Overlapping Methods of All-to-All Communication and FFT Algorithms for Torus-Connected Massively Parallel Supercomputers

Jun Doi and Yasushi Negishi

Abstract—Torus networks are commonly used for massively parallel computers, its performance often becomes the constraint on total application performance. Especially in an asymmetric torus network, network traffic along the longest axis is the performance bottleneck for all-to-all communication, so that it is important to schedule the longest-axis traffic smoothly. In this paper, we propose a new algorithm based on an indirect method for pipelining the all-to-all procedures using shared memory parallel threads, which (1) isolates the longest-axis traffic from other traffic, (2) schedules it smoothly and (3) overlaps all of the other traffic and overhead for the all-to-all communication behind the longest-axis traffic. The proposed method achieves up to 95% of the theoretical peak. We integrated the overlapped all-to-all method with parallel FFT algorithms. And local FFT calculations are also overlapped behind the longest-axis traffic. The FFT performance achieves up to 90% of the theoretical peak for the parallel 1D FFT.

Index Terms—all-to-all communication, collective communication, Blue Gene, FFT, multi-core, multi-thread, parallel algorithm, SMP, torus network

I. INTRODUCTION

All-to-all communication is one of the most performance-critical communication patterns for scientific applications on massively parallel computers. The all-to-all communication is often used for matrix or array transposition, or parallel Fast Fourier Transform (FFT) algorithms. Since the performance of the parallel FFT largely depends on the performance of the all-to-all communication, its performance is a key parameter for applications such as molecular dynamics [7] [8], quantum molecular dynamics [9], plasma physics, or digital signal processing.

In this paper, we focus on all-to-all communication on massively parallel computers with torus interconnects. Today, the torus interconnect is becoming a common interconnect topology for massively parallel computers [1] [2] [3] [4] [5] [6] since it offers high scalability, high cost effectiveness, and high flexibility. As the sizes of the torus networks become larger to scale up to petaflops and exaflops, the performance of the all-to-all communication is becoming critical to a wider range of applications because the final performance depends on the size and shape of the torus network. The bandwidth of the all-to-all communication is theoretically in inverse proportion to the longest dimension of the torus network, and the network contention caused by the asymmetry of a torus network decreases the effective bandwidth. In most cases the torus network is asymmetric. For example a standard rack of Blue Gene/P [3] machines is composed of a 3D torus whose dimensions are 8x8x16. There are only two symmetric configurations, 4 racks (16x16x16) and 32 racks (32x32x32) for machines below the petaflops (72 racks). In this paper, we propose an optimal all-to-all communication algorithm and an FFT algorithm for asymmetric torus configurations.

There are many approaches for all-to-all communication optimized for torus networks. These are classified into two types, direct and indirect methods. In direct methods the all-to-all communication is done via point-to-point communications between all of the pairs of nodes. In contrast, in indirect methods the messages are combined in packets and routed or sent through intermediate nodes. There are several indirect methods suitable for torus networks [10] [11] [12] [13], and most of the indirect methods are focused on scheduling and routing the packets in torus networks to minimize network contention. For example, [13] presented a two-phase all-to-all schedule to minimize network contention on asymmetric torus networks. In indirect methods, it is important to minimize additional software overhead such as message combining and routing in the intermediate nodes.

In this paper, we propose an overlapping method for all-to-all communications to reduce the network contention in the asymmetric torus networks. We overlap the all-to-all method based on the indirect method by pipelining in the shared memory parallel (SMP) threads on the multi-core processors, which (1) isolates the longest-axis traffic from other traffics, (2) schedules that traffic smoothly and (3) overlays all other traffic and overhead for the all-to-all communication behind the longest-axis traffic. By using threads, network latencies, startup overheads, and internal array transposes to arrange the message packets on the different threads are overlapped to maximize the performance.

We also propose a method to integrate our optimal all-to-all communication with the parallel FFT algorithms. Although the parallel FFT algorithm is usually implemented using synchronous all-to-all communication, we integrated the
parallel FFT algorithm into the pipelined all-to-all communication using SMP threads, so that local FFT calculations are also hidden behind the longest-axis traffic of all-to-all communication. We implemented parallel 1D, 2D and 3D FFT.

The remainder of this paper is organized as follows: Section II describes the performance characteristics of all-to-all communication for a torus network. Section III presents our proposal for a pipelined all-to-all method based on the indirect method, Section IV explains a way to integrate our all-to-all method with the parallel FFT algorithm. We present conclusions and future work in Section V.

II. ALL-TO-ALL COMMUNICATION ON THE TORUS NETWORK

A. Torus network

In a torus network, the nodes are arranged in an N-dimensional grid and connected to neighboring nodes so each node in the torus network has 2N nearest neighbors. Fig.1 shows an example of a 4x4 2D torus network.

![Fig. 1. Example of 4x4 2D “symmetric” torus network. In this case each node has four nearest neighbors.](image)

The packets for point-to-point communications are routed to their final destinations by routers in each node. Network contention occurs frequently when the routes of packets cross in dense communication patterns such as all-to-all, broadcast, scatter, or gather.

B. Performance of all-to-all communications

In all-to-all communication, each node exchanges messages with all of the other nodes. The all-to-all bandwidth peaks when all of the routes are filled with packets and the theoretical peak bandwidth is determined by the length of the longest axis of torus network.

The time for all-to-all communication \( T \) is given by Equation 1 [13]:

\[
T = np * \left(\frac{L}{8}\right) * m * \beta
\]

where \( np \) is the number of processors, \( L \) is the length of the longest axis, \( m \) is the total size of all of the all-to-all messages, and \( \beta \) is the per byte network transfer time. The theoretical peak bandwidth of the all-to-all communications is given by Equation 2:

\[
A = e * \left(\frac{1}{\beta}\right) * \left(\frac{8}{L}\right)
\]

where \( e \) is the efficiency and \( \left(\frac{1}{\beta}\right) \) is the network bandwidth.

Table I shows the theoretical peak bandwidth and the measured all-to-all bandwidth of an MPI library (MPI_Alltoall) [14] for the 3D torus network of Blue Gene/P. In this table, the measured bandwidth for the 3D symmetric torus networks of 512 and 4,096 nodes show much better performance than the asymmetric 3D torus networks of 1,024 and 2,048 nodes. Although MPI_Alltoall is well optimized [15], the measured bandwidth is only 72% of the theoretical peak on 1,024 nodes because of the network contention and the traffic imbalance bottleneck along the longest axis. In Table I, we show the best measured bandwidth for MPI_Alltoall. We will see that as the message size increases the effective bandwidth is reduced, as shown in Fig. 4 in Section III.C. We propose a new all-to-all method address these problems in the next section.

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>Torus Shape</th>
<th>Theoretical Peak Bandwidth [MB/s]</th>
<th>Measured Bandwidth [MB/s]</th>
<th>Efficiency [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>8x8x8</td>
<td>374</td>
<td>340.6</td>
<td>91.1</td>
</tr>
<tr>
<td>1,024</td>
<td>8x8x16</td>
<td>187</td>
<td>134.2</td>
<td>71.8</td>
</tr>
<tr>
<td>2,048</td>
<td>8x16x16</td>
<td>187</td>
<td>146.2</td>
<td>78.2</td>
</tr>
<tr>
<td>4,096</td>
<td>16x16x16</td>
<td>187</td>
<td>158.9</td>
<td>84.9</td>
</tr>
</tbody>
</table>

III. OVERLAPPING METHOD FOR ALL-TO-ALL COMMUNICATION

A. Two-phase scheduling of all-to-all communication

To reduce the network contention in the all-to-all communications on asymmetric torus networks, two-phase scheduling [13] offers advantages. The all-to-all communication is separated into two phases with the group for communication being called the communicator. The entire torus network is separated into two sub-communicators so that each sub-communicator shapes 1D axis or symmetric multi-dimensional torus and the bandwidths of the two separated all-to-all communications can approach their theoretical peaks. For example, if the configuration of the 3D torus network is X=4, Y=4 and Z=8, then the sub-communicator for phase-1 (A2A-1) contains the Z axis, while phase-2 (A2A-2) contains the X and Y axes, forming a symmetric plane as shown in Fig. 2.

![Fig. 2. Separating the all-to-all communication into 2 phases; along the longest axis and in the plane consists of the other axes.](image)

The all-to-all communication is processed in two phases:

Phase-1: For each final destination, a sender node chooses a node belonging to both the A2A-1 (including the sender) and A2A-2 (including the final destination) as an intermediate node, and sends the packets for the final destination to the chosen intermediate node.

Phase-2: Each intermediate node sends its packets to their final destinations in A2A-2.
By dividing the all-to-all communication into two phases, we can avoid network contention caused by the longest-axis traffic, but the theoretical communication time increases. Therefore we need a new approach to hide the additional communication time behind the communication time of the longest-axis all-to-all communications.

B. Overlapping all-to-all communications on multi-thread nodes

We divide the messages and pipeline the procedures of the two-phase scheduling technique so that we can overlap the two-phase all-to-all communications in the different sub-communicators each other. The pipelined procedures for handling each message consist of two all-to-all communications and three internal array transpositions:

\begin{itemize}
  \item tr1: Array transpose to arrange packets for the intermediate node
  \item tr2: Array transpose to arrange packets for the final destination
  \item tr3: Array transpose to arrange the output array
\end{itemize}

We use shared memory parallelization (SMP) on multi-thread or multi-core processors, and we simply map these pipelined procedures to each thread as shown in Fig. 3 to process the independent messages sequentially. The threads must be carefully scheduled so that only one thread at a time processes the all-to-all communication on the same sub-communicator. We use atomic counters for exclusive access control for two sub-communicators. On Blue Gene/P, we use the PowerPC® atomic load and store instructions, lwax and stwcx to refer to and increment the atomic counters. This scheduling prioritizes the longest-axis traffic to maximize the bandwidth of the all-to-all communications. The all-to-all communications along the longest-axis are the dominant procedures and the other all-to-all communications and the three internal array transpositions are hidden behind them.

In this method, all-to-all communication does not need an asynchronous implementation because each thread has to wait until all of the data is received from the earlier procedures. However by using SMP threads, the other threads can proceed independently behind the communication and it is easier to implement this complex scheduling compared to using asynchronous communication to overlap other procedures. We can implement the overlapping method by inserting OpenMP directives in front of the pipelined loop and atomic barrier operations in the loop. This pseudocode describes the overlapped two-phase all-to-all communications for \( np \) processors from array \( A(N) \) to \( B(N) \).

\begin{align*}
\text{do } i=1 \text{ to } d \{ \\
\quad \text{transpose 1: } A(N/np/d, i, np2, np1) \text{ to } T1(N/np/d, np2, np1) \\
\quad \text{wait until mod(c1,nt) becomes my thread ID} \\
\quad \text{A2A-1 from T1 to T2} \\
\quad \text{atomic increment c1} \\
\quad \text{transpose 2: } T2(N/np/d, np2, np1) \text{ to } T1(N/np/d, np1, np2) \\
\quad \text{wait until mod(c2,nt) becomes my thread ID} \\
\quad \text{A2A-2 from T1 to T3} \\
\quad \text{atomic increment c2} \\
\quad \text{transpose 3: } T3(N/np/d, np1, np2) \text{ to } B(N/np/d, i, np2, np1) 
\}\end{align*}

Since the sizes of the temporary buffers \( T \) are inversely related to the denominator \( d \), they can be chosen to maximize the performance. The larger \( d \) values increase the pipeline efficiency, while larger \( T \) values increase the efficiency of communication. It is also important to choose \( T \) so that the internal array transpose is done in the cache memory, or cache misses will extend the times for internal array transpositions beyond what can be hidden behind the times for communications.

For Blue Gene/P, we chose 512 KB for the size of the temporary buffer \( T \), because Blue Gene/P has 8 MB of L3 cache memory per node, and we need three temporary buffers per thread. Therefore 512 KB is the largest buffer size that can be processed in the L3 cache (512 KB x 3 buffers x 4 threads = 6 MB). This means that all of the data received is stored in the L3 cache and the internal array transpositions are done smoothly without any cache misses.
C. Performance of the overlapping all-to-all method

We developed the overlapping method for the all-to-all communication and measured the performance on Blue Gene/P. Blue Gene/P has four processor cores per node and SMP is supported in each node. Each node of Blue Gene/P is connected in a 3D torus network, and the smallest torus is a half-rack with an 8x8x8 torus network, with two 8x8x8 torus networks combined in each full rack of 8x8x16. The shapes of the torus networks for multiple racks are combinations of 8x8x16, so most large Blue Gene/P systems have asymmetric torus networks. We measured and compared the all-to-all performances on a half-rack (8x8x8 = 512 nodes), on one rack (8x8x16 = 1,024 nodes), two racks (8x16x16 = 2,048 nodes) and four racks (16x16x16 = 4,096 nodes) of Blue Gene/P. Fig. 4 shows the measured all-to-all bandwidths of the proposed overlapping method and MPI_Alltoall on Blue Gene/P.

On 512 nodes (symmetric torus network), the performance of MPI_Alltoall is better than our proposed method, because the advantage of our method is relatively small for a small symmetric torus network. We observed that some of the internal array transposes are not masked behind the all-to-all communication time in this case.

With the 1,024 or 2,048 nodes (asymmetric torus networks), the proposed method has 20% to 30% better performance, and achieves up to 95% of the theoretical peak bandwidth. These results show that the proposed method does a good job of overlapping all of the other overhead for the all-to-all communication behind the longest-axis traffic.

On 4,096 nodes (symmetric torus network), the performance of the proposed method is slightly better than MPI_Alltoall because of the effects of L3 cache misses.

IV. OVERLAPPING FAST FOURIER TRANSFORM

A. Parallel 1D FFT

1) Six-step FFT algorithm

A discrete Fourier transform (DFT) is defined in Equation 3.

\[ y_k = \sum_{j=0}^{n-1} x_j w_n^{jk}, \text{ where } w_n = e^{-2\pi j/n} \]

(3)

To parallelize a DFT on parallel machines, algorithms based on a six-step FFT [16] are generally used [17]. When \( n \) can be
decomposed as \( n_1 \times n_2 \). Equation 3 can be rewritten using 2D arrays as Equation 4.

\[
y(k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} x(j_1, j_2) w_{j_2 n_2}^{j_1 k_1} w_{j_1 n_1}^{j_2 k_2}
\]

When we use \( np \) parallel processes, a 1D array \( x(n) \) is distributed and each process has a partial array \( x(n_{j_1} n_{j_2}) \). This is equivalent to the 2D array \( x(n_1, n_2) \). To calculate a 1D FFT using Equation 4 on a parallel computer, three all-to-all communications are needed to transpose the global array. Usually all-to-all communication is synchronous and the local FFT calculations can be done when all of the elements of the array have been received, so the performance of parallel FFT is largely dependent on the performance of all-to-all communication.

Here is the algorithm for a parallel six-step FFT.

Step 1: Global array transpose, \( x(n_1, n_2) \) to \( t(n_2, n_1) \)
Step 2: \( n_1 \) FFT
Step 3: Multiply the elements of the array by the twiddle factor \( w_{n_2}^{j_2 k_2} \)
Step 4: Global array transpose, \( t(n_1, n_2) \) to \( t(n_2, n_1) \)
Step 5: \( n_2 \) FFT
Step 6: Global array transpose, \( t(n_2, n_1) \) to \( y(n_2, n_1) \)

In more detail, we also need internal array transposes before and after all-to-all communications of the global array transposes, and we can merge Step 3 into the internal array transpose needed in Step 4 before the all-to-all communication.

2) Overlapping parallel six-step FFT algorithm

By replacing conventional synchronous all-to-all communication with the proposed overlapping method of all-to-all communication using a multi-thread SMP, we can optimize the all-to-all communication parts of the FFT. In addition to the all-to-all communication optimization, we can also optimize the local FFT calculations and internal array transposes of the FFT by hiding their time behind the longest-axis all-to-all traffic. To integrate the overlapping method of the all-to-all communication into the parallel six-step 1D FFT, the algorithm and the array are divided for pipelining [18]. We separate the six-step FFT algorithm into two phases, Phase-1 combines Step 1 to Step 4. Phase-2 combines Step 5 and Step 6.

Then each phase is pipelined separately. Phase-1 is pipelined into \( n_1 np \) ways and Phase-2 is pipelined into \( n_2 np \) ways, and the arrays are divided in the same ways as the pipelines and the divided arrays are processed in parallel loops using SMP threads. Fig.5 shows the pipelined procedures of Phase-1 and Phase-2 using the proposed overlapping method for the all-to-all communication. In Phase-1, there are two all-to-all communications for the global array transposes, so A2A-1 and A2A-2 appears twice in a pipelined iteration. The all-to-all communication is exclusively scheduled using atomic counters as described in Section III.B. In Fig.5 A2A-1 is the all-to-all communication along the longest axis and most of the other procedures, including the local FFT calculations, are hidden behind A2A-1.

B. Parallel multi-dimensional FFT

The multi-dimensional FFT is calculated by applying successive 1D FFTs in each dimension. A multi-dimensional array is usually decomposed by mapping the problem to the shape of the torus network. For example, with a 3D problem, a 3D array would be decomposed in 3 dimensions and mapped to a 3D torus network by dividing the array by the size of the torus network in each dimension. In this case, we have three sub-communicators along each axis of the torus network and we calculate the 1D parallel FFTs using all-to-all overlapping method with these sub-communicators.

1) Overlapping parallel 2D FFT

When we map a 2D problem to a 3D torus network, we decompose the array onto two sub-communicators. One is decomposed along an axis, and the other is decomposed on a plane. We apply the overlapped parallel 1D FFTs in the two directions independently. Fig. 6 describes the pipelined 1D FFT procedures for symmetric or linear sub-communicators.
Fig. 6. Overlapping FFTs and all-to-all communications on the symmetric sub-communicator using 4 SMP threads.

Fig. 7. Overlapping FFTs and 2 phase all-to-all communications on the asymmetric sub-communicator using 4 SMP threads.

Fig. 8. Merging 2 pipelines of the overlapped method of all-to-all communications and FFTs using 4 SMP threads.

For asymmetric sub-communicators, the sub-communicator is divided into two smaller sub-communicators and overlapped as shown in Fig. 7.

2) Overlapping parallel 3D FFT

When we map a 3D problem to a 3D torus network, we decompose the array onto three sub-communicators, one for each axis. Although a 3D FFT is calculated with three independent pipelines of overlapped 1D FFTs, two of the pipelines can be merged into one pipeline because we need at least two independent pipeline loops for local parallelism to process the pipelined procedures using SMP threads. We can overlap the all-to-all communications on the different axes of the torus network as shown in Fig. 8. The rest of the pipeline is processed as shown in Fig. 6. For higher dimensions, the pipelines can be merged in the same manner to overlap the procedures for different dimensions.

C. Performance of the overlapping method of the parallel FFT

We also measured the performance of the overlapping FFT method on Blue Gene/P. We used our own optimized FFT library for the local FFT calculations to support the SIMD instructions for complex number arithmetic. We compared our overlapping FFT method to a conventional synchronous implementation using MPI_Alltoall.

The maximum performance for synchronous parallel FFT is defined by the theoretical peak bandwidth of the all-to-all communications based on dividing the number of floating point operations by the theoretical total time for the all-to-all communications. The number of floating point operations in a 1D FFT for an array of length \( N \) is defined by Equation 5.

\[
P(N) = 5N\log(N)/\log(2) \tag{5}\]

The theoretical total time for the all-to-all communication is calculated by dividing the array size by the theoretical peak bandwidth (Equation 2) and multiplying by the number of all-to-all communications needed for parallel FFT calculations.

Fig. 9 shows the measured performances for a 1D FFT on Blue Gene/P. We can see very similar curves for the all-to-all bandwidths of the MPI implementation. The performance of the overlapped FFT increases for larger arrays, which is similar to the bandwidth of the overlapping all-to-all method, and we achieved 60% to 90% of the maximum performance for the parallel 1D FFT calculation as defined by the theoretical peak bandwidth of the all-to-all communication. Our measured improvements were 1.3 to 1.8 times faster than the synchronous implementation using MPI_Alltoall.

Fig. 10 shows the measured performances for a 2D FFT on Blue Gene/P. We achieved 70% to 85% of the maximum performance for the parallel 2D FFT calculation. The measured improvements were 1.3 times faster than the synchronous implementation using MPI_Alltoall.

Fig. 11 shows the measured performances for a 3D FFT on Blue Gene/P. For 3D and higher dimension FFTs, by merging two pipelines of overlapping FFTs, two pairs of all-to-all communications are hidden behind the longest axis traffic, so the maximum performance for the proposed overlapping method is higher than the MPI implementation. We achieved 75% to 95% of the maximum performance for the parallel 3D FFT calculation defined by the theoretical peak bandwidth of the all-to-all communication, and the performance of the new overlapping method exceeded the maximum performance of the conventional implementation. The performance of the MPI implementation is stable, while the performance of the overlapping FFT improves with larger arrays, eventually becoming 2.0 times faster than a synchronous implementation using MPI_Alltoall.
V. CONCLUSIONS

In this paper, we presented an overlapping method for all-to-all communications to reduce the network contention in asymmetric torus networks. We improved the all-to-all bandwidth for asymmetric torus networks by pipelining the indirect all-to-all algorithm and by using SMP threads to overlap the all-to-all communications and the internal array transpositions. We also eliminated the performance degradation for larger arrays by processing the internal array transposes in cache memory. It is important to schedule the indirect all-to-all communications to avoid network contention on the asymmetric torus networks. However indirect methods need extra overhead such as internal array transpositions, and it is also important to reduce the overhead to improve the performance. The overlapping all-to-all method addresses these problems by hiding overhead behind the all-to-all communications along the longest axis. The proposed method improves the performance of all-to-all communication by 20% to 30% on an asymmetric torus, and achieves up to 95% of the theoretical peak bandwidth. This pipelining technique for SMP threads is not only useful for a torus network but also for other networks. We are planning to develop and analyze our overlapping all-to-all method on other parallel computers connected without torus interconnects.

We also improved the performance of the parallel FFT on torus networks by integrating the new overlapping all-to-all method into a parallel six-step FFT algorithm and by hiding the calculation time for the local FFT behind the all-to-all communications. The proposed method improves performance by 80% to 100% for parallel 1D FFT and reaches up to 90% of the maximum theoretical performance as defined by the bandwidth of the all-to-all communications. The proposed overlapping 3D FFT exceeds the maximum theoretical performance of the synchronous implementation.

We are also planning to find other applications to improve by overlapping the all-to-all communications with the calculations.

ACKNOWLEDGEMENTS

We tested and developed our experimental programs on the Watson4P Blue Gene/P system installed at IBM T.J. Watson Research Center, Yorktown Heights, N.Y. We would like to thank Fred Mintzer for helping us obtain machine time, James Sexton for managing our research project, and Shannon Jacobs for proofreading. We also would like to thank Sameer Kumar, Yogish Sabharwal and Philip Heidelberger.

REFERENCES

Fig. 9. Performance comparisons for parallel 1D FFT on Blue Gene/P. The array size is the length of the complex array and the point-to-point message length is shown as the label of the horizontal axis. The maximum performance shows the parallel 1D FFT performance of a synchronous implementation defined by the theoretical peak bandwidth of the all-to-all communications.

Fig. 10. Performance comparisons for parallel 2D FFT on Blue Gene/P. The maximum performance shows the parallel 2D FFT performance of a synchronous implementation defined by the theoretical peak bandwidth of the all-to-all communications.
Fig. 11. Performance comparisons for parallel 3D FFT on Blue Gene/P. There are 2 different maximum performances, the higher one is the maximum performance for the proposed overlapping method, and the other is the maximum performance for the conventional synchronous implementation using MPI_Alltoall as defined by the theoretical peak bandwidth of the all-to-all communications.