Collectives on Two-tier Direct Networks

Nikhil Jain, JohnMark Lau and Laxmikant Kale

Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
{nikhil,johnlau,kale}@illinois.edu

Abstract. Collectives are an important component of parallel programs, and have a significant impact on performance and scalability of an application. To obtain best performance, platform specific implementations of various parallel programming frameworks, such as MPI and Charm++, are done. As a result, when systems with new network topologies are built, new topology aware algorithms for collectives are added to these frameworks that also contain the topology oblivious algorithms. In this paper, we propose topology aware algorithms for collectives performed on two-tier direct networks such as IBM PERCS and Dragonfly. We observe that, for large message operations, significant performance gains can be made by taking advantage of large number of links in a two-tier direct network. We evaluate proposed algorithms using an analytical model based on link utilization.

Keywords: Collectives, Topology, Two-tier networks, PERCS, Dragonfly

1 Introduction

On the road to Exascale, there is a strong possibility that parallel machines of the future will have a large number of fast cores on each node and a low network bytes-to-flop ratio. Communication is becoming expensive whereas computation continues to become cheaper. Hence, scalable, low-diameter and fast networks will be desirable for building multi-Petaflop/s and Exaflop/s capability machines. New designs have been proposed recently by IBM (the PERCS topology [2]), and by the DARPA sponsored *Exascale Computing Study* on technical challenges in hardware (the dragonfly topology [6]). Both these topologies are two-tier direct networks with all-to-all connections at each level.

Many scientific applications use data movement collectives such as *Broadcast, Scatter, Gather, Allgather, All-to-all,* and computation collectives such as *Reduce, Reduce-scatter, and Allreduce* [1]. The performance of these MPI collectives is critical for improved scalability and efficiency of parallel scientific applications. In recent years, there have been an increasing number of applications such as web analytics, micro-scale weather simulation and computational nanotechnology, that involve processing extremely large scale data requiring collective operations with large messages. Performance of such large message collectives is significantly affected by network bandwidth constraints.

Most of the existing networks such as torus and fat-trees are low radix, and have constant number of links attached to a node. As such, transmitting

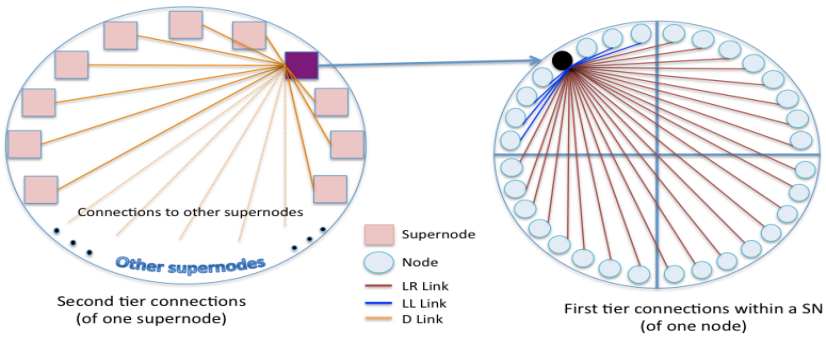

Fig. 1. The PERCS network – the right side shows first-level connections within a supernode; the left side shows second-level connections across supernodes (we show connections of only one node and one supernode respectively for clarity).

packets from a source to destination involve traversal through a large number of nodes/switches. The multiplicity of hops makes these networks congestion prone, especially when performing collectives on large data/messages. To counter the effects of congestion, carefully designed topology aware algorithms have been used for collectives on such networks [4, 5]. In addition, there is a set of topology oblivious algorithms which perform reasonably well on most systems [7, 8, 3]. However, most of these algorithms do not seem to be a good fit for two-tier direct networks as they may not be able to make full use of the all-to-all connectivity in two-tier direct networks. In this paper, we propose a new set of topology aware algorithms, which we refer to as *two-tier algorithms*, for collectives on two-tier direct networks for large messages. These algorithms exploit the high radix nodes and the multi-level structure of a two-tier direct network. Hence, they are better suited for two-tier direct networks, and as we demonstrate later should perform significantly better than other algorithms. A cost model based on the link utilization is used to evaluate the effectiveness of proposed algorithms in comparison to general topology oblivious algorithms. To best of our knowledge, this is the first paper which deals with collectives on two-tier direct networks.

2 Two-tier Direct Networks

In this section, we provide an introduction to two-tier networks using IBM's PERCS network as an example. The elementary unit is called a node: a multi-core chip with connection to and from the system network. Any communication initiated by the cores is sent to the network manager of the node. In a two-tier direct network, nodes at the first level are grouped logically to form cliques. These cliques are further grouped to form larger clusters. In the PERCS topology, these logical groups are called drawers and supernodes whereas in the dragonfly topology they are called groups and racks (or cabinets). The supernodes are connected at the second level to form a larger clique.

As a concrete example, in the right side of Figure 1, we show one supernode of the PERCS topology. Within the supernode, a circle represents a node. Eight nodes in each quadrant constitute a drawer. Every node has a hub/switch that has three types of links originating from it - LL, LR and D links. LL and LR links constitute the first tier connections that enable communication between any two nodes in one hop. On the left side of Figure 1, the second tier connections between supernodes are shown. Every supernode is connected to every other supernode by a D or L2 link. These inter-supernode connections originate and terminate at hub/switches connected to a node; a given hub/switch is directly connected to only a fraction of the other supernodes. Any packet which is to be sent from a node (N1) in a supernode(S1) to a node (N2) in another supernode (S2) first need to be sent to that node in S1 which is connected to S2. Thereafter, the packet is sent to S2 and forwarded to N2 if required using the first level links.

In this paper, we do not differentiate among first level links (LL and LR) and denote them by L1 links. The links at second level are denoted as L2 links. We also stick to core, node and supernode terminology of PERCS, but the same principles apply to Dragonfly or any other two-tier direct network. We also assume that a node is capable of sending data simultaneously on all links originating from it.

3 Cost Model and Assumptions

We assume an inorder mapping of MPI ranks or cores onto the system. Consider a system with sn supernodes, each consisting of nps (nodes per supernode) nodes with cpn (cores per node) cores each. Hence, we have $p = sn * nps * cps$ cores whose inorder mapping is performed as following. Consider a global numbering of supernodes from 0 to $sn-1$. Within a supernode and a node, nodes and cores are locally numbered from 0 to $(nps-1)$ and from 0 to $(cpn-1)$ respectively. In the global space, cores are numbered (by MPI) from 0 to $(p-1)$ using the core's supernode, node and position within the node as the key. For example, cores in supernode 0 get ranks from 0 to $(nps * cpn - 1)$. Following which, cores in supernode 1 get ranks from $nps * cpn$ to $(2 * nps * cpn - 1)$ and so on.

Further, we assume a two-tier network with round robin connection for L2 links at node level. A connection from supernode $S1$ to supernode $S2$ originates at node ($S2$ *modulo nps*) in supernode $S1$. This link connects to node ($S1$ *modulo nps*) in supernode $S2$. Therefore, each node is connected to $spn = \frac{sn}{nps}$ supernodes. We consider the case in which job allocation onto nodes and supernodes happen in a uniform manner. To keep things simple, we assume the cases where the entire machine is being used by an application. Algorithms and results for the other case, where allocation is not uniform, can be derived with minor variations and will be discussed in a future work.

As we focus on large message collectives, we use a bandwidth based model to estimate the cost of a collective algorithm. The start up cost and latency effects are ignored as the bandwidth term dominates for large messages. We assume that the time taken to send a message between any two nodes is $n\beta$, where β is the transfer time per byte, if only 1 link is being used to send n bytes of data. In

case of a computation operation, we add a γ computation cost component per byte. We also use a two step approach to find link utilization which provides a more accurate estimate of performance of an algorithm. In the first step, given a collective operation, the algorithm to use, number of MPI ranks or cores and the data length information (required by the operation), *pattern-generator* generates a list of communication exchange between every pair of MPI ranks. The data generated by *pattern-generator* is fed to *linkUsage*. Given a list of communication exchange, *linkUsage* generates the amount of traffic that will flow on each link in the given two-tier network.

4 Topology Oblivious Algorithms

This section lists the algorithms which are generally used to perform various collective operations in a topology oblivious manner for large message sizes. Many of these algorithms, which are listed in Table 1, are used in MPICH as the default option [8].

Operation	**Algorithm**	**Cost** (n bytes)
Scatter	Binomial Tree	$\frac{p-1}{p}n\beta$
Gather	Binomial Tree	$\frac{p-1}{p}n\beta$
Allgather	Ring, Recursive Doubling	$\frac{p-1}{p}n\beta$
Broadcast	DeGeijn's Scatter with Allgather [3]	$2\frac{p-1}{p}n\beta$
Reduce-Scatter	PairWise Exchange	$\frac{p-1}{p}(n\beta + n\gamma)$
Reduce	Rabenseifner's Reduce-Scatter with Gather [7]	$\frac{p-1}{p}(2n\beta + n\gamma)$

Table 1. Commonly used Algorithms

5 Two-tier Algorithms

Given the clique property and the multiple levels of connections, the two-tier networks naturally leads to a new set of algorithm which we refer to as *two-tier algorithms*. The common idea in any two-tier algorithm is stepwise dissemination, transfer or aggregation (SDTA) of data. SDTA refers to simultaneous exchange of data within a level in order to optimize the over all data exchange. Performing SDTA ensures that the algorithms use maximum possible links for best bandwidth, and collate information to minimize the amount of data exchanged at higher levels. Without loss of generality let us assume that the root of any operation is core 0 of node 0 of supernode 0. In our discussion, we use core to refer to any entity which takes part in the collective operation. An MPI Rank and Charm++ Chare are examples of such entities.

5.1 Scatter and Gather

Scatter is a collective operation used to disseminate core specific data from a source core to every other core. The two-tier algorithm for Scatter using SDTA is as follows:

1. Core 0 of node 0 of supernode 0 sends data to core 0 of every other node in supernode 0. The data sent to a core is the data required by the cores residing in the supernodes connected to the node of that core.
2. Core 0 of every node within supernode 0 sends data to core 0 of every node outside supernode 0 that the node is connected to. The data sent to a node is the data required by the cores in the supernode to which this destination node belongs.
3. Core 0 of every node that has data (including node 0 of supernode 0) sends data to core 0 of every other node within its supernode. This data is required by the cores within the node that the data is being sent to.
4. Core 0 of every node shares data, required by the other cores, with all other cores in their node.

The four step process described above implies that the source core first spreads the data within its supernode. The data is then sent to exactly one node of every other supernode by the nodes which received the data. Thereafter, nodes which have data to be distributed within their supernode spreads the data within their supernodes. Gather can be performed using this algorithm in the reverse order.

For collectives with personalized data for each core such as Scatter, the dissemination of data can also be done using direct message send. The data will take exactly the same path as described in the above scheme. We have described our approach using Scatter because of its simplicity, and ease of understanding.

5.2 Broadcast

Broadcast can be performed using the approach used for Scatter if the entire data, without personalization, is sent in the four steps. We refer to this type of Broadcast as *base broadcast*. However, using the following scheme better performance can be obtained.

1. Core 0 of node 0 of supernode 0 divides the data to be broadcasted into nps chunks and sends chunk i to core 0 of node i of supernode 0.
2. Core 0 of every node within supernode 0 sends data to core 0 of exactly one node outside supernode 0 that the node is connected to. Exactly one node is chosen to avoid duplication of data delivery in following steps.
3. Core 0 of every node that received data in the previous step sends data to core 0 of every other node within their supernode.
4. Core 0 of all the nodes that received data in Step 2 and Step 3 send data to core 0 of all other nodes outside their supernode that they are connected to.
5. Now, these cores share data with core 0 of all other nodes in their supernode.
6. Core 0 of every node shares data with all other cores in their node.

This algorithm begins with the source core dividing the data into chunks, and distributing it within its supernode (as if performing Scatter over a limited set of cores). In the second step, every node in supernode 0 share the chunk with exactly 1 node outside their supernode. Thereafter, the nodes which received the chunk in the previous step share the data with other nodes in their supernode.

As a result, all nodes in some of the supernodes have a chunk of initially divided data which needs to be sent to other supernodes. This is done in the next step, following which all nodes, which have received a chunk so far, share these chunks with other nodes in their supernode.

5.3 Allgather

An Allgather operation is equivalent to Broadcast being performed by all cores simultaneously. The SDTA based algorithm begins with all cores within every node exchanging data and collecting it at core 0 of the node. In the second step, all nodes within a supernode exchange data in an all-to-all manner using L1 links, and thus every node in every supernode contains the data which a supernode wants to broadcast to other supernodes. In the following step, supernodes exchange data in an all-to-all manner in parallel. Finally the nodes which receive data in the previous step disseminate this data to other nodes within its supernode. In addition, core 0 of every node has to share this data with all other cores in its node. This algorithm can be seen as a base broadcast being done by all nodes simultaneously (refer to §5.2).
Please note that many a times, multiple steps of SDTA can be performed by a send from the source of one step to eventual destination of the following step. An example case will be when core 0 of node 0 of supernode 0 has to send data to core 0 of nodes that are connected to other nodes of supernode 0. We have presented them as separate steps in which initially core 0 of node 0 sends the data to core 0 of other nodes of supernode 0. These nodes then forward the data to core 0 of nodes of other supernodes. This has been done only for ease of understanding, and comparison results will not reflect them.

5.4 Computation Collectives

Although the same two-tier approach presented in the previous section can be used to perform computation collectives such as Reduce, it may not result in the best performance. The inefficiency in the previous approach derives from the fact that computation collectives require some computation on the incoming data, and therefore if some node receives a lot of data from multiple sources, the computation it has to perform on the incoming data will become a bottleneck. We assume that the multiple cores do not share memory, and hence will not be able to assist in the computation to be performed on the incoming data. Also, the presented algorithms assume commutative and associative reduction operation.

Let us define an owner core as the core that has been assigned a part of the data that needs to be reduced. This core receives the corresponding part of the data from all other cores and performs the reduction operation on them. Consider a clique of k cores on which a data of size m needs to be reduced, and be collected at core 0. The algorithm we propose for such a case is the following:

1. Each core is made owner of $\frac{m}{k}$ data - assume a simple rank based ownership.
2. Every core sends the data corresponding to the owner cores (in their data) to the owner cores.

3. The owner cores reduce the data they own using the corresponding part in their data, and the data they receive.
4. Every owner core sends the reduced data to core 0.

Essentially, what we are doing is a divide and conquer strategy. The data is divided among cores, and they are made responsible for reduction on that data. Every core divides their data, and sends the corresponding portion to the owner cores. The owner cores reduce the data, and eventually send it to core 0.
Reduce - The above strategy can be used in multiple stages to perform the overall reduction in a two-tier network:

1. Perform reduction among cores of every node; collect the data at core 0.
2. Perform reduction among nodes of every supernode - owners among nodes are decided such that instead of collecting data at node 0, the data can be left with the owner nodes and directly exchanged in the next step. This may require a node to be owner of scattered chunks in the data depending on the supernode connections.
3. Perform reduction among supernodes and collect the data at supernode 0.

Reduce-Scatter - We can use the same algorithm as above to perform Reduce-scatter with a minor modification. Since the Reduce-scatter requires the reduced data to be scattered over all cores, in the last phase of reduction (i.e. reduction among supernodes), we decide owners of data such that a supernode becomes owner of the data which its cores are required to receive in a reduce-scatter. Thereafter, instead of collecting all data at supernode 0 in the final step, the algorithm scatters the data within every supernode as required by Reduce-scatter.

6 Experiments

This section presents the details and results of the experiments we have conducted. The two-tier network that has been simulated for these experiments consists of 64 supernodes. Each supernode consists of 16 nodes each of which has 16 cores. The given configuration implies that there are 4032 L2 links and 15360 L1 links in the system. Note that we ignore the time spent in sharing data within a node by the cores.

6.1 Cost Comparison

In Table 2, we present comparison of the two-tier algorithms with other algorithms using the cost model mentioned in §3. Among the data collectives, for Scatter and Gather, we observe that the two-tier algorithms which distributes data using all L1 links simultaneously within a source supernode provides theoretical speedup of factor *nps* i.e. nodes per supernode. This speedup may be affected by *sn*, i.e., the number of supernodes. If there are too few L2 links, they may become the bottleneck, and the speedup hence is bounded by $min\{nps, sn\}$. For Allgather, we find that the speedup provided by two-tier algorithms depends on both *sn* and *nps*. For Broadcast, which happens in three phases, the theoretical speedup is $\frac{nps}{3}$. Finally, for computation collectives, we observe that our

approach leads to more computation being performed. This is because the reduction happens in two phases and some computation, which could have been avoided, is performed. However, as with data collectives, the speedup for data transfer is substantial and should mask the effect of increase in computation.

Operation	**Base Cost**	**Two Tier Cost**
Scatter	$\frac{p-1}{p}n\beta$	$n\beta * max\{\frac{1}{nps}, \frac{1}{sn}\}$
Gather	$\frac{p-1}{p}n\beta$	$n\beta * max\{\frac{1}{nps}, \frac{1}{sn}\}$
Allgather	$\frac{p-1}{p}n\beta$	$n\beta(\frac{1}{nps} + \frac{1}{sn} + \frac{1}{sn*nps})$
Broadcast	$2\frac{p-1}{p}n\beta$	$n\beta(\frac{3}{nps})$
Reduce-Scatter	$\frac{p-1}{p}(n\beta + n\gamma)$	$n\beta(\frac{1}{nps} + \frac{1}{sn} + \frac{1}{sn*nps}) + 2n\gamma$
Reduce	$\frac{p-1}{p}(2n\beta + n\gamma)$	$n\beta(\frac{1}{nps} + \frac{2}{sn}) + 2n\gamma$

Table 2. Cost Model based Comparison

6.2 Scatter, Gather and Broadcast

We consider a Scatter operation in which the root sends 64 KB data to each of the remaining cores. In Table 3, we present a comparison of binomial algorithm link utilization with the two-tier algorithm. The important thing to note in the comparison is the maximum load binomial algorithm puts on a link in comparison to what two-tier algorithm puts. For L1 links, we find that two-tier algorithm puts a maximum load of 64 MB whereas binomial algorithm performs much worse, and puts a load of 141 MB. The difference is much more significant when it comes to L2 links where binomial algorithm puts a factor 32 times more load. Exactly same results are found for Gather operation due to its inverse nature to Scatter.

	Scatter		Broadcast	
	Binomial	Two-tier	DeGeijn	Two-tier
L1 Links Used	1036	960	1588	15360
L1 Links Min Traffic	1 MB	1 MB	2 MB	64 MB
L1 Links Max Traffic	141 MB	64 MB	1.1 GB	128 MB
L2 Links Used	56	63	95	3937
L2 Links Min Traffic	16.7 MB	1 MB	32 MB	64 MB
L2 Links Max Traffic	520 MB	16 MB	1.09 GB	64 MB

Table 3. Link Usage Comparison for Scatter and Broadcast

We also present the link utilization statistics for a 1 GB Broadcast in Table 3. Link utilization improves substantially both in terms of number of links used and the load which is put on links when two-tier algorithm is used. We expect an order of magnitude improvement in the execution time as the worst case link load goes down from 1.1 GB to 128 MB.

6.3 Allgather

As mentioned earlier, we study the performance of Allgather using two algorithms - recursive doubling and ring. The amount of data that each MPI rank/core wants to send is 64 KB. In Table 4, we present comparison of two-tier algorithm with the recursive doubling and ring algorithm. It can be seen that while two-tier algorithms uses all the available L1 and L2 links in the system, the other two algorithms use a very small fraction of available links. Moreover, the load which two-tier algorithm puts on the links is orders of magnitude smaller in comparison to the other algorithms. It strongly suggests that the two-tier algorithm will outperform the other two algorithms. These results also conforms with the fact that for large messages, ring algorithm is better than recursive-doubling [8].

6.4 Computation Collectives

In the Table 5, we present a comparison of link utilization for Reduce-scatter and Reduce. For this experiment, the overall reduction size is 1 GB, and hence

	Recursive Doubling	Ring	Two-tier Algorithm
L1 Links Used	10496	1080	15360
L1 Links Min Traffic	16 MB	1 GB	65 MB
L1 Links Max Traffic	15.1 GB	1 GB	65 MB
L2 Links Used	384	634	4032
L2 Links Min Traffic	4.2 GB	1 GB	16 MB
L2 Links Max Traffic	4.3 GB	1 GB	16 MB

Table 4. Link Usage Comparison for Allgather

each core receives 64 KB reduced data when Reduce-Scatter is performed. We observe an order of magnitude difference in the load put on the links by two-tier algorithms in comparison to other algorithms. This can be attributed to the step wise manner in which two-tier algorithms perform reduction. Only the necessary data go out of a node or a supernode, and hence two-tier algorithm reduces the load put on the links significantly. Given this large difference in communication load, two-tier algorithms should outperform most other algorithms despite the additional computational load they put on the cores.

	Reduce-Scatter		Reduce	
	Pairwise Exchange	Two-tier	Rabenseifner	Two-tier
L1 Links Used	15360	15360	15360	15360
L1 Links Min Traffic	2 GB	65 MB	2 GB	66 MB
L1 Links Max Traffic	2 GB	65 MB	3 GB	130 MB
L2 Links Used	4032	4032	4032	4032
L2 Links Min Traffic	4 GB	16 MB	4 GB	16 MB
L2 Links Max Traffic	4 GB	16 MB	5 GB	32 MB

Table 5. Link Usage Comparison for Reduce-scatter and Reduce

7 Conclusion and Future Work

In this paper, we presented a new set of algorithms for two-tier networks, which takes advantage of the topology. A comparison, based on a cost model and network utilization, has been done to assess the performance of these new algorithms in comparison to well know algorithms. We focused on collectives for large data sizes, and showed that the two-tier algorithms significantly outperform most other algorithms for a two-tier direct network. In future, we plan to focus on collectives for small data sizes, and potentially improve the performance for large data size. We also plan to look at cases in which only a (non uniform) part of system is allocated to an application.

Acknowledgement

This project was initiated as part of a course taught by Professor Josep Torrellas, whom the authors gratefully thank. This research was supported in part by the Blue Waters: Leadership Petascale System project (which is supported by the NSF grant OCI 07-25070).

References

1. MPI: A Message Passing Interface Standard. In: MPI Forum
2. Arimilli, B., Arimilli, R., Chung, V., Clark, S., Denzel, W., Drerup, B., Hoefler, T., Joyner, J., Lewis, J., Li, J., Ni, N., Rajamony, R.: The PERCS High-Performance Interconnect. In: 2010 IEEE 18th Annual Symposium on High Performance Interconnects (HOTI). pp. 75–82 (August 2010)
3. Barnett, M., Gupta, S., Payne, D.G., Shuler, L., Geijn, R., Watts, J.: Interprocessor Collective Communication Library (InterCom). In: In Proceedings of the Scalable High Performance Computing Conference. pp. 357–364 (1994)
4. Faraj, A., Kumar, S., Smith, B., Mamidala, A., Gunnels, J., Heidelberger, P.: Mpi collective communications on the blue gene/p supercomputer: algorithms and optimizations. In: Proceedings of the 23rd international conference on Supercomputing. pp. 489–490. ICS '09 (2009)
5. Jain, N., Sabharwal, Y.: Optimal bucket algorithms for large mpi collectives on torus interconnects. In: Proceedings of the 24th ACM International Conference on Supercomputing. pp. 27–36. ICS '10 (2010)
6. Kim, J., Dally, W.J., Scott, S., Abts, D.: Technology-driven, highly-scalable dragonfly topology. SIGARCH Comput. Archit. News 36, 77–88 (June 2008)
7. Rabenseifner, R.: A new optimized MPI reduce algorithm (1997)
8. Thakur, R., Gropp, W.D.: Improving the Performance of Collective Operations in MPICH. Lecture Notes in Computer Science 2840, 257–267 (October 2003)