Optimizing Matrix Transpose on Torus Interconnects

Venkatesan T. Chakaravarthy, Nikhil Jain, and Yogish Sabharwal

IBM Research - India, New Delhi
{vechakra,nikhil.jain,ysabharwal}@in.ibm.com

Abstract. Matrix transpose is a fundamental matrix operation that arises in many scientific and engineering applications. Communication is the main bottleneck in performing matrix transpose on most multiprocessor systems. In this paper, we focus on torus interconnection networks and propose application-level routing techniques that improve load balancing, resulting in better performance. Our basic idea is to route the data via carefully selected intermediate nodes. However, directly employing this technique may lead to worsening of the congestion. We overcome this issue by employing the routing only for selected set of communicating pairs. We implement our optimizations on the Blue Gene/P supercomputer and demonstrate up to 35% improvement in performance.

1 Introduction

Matrix transpose is a fundamental matrix operation that arises in many scientific and engineering applications. On a distributed multi-processor system, the matrix is distributed over the processors in the system. Performing transpose involves communication amongst these processors. On most interconnects with sub-linear bisection bandwidth, the primary bottleneck for transpose is the communication. Matrix transpose is also included in the HPC Challenge benchmark suite [8]. This is a suite for evaluating the performance of high performance computers. The HPCC matrix transpose benchmark mainly aims at evaluating the interconnection network of the distributed system.

We shall be interested in optimizing matrix transpose on torus interconnects. Torus interconnects are attractive interconnection architectures for distributed memory supercomputers. They are more scalable in comparison to competing architectures such as the hypercube and therefore many modern day supercomputers such as the IBM Blue Gene and Cray XT are based on these interconnects. We will be mainly interested in the case of asymmetric torus networks. Recall that a torus network is said to be symmetric if all the dimensions are of the same length; it is said to be asymmetric otherwise. Notice that increasing the size of a torus from one symmetric configuration to a larger one requires addition of nodes along all the dimensions, requiring a large number of nodes to be added. It is therefore of considerable interest to study asymmetric configurations as they are realized often in practice.

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part II, LNCS 6272, pp. 440–451, 2010.

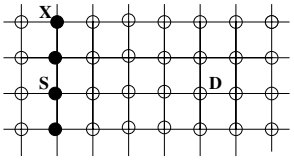


Fig. 1. Illustration of SDR technique and routing algorithms

Adaptive routing is the most common routing scheme supported by torus networks at the hardware level. In this scheme, the data may take any of the shortest paths from the source to the destination; the exact path is determined dynamically based on various network parameters such as congestion on the links. Dynamically choosing paths leads to better load balancing. The goal of this paper is to propose optimizations for matrix transpose on asymmetric torus networks when the underlying hardware level routing scheme is adaptive routing.

A standard approach for performing matrix transpose (also used in the widely used linear algebra library, ScaLAPACK [2]) is to split the processing into multiple phases. Each phase involves a permutation communication, wherein, every node sends data to exactly one node and receives data from exactly one node. In other words, the communication happens according to a permutation $\pi : \mathcal{P} \rightarrow \mathcal{P}$ and each node u sends data to the node $\pi(u)$, where $\mathcal{P}$ is the set of all nodes in the system. (If $\pi(u) = u$, then the node does not participate in the communication).

Our optimization is based on an application level routing scheme that we call *Short Dimension Routing (SDR)*, which reduces congestion for permutation communications on asymmetric torus interconnects. Apart from matrix transpose, permutation communication patterns also arise frequently in other HPC applications such as binary-exchange based fast-fourier transforms [6] and recursive doubling for MPI collectives in MPICH [11]. Our techniques are also useful in other applications that involve permutation communication.

The *SDR* technique is based on a simple, but useful, observation that on asymmetric torus networks the load on the links lying along the shorter dimensions are typically less loaded in comparison to the links lying along the longer dimensions. To see this in a quantitative manner, consider random permutation communication patterns on a 2-dimensional torus of size $N_X \times N_Y$. We show that the expected load on any X-link (namely, a link lying along the X-dimension) is $N_X/4$ and the expected load any Y-link is $N_Y/4$. Thus, in case of asymmetric torus, say a torus with $N_X \geq 2N_Y$, the expected load on the Y-links is twice that of the X-links. The *SDR* technique exploits this fact and work as follows. Instead of sending data directly from sources to destinations, we route the data through intermediate nodes. Namely, for each source-destination pair, the data is first sent from the source to the intermediate node and then from the intermediate node to the destination. In both the steps, communication takes place using the adaptive routing supported at the hardware level. The crucial aspect of the *SDR* technique is its choice of the intermediate node. We restrict the selection of the intermediate node to be from one amongst those that can

be reached by traversing only along the shorter dimensions from the source. To better illustrate the idea, consider the case of a 2-dimensional asymmetric torus of size $N_X \times N_Y$ with $N_X \geq 2N_Y$. In this case, we will choose one of the nodes lying on the same column as the source node as the intermediate node. Intuitively, the idea is that the above process improves the load balance of X-links, although increasing the load on Y-links. The increased load on Y-links is not an issue, since to begin with they are lesser loaded in comparison to the X-links. Overall, the routing leads to load balancing among the X-links, without making the Y-links potential bottleneck. This leads to reduction in the congestion resulting in better performance. The idea is illustrated in Figure 1. The figure shows a 2-dimensional torus of size $N_X = 8$ and $N_Y = 4$ (the wrap-around links are omitted for the sake of clarity). For a communication with S as the source node, one of the black nodes will be used as the intermediate node.

The next important aspect of the *SDR* technique is the exact choice of the intermediate nodes. Consider a 2-dimensional torus of size $N_X \times N_Y$, with $N_Y \geq N_X$. Consider a communication from node $u_1 = \langle x_1, y_1 \rangle$ to node $u_2 = \langle x_2, y_2 \rangle$. Our strategy is to route this data through an intermediate node $u' = \langle x_1, y' \rangle$ where $y' = (y_2 + N_Y/2) \mod N_Y$. Intuitively, this choice of u' provides the adaptive routing scheme maximum number of X-links as options to choose from, when sending data from u' to u_2. Therefore, our algorithm leads to better load-balancing on the X links. Figure 1 illustrates the idea: for a communication with S as the source node, the node labeled X will serve as the intermediate node.

However, when the torus is nearly symmetric (say $N_X \leq 2N_Y$) our particular choice of intermediate node may significantly increase load on Y links, possibly making the Y-links as the bottleneck. To overcome this hurdle, we employ the idea of *selective routing*. In this scheme, the intermediate node routing is performed only for a carefully selected subset of the source-destination pairs. To summarize, we carefully choose a subset of source-destination pairs and perform application level routing for these pairs via carefully chosen intermediate nodes. We then generalize the above ideas to higher dimensions.

We implement our optimizations on the Blue Gene/P supercomputer and demonstrate upto 35% speedup in performance for matrix transpose. Our experiments show that both the choice of the intermediate nodes and the idea of selective routing are critical in achieving the performance gain.

Related Work: Our work focuses on matrix transpose on asymmetric torus networks. The transpose problem has been well-studied on other interconnection architectures [7,4,1,5]. The basic idea of intermediate-node routing goes back to the work of Valiant [12]. Valiant considers permutation communications on hypercubes and shows that routing data via randomly chosen intermediate nodes helps in load balancing. Subsequently, oblivious routing algorithms have been studied extensively. In these work, the entire routing path for each source-destination pair is fully determined by the algorithm. They also consider arbitrary network topologies given as input in the form of graphs. We refer to the survey by Räcke [9] for discussion on this topic. In our case, we

focus on asymmetric torus networks, where the underlying hardware supports adaptive routing. Our goal is to use minimal application-level routing on top of the hardware-level adaptive routing and obtain load balancing. We achieve this by choosing a subset of source-destination pairs and routing the data for these pairs via carefully chosen intermediate nodes. We note that the routing from the source to the intermediate node and from the intermediate node to the destination is performed using the underlying hardware-level adaptive routing scheme. Our work shows that for permutation communication on asymmetric torus networks, the adaptive routing scheme can be improved.

2 Matrix Transpose Overview

In this Section, we give a brief overview of performing matrix transpose in parallel. We shall describe the underlying communication pattern which specifies the interactions amongst the processors in the system. Then, we discuss a popular multi-phase algorithm [3] that has been incorporated in the widely used ScaLAPACK [2] library. This algorithm is also used in the HPC Challenge Transpose benchmark [8].

In parallel numerical linear algebra software, matrices are typically distributed in a block cyclic manner [2]. Such a layout allows for operations to be performed on submatrices instead of individual matrix elements in order to take advantage of the hierarchical memory architecture. The processors in a parallel system are logically arranged in a $P \times Q$ grid. A matrix A of size $N \times M$ is divided into smaller blocks of size $b \times b$ which are distributed over the processor grid. For simplicity, we assume that b divides M and N; so the matrix can be viewed as an array of $M/b \times N/b$ blocks. Let $R_{p,q}$ denote the processor occupying position i, j on the processor grid where $0 \leq p < P$ and $0 \leq q < Q$. Further, let $B_{r,c}$ denote the block in the position r, c of the matrix where $0 \leq r < M/b$ and $0 \leq c < N/b$. In the block cyclic layout, the block $b_{r,c}$ resides on the processor $R_{r\%P,c\%Q}$[1].

In parallel matrix transpose, for each block $b_{r,c}$, the block must be sent to the processor containing the block $b_{c,r}$. This requires communication between the processors $R_{r\%P,c\%Q}$ and $R_{c\%P,r\%Q}$.

We now discuss the communication pattern for parallel matrix transpose. Let L denote the LCM of P, Q and G denote the GCD of P, Q. Using the generalized Chinese remainder theorem, it can be shown that a processor $R_{p,q}$ communicates with a processor $R_{p',q'}$ iff $p \equiv q' \pmod{G}$ and $q \equiv p' \pmod{G}$. Based on this fact, it can be shown that any processor communicates with exactly $(L/P \times (L/Q)$ processors. The multi-phase algorithm[3] works in $(L/P) \times (L/Q)$ phases. In each phase, every processor sends data to exactly one processor and similarly receives data from exactly one processor. The communication for processor $R_{p,q}$ is described in Figure 2. The important aspect to note is that the communication in each phase is a permutation communication.

[1] % denotes the mod operator.

$g = (q - p)\%G$
$\tilde{p} = (p + g)\%P$, $\tilde{q} = (q - g)\%Q$
For $j = 0$ to $L/P - 1$
  For $i = 0$ to $L/Q - 1$
    $p'_1 = (\tilde{p} + i \cdot G)\%P$
    $q'_1 = (\tilde{q} - j \cdot G)\%Q$
    Send data to $R_{p'_1,q'_1}$
    $p'_2 = (\tilde{p} - i \cdot G)\%P$
    $q'_2 = (\tilde{q} + j \cdot G)\%Q$
    Receive data from $R_{p'_2,q'_2}$

Fig. 2. Communication for processor $R_{p,q}$ in the multi-phase transpose algorithm

3 Load Imbalance on Asymmetric Torus Networks

As mentioned in the introduction, the shorter dimension routing technique is based on the observation that typical permutation communications on an asymmetric torus have higher load on links of the longer dimensions compared to the shorter dimensions. To demonstrate the above observation in a quantitative manner, here we consider random permutation communications on a 2-dimensional torus and show that the expected load on longer dimension links is higher than that shorter dimension links.

Consider a torus of size $N_X \times N_Y$. Consider a random permutation π. We assume that the underlying routing scheme is adaptive routing. We will show that the expected load on any X-link is $N_X/8$ and similarly the expected load on any Y-link is $N_Y/8$. For a link e, let L_e be the random variable denoting the load on the link e.

Theorem 1. *For any X-link e, $\mathrm{E}[L_e] = N_X/8$. Similarly, for any Y-link e, $\mathrm{E}[L_e] = N_Y/8$.*

Proof. Let L_X be the random variable denoting the sum of the load over all the X-links. Consider any node $u = \langle x_1, y_1 \rangle$. Every node is equiprobable to be the destination $\pi(u)$. Since we are dealing with a torus network, $\pi(u)$ is equally likely to be located to the left of u or to the right of u. In either case, the packet traverses 0 to $N_X/2$ X-links with equal probability. Therefore expected number of X-links traversed by the packet is $N_X/4$. Therefore.

$$\begin{aligned}\mathrm{E}[L_X] &= \sum_{u \in \mathcal{P}} \left(\text{ number of } X\text{-links traversed by packet from } u \text{ to } \pi(u) \right) \\ &= \sum_{u \in \mathcal{P}} (N_X/4) \;=\; |\mathcal{P}| \cdot (N_X/4),\end{aligned}$$

where $\mathcal{P} = N_X N_Y$ is the number of nodes. The total number of X-links is $2|\mathcal{P}|$ (considering bidirectionality of links) and no link is special. Thus, the expected load on the X-links is the same. It follows that

$$\mathrm{E}[L_e] = \frac{|\mathcal{P}| \cdot (N_X/4)}{2|\mathcal{P}|} = \frac{N_X}{8}.$$

The case of Y-links is proved in a similar manner. □

The above theorem shows that in case of asymmetric torus, the links of different dimensions have different expected load. For instance, if $N_X \geq 2N_Y$, the expected load on the X-links is twice that of the Y-links. Based on this observation, our application-level routing algorithms try to balance the load on the X-links by increasing load on the Y-links, while ensuring that the Y-links do not become the bottleneck.

4 Optimizing Permutation Communications

In this section, we discuss our heuristic application-level routing algorithm for optimizing permutation communications. As discussed earlier, matrix transpose is typically implemented in multiple phases, where each phase involves a permutation communication. As a consequence, we obtain improved performance for matrix transpose. Our application-level algorithm is based on two key ideas: the basic *SDR* technique and selective routing.

4.1 Basic Short Dimension Routing (SDR)

Consider a two-dimensional asymmetric torus of size $N_X \times N_Y$. Without loss of generality, assume that $N_X > N_Y$. As discussed in Section 3, for typical permutation communication patterns, the X-links are expected to be more heavily loaded in comparison to the Y-links. We exploit the above fact and design a heuristic that achieves better load balancing. Namely, for sending a packet from a source u_1 to a destination u_2, we will choose a suitable intermediate node u' lying on the same column as u_1. This ensures that the data from u_1 to u' traverses only the Y-links. The choice of u' is discussed next.

Consider a communication from a node $u_1 = \langle x_1, y_1 \rangle$ to a node $u_2 = \langle x_2, y_2 \rangle$. Recall that in adaptive routing, a packet from u_1 to u_2 may take any of the shortest paths from u_1 to u_2. In other words, the packet may traverse any of the links inside the smallest rectangle defined by taking u_1 and u_2 as the diagonally opposite ends. Let $\mathrm{D}_X(u_1, u_2)$ denote the number of X-links crossed by any packet from u_1 to u_2. It is given by $\mathrm{D}_X(u_1, u_2) = \min\{(x_1 - x_2) \mod N_X, (x_2 - x_1) \mod N_X\}$. Similarly, let $\mathrm{D}_Y(u_1, u_2) = \min\{(y_1 - y_2) \mod N_Y, (y_2 - y_1) \mod N_Y\}$. In adaptive routing, the load balancing on the X-links is proportional to $\mathrm{D}_Y(u_1, u_2)$, since a packet has as many as $\mathrm{D}_Y(u_1, u_2)$ choices of X-links to choose from. Similarly, the load balancing on the Y-links is proportional to $\mathrm{D}_X(u_1, u_2)$. Therefore maximum load balancing on the X-links can be achieved when we route the packets from u_1 to u_2 through the intermediate node $u' = \langle u_1, (u_2 + N_Y/2) \mod N_Y \rangle$. Note that the packets from u_1 to u' only traverse Y-links and do not put extra load on the X-links. Among the nodes that have the above property, u' is the node having the maximum D_Y value with respect

to u_2. Thus, the choice of u' offers maximum load balance in the second phase when packets are sent from the intermediate node to the destination.

An important issue with the above routing algorithm is that it may overload the Y-links resulting in the Y-links becoming the bottleneck. This would happen when the torus is nearly symmetric (for instance, $N_X \leq 2N_Y$). To demonstrate this issue, we present an analysis of the basic *SDR* based algorithm.

Analysis. Consider a random permutation π. The following Lemma derives the expected load on the X-links and the Y-links for the basic *SDR* algorithm. For a link e, let L_e be the random variable denoting the load on the link e.

Lemma 1. *For an X-link e, the expected load is* $\mathrm{E}[L_e] = N_X/8$ *and for a Y-link e, the expected load is* $\mathrm{E}[L_e] = 3N_Y/8$.

Proof. Let us split the overall communication into two phases: sending packets from every node to the intermediate nodes and sending the packets from the intermediate nodes to the final destination. Let us first derive the expected load for the case of X-links. In the first phase of communication, packets only cross the Y-links and do not cross any X-link. So, we need to only consider the second phase of communication. As in the case of Theorem 1, we have that the expected load $\mathrm{E}[L_e] = N_X/8$, for any X-link e.

Now consider the case of Y-links. For a Y-link e, let $L_{e,1}$ be the random variable denoting the number of packets crossing the link during the first phase.

For two numbers $0 \leq y, j < N_Y$, let $y \oplus j$ denote $y + j \mod N_Y$; similarly, let $y \ominus j$ denote $y - j \mod N_Y$. The Y-links come in two types: (i) links from a node $\langle x, a\rangle$ to the node $\langle x, a \oplus 1\rangle$; (ii) links from a node $\langle x, a\rangle$ to the node $\langle x, a \ominus 1\rangle$.

Consider a link e of the first type. (The analysis for the second type of Y-links is similar.) The set of nodes that can potentially use the link e is given by $\{\langle x, a \ominus i\rangle \ : \ 0 \leq i \leq N_Y/2\}$. Let u_i denote the node $\langle x, a \ominus i\rangle$ and let the intermediate node used by u_i be $\rho(u_i) = \langle x, y_i'\rangle$. The packet from the node $\langle x, y_i\rangle$ will cross the link e, if $y_i' \in \{a \oplus 1, a \oplus 2, \ldots, a \oplus (N_Y/2 - i)\}$ Since $\pi(u_i)$ is random, y_i' is also random. Therefore we have that

$$\Pr[\text{The packet from } u_i \text{ crosses } e \text{ in the first phase}] = \frac{(N_Y/2) - i}{N_Y}.$$

The expected number of packets that cross the link e in the first phase is:

$$\mathrm{E}[L_{e,1}] = \sum_{i=0}^{N_Y/2} \frac{(N_Y/2) - i}{N_Y} = \frac{N_Y}{8}.$$

Hence, the expected load $\mathrm{E}[L_{e,1}] = N_Y/8$.

In the second phase of the communication, every packet traverses $N_Y/2$ number of Y-links. As there are $N_X N_Y$ communicating pairs, the total load on the Y-links is $N_X {N_Y}^2/2$. The number of Y-links is $2N_X N_Y$. The expected load put in the second phase on each Y-link e is $N_Y/4$. Thus, for a Y-link e, the expected combined load on the link e is $\mathrm{E}[L_e] = N_Y/8 + N_Y/4 = 3N_Y/8$. □

This shows that our routing algorithm will perform well on highly asymmetric torus (i.e., when $N_X \geq 3N_Y$). However, when the torus is not highly asymmetric (for instance when $N_X \leq 2N_Y$), the Y-links become the bottleneck.

4.2 Selective Routing

We saw that for torus interconnects that are not highly asymmetric, the Y-links become the bottleneck in the above routing scheme. To overcome the above issue for such networks, we modify our routing algorithm to achieve load balancing on X-links without overloading the Y-links. The idea is to use application-level routing only for a chosen fraction of the communicating pairs. The chosen pairs communicate via the intermediate nodes which are selected as before. The remaining pairs communicate directly without going via intermediate nodes. Recall that for a communicating pair u_1 and u_2, the load imbalance is inversely proportional to the $\mathrm{D}_X(u_1, u_2)$. Therefore, we select the pairs having small value of D_X to communicate via intermediate nodes. The remaining pairs are made to communicate directly. We order the communicating pairs in increasing order of their D_X. Then, we choose the first α fraction to communicate via intermediate nodes and the rest are made to communicate directly. The fraction α is a tunable parameter, which is determined based on the values N_X and N_Y. When $N_X \geq 3N_Y$, we choose $\alpha = 1$; otherwise, α is chosen based on the ratio N_X/N_Y.

4.3 Extending to Higher Dimensions

Our routing algorithm can be extended for higher dimensional torus. We now briefly sketch the case of 3-dimensional asymmetric torus of size $N_X \times N_Y \times N_Z$. Without loss generality, assume that $N_X \geq N_Y \geq N_Z$. Consider communicating packets from a source node $u_1 = \langle x_1, y_1, z_1 \rangle$ to the destination node $u_2 = \langle x_2, y_2, z_2 \rangle$. Being a three dimensional torus, two natural choices exist for selecting the intermediate node: (i) $u' = \langle x_1, y_1, (z_2 + N_Z/2) \mod N_Z \rangle$; (ii) $u'' = \langle x_1, (y_2 + N_Y/2) \mod N_Y, (z_2 + N_Z/2) \mod N_Z \rangle$. We need to make a decision on which type of intermediate nodes to use. If we choose u' as the intermediate then the packets from source to the intermediate node will traverse only Z-links. On the other hand, if we choose u'' as the intermediate then the packets from source to the intermediate node will traverse both Y-links and only Z-links. In case N_Y and N_Z are close to N_X, we send all the packets directly without using intermediate nodes (Example: $N_X = N_Y = N_Z$). In case only N_Y is close to N_X, we use the first type of intermediate nodes (Example: $N_X = N_Y = 2N_Z$). Finally, if both N_Y and N_Z are considerably smaller, we use the second type of intermediate nodes (Example: $N_X = 2N_Y = 2N_Z$).

4.4 Implementation Level Optimizations

We fine tune our algorithm further, by employing the following strategies. These strategies were devised based on experimental studies.

Chunk based exchange (CBE): The source node divides the data to be sent to the destination into β smaller chunks. This allows the intermediate routing

node to forward the chunks received so far while other chunks are being received in a pipeline manner. Here β is a tunable parameter; our experimental evaluation suggests that $\beta = 1024$ gives best results.

Sending to nearby destinations: For pairs that are separated by a very small distance along the longer dimension, we choose not to perform intermediate node routing. For all pairs with distance less than γ along the longer dimension, we send the data directly. Here γ is a tunable parameter; our experimental evaluation suggests that $\gamma = 2$ gives best results.

Handling routing node conflicts: A node x may be selected as the intermediate routing node for more than one communicating pair, resulting in extra processing load on this node. To avoid this scenario, only one communicating pair is allowed to use x as an intermediate routing node. For the other communicating pairs, we look for intermediate nodes that are at distance less than δ from x. In case all such nodes are also allocated for routing, the pair is made to communicate directly. Here δ is a tunable parameter; our experimental evaluation suggests that $\delta = 2$ gives best results.

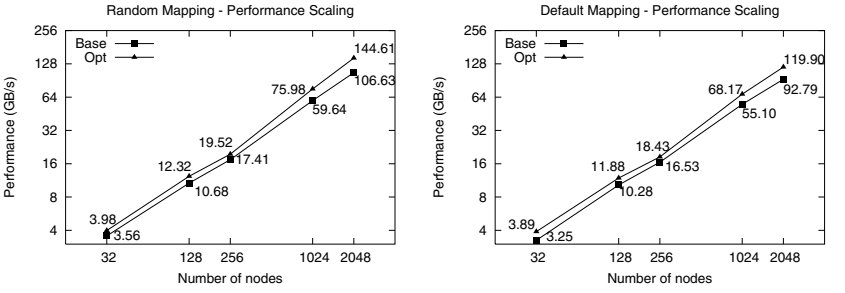
Computing intermediate nodes: Note that our algorithm needs to analyze the permutation communication pattern and choose an α fraction of source-destination pairs for which intermediate-node routing has to be performed. This process is carried out on any one of the nodes. The rest of the nodes send information regarding their destination to the above node. This node sends back information about the intermediate node to be used, if any. In case the multi-phase algorithm involves more than one phase, the above process is carried out on different nodes in parallel for the different phases.

5 Experimental Evaluation

In this Section, we present an experimental evaluation of our application-level routing algorithm on the Blue Gene/P supercomputer. We first present an overview of the Blue Gene supercomputer and then discuss our results.

5.1 Blue Gene Overview

The Blue Gene/P [10] is a massively parallel supercomputer comprising of quad-core nodes. The nodes themselves are physically small, allowing for very high packaging density in order to realize optimum cost-performance ratio. The Blue Gene/P uses five interconnect networks for I/O, debug, and various types of inter-processor communication. The most significant of these interconnection networks is the three-dimensional torus that has the highest aggregate bandwidth and handles the bulk of all communication. Each node supports six independent 850 MBps bidirectional nearest neighbor links, with an aggregate bandwidth of 5.1 GBps. The torus network uses both dynamic (adaptive) and deterministic routing with virtual buffering and cut-through capability.



(a) Random mapping (b) Default mapping

Fig. 3. Performance comparison

5.2 Experimental Setup

All our experiments involve performing transpose operations on a matrix distributed over a Blue Gene system. We consider systems of sizes ranging from 32 to 2048 nodes. The torus dimensions of these systems are of the form $d \times d \times 2d$ or $d \times 2d \times 2d$. The distributed matrix occupies 256-512 MB data on each node.

We consider two methods of mapping the MPI ranks to the physical nodes:

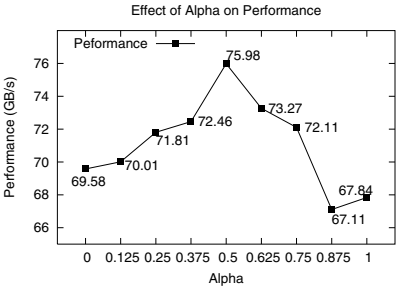
Default mapping: This is the traditional way of mapping MPI ranks to physical nodes on a torus interconnect. In this mapping, the ranks are allocated in a dimension ordered manner, i.e., the dimensions of the physical node may be determined by examining contiguous bits in the binary encoding of the MPI rank. For example the physical node for an MPI rank r may be determined using an XYZ ordering of dimensions as follows. The Z coordinate of the physical node would be $r \mod N_Z$, the Y coordinate would be $(r/N_Z) \mod N_Y$ and the X coordinate would be $r/(N_Z N_Y)$; here, divisions are integer divisions. The dimensions may be considered in other orders as well, for instance YXZ or ZXY.

Random mapping: In this case the MPI ranks are randomly mapped to the physical nodes. This results in a random permutation communication pattern. This is characteristic of the HPC Challenge Transpose benchmark [8], which is used to analyze the network performance of HPC systems.

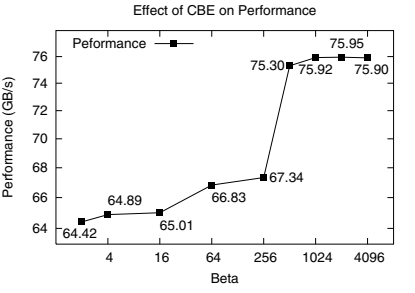
5.3 Results

We begin the result section with comparison of the performance of the multiphase transpose algorithm(*Base*) with our application-level heuristic routing algorithm (*Opt*). This comparison is followed by experimental study of effects of varying various heuristic parameters like $\alpha and \beta$. Also an attempt has been made to delineate the contribution of each optimization individually on the overall performance gain that has been obtained in *Opt*.

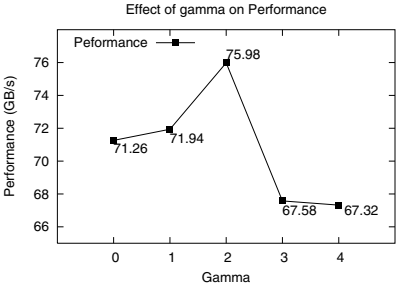
Comparison of *Opt* with *Base*: In our heuristic algorithm for optimal performance parameter α was set to .5, β to 1024, γ to 2 and δ to 2. These parameters



(a) Effect of α



(b) Effect of β



(c) Effect of γ

Version	*Performance (GB/s)*
Base	59.64
Base with CBE	60.86
Opt with $\delta = 1, \gamma = 0$	71.35
Opt with $\delta = 1, \gamma = 2$	73.74
Opt	75.98

(d) Effect of Other Factors

Fig. 4. Contributions to Performance

have been obtained experimentally. The results are shown in Figure 3. The X-axis is the system size (number of nodes) and the Y-axis is the performance of matrix transpose communication obtained in GB/s.

In Figure 3(a), the performance results are obtained by mapping the MPI tasks onto the nodes using the random mapping. We see that *Opt* provides significant improvements over *Base*. The gain varies from 12% for 32 nodes to 35% for 2048 nodes. In Figure 3(b), the performance results are obtained by mapping the MPI tasks onto the nodes using the default mapping. We see that *Opt* provides significant improvements over *Base*. The gain ranges from 11% to 29%, with the best performance being observed on 2048 nodes.

In both the graphs, we observe that the performance is significantly better for the systems of size 1024 and 2048 in comparison to the smaller system sizes. This is due to the fact the underlying interconnection network for these system sizes is a torus whereas for the smaller system sizes, it is a mesh. It is to be noted that the number of paths between intermediate routing nodes and destination nodes is double in case of a torus as compared to a mesh. Therefore, the intermediate routing node in a mesh offers limited gain in the number of paths to the destination in comparison to the torus. Our results demonstrate the benefit of our application-level routing heuristic algorithm. The overheads involved in initially determining the intermediate nodes were found to be negligible.

Effect of α: The effects of varying α for 1024 nodes is shown in Figure 4(a). The performance improves as α is increased to about 1/2. This is expected, as the load gets balanced on the longer dimension links. As we further increase α, the performance drops as the shorter dimension links become more congested.

Effect of β: Figure 4(b) shows the effect of varying β for 1024 nodes. As expected, the performance improves as β is increased. The performance saturates for large values of β.

Effect of γ: The result of varying γ is shown in Figure 4(c) for 1024 nodes. In the extreme case when $\gamma = 0$, i.e., all the communicating pairs try to use intermediate routing nodes, the performance is as low as 71 GB/s. The performance increases as *gamma* is increased to 2. As *gamma* is increased further, the performance begins to drop as a large proportion of pairs communicate directly, thereby loosing the advantage of SDR.

We also conducted experiments to separately study the contributions of the different optimizations. These results are presented in Figure 4(d).

References

1. Azari, N., Bojanczyk, A., Lee, S.: Synchronous and asynchronous algorithms for matrix transposition on mcap. In: Advanced Algorithms and Architectures for Signal Processing III. SPIE, vol. 975, pp. 277–288 (1988)
2. Blackford, L.S., Choi, J., Cleary, A., Petitet, A., Whaley, R.C., Demmel, J., Dhillon, I., Stanley, K., Dongarra, J., Hammarling, S., Henry, G., Walker, D.: Scalapack: a portable linear algebra library for distributed memory computers - design issues and performance. In: Supercomputing 1996: Proceedings of the 1996 ACM/IEEE conference on Supercomputing, CDROM (1996)
3. Choi, J., Dongarra, J., Walker, D.: Parallel matrix transpose algorithms on distributed memory concurrent computers. Parallel Comp. 21(9), 1387–1405 (1995)
4. Eklundh, J.: A fast computer method for matrix transposing. IEEE Trans. Comput. 21(7), 801–803 (1972)
5. Johnsson, S., Ho, C.: Algorithms for matrix transposition on boolean n-cube configured ensemble architecture. SIAM J. Matrix Anal. Appl. 9(3) (1988)
6. Kumar, V.: Introduction to Parallel Computing (2002)
7. Leary, D.: Systolic arrays for matrix transpose and other reorderings. IEEE Transactions on Computers 36, 117–122 (1987)
8. Luszczek, P., Dongarra, J., Koester, D., Rabenseifner, R., Lucas, B., Kepner, J., Mccalpin, J., Bailey, D., Takahashi, D.: Introduction to the hpc challenge benchmark suite. Tech. rep (2005)
9. Räcke, H.: Survey on oblivious routing strategies. In: CiE 2009: Proceedings of the 5th Conference on Computability in Europe, pp. 419–429 (2009)
10. IBM journal of Reasearch, Development staff: Overview of the ibm blue gene/p project. IBM J. Res. Dev. 52(1/2), 199–220 (2008)
11. Thakur, R., Rabenseifner, R.: Optimization of collective communication operations in mpich. International Journal of High Performance Computing Applications 19, 49–66 (2005)
12. Valiant, L.: A scheme for fast parallel communication. SIAM J. of Comp. 11, 350–361 (1982)