly efficient and scalable way. It overcomes a well-known scalability bottleneck of three-dimensional (3D) FFT implementations by using two-dimensional domain decomposition. Designed for portable performance, P3DFFT achieves excellent timings for a number of systems and problem sizes. On a Cray XT5 system P3DFFT attains 45% efficiency in weak scaling from 128 to 65,536 computational cores. Library features include Fourier and Chebyshev transforms, Fortran and C interfaces, in- and out-of-place transforms, uneven data grids, and single and double precision. P3DFFT is available as open source at <http://code.google.com/p/p3dfft/>. This paper discusses P3DFFT implementation and performance in a way that helps guide the user in making optimal choices for parameters of their runs.

**Key words.** fast Fourier transforms, parallel computing, scalable algorithms, all-to-all communication, numerical libraries, petascale computation, bisection bandwidth

**AMS subject classifications.** 65T50, 97N80, 97N80

**DOI.** 10.1137/11082748X

**1. Introduction.** A large number of applications in computational sciences are written to solve partial differential equations on three-dimensional (3D) grids. Common examples are pseudospectral solvers employing fast Fourier transforms (FFTs), as well as codes using Chebyshev transform and compact finite difference algorithms. Most of these algorithms share the feature of needing local (with respect to a processing element) access to all data in a given dimension of a 3D array at a certain stage, and then the second and third dimensions at other stages. This paper presents a software framework for such operations, and while we focus our discussion on FFTs it is naturally extended to other algorithms in this class.

The FFT in one, two, or three dimensions is an efficient algorithm well known for decades (see Cooley and Tukey (1965)) as well as a traditional component of any serious numerical library. 3D FFT happens to be one of the most compute- and communication-intensive components in applications from a range of fields (for example, in turbulence, molecular dynamics, 3D tomography, and astrophysics), and for this reason there are many excellent parallel implementations in existence today. Some of the implementations are supplied by well-respected libraries and packages, whether open-source or commercial (see FFTW, Frigo and Johnson (2005), and PESSL, NAG, Agarwal, Gustavson, and Zubair (1994)). While these are typically highly optimized and produce good performance at moderate scales, most of them were not designed

---

\*Submitted to the journal's Software and High Performance Computing section March 15, 2011; accepted for publication (in revised form) April 19, 2012; published electronically August 15, 2012. This work was sponsored by NSF grant OCI-085-684. The author acknowledges use of NSF computer resources at NICS/ORNL, and the TACC/University of Texas Ranger system, under a Teragrid startup allocation award, as well as Cray XT5 (Jaguar) system at NCCS/ORNL under a DOE INCITE award "A Petascale Study of Turbulent Mixing in Non-Stratified and Stratified Flows."

<http://www.siam.org/journals/sisc/34-4/82748.html>

†UC San Diego, San Diego Supercomputer Center, 9500 Gilman Drive, MC 0505, La Jolla, CA 92093 (dmitry@sdsc.edu).

to address one increasingly important consideration, namely, the limit of scalability. Most of the parallel 3D FFT libraries to date use *one-dimensional* (1D), or *slab*, domain decomposition (described in section 2), which allows scaling only up to the linear grid size. At the age of petascale platforms more and more systems typically have numbers of processing elements (PEs) far exceeding this limit for many applications. For example, cutting-edge turbulence simulations today (Donzis, Yeung, and Pekurovsky (2008); Donzis and Yeung (2010)) use  $4096^3$  grids and so with 1D decomposition would scale only to 4096 PEs at most. In order to handle the enormous volume of data and compute operations such applications are run on some of the most powerful supercomputers of today, involving  $O(10^4-10^5)$  PEs. The P3DFFT package [P3DFFT, D. Pekurovsky] (the subject of this paper) overcomes this scalability barrier by employing *two-dimensional* (2D), or *pencils*, domain decomposition.

While the idea of 2D decomposition applied in the 3D FFT algorithm is not new, few general-purpose implementations are known to date. Wapperom, Bens, and Straka (2006) investigated a 2D decomposition applied in an FFT/Chebyshev transform and in a pseudospectral DNS turbulence code. Several interesting studies of 2D decomposed 3D FFT on various modern supercomputer architectures have been reported recently (Takahashi (2010), Jagode (2008), Hein et al. (2008), Kirker (2009)); however, these codes are not available to the public. In terms of general-purpose libraries, at this time there are only a small number of offerings. An effort by an IBM team (Eleftheriou et al. (2005)) resulted in a library of 3D decomposed parallel FFTs optimized for 3D torus architecture of an IBM BG/L supercomputer. Unfortunately this software provides a very limited set of options in terms of input sizes and datatypes and is available only as a precompiled binary for IBM BG/L. An FFT package related to LAMMPS code (Plimpton) provides array remapping tools for employing 2D decomposition and 3D FFT implementation; however, this package implements only complex-to-complex transforms.

P3DFFT fills the niche by providing an open source, easy to use implementation (available from <http://code.google.com/p/p3dfft>) employing 2D decomposition. By design it is portable to a large number of platforms, adaptable for diverse applications, and at the same time is built for maximum performance. P3DFFT is currently built on top of Message Passing Interface (MPI), which provides portable performance and ease of use. Its framework has been successfully applied in applications ranging from direct numerical simulations of turbulence to oceanography, astrophysics, material science, and chemistry (Donzis and Yeung (2010), Homann et al. (2009) and (2010), Chandy and Frankel (2009), Schumacher and Putz (2007), Schumaker (2009), Peters et al. (2010), Bodart (2009), Grafke (2008), Weidauer et al. (2010), Laizet et al. (2010), Schaeffer et al. (2010)). Its performance has been rigorously studied on a torus interconnect architecture of the IBM BG/L supercomputer (Chan et al. (2008)).

The paper is structured as follows: In section 2 we introduce the parallel 3D FFT algorithm implemented in P3DFFT as well as the two most relevant decomposition schemes. In section 3 we delve into details about P3DFFT implementation. In section 4 we present results of some benchmark tests and analyze performance of the library, comparing it with an asymptotic model. Section 5 contains a summary and our conclusion.

**2. Parallel implementation of Fourier transforms.** There are many publications devoted to parallel implementation of FFTs (see, for example, Foster and Worley (1997), Swartztrauber (1987), and Sweet et al. (1991)). Let us begin by considering how the grid is decomposed among parallel tasks. The simplest way to distribute

## 1D decomposition      2D decomposition

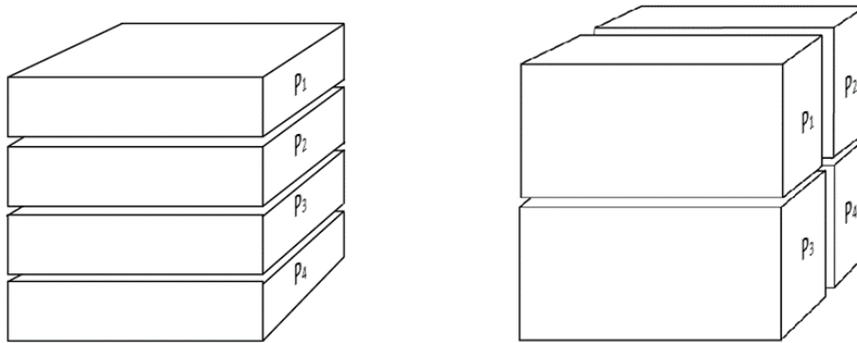


FIG. 1. 1D and 2D domain decomposition. The number of data points assigned to a processing element  $P_1$  through  $P_4$  is essentially the same, but they represent different geometrical shapes.

the data is by assigning each task one or more 2D planes (a *slab*) of the array (see Figure 1). This is known as 1D domain decomposition because the grid is divided along one dimension. As already mentioned, this approach works well only as long as the task count  $P$  does not exceed the linear grid size  $N$ . If taken beyond this point, 1D decomposition results in a sudden loss of scalability due to load imbalance, as some tasks are without work. 2D, or *pencils*, decomposition, used in P3DFFT, is the next logical step. Instead of slabs, each processor/task is responsible for a rectangular column (pencil) of the data array. The tasks are arranged in a 2D virtual grid with dimensions  $M_1 \times M_2$  and this controls the dimensions of each pencil. There is some freedom in choosing the grid dimensions, as long as the product of  $M_1$  and  $M_2$  equals  $P$ , the total number of tasks. With 2D decomposition the algorithm theoretically scales up to  $N^2$  tasks, and while it is somewhat more cumbersome to implement, as noted in the introduction, a number of applications running on subpetascale and petascale platforms critically depend upon it.

Regardless of decomposition, a Fourier transform in three dimensions is comprised of three 1D FFTs in the three dimensions (X, Y, and Z) in turn. When all of the data in a given dimension of the grid resides entirely in a processor's memory (i.e., it is *local*) the transform consists of a 1D FFT done over multiple grid lines by every processor, which can be accomplished by a serial algorithm provided by many well-known FFT libraries and is usually a fairly fast operation. The transforms proceed independently on each processor with regard to its own assigned portion of the array. When the data are divided across processor boundaries (i.e., nonlocal), several approaches are possible. One approach could be to use a parallel implementation of 1D FFT that exchanges the data elements as they are needed on other processors. Such method (*distributed* transform in the terminology of Foster and Worley (1997)) involves multiple data exchanges. An alternative is the *transpose* transform, where the array is reorganized by a single step of global transposition so that the dimension to be transformed becomes local, and then serial 1D FFT can be applied. As shown in Foster and Worley (1997) (see Table 1 therein), the data volume exchanged in the transpose method is approximately  $\log(M_1)/2$  or  $\log(M_2)/2$  times less than that involved in the distributed approach. As will be seen below, data volume has almost

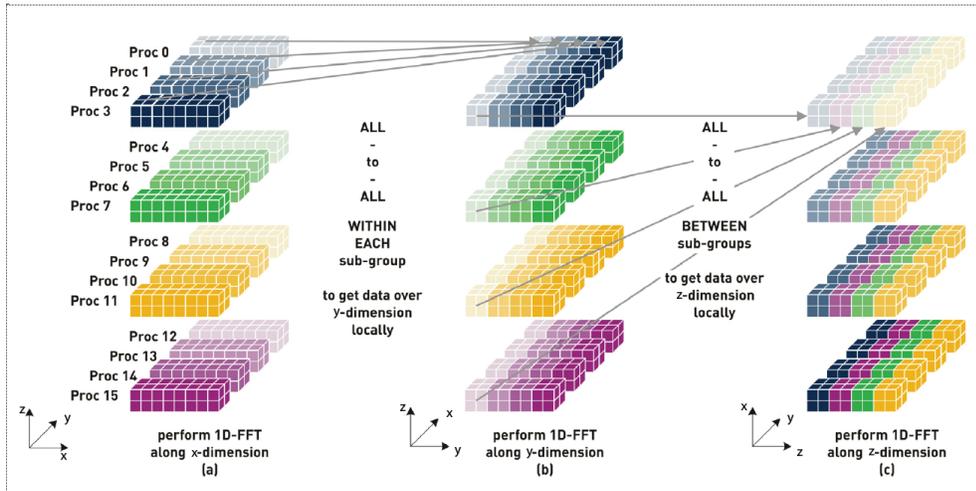


FIG. 2. Three computation and two transpose stages of 3D FFT algorithm 2D parallel decomposition. In this example there are 16 tasks arranged in a  $4 \times 4$  processor grid. Each rectangular column of boxes of the same color represents data contained locally in a processor's memory. (Going from left to right) 1D FFT in X dimension for all X lines performed independently by each task is followed by an exchange/transpose among processors belonging to the same subgroup (a row in processor space). This is followed by a 1D FFT in Y followed by the second transpose among different rows, followed by the final 1D FFT in Z dimension.

linear relation to the total execution time since communication, which comprises most of the elapsed time in many cases of interest, is dominated by the time to push the data volume through network links with limited bandwidth. Therefore, the transpose approach is used in P3DFFT and throughout this paper.

Thus the parallel algorithm employed in P3DFFT (Figure 2) has three compute stages corresponding to 1D FFT transforms in X, Y, and Z dimensions over the entire volume of the data grid, interspersed with two parallel transposes in order to first rearrange data from X- to Y-oriented pencils, and then from Y- to Z-oriented pencils. Corresponding to the idea of the transpose approach, the 1D FFT compute stages always proceed on pencils oriented in the direction of the transform, which means the transform is local. Each transpose involves an all-to-all exchange within subgroups of tasks comprising either a row or column of the virtual 2D processor grid.

Many applications study physical problems with one dimension of nonhomogeneity—for example, a wall-bounded turbulent flow where two dimensions have periodic boundary conditions while the third dimension has rigid walls on both ends. For such situations it is common to use transforms other than Fourier in the third dimension—for example, Chebyshev transforms, compact schemes, etc. The set-up for these problems is essentially identical to the FFT scenario, with substitution of the appropriate third transform in place of the FFT.

### 3. P3DFFT design and use.

**3.1. Library overview.** P3DFFT is available as an open-source library package with an easy-to-use interface, providing implementation of 3D FFT with strong performance and scalability on a variety of platforms. Since it implements 2D decomposition, it can scale up to  $N^2$  cores as discussed above. 1D decomposition is also included as a special case of 2D decomposition as an option for running on moderate core counts. Both Fortran and C interfaces are supported, and comprehensive

TABLE 1

Local array dimensions ( $L_1, L_2, L_3$ ) and logical memory storage order (Fortran convention) for 3D local arrays comprising X-, Y-, and Z-pencils. R2C transform takes X-pencils as input and Z-pencils as output. In C2R the input/output is reversed. XYZ in this format means that the index spanning the X dimension runs fastest, followed by the Y and Z indices.  $M_1$  and  $M_2$  are dimensions of the processor grid ( $M_1 \times M_2 = P$ ), and  $N_x, N_y, N_z$  are dimensions of the 3D data grid.

	X-pencil			Y-pencil		Z-pencil	
STRIDE1 defined	$L_1$	$N_x$	XYZ	$N_y$	YXZ	$N_z$	ZYX
	$L_2$	$N_y/M_1$		$(N_x+2)/(2M_1)$		$N_y/M_2$	
	$L_3$	$N_z/M_2$		$N_z/M_2$		$(N_x+2)/(2M_1)$	
STRIDE1 undefined	$L_1$	$N_x$	XYZ	$(N_x+2)/(2M_1)$	XYZ	$(N_x+2)/(2M_1)$	XYZ
	$L_2$	$N_y/M_1$		$N_y$		$N_y/M_2$	
	$L_3$	$N_z/M_2$		$N_z/M_2$		$N_z$	

documentation is available. Also supplied in the distribution are sample programs in both Fortran and C, illustrating several usage scenarios. P3DFFT supports both in-place and out-of-place transforms in both single and double precision. It can handle any grid dimensions (i.e., not a power of two) as long as the underlying FFT library (ESSL or FFTW) supports them. P3DFFT is capable of handling problems with uneven decomposition among processors—for example, a  $256^3$  grid on 24 MPI tasks. Although P3DFFT works for grids of any size, it is designed to be most effective for large datasets. The present version 2.5 of P3DFFT implements real-to-complex (R2C forward) and complex-to-real (C2R backward) Fourier transforms, sine/cosine (Chebyshev) transforms, and an empty transform which allows the user to substitute a custom transform of their own choice. P3DFFT has been tested and benchmarked on many parallel machines.

**3.2. User interface.** An X-pencil is defined as a local array shape such that the X dimension is entirely local within a given task's memory. The Y dimension is spread over the tasks in the rows of the virtual processor grid, while the Z dimension is spread over columns. Similar definitions apply for Y-pencils and Z-pencils (see Table 1). P3DFFT accepts the input data array for R2C shaped as X-pencils, while the output array is defined as Z-pencils. Conversely, the C2R transform expects the input array in Z-pencils, while its output comes as X-pencils. Significant resources are saved by avoiding transpose back to the original distribution shape. This method is suited for a great many applications (such as convolution and differentiation algorithms) that require forward and backward transforms in sequence.

While pencil type fixes the shape of local arrays, memory storage order within the arrays can vary, depending on the pencil shape and the optional user flag STRIDE1. Memory storage dimensions are summarized in Table 1.

P3DFFT is written in Fortran 90 and uses MPI for communication. While the library can be called from C as well, we follow the Fortran convention, and when talking about XYZ storage order we mean that the X dimension is the first in the order of storage; i.e., elements with different X for given Y and Z indices are stored contiguously.

In an R2C 3D transform the first transform (in X) is a real-to-complex 1D FFT, while the following Y and Z 1D FFTs are complex-to-complex. The real-to-complex transforms are known to have complex conjugate symmetry, namely,  $F(x) = F^*(N-x)$ , which means half of the output modes are redundant due to the conjugate symmetry. In particular, in an R2C transform of size  $N$ , modes  $N/2+2$  through  $N$  can be reconstructed from those numbered 1 through  $N/2$ . Modes 1 (the average) and  $N_x/2+1$  (the Nyquist frequency) have zero imaginary components. In practice

this means that the output (reflected in Table 1) is represented by complex numbers in an array of dimensions  $((N_x + 2)/2, N_y, N_z)$  distributed among the processors in a suitable way.

The library is called from a user's code via module interface. Internal library variables are hidden from the user while special routines are used to set or get relevant parameters such as the grid size and dimensions of pencil arrays (details of usage are described in [P3DFFT manual]). The package is easily installed using *configure* command.

**3.3. Implementation strategy.** As already mentioned, the design of P3DFFT followed the principle of portable performance. This implies that sometimes a choice needed to be made between achieving performance on a given platform and portability. A number of intermediate kernels have been created, incorporating various ideas mentioned below, which were then tested on several platforms. Some options have been found to underperform universally across the platforms, in which case the decision was clearly against them. In other cases there was no clear winner, or different variants showed up better or worse depending on the platform and/or usage scenario. In such cases we attempted to implement all of the available options and delegate the selection to the user. Thus the user has the ability to take an active part in performance, tuning on their platform of choice. On the other hand, a set of reasonable defaults guarantees ease of use for those not inclined for performance timing experiments. Some of the flexibility has to do with data layout, and in this case considerations other than performance may be in place, i.e., ease of incorporating with the rest of the user's application.

As mentioned in the previous section, the compute stages of the parallel 3D FFT algorithm consist of 1D FFT on local data. P3DFFT uses an established FFT library of the user's choice (currently FFTW or ESSL) for this task. Each of the three FFTs is done as a single call, combining transforms of multiple sets of data. For transforms in the second and third dimensions it is necessary to operate on data sets that are not contiguous in memory (though always local in the memory of a core/task). This can be handled in several ways. Typically FFT libraries allow both stride-1 and non-unit-stride data access patterns. Thus one approach is to let the FFT library handle the non-unit strides. Another approach is to transpose the data first to arrange them in a stride-1 format before calling the FFT library. Both of these approaches are available in P3DFFT; the second is triggered by setting the STRIDE1 flag. Loop blocking is used with the memory transpose to optimize cache use. Relative performance advantages of these approaches are discussed in the next section. A third alternative can also be conceived, wherein FFT sublibraries are called with input, but not the output, having unit stride (or vice versa). For example, if the input is in the form  $(y,z,x)$ , where  $y$  is the fastest changing index, a call to an FFT library could transpose the input, calculate the FFT in  $Z$ , and then write the output in the form  $(z,y,x)$ , which is the optimal form for backward transform that follows. This approach was explored and, at least with FFTW, found to lead to inferior performance. The reasons for this are not clear at this time and may be specific to FFTW implementation. This approach is not used in P3DFFT.

As already mentioned, transposing data between pencils of different orientation between the three stages requires an all-to-all exchange within subgroups of processors. We define MPI cartesian subcommunicators ROW and COLUMN based on each task's placement in the virtual 2D processor grid. In order to transpose from X- to Y-pencils, an all-to-all exchange is needed among tasks on the same rows of the

processor grid. Similarly, to transpose from Y- to Z-pencils, an all-to-all exchange within columns is needed. These exchanges are implemented by a call to *MPI\_Alltoall* or *MPI\_Alltoallv* with ROW (COLUMN) used as the communicator argument. This relies on an optimized implementation of *MPI\_Alltoall(v)* as part of the MPI library on a given system. In most cases *MPI\_Alltoall(v)* can be expected to work faster than an equivalent collection of point-to-point send/receive calls. This was confirmed in numerical experiments on several platforms. (The above is valid for blocking operations. Using nonblocking point-to-point sends/receives, or PGAS languages such as Co-Array Fortran, allows overlapping communication and computation. This is an interesting alternative; however, it is not always a portable choice. Studies are ongoing, and future releases of the package may include a PGAS or overlap option.)

The parallel transposes involve packing/unpacking data into/out of send/receive buffers. Essentially this procedure involves a memory copy. In addition, when the STRIDE1 option is defined the packing/unpacking may involve a local memory transpose; i.e., the copy operation does not follow a stride-1 pattern. We use loop blocking to minimize cache misses. While the MPI standard provides a mechanism for such pack/unpack operations through MPI datatypes, performance is dependent on the particular MPI implementation. In experiments to date no measurable gain in performance was found for a version implemented with MPI Datatypes. Therefore, we chose to perform the packing/unpacking explicitly.

**3.4. Load balancing.** The algorithm lends itself to a symmetric work distribution, and P3DFFT attempts to divide the work evenly among the tasks (this also includes communication). In cases of uneven dimensions for local grid (for example when using  $128 \times 128 \times 128$  data grid on 6 CPUs) load imbalance is naturally present. In practice there is a small load imbalance even in the case when the processor grid evenly divides the data grid due to the nature of R2C and C2R transforms, since as mentioned earlier the dimensions of the output array is  $((N_x + 2)/2, N_y, N_z)$ . This leads to a slight load imbalance, even if the initial problem is evenly decomposed by the processor grid. Since the number of elements exchanged among tasks may not be the same, the *MPI\_Alltoallv* call is used for exchanges as it allows for more flexibility in defining data than *MPI\_Alltoall*. The mentioned load imbalance is quite small for a reasonably large data grid volume, and in practice does not cause performance problems. This implementation is therefore well suited for evenly as well as unevenly distributed data grids.

While *MPI\_Alltoallv* is the simplest solution, it is not necessarily the best one in some cases. In particular, it has been reported [Schulz] that implementation of *MPI\_Alltoallv* on Cray XT platforms is inferior in performance to *MPI\_Alltoall* for comparable (evenly divided) buffers. For this reason P3DFFT provides an option (USEEVEN) for reverting to *MPI\_Alltoall* by simply padding send buffers with a few extra elements in order to make the exchanges even. When load imbalance is small these extra elements do not substantially add to the volume of data exchanged and the end result is better than using *MPI\_Alltoallv* on these systems, as shown below in the results section.

## 4. Results and discussion.

**4.1. Benchmark tests description.** In this section we describe benchmark results obtained by using one of the sample programs (test\_sine) supplied with the P3DFFT package. This program initializes a 3D array of specified dimensions, performs a forward and backward 3D FFT, and then checks to make sure the data is the

same (apart from a scale factor) as the initial array. The forward and backward FFT can be repeated in a timed loop for a specified number of iterations. The program reports time results averaged over the iterations' count.

P3DFFT has been tested on many platforms, and in this paper we discuss results obtained on Cray XT5 partition of *Jaguar* at NCCS/ORNL, *Kraken* at NICS/University of Tennessee, and the *Ranger* system at TACC/University of Texas at Austin. Jaguar is a Cray XT5 computer with 2.6 GHZ AMD Opteron processors, 12 cores per node (2 sockets with 6 cores per socket), 16 GB of memory per node, and Cray SeaStar2 3D torus interconnect [Jaguar], with a total of 224,256 compute cores. GNU compilers were used in these tests. Kraken is a similar but somewhat smaller system at NICS/UT [Kraken]. Ranger is a Sun/AMD platform with 2.3GHz AMD Opteron processors, 16 cores per node (4 quad-core), 32 GB of memory per node, and a Closs interconnect (InfiniBand switch), with a total of 62,976 compute cores [Ranger]. PGI compilers were used in our tests on Ranger.

**4.2. Achieving optimal performance with P3DFFT.** As mentioned earlier, P3DFFT was designed with reasonable defaults for portable performance, as well as several user-controlled settings that can be adjusted for maximum performance. These settings include the STRIDE1 option, the USEEVEN option, and a suitable choice of the aspect ratio of the virtual processor grid. We are going to discuss these in turn.

1. *STRIDE1 option.* Recall that STRIDE1 option controls whether the local memory transpose is performed in P3DFFT or delegated to the FFT sublibrary. There are pros and cons for each choice. On the one hand, delegating the transpose to the FFT library allows it to optimize cache flow in combination with the FFT itself. On the other hand, in this case the non-stride-1 access is needed both on input and output of the FFT. If the data is transposed and written to a temporary buffer, as in P3DFFT, separately from the FFT, the out-of-cache access pattern is needed in only one of either input or output. The cost of this is an extra memory write/read, which is done in small chunks, guaranteeing that the data reside in cache. In general there is no clear winner between these two approaches, as many factors (such as the size of transforms, the cache architecture of a given machine, and the sophistication of a given FFT implementation's handling of non-stride-1 data) contribute. In many tests performance of the two approaches has been found to be quite close. In practice this choice is often tied up with how the data structures are set up in the application that uses P3DFFT (Fourier space arrays are ordered ZYX as opposed to XYZ by default, per Table 1).

2. *USEEVEN option.* In the previous section we mentioned a potential advantage of using the *MPI\_Alltoall* call to transpose arrays even if the buffers are of different sizes for all the tasks, by padding the buffers. P3DFFT provides the USEEVEN option to use this implementation, whereas the default is *MPI\_Alltoallv*. The effect of using this option is quite noticeable on Cray XT5 (see Figure 4 below), due to an abnormal difference between *MPI\_Alltoall* and *MPI\_Alltoallv* performance mentioned earlier [Schulz]. The USEEVEN option is therefore strongly recommended on Cray XT5 but is not necessary on other platforms.

3. *Processor grid dimensions and task placement.* Below we consider the main performance bottleneck for P3DFFT, namely, the two transposes involving all-to-all communication.

Let us assume that the messages are large enough so that the bandwidth of the network is the main bottleneck for interprocessor communication, rather than the latency. The all-to-all communication pattern in 3D FFT depends mainly on two

factors: bisection bandwidth of the network and the degree of contention between different messages in utilization of the network links. The latter contention is a function of a given network topology, as well as the efficiency of the all-to-all algorithm and its implementation in the system. Since current generation of P3DFFT uses `MPI_Alltoall(v)` (in order to be portable yet use each system's optimized all-to-all), in this paper we do not attempt to model network contention and related effects but focus on network throughput defined by the bisection bandwidth as a rough upper limit estimate. Some in-depth modeling work for the 3D torus architecture of the IBM BG/L network, taking into account contention effects, can be found in Chan et al. (2008), Kumar et al. (2008), and Pekurovsky et al. (2006).

First consider a 3D grid decomposed in one dimension. In the 3D FFT problem the number of floating point operations in the computation phases is  $O(N^3 \log(N))$ , while the data volume for all-to-all communication is  $O(N^3)$ . As the number of tasks (or cores)  $P$  is increased, the computation time scales as  $O(1/P)$  since the work is nearly evenly distributed over  $P$  tasks. An all-to-all exchange involves each task sending and receiving  $P - 1$  messages of size  $N^3/P^2$ , so the total volume of data is approximately  $m * N^3$  (here  $m$  is the size of each array element in bytes, and  $N^3$  can be substituted for the product of  $N_x * N_y * N_z$  for noncubic grids). By definition, in the all-to-all exchange every second message will cross the network bisection, and therefore the time for such a transfer is approximated by

$$(1) \quad T_{network} = mcN^3 / (2\sigma_{bi}(P)),$$

where  $\sigma_{bi}(P)$  is the bisection bandwidth of the network portion containing  $P$  tasks, and  $c$  is a constant containing the network contention and other effects.

As already mentioned, in a 2D decomposition there are two all-to-all exchanges involved, each performed within a ROW or COLUMN subcommunicator. Although they exchange the same volume of data, they have rather different properties due to the patterns of topological placement of tasks on the network. The ROW subcommunicators normally consist of tasks on adjacent nodes (or even within a single node). These nodes are going to end up somewhere close in their location on the network graph. However, each COLUMN subcommunicator will span tasks on nodes that are spread out across the network. Since performance characteristics of all-to-all exchanges substantially depend on how far the nodes are from each other on the network graph, typically the row and column exchanges have very different properties, so there is an asymmetry in their parameters. For this reason, when choosing the processor grid dimensions, a square grid is often not optimal, even if this may seem counterintuitive. The optimal choice depends on specifics of the architecture of a given platform. The dimensions should satisfy

$$(2) \quad M_1 * M_2 = P, M_1 \leq (N_x/2, N_y), \text{ and } M_2 \leq (N_y, N_z).$$

Assuming there is no problem with injecting many messages into the network, often the best dimensions are such that the ROW exchange is done entirely within the node, or among a few neighboring nodes. (This means that  $M_1 \ll M_2$ .) In such a scenario, the ROW exchange will be defined by memory bandwidth on the node and quite cheap in comparison with the COLUMN exchange. In contrast, in the case where  $M_1$  and  $M_2$  are of the same order, both ROW and COLUMN exchanges involve network traffic across nodes, resulting in a higher total cost.

*Therefore choosing  $M_1$  such that the ROW transpose occurs within one node, if feasible, appears to be a good strategy.*

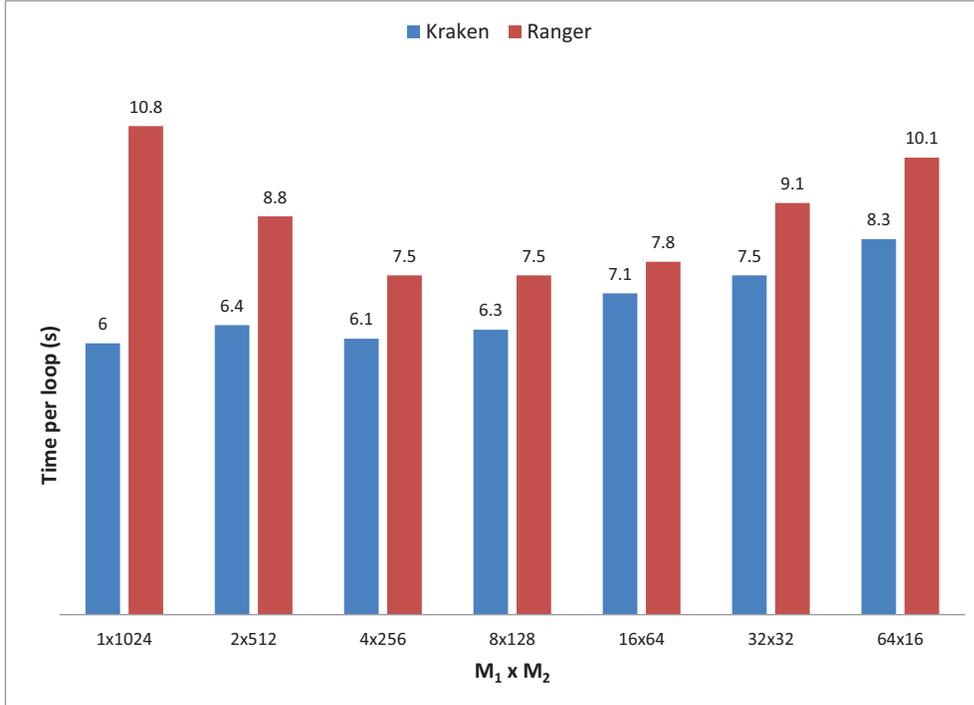


FIG. 3. Performance dependence on processor grid aspect ratio  $M_1 \times M_2$ . Each bar's height shows the time to complete one forward and one backward 3D FFT on a  $2048^3$  grid using 1024 cores on Cray XT5 (Kraken) and Sun/AMD (Ranger).

In Figure 3 we study the dependence of performance on processor grid dimensions. Data shown confirm the benefit of choosing  $M_1$  such that the ROW transpose occurs within a node. We see that the performance of P3DFFT can depend noticeably on the choice of processor grid aspect ratio. In this example (a  $2048^3$  grid on 1024 cores) the time to solution rises as  $M_1$  crosses the threshold equally to the number of cores per node (in this case, 12 for Kraken and 16 for Ranger). Note that the square grid  $32 \times 32$ , which may seem intuitively to be the best one, does not yield the optimal performance. The reason for the reduced performance on Ranger with low  $M_1$  is not clear at this time but is possibly related to more messages leaving each node in the COLUMN exchange when  $M_1$  is low.

It is interesting to note that in our tests on Cray XT5 (Kraken) at high core counts (see Figure 4) the situation is somewhat different, and performance favors more square-shaped processor grids. One hypothesis explaining this is a limitation on the number of messages that the Cray's SeaStar interconnect can handle at a given time.

We conclude that the choice of optimal processor grid dimensions appears to be architecture- and problem-specific. The value of  $M_1$  being close to the number of cores on the node is a good starting point in many cases; however, the user is advised to carry out a few test runs with higher and lower values for each case.

Assuming the above is the case, (1) is still a good approximation for total communication time, since now the COLUMN exchange follows the same pattern as the all-to-all exchange in 1D decomposition. Therefore, total execution time for the 3D FFT can be approximated by the following:

$$(3) \quad T_{FFT} = N^3[2.5\log(N)/(PF) + bm/(P\sigma_{mem}) + cm/(2\sigma_{bi}(P))].$$

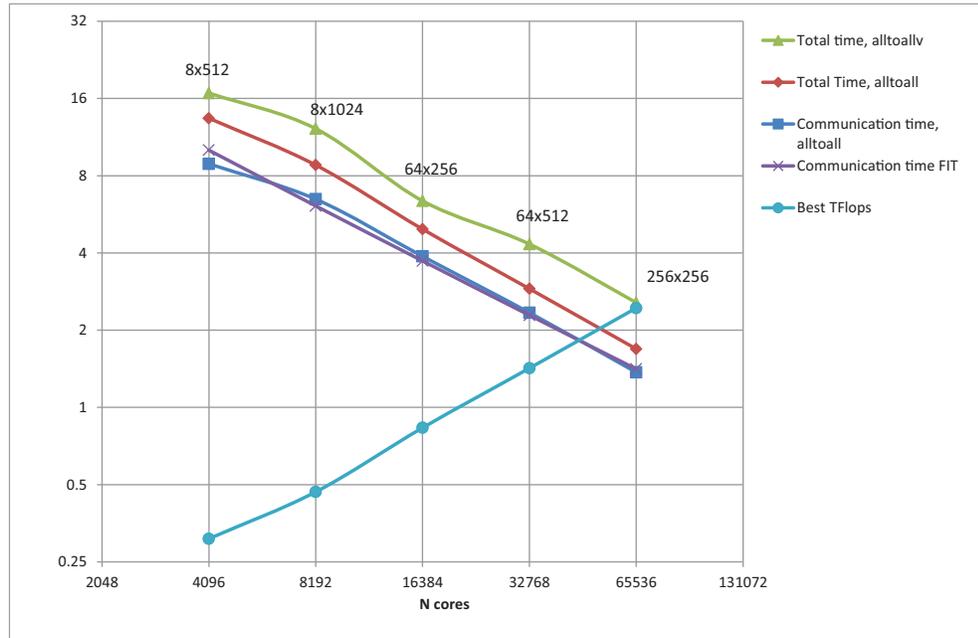


FIG. 4. Strong scaling of P3DFFT benchmark for a  $4096^3$  grid in double precision on Cray XT5. This log-log plot compares performance of versions implemented with `MPI_Alltoall` (`USEEVEN` option, red diamonds) and `MPI_Alltoallv` (default, green triangles) for parallel transposes. Both versions use the `STRIDE1` option; however, results without `STRIDE1` are very close to these numbers. Also shown is the communication time corresponding to the `MPI_Alltoall` version (blue squares) and a calculated fit to this data of the function  $a/P + b/(P^2/3)$  (magenta crosses). Numbers next to each data point show the processor grid dimensions used in obtaining the timing, corresponding to best result among several geometries explored for each case. Time in both seconds (for a forward/backward transform pair) and TeraFlops (achieved number of floating point operations per second, in  $10^{12}$ ) is plotted. (TeraFlop numbers are based on best results with all-to-all).

Here the parameter  $b$  contains the number of memory accesses per data element, both in FFT operations and in all the local and nonlocal transposition steps, while  $\sigma_{mem}$  is memory bandwidth per task on a node. Parameter  $F$  reflects the floating point operations per second in FFT. The values of coefficients  $F$ ,  $b$ , and  $c$  vary depending on the system hardware, as well as implementation of MPI and the FFT library. The above is a model that will be used in the following sections to fit the measured timing data. For cases when  $M_1$  is larger than the number of cores on the nodes, both ROW and COLUMN exchanges involve network traffic, but even in the worst case both of them are limited by bisection bandwidth, and therefore (3) should still hold.

In our studies to date, the default placement of tasks on the physical network (i.e., cores on a node are populated with contiguous task IDs) has proven to be optimal, compared with several other task mapping algorithms, when performing 3D FFT on cubic grids. Essentially this is due to the fact that all-to-all operations are asymmetric and not sensitive to task placement. However, for noncubic grids the situation is different. An interesting study was reported by an ANL group (Chan et al. (2008)) where up to 48% improvement in performance was found due to optimized task placement based on the network topology when using P3DFFT with noncubic domains on an IBM BG/L. While further work here is needed, in this paper we focus on results obtained when using simple contiguous task placement and study cubic grids only.

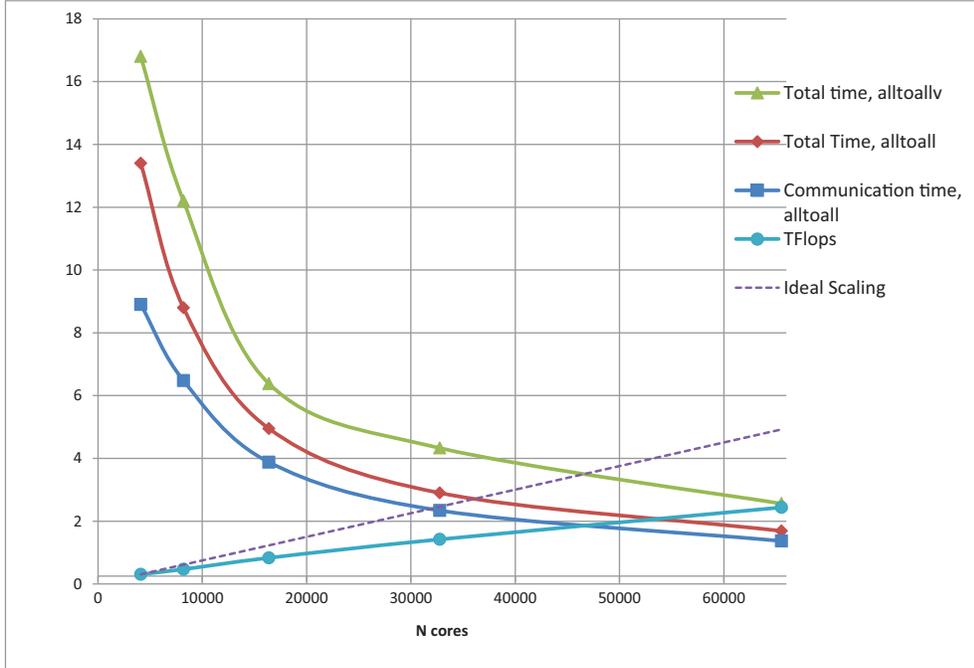


FIG. 5. Same data as in Figure 4 plotted with linear (instead of logarithmic) axes.

**4.3. Benchmark results and discussion.** Figures 4 and 5 show strong scaling of the P3DFFT forward-and-backward 3D FFT test of size  $4096^3$  using double precision on Cray XT5 (Kraken). Since we have seen that the processor grid aspect ratio has some effect on performance, only the best  $M_1 \times M_2$  combination is taken as a data point for each core count. It is clear from these figures that communication time is a substantial part of the total execution time.

Recall that this platform has 3D torus interconnect, and therefore bisection bandwidth scales asymptotically as  $O(P^{2/3})$ . Therefore (3) becomes

$$(4) \quad T_{FFT} = a/P + d/P^{2/3}.$$

A least-squares fit to the data to a curve defined by (4) produces an excellent match. Taking the value of coefficient of the  $P^{2/3}$  term obtained from the fit, assuming that half of all messages will need to pass through the network bisection, and considering that there are two transposes, we obtain a value of 212 GB/s for effective bandwidth for the network portion containing 65,536 cores (or 5,462 nodes). It is not easy to contrast this with a predicted value since the placement of tasks on the torus is unknown. If we assume a  $15 \times 16 \times 24$  partition with wraparound torus links in the second and third dimensions, and a peak bandwidth of 9.6 GB/s per link, the expected bisection bandwidth is  $16 \times 24 \times 9.6$  GB/s = 3,686 GB/s, which implies about 6% efficiency. This number parameterizes network contention and any other loss of efficiency in the network. This is a rough estimate and should be taken with caution due to many unknowns, such as how the nodes were grouped together, and whether other users' traffic affected the network throughput. However, the results of this analysis are in agreement with the assumption that communication time scaling is inversely proportional to bisection bandwidth.

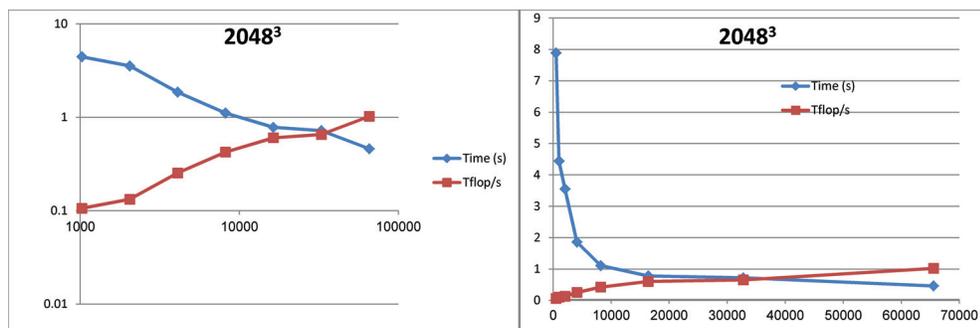


FIG. 6. Performance of P3DFFT on a  $2048^3$  transform, running on Cray XT5. Both time in seconds (for a forward/backward transform pair) and number of floating points per second (in  $10^{12}$ , or TeraFlops) are plotted, and both linear and log-log plots are presented.

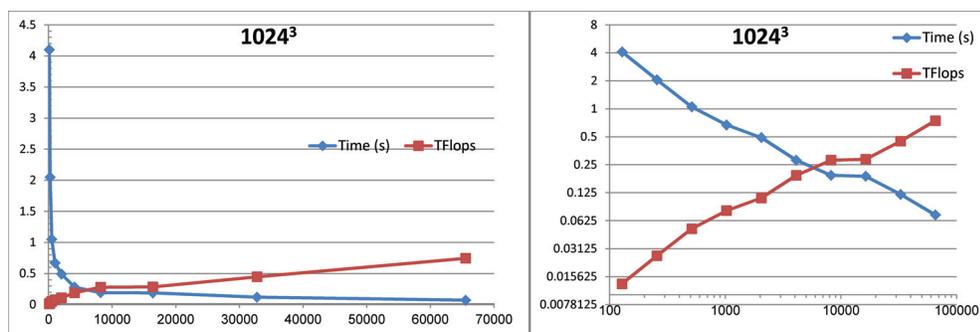


FIG. 7. Performance of P3DFFT on a  $1024^3$  transform, running on Cray XT5. Both time in seconds (for a forward/backward transform pair) and number of floating points per second (in  $10^{12}$ , or TeraFlops) are plotted, and both linear and log-log plots are presented.

P3DFFT performs and scales reasonably well not only for large transforms but also for medium grid sizes. Figures 6–8 present benchmark results for tests with linear grid sizes 2048, 1024, and 512.

A weak scaling study is presented in Figure 9. We compare timings for five cases, involving 16, 128, 1024, 8,192, and 65,536 compute cores, with corresponding grid sizes  $512^3$ ,  $1024^3$ ,  $2048^3$ ,  $4096^3$ , and  $8192^3$ . Computational intensity of the 3D FFT is  $O(N^3 \log(N))$ . Therefore, the weak scaling can be approximately defined by increasing the core count eight times with each two-fold increase in grid size, and including a factor of  $\log(N)$  in the efficiency. With this definition P3DFFT achieves parallel efficiency of 45% over the range of core counts from 128 to 65,536.

Finally, we touch on advantages and disadvantages of using 2D decomposition versus 1D decomposition in 3D FFT at large and moderate scales. Both 1D and 2D decompositions are implemented by P3DFFT (the 1D version is simply a special case of the 2D version with virtual processor grid geometry  $1 \times P$ ). While the 2D version is free from the scaling limitation of slabs distribution, at moderate scales (namely,  $P \leq N$ ) the 1D decomposition may be better, and our study in Figure 10 confirms this. The main reason for the difference is that in the 1D case there is only one transpose instead of two. The difference gets smaller as the core count  $P$  is increased,

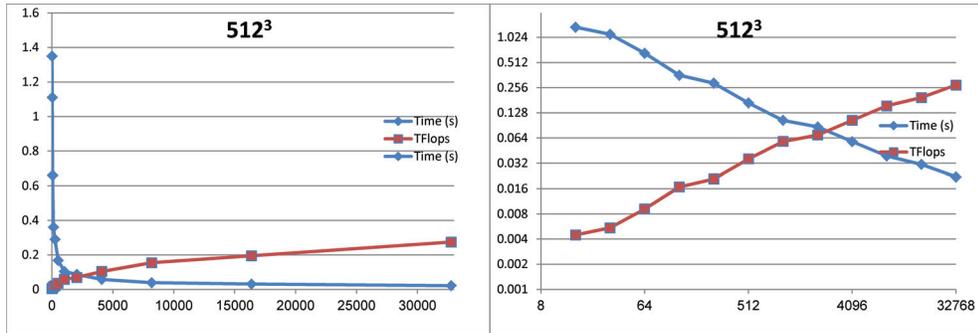


FIG. 8. Performance of P3DFFT on a 5123 transform, running on Cray XT5. Both time in seconds (for a forward/backward transform pair) and number of floating points per second (in  $10^{12}$ , or TeraFlops) are plotted, and both linear and log-log plots are presented.

### Weak scaling

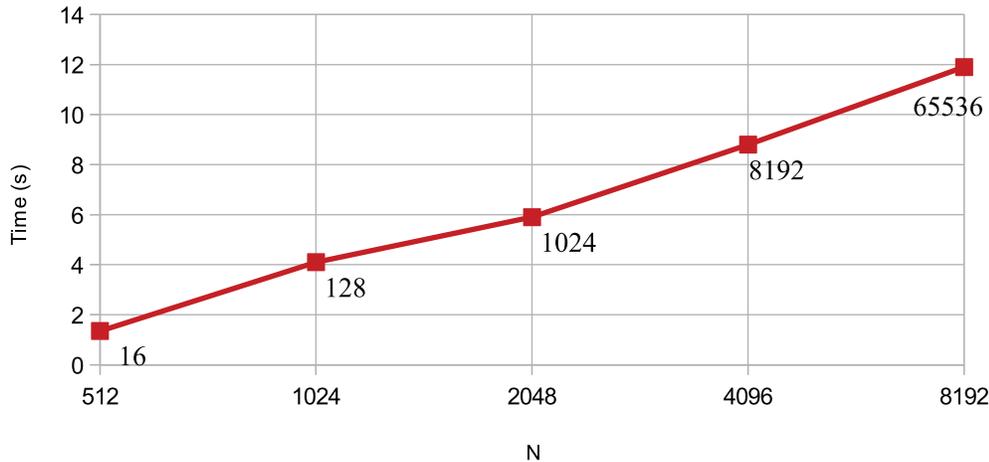


FIG. 9. Weak scaling of P3DFFT test on Cray XT5. The horizontal axis shows linear data grid size. The numbers next to data points indicate the core count. This is a log-linear plot.

becoming negligible at  $P = N$ . As  $P$  goes beyond  $N$ , it is clear that the 1D version cannot scale; however, the 2D version continues to scale. Therefore, the main reason for using the 2D approach is maintaining scalability beyond the  $P = N$  point.

**5. Conclusion and related work.** In this paper we have presented P3DFFT, a software package for 3D FFTs and related algorithms on parallel computers. The algorithm is implemented using 2D decomposition, which significantly increases the range of scaling compared with 1D decomposition used in many other implementations. The design of P3DFFT followed the goal of portability and performance and is based on a thorough set of experiments.

The main factors affecting performance were analyzed in this paper in a way that helps guide the user in choosing optimal parameters for their run. The paper also contains some benchmark data which agree well with an asymptotic model based on

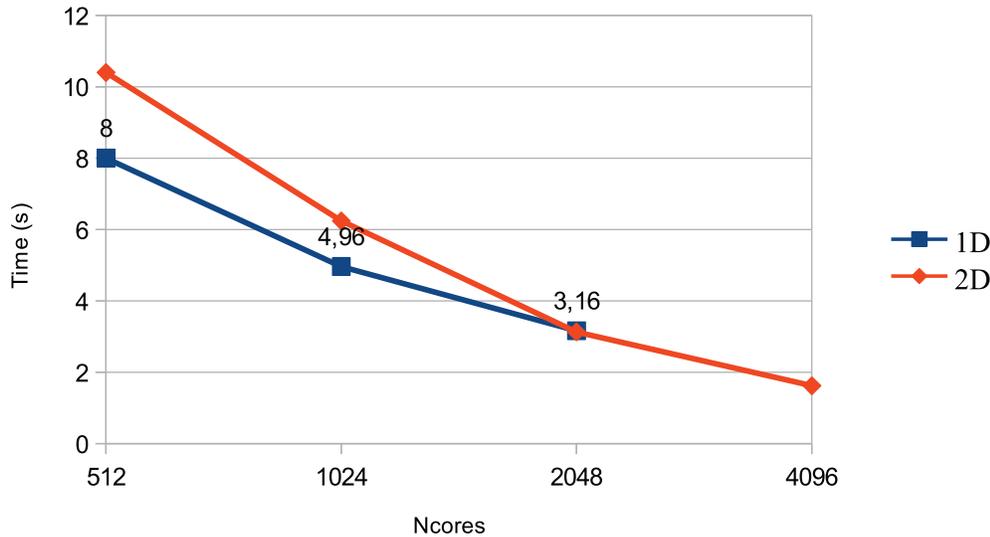


FIG. 10. Scaling of 1D and 2D versions of the test FFT program. This study was done on Cray XT5 (Kraken) using  $2048^3$  grid in double precision, with GNU compilers. There is no 1D data at 4096 cores since the number of tasks exceeds the number of slabs available (in this case 2048).

the network bisection bandwidth for the studied architectures. The reader is given a realistic level of expected performance achievable with P3DFFT at high scales.

On a Cray XT5 system P3DFFT achieves a weak scaling efficiency of 45% when scaling from 128 to 65,536 compute cores. Loss of efficiency is attributed to the difficulty of maintaining scalable sustained bisection bandwidth on systems with core counts reaching into tens and hundreds of thousands. While this tendency is likely to be exacerbated in the future, at present this is a reasonable level of scalability to enable many critical computations at large core counts. P3DFFT has already found application in many scientific projects (Donzis, Yeung, and Pekurovsky (2008), Donzis and Yeung (2010), Homann et al. (2009), Homann et al. (2010), Chandy and Frankel (2009), Schumacher and Putz (2007), Schumacher (2009), Peters et al. (2010), Bodart (2009), Grafke (2008), Weidauer et al. (2010), Laizet et al. (2010), and Schaeffer et al. (2010)).

Studies involving Unified Parallel C (UPC) implementation of 3D FFT have been reported in a number of papers including Bell et al. (2006) and Nishtala et al. (2009). The authors report seeing an advantage in performance with UPC over MPI (using `MPI_isend` and `MPI_irecv`) at medium scales, which they attribute to overlap of communication with computation. By using communication protocols such as GASNET that are closer to the low-level network fabric than MPI, a higher efficiency of overlap is achieved. Another work with communication overlap in 3D FFT has been reported in Kandalla et al. (2011), where a gain in performance is shown due to overlap of communication and computation using `MPI_Put` calls. In general the ability to achieve overlap is highly dependent on the capabilities of the hardware and software of a given system. It may be also noted that while this avenue deserves careful attention and study, measurements on production of high-end systems (for example, Cray XT) show that on the order of 80% of the total time is spent in communication at high core counts with 3D FFT, which unfortunately limits the gains achievable with overlap of communication and computation in this case.

Several teams have studied hybrid MPI/OpenMP implementation of pseudospectral CFD codes employing 3D FFTs [Mininni, Gorobets, Takahashi 2006, Itakura]. Using hybrid code may potentially help performance somewhat by aggregating messages in all-to-all exchanges, although it is not likely to relieve the fundamental problem of limited bisection bandwidth. The downside is possible performance loss due to OpenMP overhead and false sharing, as well as increased level of programming difficulty. While questions remain about whether the hybrid approach provides a decisive advantage over pure MPI, it is being investigated and may be implemented in later versions of P3DFFT package.

As mentioned earlier, P3DFFT implements several transform types in the third dimension, including Chebyshev and empty transform, for those cases where the third dimension requires special treatment with respect to the first two. Future evolution of the package will follow the need of its users, and may include increased flexibility of array layout, as well as a versatile collection of isolated array transpose calls. This will broaden the range of applicability of P3DFFT in scientific computing.

**Acknowledgments.** The author would like to thank P. K. Yeung, D. Donzis, R. Schulz, and G. Brethouwer for helpful discussions.

#### REFERENCES

- R.C. AGARWAL, F.G. GUSTAVSON, AND M. ZUBAIR (1994), *An efficient parallel algorithm for the 3-D FFT NAS parallel benchmark*, in Proceedings of the Scalable High-Performance Computing Conference, pp. 129–133.
- C. BELL, D. BONACHEA, R. NISHTALA, AND K. YELICK (2006), *Optimizing bandwidth limited problems using one-sided communication and overlap*, in Proceedings of the Parallel and Distributed Processing Symposium (IPDPS 2006).
- J. BODART (2009), *Large scale simulation of turbulence using a hybrid spectral/finite difference solver*, in Parallel Computational Fluid Dynamics: Recent Advances and Future Directions, DE Stech Publications, Lancaster, PA, pp. 473–482.
- A. CHAN, P. BALAJI, W. GROPP, AND R. THAKUR (2008), *Communication analysis of parallel 3D FFT for flat cartesian meshes on large Blue Gene systems*, in Proceedings of the 15th International Conference on High Performance Computing, Bangalore, India, pp. 350–364.
- A.J. CHANDY AND S.H. FRANKEL (2009), *Regularization-based sub-grid scale (SGS) models for large eddy simulations (LES) of high-Re decaying isotropic turbulence*, Journal of Turbulence, 10, p. 1.
- J.W. COOLEY AND J.W. TUKEY (1965), *An algorithm for the machine calculation of complex Fourier series*, Math. Comput., 19, pp. 297–301.
- D.A. DONZIS, P.K. YEUNG, AND D. PEKUROVSKY (2008), *Turbulence simulations on  $O(10^4)$  processors*, in TeraGrid’08 Conference, Las Vegas, NV.
- D.A. DONZIS AND P.K. YEUNG (2010), *Resolution effects and scaling in numerical simulations of passive scalar mixing in turbulence*, Phys. D, 239, pp. 1278–1287.
- M. ELEFTHERIOU, B. FITCH, A. RAYSHUBSKIY, T.J.C. WARD, AND R.S. GERMAIN (2005), *Performance measurements of the 3D FFT on the Blue Gene/L supercomputer*, in Euro-Par 2005 Parallel Processing: 11th International Euro-Par Conference, Lecture Notes in Comput. Sci. 3648, J.C. Cunha and P.D. Medeiros, eds. Springer-Verlag, Berlin, pp. 795–803.
- I. FOSTER AND P. WORLEY (1997), *Parallel algorithms for the spectral transform method*, SIAM J. Sci. Comput., 18, pp. 806–837.
- M. FRIGO AND S.G. JOHNSON (2005), *The design and implementation of FFTW3*, Proc. IEEE, 93, pp. 216–231.
- A. GOROBETS, F.X. TRIAS, R. BORRELL, O. LEHMKUHL, AND A. OLIVA (2011), *Hybrid MPI + OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction*, Computers Fluids, 49, pp. 101–109.
- G. GRAFKE (2008), *Numerical simulations of possible finite time singularities in the incompressible Euler equations: Comparison of numerical methods*, Phys. D, 237, pp. 1932–1936.
- J. HEIN, H. JAGODE, U. SIGRIST, A. SIMPSON, AND A. TREW (2008), *Parallel 3D-FFTs for multicore nodes on a mesh communication network*, in Proceedings of CUG, 2008.
- H. HOMANN, O. KAMPS, R. FRIEDRICH, AND R. GRAUER (2009), *Bridging from Eulerian to La-*

- grangian statistics in 3D hydro- and magnetohydrodynamic turbulent flows*, New J. Phys., 11, 073020.
- H. HOMANN, J. BEC, AND R. GRAUER, *DNS of finite-size particles in turbulent flows* (2010), in Proceedings of the John von Neumann Institute for Computing NIC Symposium, Juelich, Germany, 2010, pp. 357–364.
- K. ITAKURA, A. UNO, M. YOKOKAWA, T. ISHIHARA, AND Y. KANEDA (2004), *Scalability of hybrid programming for a CFD code on the Earth Simulator*, Parallel Comput., 30, pp. 1329–1343.
- H. JAGODE (2008), *Custom assignment of MPI ranks for parallel multi-dimensional FFTs: Evaluation of BG/P versus BG/L*, in Parallel and Distributed Processing with Applications, ISPA '08, pp. 271–283.
- Jaguar, <http://www.nccs.gov/computing-resources/jaguar/>
- K. KANDALLA, H. SUBRAMONI, K. TOMKO, D. PEKUROVSKY, S. SUR, AND D. PANDA (2011), *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, Volume 26, Issue 3, Pages 237–246 (ISC'11)
- KRAKEN, <http://www.nics.tennessee.edu/?q=node/38>.
- I. KIRKER (2009), *Demanding Parallel FFTs: Slabs & Rods*, Master's Thesis, EPCC, available online from [http://www.epcc.ed.ac.uk/wp-content/uploads/2009/01/Ian\\_Kirker.pdf](http://www.epcc.ed.ac.uk/wp-content/uploads/2009/01/Ian_Kirker.pdf)
- S. KUMAR, Y. SABHARWAL, R. GARG, AND P. HEIDELBERGER (2008), *Optimization of all-to-all communication on the Blue Gene/L supercomputer*, in Proceedings of the 37th International Conference on Parallel Processing, 2008, pp. 320–329.
- S. LAIZET, E. LAMBALLAIS, AND J. C. VASSILICOS (2010), *A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence*, Computers Fluids, 39, pp. 471–484.
- P.D. MININNI, D.L. ROSENBERG, R. REDDY, AND A. POUQUET (2011), *A hybrid MPI-OpenMP scheme for scalable parallel pseudospectral computations for fluid turbulence*, Parallel Comput., 37, pp. 316–326.
- NAG (Numerical Algorithms Group) library, <http://www.nag.com>.
- R. NISHTALA, P. HARGROVE, D. BONACHEA, AND K. YELICK (2009), *Scaling communication-intensive applications on BlueGene/P using one-sided communication and overlap*, in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2009), IEEE, Washington, DC, pp. 1–12.
- D. PEKUROVSKY, P.K. YEUNG, D. DONZIS, S. KUMAR, W. PFEIFFER, AND G. CHUKKAPALLI (2006), *Scalability of a pseudospectral DNS turbulence code with 2D domain decomposition on Power4+/Federation and Blue Gene systems*, in Proceedings of ScicomP12, Boulder, CO, available online from [http://www.spscicomp.org/ScicomP12/www.cisl.ucar.edu/info/scicomp/Presentations%20CD/7-20\\_Thursday/Session%201/Pekurovsky.pdf](http://www.spscicomp.org/ScicomP12/www.cisl.ucar.edu/info/scicomp/Presentations%20CD/7-20_Thursday/Session%201/Pekurovsky.pdf).
- PESSL (Parallel Engineering and Scientific Subroutine Library), [http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.essl.doc/pessl\\_aix33/am60130515.html](http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.essl.doc/pessl_aix33/am60130515.html).
- D. PEKUROVSKY, *P3DFFT Manual*, [http://code.google.com/p/p3dfft/downloads/detail?name=P3DFFT\\_USER\\_GUIDE\\_2.4.pdf](http://code.google.com/p/p3dfft/downloads/detail?name=P3DFFT_USER_GUIDE_2.4.pdf).
- N. PETERS, L. WANG, J.-P. MELLADO, J. H. GÖBBERT, M. GAUDING, PH. SCHÄFER, AND M. GAMPERT (2010), *Geometrical properties of small scale turbulence*, in Proceedings of the John von Neumann Institute for Computing NIC Symposium, Juelich, Germany, pp. 365–371.
- S. PLIMPTON, *Parallel FFT Package*, <http://www.sandia.gov/~sjplimp/docs/fft/README.html>.
- RANGER, <http://services.tacc.utexas.edu/index.php/ranger-user-guide>
- P. SCHAEFFER, M. GAMPERT, J. H. GÖBBERT, L. WANG, AND N. PETERS (2010), *Testing of model equations for the mean dissipation using Kolmogorov flows*, Flow, Turbulence and Combustion, 85 (2010), pp. 225–243.
- J. SCHUMACHER AND M. PUTZ (2007), *Turbulence in laterally extended systems*, in Parallel Computing: Architectures, Algorithms and Applications, C. Bischof, M. Buckner, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, and F. Peters, eds., NIC Ser., 38, John von Neumann Institute for Computing, Juelich, Germany, pp. 585–592.
- R. SCHULTZ, *3D FFT with 2D Decomposition*, Technical report available online from <http://cmb.ornl.gov/members/z8g/csproject-report.pdf>.
- J. SCHUMACHER (2009), *Lagrangian studies in convective turbulence*, Phys. Rev. E, 79, 056301.
- R. SWEET, W. BRIGGS, S. OLIVEIRA, J. PORSCHKE, AND T. TURNBULL (1991), *FFT's and three-dimensional Poisson solvers for hypercubes*, Parallel Computing, 17, pp. 121–131.
- P. SWARZTRAUBER (1987), *Multiprocessor FFTs*, Parallel Comput., 5, pp. 197–210.
- D. TAKAHASHI (2006), *A hybrid MPI/OpenMP implementation of a parallel 3-D FFT on SMP clusters*, in Parallel Processing and Applied Mathematics, Lecture Notes in Comput. Sci., 3911, Springer, Berlin, pp. 970–977.

- D. TAKAHASHI (2010), *An implementation of parallel 3-D FFT with 2-D decomposition on a massively parallel cluster of multi-core processors*, in *Parallel Processing and Applied Mathematics*, Lecture Notes in Comput. Sci., 6067, Springer, Berlin, pp. 606–614.
- P. WAPPEROM, A.N. BERIS, AND M.A. STRAKA (2006), *A new transpose split method for three-dimensional FFTs: Performance on an Origin2000 and Alphaserver cluster*, *Parallel Comput.*, 32, pp. 1–13.
- T. WEIDAUER, C. JUNGHANS, O. PAULUIS, M. PÜTZ, AND J. SCHUMACHER (2010), *Shallow moist convection*, in *Proceedings of the John von Neumann Institute for Computing NIC Symposium*, Juelich, Germany, pp. 373–380.