Fat-Trees Routing and Node Ordering Providing Contention Free Traffic for MPI Global Collectives

Eitan Zahavi
Mellanox Technologies LTD
Yokneam, Israel
eitan@mellanox.co.il

***Abstract*—As the size of High Performance Computing clusters grows, the increasing probability of interconnect hot spots degrades the latency and effective bandwidth the network provides. This paper presents a solution to this scalability problem for real life constant bisectional-bandwidth fat-tree topologies. It is shown that maximal bandwidth and cut-through latency can be achieved for MPI global collective traffic. To form such congestion-free configuration, MPI programs should utilize collective communication, MPI-node-order should be topology aware, and the packets routing should match the MPI communication patterns. First, we show that MPI collectives can be classified into unidirectional and bidirectional shifts. Using this property, we propose a scheme for congestion-free routing of the global collectives in fully and partially populated fat trees running a single job. Simulation results of the proposed routing, MPI-node-order and communication patterns show a 40% throughput improvement over previously published results for all-to-all collectives.**

Keywords- Network Topologies; Routing Algorithms and Techniques; Collective Communication

I. INTRODUCTION

In recent years, the ever increasing demand for compute-power is addressed by building multiple thousand nodes High Performance Computing (HPC) clusters exceeding the peta-flop per second barrier [1]. State of the art parallel programs running on these clusters utilize the standard Message Passing Interface (MPI). Message passing overcomes the need to implement concurrent shared memory among the distributed processes. Many MPI applications exhibit a ratio of communication to computation time which is proportional to the number of nodes they use. For these parallel applications, the network latency and bandwidth may inhibit the desired linear performance acceleration with cluster size. There are two main network topologies used by HPC clusters: fat-trees and 3D tori. Fat-trees are used for diverse traffic patterns while tori are used for problems limited to nearest neighbor communication.

With the rise in HPC cluster size, a recent study [2] has measured degradations of network performance, down to 40% of the bandwidth provided by fat-trees to MPI communication. The source of the degradation is attributed to cases where traffic from multiple sources congests a particular network link, creating a "hot spot". Simulations show that the average degradation is 40% of the nominal network bandwidth for random MPI-node-order. Furthermore, for adversarial node-order, the obtained bandwidth is only 7.1% of the network bandwidth which represents a degradation of 92.9%.

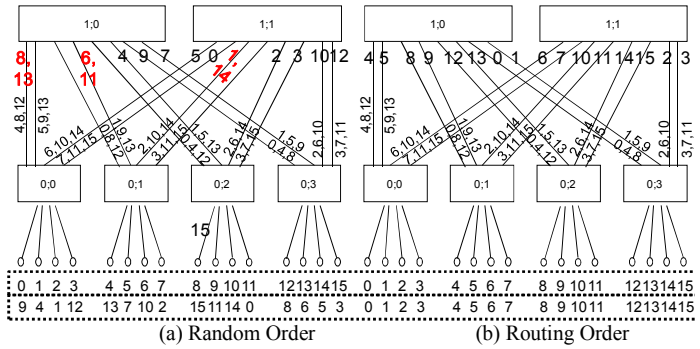

Figure 1. Routing and MPI Node Order Cause or Prevent Blocking: Destination = (Source + 4) mod 16; (a) Random Order Cause 3 Hot-Spots; (b) Routing Order is Congestion-free

The example provided in Figure 1 demonstrates how congestion can be formed if node-ordering is not considering the routing, and how routing aware ordering is congestion-free. The figure shows the routing, node-order and resulting flows for the traffic pattern: destination = (source + 4) % 16. The lowest row of numbers represents the MPI node order. The row above provides the node numbers used as destination addresses by the routing. Routing appears right above the switches as the set of destination routed through each link. E.g. the left most switch routes packets to destinations 4, 8 and 12 on the first link from the left. The numbers on the top represent the destination MPI-node-numbers flowing through each link. For the random MPI node-order on (a) pairs of nodes appear on three links – representing three links which will cause blocking. When applying routing aware order (b) each link carry one flow – resulting with congestion-free traffic.

This paper describes how to find a routing aware node-ordering and prove that using such ordering with D-Mod-K routing (enhanced to handle real-life-fat-trees) solves the network scalability problem for a single job running global MPI collectives on the cluster. Once used, the network is congestion-free, providing full bandwidth and cut-through latencies. Unlike the Adaptive-Routing solutions to the same problem, which are reactive and thus suffer from inherent loss of throughput during the adaptation stage, our approach is proactive in preventing the hot-spots from occurring.

Furthermore, not all network transports (e.g. InfiniBand™ Reliable Connected) can be adaptively routed due to the resulting out-of-order packets delivery while our approach is agnostic to the transport used.

The paper starts with a discussion about the problem in Section II and continues with a survey of the MPI communication patterns leading to important observations about their common characteristics in section III. Then, an extension of the known formal definition of fat-tree topologies to support real life HPC topologies is presented in section IV. Utilizing MPI collectives and node ordering it is shown in sections V and VI that a D-Mod-K routing algorithm enables global communicator traffic to pass in a congestion-free fashion on these topologies. Section VII presents simulation results of the routing, MPI-node-order and traffic sequencing for several large clusters of InfinibandTM nodes, running multiple MPI communication patterns without congestion.

II. MPI Collectives Scalability With Fat Tree Size

MPI collectives are functions implementing communication algorithms [21]. They are optimized to give the lowest latency and hide the details of buffering and communication from the programmer. MPI communicators are aliases for the set of nodes that participate in the collectives. The "global" communicator represents all the nodes. Detailed description of MPI collectives is provided in section III. The measurements performed by [2] have shown that the effective bandwidth for various MPI global communicator collectives degrades down to ~40% of the network capacity. This bandwidth loss is attributed to hot-spots caused by the communication pattern. These results were reproduced by a simulation of a 1944 nodes InfiniBand™ cluster, routed deterministically, using a OMNeT++ simulation platform [20]. The simulation model is calibrated against InfiniBand™ QDR links (4000MBps unidirectional bandwidth) of Mellanox IS4 switches (36 ports) connected to hosts with PCIe Gen2 8X slots (supporting 3250MBps unidirectional bandwidth).

Two traffic patterns representative of collective communications (presented later) named "Shift" and "Recursive-Doubling" were simulated by translating their algorithm into sequences of destinations specific for each end-port. A random node-ordering of the MPI node-number to cluster end-ports is used in the translation. During the simulation, the end-ports progress through their destinations sequence independently when the previous message has been sent to the wire. The resulting normalized effective bandwidth (total-bytes / time) for different message sizes is provided in Figure 2 (normalized to the full PCIe bandwidth).

It can be seen from the graph that as the message size is increases the effective bandwidth is decreases. The reason for that is the increased probability for head of line blocking as the message size grows. The Shift collective translates into a long destination sequence (1943 stages for 1944 nodes job, each stage every node communicates with another node) thus congestion short delays are averaged over the entire sequence. However, for the short destination sequence (of length $\lceil log_2(1944) \rceil = 11$stages) required by the Recursive-Doubling such averaging does not happen and the effective bandwidth is reduced even for short messages.

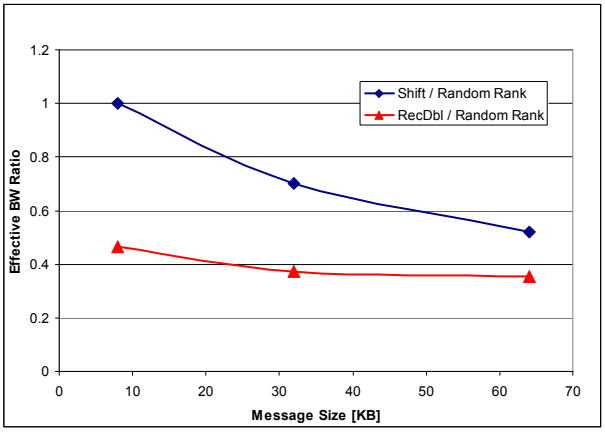

Figure 2. Shift and Recursive Doubling Collectives Normalized BW vs. Message Size

To further illustrate the importance of correct routing and MPI-node-order (i.e. MPI "rank" assignment) a simulation is performed for a Ring permutation sequence with deterministic routing and adversary node ordering. The adversary node-order was made such that all of the nodes of each leaf switch send data to nodes of other leaf switches. The selection of the particular destinations for each node is done such that for each leaf switch all flows congest on a single up-going port. The results show that the effective bandwidth can drop by a factor of 14 for some permutation sequences. The measured average bandwidth for this traffic pattern is 231.5MBps which is close to the network bandwidth of 4000MBps divided by the worst possible link oversubscription of 18. The normalized effective bandwidth ratio obtained is 7.1%.

In all the above simulations it was assumed that end-ports progress through the stages of the permutation sequence in an asynchronous manner, so periods of contention and periods of no contention may average out on each link. When synchronization is performed all the end-ports change their destination synchronously. So the maximal number of flows contending on all the links dictates the worst completion time for each stage in the destination sequence.

Further study of the impact of cluster size on the probability for hot spot and its degree was conducted using a tool (based on ibdm [23]) that given a topology, traffic pattern and routing computes per-link Hot Spot Degree (HSD) which is the number of flows sent through the link. Fat-tree topologies for 128, 324, 1728 and 1944 nodes were studied. The analyzed traffic is representative of global communicator Binomial, Butterfly, Dissemination, Ring,

Shift, Tournament collectives (presented and analyzed later in section III). Figure 3 shows the average of the maximal hot-spot-degree of all links, over all stages of the collective algorithm. The result is further averaged over 25 different random MPI-node-orders cases. Error bars show minimal and maximal average HSD obtained for all MPI-node-order. The "Ring", "Shift" and "Butterfly" collectives exhibit exponential behavior such that without routing aware MPI-node-order and permutation aware routing the effective bandwidth for large jobs is significantly degraded.

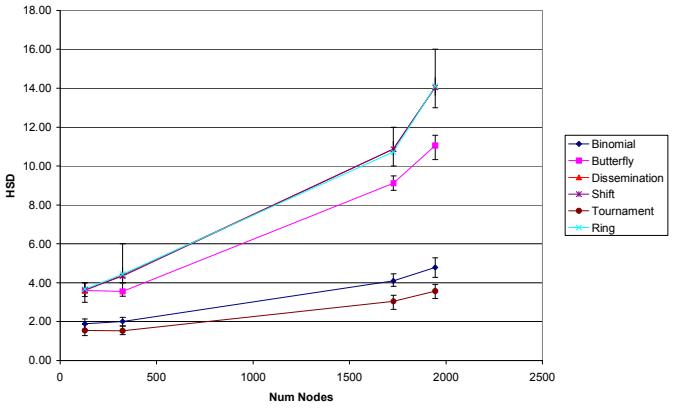

Figure 3. MPI Collectives Hot-Spot-Degree vs. Cluster Size for various MPI Collective Communication Patterns

III. MANY MPI COLLECTIVES, JUST TWO PERMUTATION SEQUENCES

This section presents an analysis of MPI collective communication primitives that are shown to be key to many HPC applications [4][5][6]. Consequently, in the last years, researchers have focused on optimizing MPI collective's algorithms, and proposing new ways to organize the communication for different message and job sizes [7][8][9]. However, the performance model used by all known published studies assumes a perfect network and ignores the added latency imposed by network hot-spots. The simulations presented in section II show that this assumption is inaccurate and leads to the performance scalability problems reported by [2]. To better understand the impact of collective communication, a novel decomposition of collective algorithms into two parts is presented. The first part, named Collective Permutation Sequence (CPS) classifies the permutation of source-destination pairs that communicate at each stage of the collective. The second part defines the content of the communication, i.e. the data exchanged. Applying this decomposition to MVAPICH and OpenMPI collectives we realized only 8 different permutation sequences are being used[1]. Finally, we identify two sequence patterns that combine all the permutations used by these collectives.

[1] Although more CPS can be defined we assume by now the design-space of possible algorithms is mostly investigated.

In order to provide a non blocking routing solution for MPI collectives, there is a need to identify common characteristics of these collectives that may enable the desired routing solution. We have studied the collectives algorithms described by [7][8][10], and our conclusions are summarized in this section. Utilizing the decomposition of collective algorithms into CPS and message content, our study is focused on the CPS part of the algorithms and a survey of the collectives used by OpenMPI and MVAPICH was conducted. The resulting set of permutation sequences[2] and their usage by the collective algorithms of the two MPI implementations is provided in Table 1.

TABLE 1 MPI COLLECTIVE PERMUTATION SEQUENCE USAGE BY MVAPICH AND OPENMPI COLLECTIVE ALGORITHMS

Collective / Algorithm / Sequence	AllGather			Barrier	Broadcast		AllToAll			Reduce Scatter		Reduce Scatter Commutative	AllReduce		Reduce		Scatter	Gather
	Ring	Bruck	Doubling		Old	V.d.Geijn	Shift	Xor	Bruck	Old	New	New	Old	Rabenseifner	Old	Rabenseifner		
Binomial Tree				o	m o O	m M				m M				M	o m M	m	m o O	m M o O
Butterfly Recursive Double	m 2		o 2	m o	o	m		M 2				m o	o m M					
Recursive Half										o	m M			M		m		
Dissemination				m o					m o									
Reverse Dissemination		mo																
Tournament																		
Ring	M O				M O	M				m M O			O		O		M O	
Shift							M O					M O						

The rows of Table 1 represent different CPS while the columns represent different MPI collective's algorithms. Each cell marks the use of a permutation sequence by one of the MPI implementations. The labeling is: 'm' or 'M' for MVAPICH and 'o' or 'O' for OpenMPI, small or large messages respectively. A marking of '2' means usage is limited to a node count which is a power of 2. The table contains 18 different algorithms used by these MPI libraries which employ only 8 different CPS.

A formal definition of the different permutation sequences is provided in Table 2. The formulation uses: N as number of end-ports, n_i and n_j as end-port index i and j respectively and s as stage indices. Three key observations can be made. First, all CPS conform to the principle of constant displacement, i.e. for each stage of communication, for all pairs communicating in that stage; the distance

[2] A few small message collective algorithms (like k-tree) exists which are relying on the network buffering ability to absorb short congestion events. These algorithms are not utilizing permutation sequences and as such will always create short hot spots. Due to the characteristics of these algorithms they are left out of the scope of this paper

(modulo the number of end-ports) between the index of the source and the index of the destination is constant. Second, there are only two types of CPS: unidirectional and bidirectional. For unidirectional CPS the displacement is always positive. For the bidirectional CPS the inclusion of a pair in a communication stage implies that the reverse pair also exists in that stage. Finally, since the Shift CPS incorporates stages for all the distance values 1 to ***N-1*** it is a super set of the permutations used by all other unidirectional CPS.

TABLE 2 COLLECTIVE PERMUTATION SEQUENCES FORMAL DEFINITION

CPS		Definition
Dissemination	Unidirectional	$n_i \rightarrow n_{(i+2^s) mod\ N} \mid \forall i: 0 \le i < N, \quad 0 \le s \le \lceil log_2 N \rceil$
Tournament	Unidirectional	$n_{(i+2^s)} \rightarrow n_i \mid \forall i: 0 = i\ mod\ 2^{s+1}; \quad 0 \le s \le \lceil log_2 N \rceil$
Shift	Unidirectional	$n_i \rightarrow n_{(i+s)\ mod\ N} \mid \forall i: 0 \le i < N, \quad 1 \le s \le N-1$
Ring	Unidirectional	$n_i \rightarrow n_{(i+1)\ mod\ N} \mid \forall i: 0 \le i < N$
Binomial*	Unidirectional	$n_i \rightarrow n_{(i+2^s)} \mid \forall i: 0 \le i < 2^s \wedge i + 2^s < N \quad 0 \le s \le \lceil log_2 N \rceil$
Recursive Doubling	Bidirectional	$n_i \leftrightarrow n_{(i \oplus 2^s)} \mid \forall i: 0 \le i < N \wedge i \oplus 2^s < N \quad 0 \le s \le \lceil log_2 N \rceil$
Recursive Halving	Bidirectional	$n_i \leftrightarrow n_{\left(i \oplus \frac{N}{2^s}\right)} \mid \forall i: 0 \le i < N \wedge i \oplus \frac{N}{2^s} < N \quad 0 \le s \le \lceil log_2 N \rceil$

* Algorithms of Binomial CPS use either the left or right arrow

For example, the Ring CPS has a single stage in which node-0 sends data to node-1, node-1 to node-2 and so on. The Binomial CPS for 1024 nodes has 11 stages. On the first stage, ***s=0***, only node-0 is sending data to node-1. On the second stage, ***s=1***, node-0 sends to node-2 and node-1 to node-3. On the third stage node-0 sends to node-4, node-1 to node-5, node-2 to node-6 and node-3 to node-7. Note that for the Binomial CPS the distance of the destination is constant in every stage. So the permutations of the Binomial CPS are contained (the source destination pairs) in one of the permutations making a Shift CPS stage.

IV. REAL LIFE FAT-TREE FORMULATION

This section introduces a new family of fat-tree topologies which are being used in real clusters. Although there are several fat-tree representations in the literature, there is none that is sufficient to describe real life fat-trees building today's HPC clusters. The progress from k-ary-n-trees [11][12] through Extended Generalized Fat-Trees (XGFT) [13], to this paper contribution of Parallel Ports Fat-Trees (PGFTs) and Real Life Fat-Trees (RLFTs) will be discussed in the following.

A. Why Parallel Ports Generalized Fat-Trees are required?

Even though XGFTs support a different number of up-going or down-going ports at each level of the tree they cannot describe existing real life topologies. For economical reasons real clusters are built from switches with high port-count as the integration results with lower price per port. With k-ary-n-tree or GFTs the switches at the root of the tree have only half of their ports connected. XGFTs fix that by allowing different number of ports at each level. So the maximal sized topology built using the same switch can be expressed as XGFTs. However, when a smaller topology is required XGFTs cannot represent the desired topology.

For example, to build a 16-nodes-cluster with 8 port switches. Only 4 leaf switches are required having 4 end-nodes connected to each. In order to preserve the Cross Bisectional Bandwidth (CBB) 16 links are connected to the second level switches, 4 from each leaf switch. Since XGFT only allows a single link between two switches we end up with 4 second level switches. The resulting XGFT topology is provided in Figure 4(a) which is only using 4 out of the 8 ports available for each spine switch. To overcome this limitation of XGFTs they are extended by introducing the concept of Parallel Ports Generalized Fat Trees (PGFT). The PGFT in Figure 4(b) uses two parallel ports to keep the CBB thus require only 2 spine switches.

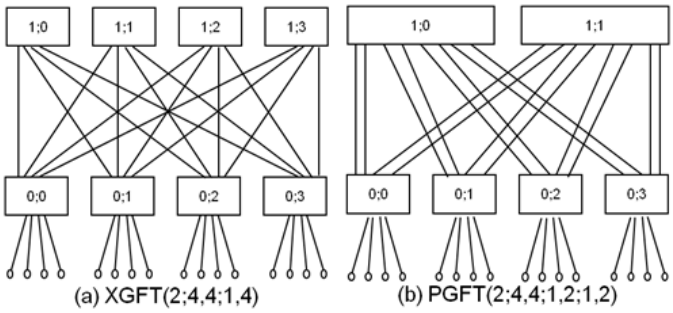

Figure 4. An example of a Non-Maximal Fat-Tree showing XGFT cannot describe full CBB while PGFT can

B. PGFT Definition

PGFTs are canonically defined by the tuple: $PGFT(h; m_1, ...m_h; w_1, ...w_h; p_1, ...p_h)$ where ***h*** is the number of levels in the tree; $\boldsymbol{m_i}$ is the number of different lower level nodes connected to nodes on level ***l***; $\boldsymbol{w_i}$ is the number of different upper level nodes connected to nodes on level ***l-1*** and $\boldsymbol{p_i}$ is the number of parallel links connecting between nodes in level ***l*** and ***l-1***. The XGFT defined by [13] is extended to PGFT by introducing up-going and down-going port objects. Like in XGFT tuples notation, each node is assigned a tuple $(l, a_h, ...a_1)$, where ***l*** is its level and the vector of digits $\mathbf{a}_i$ describe the sub-trees the node is located at. The $\boldsymbol{a_h}$ digit represents the switch index within the top most sub-tree. Recursively $\boldsymbol{a_{h-1}}$ represents the index of the sub-tree within that first sub-tree, and so on.

Figure 5(a) shows a single node (circle) and its ports (hexagons). There are $\boldsymbol{w_{l+1}p_{l+1}}$ up-going ports and $\boldsymbol{w_l p_l}$ down-going ports. Each port is assigned a tuple of the form: $(l, a_h, ...a_1, q)$ which is equal to its node tuple with the addition

of a port number - ***q***. To construct a PGFT one can first draw the nodes and ports on each level and then connect the ports between the levels using the following rules: ports $(l, a_h, ...a_{l+1}...a_1, q)$ and $(l+1, b_h, ..., b_{l+1}...b_1, r)$ are connected if and only if $b_i = a_i$ for all the digits except for $i = l+1$, and the first of the p_{l+1} connections will be between the up-going port $q = b_{l+1}$ and the down-going port $r = a_{l+1}$. The ***k*** connection is between the up-going port $q = b_{l+1} + kw_{l+1}$ and the down-going port $r = a_{l+1} + km_{l+1}$. Figure 5(b) demonstrates these connections for two nodes at levels 2 and 3: the 2 LSB digits of these two nodes match – so they must be connected. The first connecting link is between port number 0 of the lower node (which equals the 3rd digit of the upper node) and port number 1 of the upper node (which equals the 3rd digit of the lower nodes).

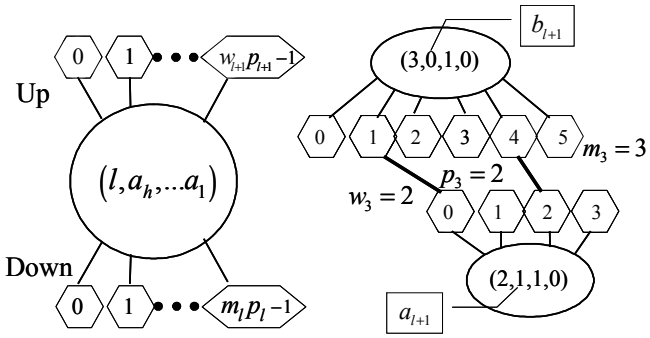

Figure 5. Introducing PGFT Nodes, Ports and their Connections

C. Real Life Fat-Trees (RLFT)

XGFTs and PGFTs support the definition of a large variety of topologies. Not all of them are practical to build. This section describes the characteristics of the Real Life Fat-Trees (RLFT), a sub-class of PGFTs, which are further studied by the rest of this paper. The set of attributes that makes a PGFT into an RLFT are presented below.

The first restriction for a PGFT to be routed in a non blocking manner is that it preserves a constant CBB. If CBB is not constant and is reduced going up the tree, some links must carry more than one flow at a given communication stage of a Shift CPS (since in each stage all of the nodes send data). The constant CBB requirement means that the nodes input BW equals their output BW or: $m_l p_l = w_{l+1} p_{l+1}$.

The second restriction applied to the PGFT is that the end-ports are actually not switches but host network interface cards which connect to the PGFT via a single cable i.e. $w_1 = p_1 = 1$.

The third restriction stems from the practical cost aspect of large HPC PGFTs: Real life HPC are designed using the same port-count cross-bar switches. RLFTs addressed in the rest of the paper will assume all of the switches have the same number of ports. At each level > 0: $m_l p_l + w_{l+1} p_{l+1}$. It is common to define switch *arity* ≡ K which is half of the switch ports: $K = \left(m_l p_l + w_{l+1} p_{l+1}\right)/2$. The top level of the tree has only down-going ports and thus: $m_h p_h = 2K$.

V. Non Blocking Routing for Unidirectional CPS

Several previous studies [14][15][16][17][18] have addressed the topic of providing deterministic routing for fat-trees for specific topologies. Most of them describe routing which we refer to as D-Mod-K. The D-Mod-K routing was extended by [22] to RLFTs. The basic property of the D-Mod-K routing is that each down-going port is used to pass traffic to a single end-port, so blocking may only happen for traffic going up the tree. This section focuses on the evaluation of RLFT D-Mod-K routing showing it provides congestion free traffic under conditions that are satisfied by running a unidirectional MPI global collective of a single job.

A. Fully Populated RLFTs

The RLFT D-Mod-K routing provides a close form function defining the up-going-port to be used to route to destination j on a specific switch $(l, s_h..s_1)$:

$$P_l^U(j) \equiv (l, s_h..s_1, q_l^U) \mid q_l^U(j) = \left\lfloor j / \prod_{k=1}^{l} w_k \right\rfloor \bmod \left(w_{l+1} p_{l+1}\right) \quad (1)$$

The following theorems proved in this paper appendix show that D-Mod-K routing is providing congestion-free Shift CPS when all the nodes of the RLFT participate in the Shift and are allocated according to the RLFT node indexing order:

Theorem 1: The routing according to (1) guarantees that no more than one destination is routed through any of the up-going ports in the network, for all stages of the Shift CPS on a complete RLFT.

Theorem 2: The routing according to (1) guarantees that no more than one destination is routed through all the network down-going ports of a complete RLFT.

Section III concludes that Shift CPS is a superset of all unidirectional CPS. Combining it with RLFT ordered MPI-node-order and D-Mod-K routing not only provides congestion-free for Shift CPS but to the rest of the unidirectional CPS.

An intuitive way to understand this routing and the claim for its non-blocking property is to consider traffic flowing up the tree from a fully populated leaf switch (switch at level 1). A Shift permutation sequence guarantees that for a

contiguous set of traffic sources the set of traffic destinations is also contiguous and in the same order. To avoid congestion on up-going links, the proposed routing scheme spreads the traffic among all up-going ports. For the lowest level leaf-switches, the index of the up-going port route for a given destination is set to be the destination index modulo the total number of up-going ports. The up-going port assignment is cyclic with the destination number such that at any given stage of the Shift, the contiguous range of destinations is evenly distributed through all of the up-going ports in a non-overlapping manner. Destinations routed through a second level switch share the same port index at the first level switches. These destinations form an arithmetic sequence with a difference equal to the number of up-going ports of the first level switches. Such a sequence can be spread without overlaps on the up-going ports of the second level switches by dividing the destination index by the sequence distance modulo the number of up-going ports. Applying this principle recursively on all of the tree levels provides a non blocking routing for the Shift CPS.

Figure 6 holds a fragment of a PGFT to demonstrate the routing described above. Up-going ports are marked with their port number and the level 1 routing is shown to be through port number q=j mod 4. The set of destinations through each port is provided on the right most leaf switch. For example, consider traffic flowing from the right most node at level 1 toward destination 9. Traffic traverses the node at level 1 through port 1 and then the node at level 2 through port 2. Note that every 4 contiguous destinations are to be routed through different up-going ports. For the second level switches the routing to destination j is through port number q=(j/4) mod 4, and the same property holds.

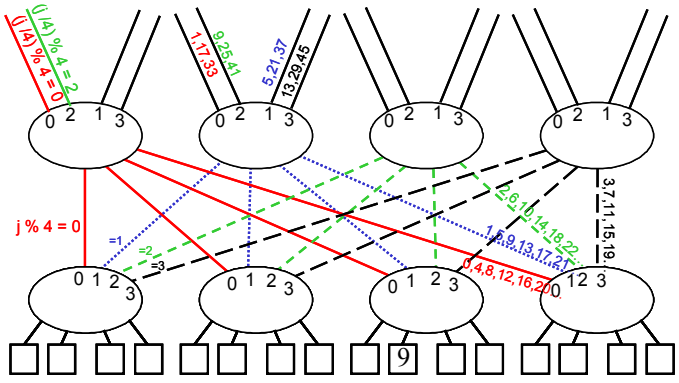

Figure 6. Example Routing on a PGFT Sub-Section

B. *Partially Populated RLFTs*

The proof of theorem 1 does not depend on the exact location in the tree of the destination. It only requires that for all Shift stages, destinations will be contiguous modulo $\prod_{k=1}^{l} w_k$ for every $1 \le l \le h$ which translates to $N = iN_{CF} = i\prod_{k=1}^{h} w_k$ (where ***i*** is an arbitrary integer value). This conclusion provides a key contribution for this paper: Routing (1) provides congestion-free Shift permutation sequence (and thus, all of the other unidirectional permutation sequences) for any set of $N = i\prod_{k=1}^{h} w_k$ nodes. For example, the maximal three layer RLFT built from 36 ports switches, RLFT(3; 18, 18, 36; 1, 18, 18; 1, 1, 1), has 36 different sub allocations that can provide congestion-free unidirectional MPI collective in multiplications of 324 nodes.

Many large scale HPC installations are used as utility clusters where several jobs are run in parallel at any given time. Since this paper focuses on running a single very large job, it is beyond the scope of this paper to describe how they can be allocated and routed to meet congestion-free traffic.

VI. Non Blocking Traffic for Bidirectional CPS

Both bidirectional CPS Recursive-Doubling and Recursive-Halving use the same permutation sequence played in reverse. The provided definition for the Recursive-Doubling is:

$$n_i \leftrightarrow n_{i\oplus 2^s} \mid \forall i : 0 \le i < P \wedge i \oplus 2^s < P \, s \in \{0 \cdots \lfloor \log_2(N) \rfloor\} \qquad (2)$$

It lacks the handling of cases where the number of nodes is not a power of 2. A common solution to handling non-power-of-2 job size, by the MPI implementations, is to precede the CPS with a "pre" permutation:

$$n_i \leftarrow n_{i+2^s} \mid \forall i : 0 \le i < N - 2^S; s = \lfloor \log_2(N) \rfloor$$

and conclude with additional "post" permutation:

$$n_i \rightarrow n_{i+2^s} \mid \forall i : 0 \le i < N - 2^S; s = \lfloor \log_2(N) \rfloor$$

This way the first set of nodes act like proxies for the reminder nodes above the highest power of 2 that fit within N. To obtain congestion-free traffic for the Bidirectional CPS on the global communicator, it is possible to extend the idea of adding "pre" and "post" permutations and apply the same concept for every layer of the tree.

Theorem 3: If traffic going up the tree through a switch is guaranteed to be either $n_i \rightarrow n_{i\oplus 2^s}$ or the reverse $n_i \leftarrow n_{i\oplus 2^s}$ no blocking will occur.

The proof is simple and builds on theorem 1: Since the distance between the source and the destination is fixed during each stage, the traffic flowing up the tree is matching a Shift stage so theorem 1 conditions are met. To construct traffic that meets the requirement of theorem 3 it is sufficient to restrict the permutation sequence to follow the tree structure: In each stage the communication is allowed between end-ports grouped by their leaf switches and is of the same hierarchical distance from each other. Formally: given a $PGFT(h; m_h, \ldots m_1; w_h, \ldots w_1; p_h, \ldots p_1)$ congestion-free Recursive Doubling permutation sequence is defined by a set of group of communication stages denoted by *grp.stage* .

Each group matches one level of the tree and defines communication between nodes that have their first common parent at that level of the tree.

For each level of the tree the following constants are defined:

$$L_l = \lfloor \log_2(m_l) \rfloor ; M_l = \prod_{k=1}^{l} m_k ; E_l = M_{l-1} 2^{L_l} \quad (3)$$

In each group of stages there may be a pre-stage (denoted -1) and post-stage (+1) if $E_l \neq M_l$, that deal with the non power of 2 conditions. These pre and post stages are:

$$P_{l.-1} : \{ j \to i \mid E_1 \le j \bmod m_l \wedge j = i + E_l \wedge 0 \le i < N \} \quad (4)$$

$$P_{l.+1} : \{ i \to j \mid E_l \le j \bmod M_l \wedge j = i + E_l \wedge 0 \le i < N \} \quad (5)$$

The bulk of the group stages is different for ***l=1*** and the rest of the groups:

$$P_{1.s} : \left\{ \begin{array}{l} i \to j \mid j = \left(i \bmod m_l \right) \oplus 2^S + \left\lfloor \frac{i}{m_l} \right\rfloor m_l \wedge \\ 0 \le s < L_1 \wedge i \bmod m_l < E_1 \wedge 0 \le i < N \end{array} \right\} \quad (6)$$

$$P_{l.s} : \left\{ \begin{array}{l} i \to j \mid j = \left(\left\lfloor \frac{i}{M_{l-1}} \right\rfloor \oplus 2^S \right) M_{l-1} + i \bmod M_{l-1} \wedge \\ 0 \le s < L_l \wedge i \bmod M_l < E_1 \wedge 0 \le i < N \end{array} \right\} \quad (7)$$

For a fully populated RLFT there are at most 2 extra stages required for every level of the tree if ***K*** is not a power of 2. For a partially populated tree, the number of stages used does not reflect the actual number of the used end-ports but the number of leaf switches they occupy. In any case, a topology aware permutation sequence as described above is sufficient to avoid congestion if applied on top of routing (1).

VII. Experimental Results and Verification

To validate the performance of the proposed routing, the routing (1) was coded on top of "ibdm" - an InfinibandTM data-model [23] - which is freely available as part of the OFA "ibutils" package. The package provides various utilities for parsing a file holding the topology and then manipulating the resulting in-memory data-structures. The calculated routing tables are then used by the simulation model presented in section II. Repeating the Shift and Recursive-Doubling permutation sequence simulations on this model, while using MPI-node-order matching the routing algorithm, provides the expected full bandwidth and cut-through latency.

To enable evaluation of the HSD for many topologies and with reasonable runtime, an analytical model was used to compute the flow of traffic through the network links. This model is also available as part of the "ibdm" package. Table 3 lists the various cases that were tested. Cases of partial tree job size (marked in the table as "Cont. –X") were generated by randomly selecting a set of nodes excluded from the communication. The calculated HSD, matches the behavior of the MPI in case of a fully synchronized move between stages and is a worst case analysis, i.e. if an average HSD of one is reported, there is no congestion in any of the stages of the permutation sequence. All of the cases tested, covering 2 and 3 level fat-trees, fully and partially populated show that the proposed routing and mpi-node-order results in congestion-free traffic, i.e. HSD = 1. With such routing, distributed synchronization issues, such as the OS jitter, may still prevent the MPI collectives from obtaining the full network bandwidth and experiencing the minimal latency. However they have been shown to be reduced by using clock synchronization protocols. To show the relative improvement of the proposed MPI-node-order versus random MPI-node-order, the last column, marked as "Random Ranking Avg HSD" holds the average of the worst number of contending paths in all the permutation sequence stages under random MPI-node-order. Improvement factors up to 5.2 are observed.

Table 3 Hot Spot Analysis of Various Topologies and CPS

K	Topology: PGFT tuple	Total End-ports	CPS End-Ports	End-Port Selection	CPS	Rand Rank Avg HSD
12	2;12,12;1,12;1,2	144	144	ALL	Shift	3.75
18	2;16,16;1,16;1,2	324	324	ALL	Shift	4.32
12	3;12,12,12;1,12,12;1,1,2	1728	1728	ALL	Shift	5.24
18	3;18,18,6;1,18,6;1,1,3	1944	1944	ALL	Shift	5.41
12	2;12,12;1,12;1,2	144	120	Cont. -2K	Shift	3.50
18	2;16,16;1,16;1,2	324	288	Cont. -2K	Shift	4.28
12	3;12,12,12;1,12,12;1,1,2	1728	1584	Cont. -K^2	Shift	1.98
18	3;18,18,6;1,18,6;1,1,3	1944	1296	Cont. -2K^2	Shift	5.22
12	2;12,12;1,12;1,2	144	144	ALL	RecDbl	2.9
18	2;16,16;1,16;1,2	324	324	ALL	RecDbl	3.25
12	3;12,12,12;1,12,12;1,1,2	1728	1728	ALL	RecDbl	4.26
18	3;18,18,6;1,18,6;1,1,3	1944	1944	ALL	RecDbl	4.26
12	2;12,12;1,12;1,2	144	120	Cont. -2K	RecDbl	2.8
18	2;16,16;1,16;1,2	324	288	Cont. -2K	RecDbl	3.7
12	3;12,12,12;1,12,12;1,1,2	1728	1584	Cont. -K^2	RecDbl	4.47
18	3;18,18,6;1,18,6;1,1,3	1944	1296	Cont. -2K^2	RecDbl	4.06

VIII. Conclusion

As demonstrated in Table 3 the proposed methodology of using coherent node-ordering, D-Mod-K routing and MPI collective permutation sequences provides congestion-free traffic for a single job running on fully or partially populated Real Life Fat-Trees. This result improves network performance by 40% over published measurements. Large HPC computers, designed today, may benefit from the results of this paper for preserving network scalability for MPI collectives. It is the combination of the two worlds: routing and collective design that enables these results.

IX. APPENDIX: PROOFS FOR THEOREMS 1 AND 2

Theorem 1: The routing according to (1) guarantees that no more than one destination is routed through any of the up-going ports in the network, for all stages of the Shift CPS on a complete RLFT.

The following lemmas are used to prove theorem 1:

Lemma 1: The set of destinations routed through up-going ports of a node $(l, b_h, ...b_1)$, not including the top nodes, is a sub-set of the algebraic sequence:

$$J_{(l,b_h,...b_1)} = \left\{ \sum_{t=1}^{l} b_t \prod_{k=1}^{t-1} w_k + i \prod_{k=1}^{l} w_k \mid 0 \le i < N \Big/ \prod_{k=1}^{l} w_l \right\} \quad (8)$$

The motivation for using the super-set described by (8) rather than the actual sequence is that the expression for the accurate set is much more complex than (8) and is not required for proving theorem 1. Note that in the accurate sequence, destinations that are descendants of a node pass through it but are not routed through the up-going ports (but the down-going ports).

The proof of lemma 1 is based on a recursion starting with the destinations of end-ports: Each end-point may send data to all of the other end-points: $J_{(0,b_h,...b_1)} = \{ i \mid 0 \le i < N \}$

Based on the routing (1) for level 0: $q_0^U(j) = j \bmod (w_1 p_1)$ and the PGFT connection rule $b_{l+1} = q \bmod w_{l+1}$ the sequence of destinations passing through the parent with $b_1 = q$ start with destination b_1 and a step of w_1 so $J_{(1,b_h,...b_1)} = \{ b_1 + i w_1 \mid 0 \le i < N/w_1 \}$. Since all of the children nodes of $(1, b_h, ...b_1)$ connect to it through $b_1 = q$ they all pass the same sequence of destinations to that parent.

Similarly the sequence of destinations passing through second level nodes, is obtained using the routing (1) on each element of the set $J_{(1,b_h,...b_1)}$:

$$J_{(2,b_h,...b_1)} = \{ b_1 + b_2 w_1 + i w_1 w_2 \mid 0 \le i < N/w_1 w_2 \} \quad (9)$$

Finally for a node at arbitrary level the sequence of destinations passing through it is:

$$J_{(l,b_h,...b_1)} = \left\{ \sum_{t=1}^{l} b_t \prod_{k=1}^{t-1} w_k + i \prod_{k=1}^{l} w_k \mid 0 \le i < N \Big/ \prod_{k=1}^{l} w_l \right\}$$

□

Lemma 2: Routing (1) is non-blocking for any continuous sub-sequence of destinations passing through a node (not including the top level nodes) of size equal to the number of up-going ports.

Proof of lemma 2: Given a destination of index ***i***, from the above sequence, the up-going port obtained by (1) is:

$$q_l^U(j) = \left\lfloor \frac{\sum_{t=1}^{l} b_t \prod_{k=1}^{t-1} w_k + i \prod_{k=1}^{l} w_k}{\prod_{k=1}^{l} w_k} \right\rfloor \bmod (w_{l+1} p_{l+1}) \quad (10)$$

$$q_l^U(j) = (C + i) \bmod (w_{l+1} p_{l+1}) \quad (11)$$

So a contiguous sub-sequence of $w_{l+1} p_{l+1}$ destinations which makes a contiguous range of index values maps to a cyclic set of $w_{l+1} p_{l+1}$ up-going ports. So for such sub-sequence all up-going ports are used, each for exactly one destination □

Lemma 3: For RLFTs routing (1) is non-blocking for any contiguous sub-sequence of destinations passing through a node (not including the top level nodes) of size equal to the number of up-going ports which may wrap around from the last possible index to the first element of the destination sequence

Proof of lemma 3: To prove lemma 3 we show that the up-going port used for routing the next destination past the last one is the same as the one used for routing to the first destination. This condition is met if the expression for the number of destinations in the sequence is a multiple of $w_{l+1} p_{l+1}$.

For RLFTs the number of nodes is:

$$N = \prod_{i=1}^{h} m_i = \frac{2K^h}{\prod_{k=1}^{h} p_k} \quad (12)$$

The index after the last index is:

$$\frac{N}{\prod_{k=1}^{l} w_k} = \frac{2K^h}{\prod_{i=1}^{l} w_k \prod_{k=1}^{h} p_k} = \frac{2K^{h-l}}{\prod_{k=l+1}^{h} p_k} \quad (13)$$

Expression (14) is a product of $w_{l+1} p_{l+1} = K$ if (13) is an integer. On RLFTs with constant radix for all levels:

$$\frac{2K^{h-l-1}}{\prod_{k=l+1}^{h} p_k} = 2\prod_{k=l+1}^{h} \frac{K}{p_k} = 2\prod_{k=l+1}^{h} m_k \text{ is an integer } \square$$

Lemma 4: For each switch at levels $l = 1..h-1$ the sub-sequence of destinations routed through it, in every stage of a Shift CPS is contiguous or wraps around the last destination and do not exceed K elements.

Proof of lemma 4: A switch at level ***l*** may receive up-going traffic from its descendent leafs. Based on the recursive nature of the tree there are $\prod_{k=1}^{l} m_k$ such descendant leafs for switch at level ***l***. If these descendants end-ports send to a contiguous set of destinations (Shift CPS) the switch will only route the subset which are $\prod_{k=1}^{l} w_k$ apart.

The number of destinations passing through a switch in a single Shift CPS stage is:

$$\frac{\prod_{k=1}^{l} m_k}{\prod_{k=1}^{l} w_k} = \frac{\prod_{k=1}^{l} \frac{K}{p_k}}{\prod_{k=2}^{l} \frac{K}{p_k}} = K \quad \square \tag{14}$$

Proof of theorem 1: Since lemma 4 show that there will be no more than K destinations routed up through a switch in an RLFT Shift permutation sequence stage and lemma 3 shows that routing of these destinations is through different up-going ports the result is that the routing through that switch is non-blocking $\square$

Theorem 2: The routing according to (3) guarantees that no more than one destination is routed through all the network down-going ports of a complete RLFT.

The following lemmas are used to prove theorem 2:

Lemma 5: For routing (1) a single top level switch is passing all the flows to a specific destination

We prove by induction on the tree levels and show that routing towards a destination ***j*** from an arbitrary end-port $(0, s_h..s_1)$ at level ***l*** the traffic will pass trough switch whose first tuple digits are independent of the source end-port. The result is that all the digits of the top level switch that a packet will traverse, are independent of the originating end-port. Which means a single top level switch will pass the traffic to a specific destination.

The base of the induction is for $l = 0$: The up-going port is obtained using the routing (1) formulation $q_0^U(j) = j \bmod (w_1 p_1)$; then using the PGFT connections rule:

$$\left\{ \begin{array}{l} \{(0, a_h, ...a_{l+1}...a_1, q)_U, (1, b_h, ...b_{l+1}...b_1, r)_D\} \mid \\ 0 \le q < w_1 p_1 \wedge 0 \le r < m_1 p_1 \wedge \\ \forall j \in [2..h](a_j = b_j) \wedge b_1 = q \bmod w_1 \wedge \\ a_1 = r \bmod m_1 \end{array} \right\} \tag{15}$$

Since ***q*** is known and constant, b_1 is also known and constant which prove that routing from arbitrary end-port to destination j is going through first level switches sharing the first digit of the tuple.

The induction step assumes at level ***l*** the first ***l*** digits are known and constant and show that the PGFT connection rule adds the digit at place ***l+1*** :

$$\left\{ \begin{array}{l} \{(l, a_h, ...a_{l+1}...a_1, q)_U, (l+1, b_h, ...b_{l+1}...b_1, r)_D\} \mid \\ 0 \le q < w_{l+1} p_{l+1} \wedge 0 \le r < m_{l+1} p_{l+1} \wedge \\ \forall j \in [1..h](j \ne l+1 \rightarrow a_j = b_j) \wedge \\ \lfloor q / p_{l+1} \rfloor = \lfloor r / p_{l+1} \rfloor \wedge b_{l+1} = q \bmod w_{l+1} \\ \wedge a_{l+1} = r \bmod m_{l+1} \end{array} \right\} \tag{16}$$

All digits b_j with $j \ne l+1$ are preserved and $b_{l+1} = q \bmod w_{l+1}$ is constant since ***q*** is independent of the lower level switch $\square$

Lemma 6: The number of destinations passing through an RLFT top level switch is at most the number of its ports

Proof of lemma 6: Using expression (8) for the top level we get the number of destination is:

$$\frac{N}{\prod_{k=1}^{h} w_k} = \frac{2K^h}{\prod_{k=1}^{h} w_k \prod_{k=1}^{h} p_k} = \frac{2K^h}{\prod_{k=1}^{h} p_k w_k} \tag{17}$$

For RLFT the first level is of single connection and the rest of the levels have K connections so:

$$\frac{N}{\prod_{k=1}^{h} w_k} = \frac{2K^h}{p_1 w_1 \prod_{k=2}^{h} p_k w_k} = \frac{2K^h}{K^{h-1}} = 2K \quad \square \tag{18}$$

Proof of theorem 2: Combining lemmas 5 and 6 different sets of exactly 2K destinations are routed through each top level switch which as 2K ports. So each port may be assigned just one destination. Applying these lemmas to sub-trees in a recursive manner show that theorem 2 holds for every switch in the tree.

ACKNOWLEDGMENT

I would like to thank my advisors Avinoam Kolodny and Israel Cidon for their support. Special thanks to Jose Flich for his insights, careful review and suggestions. Many thanks, also, to my colleagues and managers in Mellanox for promoting research and to my dear family supporting my efforts.

REFERENCES

[1] http://www.top500.org

[2] Torsten Hoefler, Timo Schneider, Andrew Lumsdaine "Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks," Cluster Computing, IEEE International Conference, pp.116-125, 2008

[3] K. Asanovic et al. "The landscape of parallel computing research: a view from Berkeley," Technical report, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006

[4] Kerbyson D.J., Barker K.J, "Automatic Identification of Application Communication Patterns via Templates". Int. Conf. on Parallel and Distributed Computing Systems (PDCS), Las Vegas, NV, 2005

[5] Alme H.J., Hoisie A., Petrini F., Wasserman, H.J., Gittings M.L., Kerbyson D.J., "Predictive Performance and Scalability Modeling of a Large-scale Application", SC'01, Denver, CO. 2001

[6] Wolfgang E. Denzel , Jian Li , Peter Walker , Yuho Jin, "A framework for end-to-end simulation of high-performance computing systems," Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, March 03-07, 2008, Marseille, France

[7] Petrini, J. Fernandez, E. Frachtenberg, and S. Coll F., "Scalable collective communication on the asci q machine," Hot Interconnects 12, 08 2003

[8] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective communication operations in MPICH," Int'l Journal of High Performance Computing Applications, 19(1):49–66, Spring 2005

[9] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance Analysis of MPI Collective Operations," Proceedings of the 19th International Parallel and Distributed Processing Symposium, 4th International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS 05), Denver, CO, April 2005

[10] OpenMPI implementation of tuned collectives layer: http://svn.open-MPI.org/svn/oMPI/trunk/oMPI/mca/coll/tuned/

[11] C.E. Leiserson, "Fat-trees: Universal Networks for Hardware-Efficient Supercomputing," IEEE Transactions on Computers, 34(10):892-901, Oct. 1985

[12] Fabrizio Petrini, Marco Vanneschi, "k -ary n -trees: High Performance Networks for Massively Parallel Architectures," 11th International Parallel Processing Symposium (IPPS '97), ipps, pp.87, 1997

[13] Sabine R. Öhring , Maximilian Ibel , Sajal K. Das , Mohan J. Kumar, "On generalized fat trees," Proceedings of the 9th International Symposium on Parallel Processing, p.37, April 25-28, 1995

[14] S, Heller. [ed.] K. Bolding and L. Synder. "Congestion-Free Routing on the CM-5 Data Router," First International Workshop PCRCW, Seattle, Washington, LNCS, Vol. 853, pp. 176-184, May 1994

[15] Sameer Kumar, Laxmikant V. Kale, "Scaling All-to-All Multicast on Fat-tree Networks," Proceedings of the Parallel and Distributed Systems, Tenth International Conference, p205 2004 DOI:10.1109/ICPADS.2004.77

[16] Xuan-Yi Lin, Yeh-Chin Chung, and Tai-Yi Huang. "A Multiple LID Routing Scheme for Fat-Tree-Based InfiniBand Networks," IEEE International Parallel and Distributed Processing Symposium (IPDPS'04). pp. 1-13, 2004

[17] C. Gomez, F. Gilabert, M.E. Gomez, P. Lopez, J. Duato, "Deterministic versus Adaptive Routing in Fat-Trees," IEEE International Parallel and Distributed Processing Symposium, (IPDPS'07), pp.292, 2007

[18] Eitan Zahavi, Gregory Johnson, Darren J. Kerbyson, and Michael Lang. "Optimized InfiniBand Fat-tree Routing for Shift All-To-All Communication Patterns," Concurrency and Computation: Practice and Experience. Volume 22 Issue 2, Pages 217 – 231. 2009. DOI: 10.1002/cpe.1527

[19] P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan, "The Influence of Operating Systems on the Performance of Collective Operations at Extreme Scale," in IEEE International Conference on Cluster Computing, 2006

[20] OMNeT++: an extensible, modular, component-based C++ simulation library and framework: http://www.omnetpp.org/

[21] J. J. Dongarra, S. W. Otto, M. Snir, and D. Walker, A Message Passing Standard for MPP and Workstations," Comm. ACM, Vol. 39, No. 7, July 1996, pp.~84—90

[22] E. Zahavi, D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees, Technical Report CCIT Report #776, Aug. 2010.

[23] Open Fabrics Alliance download packages page: http://www.openfabrics.org/downloads/ibutils/.