Communication Analysis of Parallel 3D FFT for Flat Cartesian Meshes on Large Blue Gene Systems

A. Chan, P. Balaji, W. Gropp and R. Thakur

Preprint
Argonne National Laboratory #

Communication Analysis of Parallel 3D FFT for Flat Cartesian Meshes on Large Blue Gene Systems*

A. Chan[1], P. Balaji[2], W. Gropp[3], and R. Thakur[2]

[1] ASCI FLASH Center, University of Chicago
chan@mcs.anl.gov

[2] Math. and Comp. Sci. Division, Argonne National Laboratory
{balaji,thakur}@mcs.anl.gov

[3] Dept. of Computer Science, University of Illinois, Urbana-Champaign
wgropp@illinois.edu

Abstract. Parallel 3D FFT is a commonly used numerical method in scientific computing. P3DFFT is a recently proposed implementation of parallel 3D FFT that is designed to allow scalability to massively large systems such as Blue Gene. While there has been recent work that demonstrates such scalability on regular cartesian meshes (equal length in each dimension), its performance and scalability for flat cartesian meshes (much smaller length in one dimension) is still a concern. In this paper, we perform studies on a 16-rack (16384-node) Blue Gene/L system that demonstrates that a combination of the network topology and the communication pattern of P3DFFT can result in early network saturation and consequently performance loss. We also show that remapping processes on nodes and rotating the mesh by taking the communication properties of P3DFFT into consideration, can help alleviate this problem and improve performance by up to 48% in some special cases.

1 Introduction

Fast Fourier Transform (FFT) has been one of the most popular and widely used numerical methods in many areas of scientific computing, including digital speech and signal processing, solving partial differential equations, molecular dynamics [3], many-body simulations and monte carlo simulations [1, 2, 14]. Given its importance, there have been a large number of libraries that provide different implementations of FFT (both sequential and parallel) aimed at achieving high-performance in various environments. FFTW [15], IBM PESSL [13], and the Intel Math Kernel Library (MKL) [9] are a few examples of such implementations. P3DFFT [6] is a recently proposed parallel implementation of 3D FFT

* This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. We also acknowledge IBM for allowing us to use their BG-Watson system for our experiments. Finally, we thank Joerg Schumacher for providing us his test code that allowed us to understand the scalability issues with P3DFFT on flat cartesian meshes.

that is designed to allow scalability for large problem sizes on massively large systems such as Blue Gene (BG) [16]. It aims at achieving such scalability by limiting communication to processes in small local sub-communicators instead of communicating with all processes in the system.

While there has been previous work that demonstrates the scalability of P3DFFT for regular 3D cartesian meshes, where all dimensions of the mesh are of equal length [7], its inability to achieve similar scalability for flat 3D cartesian meshes, where one dimension is much smaller than the other two, is a known problem [18]. Flat 3D cartesian meshes are a good tool in studying quasi-2D systems that occur during the transition of 3D systems to 2D systems (e.g., in superconducting condensate [17], Quantum-Hall effect [20] and turbulence theory in geophysical studies [19]). Thus, such loss of scalability can be a serious problem that needs to be addressed.

In this paper, we analyze the performance of P3DFFT for flat 3D cartesian meshes on a large 16-rack (16384-node) Blue Gene/L (BG/L) system. Specifically, we perform detailed characterization of the communication pattern used by P3DFFT and its behavior on the BG network topology. We observe that a combination of the network topology and the communication pattern of P3DFFT can result in parts of the communication to over-saturate the network, while other parts under-utilize the network. This causes overall loss of performance on large-scale systems. We also show that carefully remapping processes on nodes and rotating the mesh by taking the communication properties of P3DFFT into consideration can help alleviate this problem. Our experimental results demonstrate up to 48% improvement in performance in some cases.

2 Overview of Parallel 3D FFT Techniques

FFT [8] is an efficient algorithm to compute the Discrete Fourier Transform (DFT) and its inverse. Fourier transform consists of a forward and a backward transform. The forward operation transforms a function *f(x)* in real space *X* to a function *F(k)* in Fourier space *K*. The backward transform does the reverse operation that transforms *F(k)* in Fourier space *K* to *f(x)* in real space *X*. In this section, we will mainly discuss the forward fourier transform, but the backward fourier transform can be similarly performed by reversing the steps in the forward transform.

A typical 3D forward fourier transform for a real-space function *f(x,y,z)* can be expressed as follows:

$$f(k_x, k_y, k_z) = \sum_z \left[\sum_y \left[\sum_x f(x, y, z) \cdot e^{ik_x x} \right] e^{ik_y y} \right] e^{ik_z z} \tag{1}$$

The goal here is to perform a 1D fourier transform on each of the three dimensions of the 3D data mesh, distributed over P processes. There are two basic approaches for doing this [11], distributed FFT and transpose-based FFT. Distributed FFT relies on a parallel implementation of 1D-FFT, with each process communicating the necessary data with other processes. Transpose-based FFT, on the other hand, relies on a sequential version of 1D-FFT that performs

the transform on one dimension at a time, and transposing the data grid when needed. There are two different transpose-based FFT strategies for 3D meshes, which differ in their data decomposition pattern. Let us consider a data grid of size: $n_x \times n_y \times n_z$.

- *1D Decomposition:* In the 1D data decomposition technique, the data grid is divided across P processes such that each process gets a 2D slab of the grid (size of $n_x \cdot n_y \cdot n_z / P$). Each process carries out a typical sequential 2D-FFT on its local slab, and thus does not require any communication during this operation. Once the 2D-FFT has completed, it transposes the mesh using an `MPI_Alltoallv()` operation. This allows it to receive data corresponding to the third dimension, on which a 1D-FFT is applied. Thus, only one global transpose is used in this technique. However, the drawback is that it only scales $\texttt{max}(n_x, n_y, n_z)$ number of processes.
- *2D Decomposition:* In the 2D data decomposition technique (shown in Figure 1), one face (2D) of the mesh is divided over $P = P_{row} \times P_{col}$ processes, so each process contains a column (pencil) of the data mesh of size $n_x \times (n_y/P_{row}) \times (n_z/P_{col})$. Each process first performs a 1D-FFT along the length of the column (say x-axis). Then it does a transpose on the remaining two axes (y- and z-axis) and performs 1D-FFT on the y-axis. Finally, it performs a transpose on the y- and z-axes and performs a 1D-FFT on the z-axis. Two transposes are performed altogether. P3DFFT uses this strategy as it can theoretically scale up to $n_x \cdot n_y \cdot n_z$ / $\texttt{min}(n_x, n_y, n_z)$ processes.

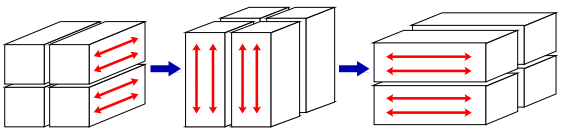

Fig. 1. 2D Decomposition: 1D FFT in each dimension followed by a transpose.

Neither of the transpose based FFT techniques allows for easy overlap of communication and computation as the transpose where the communication takes places has to be finished before the local 1D-FFT can be carried out.

3 Related Work

A number of implementations of Parallel 3D FFT exist. FFTW [4] has been a popular implementation of parallel 3D FFT. While there has been prior literature [10] that identified issues with its performance and improved its scalability to some extent, FFTW itself relies on 1D decomposition (described in Section 2) which allows it to only scale up to a theoretical limit of $\texttt{max}(n_x, n_y, n_z)$ number of processes. That is, with a problem size of 4096^3, FFTW cannot use more than 4096 processors. Thus, it is not ideal to use on massively parallel systems such as BG which support hundreds of thousands of processors. P3DFFT has recently been proposed to deal with such scalability issues and allow 3D FFT to be effectively used on such systems.

Like P3DFFT, IBM recently proposed an alternate implementation of Parallel 3D FFT, specifically for their BG system, known as BGL3DFFT [12]. However, BGL3DFFT has several limitations. First, it is a closed source implementation that restricts much utility for open research. Second, owing to its lack of Fortran support, it has not gained too much popularity in mainstream scientific computing. Third, while there is published literature that shows its scalability for small 3D grids (up to 128×128×128) [12], there is no evidence of its scalability for larger problem sizes. Keeping the drawbacks of BGL3DFFT aside, we believe that the problems in handling flat cartesian problems exist in the BGL3DFFT implementation as well, and that our observations are relevant there too.

There is also previous literature that shows that P3DFFT scales reasonably well with large regular cubical 3D meshes [7]. However, recently, Joerg Schumacher pointed out the importance of 3D-FFT on flat cartesian meshes where $n_x = n_y > n_z$ in his crossover study from 3D to quasi-2D turbulence systems [18] and found that 3D-FFT on flat cartesian meshes does not scale as well as regular cartesian meshes. Our paper uses Joerg's study as a motivation to understand the scalability issues of P3DFFT for flat cartesian meshes.

4 Communication Overheads in P3DFFT

In this section, we first present relevant details about the BG network in Section 4.1. We next present the communication characteristics of P3DFFT in Section 4.3 and an analysis of network saturation caused by such communication in Section 4.2.

4.1 BG/L Network Overview

BG/L has five different networks [5]. Two of them (1G Ethernet and 100M Ethernet with JTAG interface) are used for file I/O and system management while the other three (3-D Torus Network, Global Collective Network and Global Interrupt Network) are used for MPI communication. The 3-D torus network is used for point-to-point MPI and multicast operations and connects all compute nodes to form a 3-D torus; thus, each node has six neighbors. Each link provides a bandwidth of 175 MB/s per direction for a total of 1.05 GB/s bidirectional bandwidth per node.

4.2 Analyzing Network Saturation Behavior in P3DFFT on BG/L

As described earlier, unlike regular clusters that use switched network fabrics, the Blue Gene family of supercomputers relies on a torus network for interconnectivity. Thus, each node is directly connected with only six other nodes. To reach any other node, the message has to traverse more than one link; this leads to network link sharing by multiple messages, leading to network saturation.

Since P3DFFT does not directly perform communication with all processes in the system, but rather communicates only with processes in its row and column sub-communicators, the network saturation behavior is tricky. In Figure 2(a), we show the mapping of the processes in the row and column sub-communicators

to the physical torus on BG/L. This example considers a system size of 512 processes, with the row sub-communicator containing 32 processes and the column sub-communicator containing 16 processes, i.e., a 32×16 process grid. Thus, the first row would have processes 1 to 32, the second row would have processes 33 to 64 and so on.

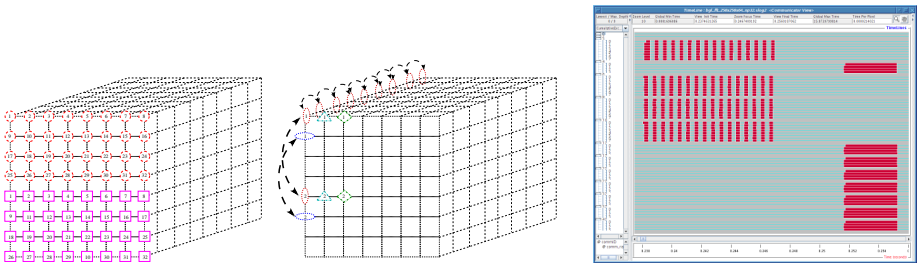

Fig. 2. (a) Mapping the row and column communicator processes in a 2D process grid to a 3D torus; (b) Jumpshot's communicator view on P3DFFT's communication.

Note that the size of different dimensions in the BG/L torus is fixed based on the available allocation. In this case, we pick a torus topology of $8 \times 8 \times 8$. Therefore, the first row of processes in the process grid (1 to 32) map to the first four physical rows on the BG/L torus (shown as red circles in Figure 2(a)). Similarly, the second row of processes in the process grid (33 to 64) map to the next four physical rows (shown as pink rectangles). It is to be noted that all processes in the row communicator are always allocated adjacent to each other. That is, any communication within the row sub-communicator will not require the message to go outside these four rows.

The mapping of the processes corresponding to the column communicator is, unfortunately, more complicated than the row communicator. Processes corresponding to the first column are 1, 33, 65, 97, etc. These processes are not all topologically adjacent. In other words, as shown in Figure 2, messages traversing the non-adjacent portions of the column communicator have to pass through more links, oftentimes contending with messages from other communicators, and can thus saturate the network significantly faster as compared to the row communicator.

4.3 Communication Characterization of P3DFFT

Consider a 3D data grid of size $N = n_x \times n_y \times n_z$ which needs to be solved on a system with P processes. P3DFFT decomposes the 3D grid into a 2D processor mesh of size $P_{row} \times P_{col}$, where $P_{row} \times P_{col} = P$. It splits the 2D processor mesh into two orthogonal sub-communicators—one in each dimension. Thus, each process will be a part of a *row* and a *column* sub-communicator. As shown in Figure 2(b), the first global transpose of the forward 3D-FFT consists of n_z/P_{col} iterations of `MPI_Alltoallv` over the row sub-communicator (the short red states), with the message count per process-pair being m_{row} defined in Equation 2. The total message count per process for the first transpose becomes $n_x \cdot n_y \cdot n_z/(P_{row} \cdot P_{col})$.

$$m_{row} = \frac{n_x \cdot n_y}{P_{row}^2} = \frac{N}{n_z \cdot P_{row}^2} \tag{2}$$

The second global transpose consists of one single iteration of `MPI_Alltoallv` over the column communicator (the long red states in Figure 2(b)), with message count per process being $n_x \cdot n_y \cdot n_z/(P_{row} \cdot P_{col})$, which is the same as the first transpose. The corresponding message count per process-pair is m_{col}, where

$$m_{col} = \frac{n_x \cdot n_y \cdot n_z}{P_{row} \cdot P_{col} \cdot P_{col}} = \frac{N \cdot P_{row}}{P^2} \tag{3}$$

The total communication cost for the two global transposes becomes:

$$\begin{aligned} T(n_z, P_{row}) &= \frac{n_z}{P_{col}} \cdot T_{row}\left(m_{row}\right) + T_{col}\left(m_{col}\right) \\ &= \frac{n_z \cdot P_{row}}{P} \cdot T_{row}\left(\frac{N}{n_z \cdot P_{row}^2}\right) + T_{col}\left(\frac{N \cdot P_{row}}{P^2}\right) \end{aligned} \tag{4}$$

where $T_{row}()$ and $T_{col}()$ are functions of communication latency for the row and column communicators. The 2D processor decomposition and the symmetry requirement of the real-to-complex 3D-FFT together demands the following conditions:

$$\frac{n_z}{P_{col}} \geq 1, \quad \frac{n_y}{P_{row}} \geq 1, \quad \text{and} \quad \frac{n_x}{P_{row}} \geq 2 \tag{5}$$

P_{row} and n_z are chosen as independent variables that affect the total communication time. P_{row} can take different values depending on how the processors are arranged as a 2D processor mesh, while satisfying the validity conditions presented in Equation 5. As P_{row} decreases, P_{col} could become bigger than n_z and violates the first condition in Equation 5. However, by rotating this grid, the values of n_x and n_z can be interchanged to maintain the inequality as P_{row} decreases further. We will study this possibility in our experiments later.

4.4 Understanding the Trends in P3DFFT Performance

The total communication time in P3DFFT is impacted by three sets of variables: (i) message size, (ii) communicator size and (iii) congestion in the communicator topology. The first two variables (message size and communicator size) are directly related to the P_{row} parameter described in Equation 4. The third parameter, however, depends on the physical topology of the processes present in the communicator and their communication pattern, as these conditions determine how many messages share the same link on the torus network.

P3DFFT internally uses `MPI_Alltoallv` to transpose the data grid. For most implementations of MPI, including the one on Blue Gene, this is implemented as a series of point-to-point communication operations, with each process sending and receiving data to/from every other process in the communicator. For this communication pattern, even in a communicator where all the processes are topologically adjacent (row communicator), the number of messages that need to traverse the same network link in the torus network can increase quadratically.

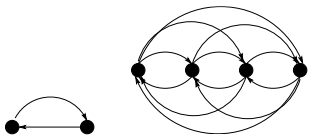

Fig. 3. `MPI_Alltoallv` Congestion Behavior

Figure 3 illustrates this behavior. Let us consider a torus of $8 \times 8 \times 8$ processes. For a row communicator with only 2 processes, the two processes just exchange data between each other. Thus, there is only one message per link in each direction (BG/L links are bidirectional). For a row communicator with 4 processes, the exchange is more complicated with the busiest link serving up to 4 messages in each direction. Though not represented in Figure 3, it can be shown that the number of messages traversing the busiest link in each direction increases quadratically with increasing communicator size. The exception to this rule is when one dimension of the torus completes. For example, for a communicator with 8 processes, the first dimension in the torus is fully utilized; thus, since BG/L uses a 3-D torus, this would mean that these processes can use an extra wrap-around link along this dimension. In this case, the maximum number of messages on the busiest link would be half the value it would have been without this wrap-around link.

In summary, if the first dimension of the torus has a processors, for communicator sizes of 1, 2, 4, ..., a/2, a, the number of messages on the busiest link would increase as 1, 4, 16, ..., $(a/4)^2$, $(a/2)^2 \times 2$, i.e., a quadratic increase in congestion with increasing communicator size. This trend continues for the second and third dimensions as well. Using this analysis, we can observe that a small system that has a torus configuration of $8 \times 8 \times 8$ would have a much smaller amount of congestion as compared to a large system that has a torus configuration of $8 \times 32 \times 16$.

The top 4 graphs in Figure 5 illustrate the total bandwidth per process achieved by `MPI_Alltoallv` for different message sizes on a small system ($P = 512$). The diamonds and triangles marked on the figures show the different message sizes (and corresponding bandwidths) that are used within P3DFFT for data grid configurations of $512 \times 512 \times 128$ and $128 \times 512 \times 512$. We notice that as long as the message size is larger than about 1 KB, both the row and column communicator achieve the peak bandwidth; thus, for best performance, it is preferred that a large message size be used. However, as illustrated in Equation 4, when P_{row} becomes large, the message size used by the row communicator drops quadratically. This causes it to use a very small message size for large P_{row} values resulting in the network not being saturated, and consequently performance loss. Thus, a small P_{row} value is preferred. For large systems ($P = 4096$), the large impact of congestion, as described above, can be observed in the bottom 4 graphs in Figure 5. The congestion causes a two-fold difference in the `MPI_Alltoallv` bandwidths achieved by the row and column communicators.

In the network saturation region, the time taken by `MPI_Alltoallv` can be approximated as a linear function, i.e. $T_{sub}(m_{sub}) \approx \alpha_{sub} \cdot (r \cdot m_{sub}) + \beta_{sub}$ where sub is the sub-communicator label for either row or col, and r is the precision of the datatype that $r \cdot m_{sub}$ is the message size in byte. In order

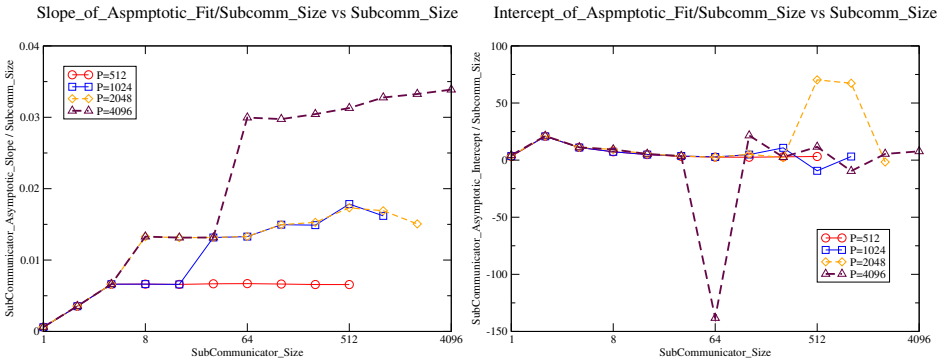

Fig. 4. Plots of scaled slope, $S()$, and intercept, $I()$, of asymptotic fit of the latency of the `MPI_Alltoallv` at P=512,1024,2048,4096. The graphs show that the slope and intercept scales with the subcommunicator size in a meaningful way.

to use the asymptotic function meaningfully, we investigated how the α and β change with their corresponding sub-communicator size. The results are shown in Figure 4. Notice that the Y-axes of both pictures are divided by the size of the sub communicator. This is necessary to scale out the effect of the communicator size. Four system sizes, $P = 512, 1024, 2048, 4096$ are plotted in the figures. They all overlap nicely to some universal functions. The scaled slope and intercept functions will be called $S(P_{sub})$ and $I(P_{sub})$ respectively. They are defined as

$$S\left(P_{sub}\right) = \frac{\alpha_{sub}\left(P_{sub}\right)}{P_{sub}} \quad and \quad I\left(P_{sub}\right) = \frac{\beta_{sub}\left(P_{sub}\right)}{P_{sub}} \tag{6}$$

For small systems $P = 512, 1024$, $S(P_{sub})$ increases linearly in $P_{sub} = 1, 2, 4$ and then becomes a constant afterward. But for system $P = 1024$ which is similar to $P = 512$, except a step jump appears from $P_{sub} = 16$ to $P_{sub} = 32$. For $P = 2048$, $S(P_{sub})$ increases linearly in $P_{sub} = 1, 2, 4, 8$, and then stays as a constant afterward. For $P = 4096$ which is similar to $P = 2048$, except with a step jump from $P_{sub} = 32$ to $P_{sub} = 64$. We believe the step jumps are due to the sudden increase of contention as P_{sub}'s topology changes as explained earlier in this section.

With Equations 6, 2 and 3, Equation 4 can be simplified to

$$\begin{aligned} T(n_z, P_{row}) &\approx \frac{n_z}{P_{col}}\left(\alpha_{row}(r \cdot m_{row}) + \beta_{row}\right) + \left(\alpha_{col}(r \cdot m_{col}) + \beta_{col}\right) \\ &= \frac{n_z}{P_{col}}\left(\frac{\alpha_{row}}{P_{row}}\frac{rN}{n_z P_{row}} + \beta_{row}\right) + \left(\frac{\alpha_{col}}{P_{col}}\frac{rN}{P} + \beta_{col}\right) \\ &= r\left[S\left(P_{row}\right) + S\left(\frac{P}{P_{row}}\right)\right]\frac{N}{P} + I\left(P_{row}\right) n_z \frac{P_{row}^2}{P} + I\left(\frac{P}{P_{row}}\right)\frac{P}{P_{row}} \end{aligned} \tag{7}$$

The $T(n_z, P_{row})$ is linear in n_z but its dependence on P_{row} is rather complicated. Since the behaviors of $S()$ and $I()$ in P_{sub} are known from Figure 4, we can reasonably describe how the total transpose time changes with P_{row}. Based on Figure 4, $S()$ is always positive and monotonic in P_{sub}. For large systems ($P = 4096$), $S() < 0.035$. For small systems ($P = 512$), $S() < 0.007$. $I()$ is more

of less a positive constant of order $O(10)$ except the big negative spike occurs at $P_{sub} = 64$. For the system parameters being considered here, we are mainly interested in $P_{row} < \sqrt{P}$, each term in Equation 7 can be estimated as follows:

$$\begin{matrix}\text{1st}\\ \text{term}\end{matrix} \propto \frac{N}{P} \gg P, \quad \begin{matrix}\text{2nd}\\ \text{term}\end{matrix} \propto n_z \frac{P_{row}^2}{P} < n_z < P, \quad \begin{matrix}\text{3rd}\\ \text{term}\end{matrix} \propto P_{col} < P$$

However, all the terms in Equation 7 are made equally important by $S() \ll I()$. For simplicity, let's ignore any terms that is $O(P_{row}^2)$ or higher and consider the small P_{row} limit, where $S(P_{row}) \rightarrow s_0 \cdot P_{row}$, $S(P/P_{row}) \rightarrow S_\infty$, and $I(P/P_{row}) \rightarrow I_\infty$. Equation 7 can be approximated as:

$$T(n_z, P_{row}) \approx r\left[s_0 \cdot P_{row} + S_\infty\right]\frac{N}{P} + I_\infty \frac{P}{P_{row}} \tag{8}$$

$$\Longrightarrow \quad P_{row}^{min} = P\sqrt{\frac{I_\infty}{rs_0N}} \quad \text{and} \quad T(n_z, P_{row}^{min}) \approx rS_\infty \frac{N}{P} + 2\sqrt{rs_0NI_\infty} \tag{9}$$

P_{row}^{min} is where the minimum of $T(n_z, P_{row})$ occurs.

For $N = 512\times512\times128$ and $P = 512$, $r = 4$, $s_0 \sim 0.002$, $S_\infty \sim 0.007$, $I_\infty \sim 3$, then $P_{row}^{min} \sim \sqrt{3} \sim 1.73$ and $T(n_z, P_{row}^{min}) \sim 3.5$ *msec*. For $N = 2048\times2048\times512$ and $P = 4096$, $r = 4$, $s_0 \sim 0.002$, $S_\infty \sim 0.035$, $I_\infty \sim 7$, then $P_{row}^{min} \sim 2\sqrt{7} \sim 5.3$ and $T(n_z, P_{row}^{min}) \sim 93$ *msec*. Both predicted $T(n_z, P_{row}^{min})$ values are within few percents of the actual measured experimental values.

For more accurate estimation, $O(P_{row}^2)$ terms and the full features of $S()$ and $I()$ are all needed. $I()$ has a negative spike of $O(10^2)$ at $P_{row} = 64 = \sqrt{P}$ as in Figure 4. The negative spike will certainly produce a local minimum of total transpose time for $N = 4096$. If the flat cartesian grid is rotated to increase n_z to avoid violating the validity conditions in Equation 5, P_{row} can get a lot closer to 1. Also, the discrete jumps seen in $S()$ in Figure 4 could be reflected in the observed total trasnpose time as sudden jump seen in the corresponding $S()$.

5 Experimental Evaluation and Analysis

In this section, we experimentally evaluate the P3DFFT library using a fortran physics program[4] that uses a flat cartesian mesh. Specifically, in this program, some of the variables only have x- and y-components, but no z-component. This means that the physical system emulated by this program is a quasi-2D system with preferential treatment in the z-axis. Therefore, our analysis of total communication time with respect to change in P_{row} in Equation 7 is applicable to this program, but not the communication analysis with respect to change in n_z. This is because the variation in P_{row} and P_{col} affects only the `MPI_Alltoallv` used in the two global transposes employed by the 3D-FFT which is being applied uniformly to all variables. In order words, the variation of the fortran program with respect to P_{row} is equivalent to the variation of 3D-FFT algorithm. But the

[4] The program, provided by Joerg Schumacher, has been modified for our benchmarking purpose.

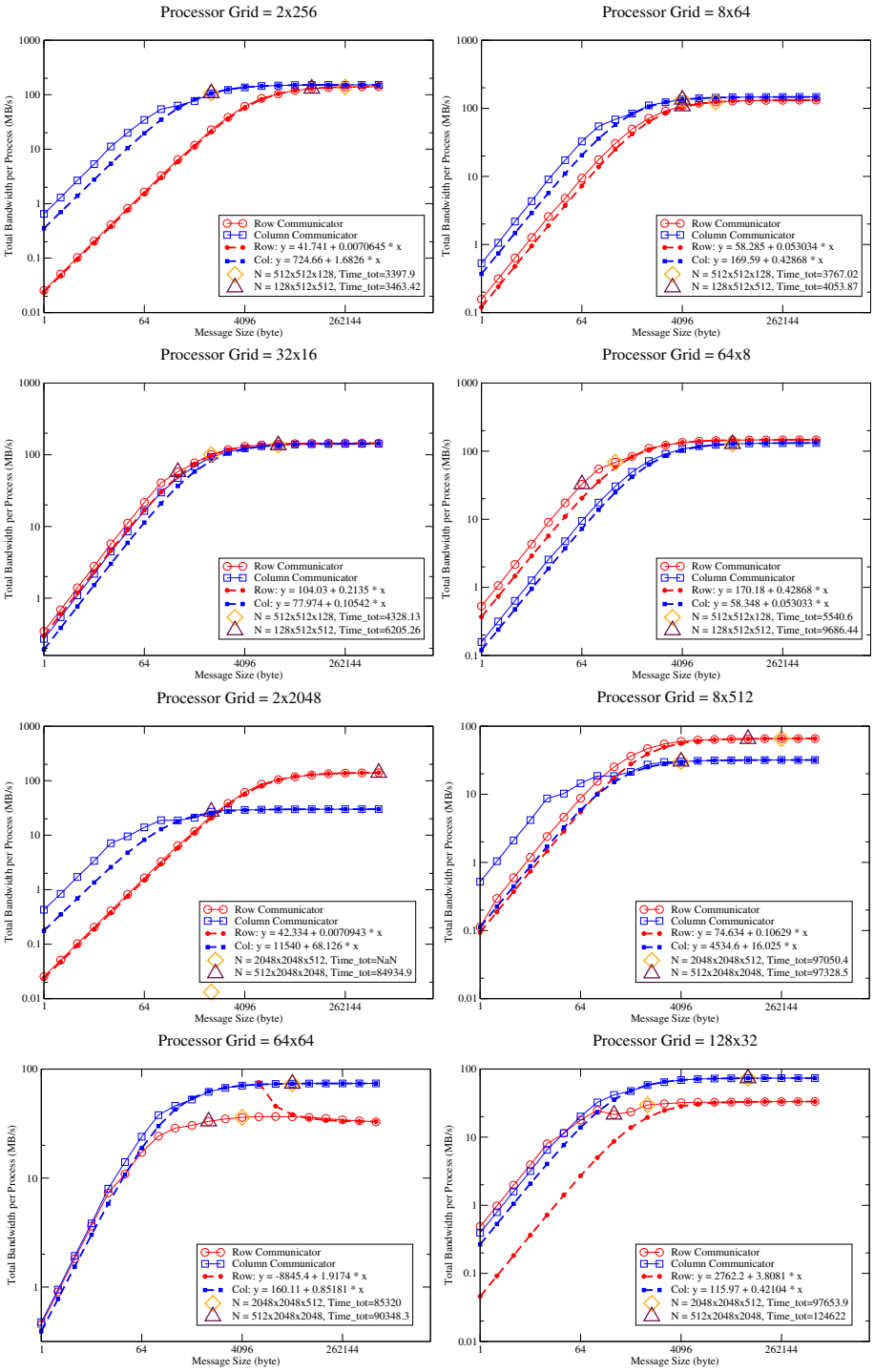

Fig. 5. Total bandwidth per process vs message size of small ($P = 512$) and large systems ($P = 4096$).

variation of the fortran program with respect to n_z includes both the variation of the 3D-FFT algorithm and the special asymmetric treatment of the z-axis in the phyical problem. In Section 5.1, we first observe the communication behavior for a small half-rack (512-node) system and verify our analysis. Then, in Section 5.2, we utilize this understanding to evaluate and optimize the performance of P3DFFT for a large-scale system.

5.1 Communication Analysis on a Small-scale System

In this section, a small BG/L system of 512 nodes is used to study the behavior of P3DFFT. These 512 nodes form a regular torus of $8\times8\times8$ dimensions. We ran our fortran program that uses the P3DFFT library with various data grid configurations on different processor mesh arrangements. Table 1 presents the timing data from this run.

Table 1. Timing of the fortran P3DFFT program (in second) with $P=512$ ($8\times8\times8$ torus). P: Processor mesh configuration. N: FFT data grid configuration.

P \ N	$256\times256\times64$	$64\times256\times256$	$512\times512\times128$	$128\times512\times512$
8×64	1.294	1.37	9.08	9.98
16×32	1.276	1.65	9.08	10.73
32×16	1.41	2.34	9.62	11.86
64×8	1.74		10.66	15.01

Four data grid configurations ($256\times256\times64$, $64\times25\times256$, $512\times512\times128$ and $128\times512\times512$), and four different processor mesh decompositions (8×64, 16×32, 32×16 and 64×8), were attempted on the 512-node system. In Table 1, we can see that the best timing for each data grid configuration occurs at the processor-mesh with the shortest row dimension, i.e., shortest P_{row}. Also, we see that the fortran program is taking longer to finish as P_{row} increases. Both features are consistent with our findings with Equation 8 in the last section.

5.2 Evaluation on a Large-scale System

In this section, we evaluate the performance of P3DFFT on a large-scale (16-rack or 16384-node) BG/L system. Specifically, we evaluated the performance of a $2048\times2048\times512$ data grid, with different processor-mesh configurations, on 4 racks (4096 nodes), 8 racks (8192 nodes) and 16 racks (16384 nodes). For the 4-rack system, we also tried out two different torus topologies ($16\times32\times16$ and $16\times16\times16$). Further, we also study the impact of rotating the data grid.

Tables 7, 6, 4 and 3 show the performance results for the different system sizes and configurations with our fortran test program. All these results indicate that the best performance occurs at the smallest P_{row} and largest n_z, i.e. rotated data grid, shown in the tables, when the number of processors P and the problem size N (FFT data grid size) are fixed. The small P_{row} giving the best performance is consistent with our findings of Equation 8. The best performance occuring at

largest n_z for fixed problem size N is more a feature of the physics problem being solved in the fortran program and not a feature of P3DFFT as explained in the beginning of section 5.

Table 2. Timing from fortran P3DFFT program (in second) with one processor size 4096 but different torus configurations. P: Processor mesh configuration. N: FFT data grid configuration.

Table 3. P = 4096 (16×16×16 torus)

P \ N	2048×2048×512	512×2048×2048
16×256	185.9	151.2
64×64	179.2	
256×16	194.1	

Table 4. P = 4096 (8×32×16 torus)

P \ N	2048×2048×512	512×2048×2048
8×512	215.2	181.36
32×128	218.4	190.4
64×64	201.7	179.3
128×32	198.2	194.4
512×8	239.1	

Tables 4 and 3 show the performance numbers of the fortran with the same problem sizes (2048×2048×512 and 512×2048×2048) and the same number of nodes (4096 nodes). The only difference between these two tables is that the different torus configurations are used. Table 4 is evaluated on a $8 \times 32 \times 16$ torus, while Table 3 is evaluated on a 16×16×16 torus. The fastest performance at 64×64 can be explained by the big negative spike of $I()$ seen in Figure 4 and Equation 7. We notice that for the processor-mesh, 64×64, P3DFFT is 10% faster in the 16×16×16 torus as compared to the $8 \times 32 \times 16$ torus. The reason for this behavior is the layout of the column communicator as described in Section 4.2. Specifically, in the 8×32×16 torus configuration, each row (64 processes) takes up eight physical rows on the torus. Thus, the processes in the column communicator can be up to 8 rows apart. On the other hand, in the $16 \times 16 \times 16$ torus configuration, each row takes up only four rows, thus reducing the distance between the processes in the column communicator and consequently improving their performance.

Table 5. Timing from fortran P3DFFT program (in second) with two different processor sizes, 8192 and 16384. P: Processor mesh configuration. N: FFT data grid configuration.

Table 6. P = 8192 (16×32×16 torus)

P \ N	2048×2048×512	512×2048×2048
16×512	146.5	113.1
64×128	142.5	115.3
128×64	144.3	125.7
512×16	165.0	

Table 7. P = 16384 (32×32×16 torus)

P \ N	2048×2048×512	512×2048×2048
16×1024		79.6
32×512	117.9	84.4
128×128	118.1	91.1
512×32	128.2	
1024×16	160.1	

Next, let us consider Table 6 that shows the performance for 8192 processors. If we notice the 2048×2048×512 FFT data grid configuration, we see that in this case, the smallest value of P_{row} (16×512 configuration) does not provide the best performance. Instead, 64×128 provides a better performance. This again can be explained by the big negative spike of $I()$ seen in Figure 4 and Equation 7. This

suggests 8192 processor configuration is similar to the non-cubical torus 4096 processor configuration discussed earlier in Equation 8 with the existence of at least two optimal configurations. Comparing the performance impact of the data grid rotation from the $2048 \times 2048 \times 512$ configuration to the $512 \times 2048 \times 2048$ configuration, we notice that the performance improves by about 26%. Not all the improvement is from the communication time.

The final result we present is for the large 16-rack (16384-node) system. We notice that this case is a little different from the 4-rack (4096-node) and the 8-rack (8192-node) results where two optimal configurations can be obtained. However, the overall performance is still consistent with the other results. That is, performance improves with decreasing P_{row} and with increasing n_z. Specifically, reducing P_{row} can improve performance by about 15% as compared to the default 128×128 processor mesh configuration. Increasing n_z, on the other hand, can improve performance by close to 48%.

Based on all the experimental results, we notice that there could be multiple optimal system configurations, two possible ones are 1) small P_{row} in rotated data grid with larger n_z. 2) $P_{row} \simeq \sqrt{P}$ in the regular data grid with smaller n_z. The later optimal configuration may not exist in all system sizes and configurations. But the first optimal configuration seems to always exist. Rotating the FFT data grid furthers the path of performance improvements that have been stopped by Equation 5. This indicates that as we move to even larger problem sizes, the lessons learnt in this paper will have increasingly higher importance.

6 Concluding Remarks and Future Work

P3DFFT is a recently proposed implementation of parallel 3D FFT for large-scale systems such as IBM Blue Gene. While there have been a lot of studies that demonstrate the scalability of P3DFFT on regular cartesian meshes (where all dimensions are equal in length), there seems to be no previous work that studies its scalability for flat cartesian meshes (where the length of one dimension is much smaller than the rest). In this paper we studied the performance and scalability of P3DFFT for flat cartesian meshes on a 16-rack (16384-node) Blue Gene system and demonstrated that a combination of the network topology and the communication pattern of P3DFFT can result in parts of the communication to over-saturate the network, while other parts under-utilize the network. This can cause overall loss of performance on large-scale systems. We further showed that remapping processes on nodes and rotating the FFT data grid by taking the communication properties of P3DFFT into consideration, can help alleviate this problem and improve performance by up to 48% in some cases.

While our work alleviates the issue of network saturation, it does not completely avoid it. For future work, we would like to further the study of alleviation of network contention by rotating the torus configuration through environment variable `BG_MAPPING` which allows user to rearrange process layout in the torus, and we would also like to study the impact of split-collectives to hide communication time that can be aggravated due to such saturation.

References

1. http://128.55.6.34/projects/paratec/.
2. http://eslab.ucdavis.edu/software/qbox/.
3. http://www.cse.scitech.ac.uk/ccg/software/dl_poly/.
4. http://www.fftw.org.
5. http://www.research.ibm.com/journal/rd/492/gara.pdf.
6. http://www.sdsc.edu/us/resources/p3dfft.php.
7. http://www.spscicomp.org/scicomp12/presentations/user/pekurovsky.pdf.
8. James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1964.
9. Intel Corporation. Intel Math Kernel Library (MKL). http://www.intel.com/cd/software/products/asmo-na/eng/307757.htm.
10. C. E. Cramer and J. A. Board. The Development and Integration of a Distributed 3D FFT for a Cluster of Workstations. In *Proceedings of the 4th Annual Linux Showcase and Conference*, volume 4. USENIX Association, 2000.
11. Anshu Dubey and Daniele Tessera. Redistribution strategies for portable parallel FFT: a case study. *Concurrency and Computation: Practice and Experience*, 13(3):209–220, 2001.
12. M. Eleftheriou, B. G. Fitch, A. Rayshubskiy, T. J. C. Ward, and R. S. Germain. Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: implementation and early performance measurements. *IBM Journal of Research and Development*, 49:457–464, 2005.
13. Salvatore Filippone. The IBM Parallel Engineering and Scientific Subroutine Library. In *International Workshop on Applied Parallel Computing, Computations in Physics, Chemistry and Engineering Science*, pages 199–206, London, UK, 1996. Springer-Verlag.
14. B. G. Fitch, A. Rayshubskiy, M. Eleftheriou, T. J. C. Ward, M. E. Giampapa, M. C. Pitman, J. W. Pitera, W. C. Swope, , and R. S. Germain. Blue Matter: Scaling of N-body simulations to one atom per node. *IBM Journal of Research and Development*, 52(1/2), 2008.
15. M. Frigo and S. 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".
16. A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49:195–212, 2005.
17. C. J. Olson, G. T. Zimnyi, A. B. Kolton, and N. Grnbech-Jensen. Static and Dynamic Coupling Transitions of Vortex Lattices in Disordered Anisotropic Superconductors. *Phys. Rev. Lett.*, 85:5416, 2000.
18. Jorg Schumacher and Matthias Putz. Turbulence in Laterally Extended Systems. In *Parallel Computing: Architectures, Algorithms and Applications*, volume 15 of *Advances in Parallel Computing*. IOS Press, 2008.
19. D.N. Straub. Instability of 2D Flows to Hydrostatic 3D Perturbations. *J. Atmos. Sci.*, 60:79–102, 2003.
20. K. v. Klitzing, G. Dorda, and M. Pepper. New Method for High-Accuracy Determination of the Fine-Structure Constant Based on Quantized Hall Resistance. *Phys. Rev. Lett.*, 45:494–497, 1980.