

Multicast Performance Analysis for High-Speed Torus Networks

S. Oral and A. George

HCS Research Lab, ECE Dept., University of Florida, 32611, Gainesville, FL
{oral, george}@hcs.ufl.edu

Abstract

Overall efficiency of high-performance computing clusters not only relies on the computing power of the individual nodes, but also on the performance that the underlying network can provide to the computational application. Although modern high-performance networks, especially System Area Networks (SANs), have high unicast performance, they do not support multicast communication in hardware. This research experimentally evaluates the performance of various protocols for unicast-based and path-based multicast communication on high-speed torus networks. Software-based multicast performance results of selected algorithms on a 16-node Scalable Coherent Interface (SCI) torus are given. The strengths and weaknesses of the various protocols are illustrated in terms of startup and completion latency, CPU utilization, and link utilization and concurrency.

Keywords: collective communication, unicast-based multicast, path-based multicast, torus networks, Scalable Coherent Interface.

1. Introduction

Communication primitives for message-passing parallel computing can be classified as *point-to-point (unicast)*, involving a single source and destination node, or *collective*, involving more than two processes. Even if collective communication is not strictly necessary for the development of parallel programs, it usually plays a major role both by simplifying programming tasks and enabling a greater degree of portability among different platforms. Therefore, efficient support of collective communications is a critical issue in the design of networks for high-performance parallel systems.

An important primitive among collective communication operations is *multicast* communication. Multicast communication is concerned with sending a single message from a source node to a set of destination nodes. This primitive can be used as a basis for many collective operations, such as barrier synchronization and

global reduction, as well as cache invalidations in shared-memory multiprocessors [1]. The multicast primitive also functions as a useful tool in parallel numerical procedures such as matrix multiplication and transposition, eigenvalue computation, and Gauss elimination [2]. Moreover, this type of communication is used in parallel search [3] and parallel graph algorithms [4]. Special cases of this primitive include *unicast*, in which the source node must transmit a message to a single destination node, and *broadcast*, in which the destination node set includes all network nodes.

Multicast communication algorithms can be classified as unicast-based, path-based or tree-based [5]. In a *unicast-based* algorithm, the source node sends the message to the destination node set as unicast-routed messages. This type of communication does not require any additional hardware support but incurs some degree of message relaying overhead [1].

Unlike the unicast-based algorithms, *path-based* ones require some degree of hardware support. This type of communication is based on having separate processor and router elements in each node, where the router has the ability to relay the message to multiple output channels at the same time. A worm that contains multiple destination addresses in its header is the basis for path-based communication. When a destination node receives this worm, it simply removes its address from the header, routes the message to the next destination node, copies the message to its local buffer and replicates it on different output channels if necessary. Finally, the last destination node of each path entirely removes the worm from the network [6].

For switched (*indirect*) networks, the message routing is handled by the central switching elements instead of the distributed routing units in each node. Therefore, there is no direct mapping between the switching elements and the processors, so the path-based algorithms are inadequate for these types of networks. To this limitation, *tree-based* algorithms were developed, in which the source node injects a multi-destination worm into the network and, at each central switching element this worm is replicated to appropriate output ports as separate multi-destination worms. Consequently, all these worms follow different paths in the network,

forming a multicast tree to deliver the message to appropriate destination nodes [7].

In the context of this study, we have selected five relevant multicast algorithms from the literature to be evaluated for high-speed torus networks. Among these algorithms, the *separate addressing* (also known as multi-unicast) and the *U-torus* [8] protocols are unicast-based. The other three are path-based multicast communication algorithms, namely *S-torus*, *M_d-torus*, and *M_u-torus* [9].

As a case study, the tradeoffs in the performance of the selected algorithms are evaluated experimentally on a 2D torus network of *Scalable Coherent Interface (SCI)* using the Wulfskit products from Dolphin/Scali [10]. The Dolphin/Scali Wulfskit is a commercial implementation of the SCI standard [11] that addresses networking and high-performance computing domains. SCI is a *System Area Network (SAN)* based on point-to-point links to form unidirectional ringlets. Using these ringlets as a basic building block, a large variety of topologies are possible, such as counter-rotating rings and unidirectional or bi-directional tori. Recent Wulfskit SCI networks have become available that achieve ultra-low latencies (e.g. $< 2 \mu\text{s}$ for one-way messaging) and high link data rates (e.g. 5.3 Gb/s) over point-to-point links with cut-through switching [12].

The next section summarizes related work, followed by Section 3 where a brief overview of the selected multicast algorithms is provided. Section 4 provides a detailed discussion of the setup, results and analysis from the case study experiments. Finally, Section 5 provides conclusions and directions for future work.

2. Related research

Research for multicast communication in the literature can be briefly categorized into two groups, unicast-based and multi-destination-based. Among the unicast-based multicasting methods, separate addressing is the simplest one, in which the source node iteratively sends the message to each destination node one after another as separate unicast transmissions. Another approach for unicast-based multicasting is to use a multi-phase communication configuration for delivering the message to the destination nodes. In this method, the destination nodes are organized in some sort of a binomial tree, and at each communication step the number of nodes covered increases by a factor of n . The *U-torus* multicast algorithm proposed by Robinson *et al.* [8] is a slightly modified version of this binomial-tree approach for direct torus networks that use wormhole routing.

Lin and Ni [13] were the first to introduce and investigate the path-based multicasting approach. Subsequently, path-based multicast communication has received attention and has been studied for direct

networks [8, 9, and 14]. Regarding path-based studies, this research will concentrate on the work of Robinson *et al.* [8, 9] in which they have defined *S-torus*, *M_d-torus*, *M_u-torus* algorithms. These algorithms were proposed as a solution to the multicast communication problem for generic, wormhole-routed, direct unidirectional and bi-directional torus networks. More details about path-based multicast algorithms for wormhole-routed networks can be found in the survey of Li and McKinley [15]. Tree-based multicasting also received attention [16, 17] and these studies focused on solving the deadlock problem for indirect networks.

SCI unicast performance analysis and modeling has been discussed in literature [18-21], while collective communication on SCI has received little attention and its multicast communication characteristics are still unclear. Limited studies on this avenue have used collective communication primitives for assessing the scalability of various SCI topologies from an analytical point of view [22, 23], while no known study has yet investigated the multicast performance of SCI.

This research focuses on evaluating the performance of the selected unicast-based and path-based multicast algorithms over high-speed torus networks. As a case study, the selected algorithms are analyzed for the Dolphin/Scali Wulfskit SCI network using a user-level API. The algorithms are comparatively evaluated using various metrics, including multicast completion latency, start-up latencies, CPU load, link contention, and concurrency.

3. Selected multicast algorithms

The algorithms analyzed in this study were defined in the literature [8, 9]. This section will simply provide an overview of how they work and briefly point out their differences. Bound by the limits of available hardware, we selected two unicast-based and three path-based multicast algorithms for our research, thereby keeping an acceptable degree of variety among different classes of multicast routing algorithms. In this work, the aggregate collection of all destination nodes and the source node is called the *multicast group*. Therefore, for a given group with size d , there are $d-1$ destination nodes.

Separate addressing

Among the selected unicast-based algorithms, separate addressing is the simplest one. In this protocol, the root node iteratively sends the message to all destination nodes one after another as unicast messages. For small group sizes and short messages, separate addressing can be a cost-effective approach. However, for large messages and large group sizes, the iterative unicast transmissions may result in large host-processor overhead. Another drawback of this protocol is the

linearly increasing multicast completion latencies with the increasing group sizes. In a separate addressing algorithm, for a given group of size d , there will be $d-1$ total communication steps in order to deliver to all of the destination nodes. Figure 1(a) shows a typical scenario for a group size of 10. The root node and the destination nodes are clearly marked and the message transfers are indicated. Alphabetic labels next to each arrow indicate the individual paths, and the numerical labels represent the logical communication steps (i.e. not the physical number of hops involved).

U-torus

U-torus [8] is another unicast-based multicast algorithm and it uses a binomial-tree approach to reduce the required communication steps. For example, if we assume a group with the size d , the lower bound on the number of steps required to complete the multicast by U-torus will be $\lceil \log_2 d \rceil$. This reduction is achieved by increasing the number of covered destination nodes by a factor of 2 in each communication step.

Figure 1(b) illustrates a typical U-torus multicast scenario for a group size of 10. Applying U-torus to this group starts with dimension ordering of all the nodes, including the root, and then rotating around to place the root node at the beginning of the ordered list as given below:

$$\begin{aligned}\Phi &= \{(1,1), (1,2), (1,3), (2,2), (2,4), (3,1), (3,3), (4,1), \\ &\quad (4,2), (4,4)\} \\ \Phi' &= \{(2,2), (2,4), (3,1), (3,3), (4,1), (4,2), (4,4), (1,1), \\ &\quad (1,2), (1,3)\}\end{aligned}$$

where Φ denotes the dimension-ordered group and Φ' denotes the rotated version of Φ . The order in Φ' also defines the final ranking of the nodes, as they are sequentially ranked starting from the leftmost node. As an example for the Φ' given above, node (2,2) has a ranking of 0, node (2,4) has a ranking of 1, and the node (1,3) has a ranking of 9.

After obtaining the Φ' , the root node sends the message to the *center* node of the Φ' to partition the multicast problem of size d into two subsets of size $\lceil d/2 \rceil$ and $\lfloor d/2 \rfloor$. The center node is calculated by Eq. 1 as in [8], where *left* denotes the ranking of the leftmost node, and *right* denotes the ranking of the rightmost node.

$$center = left + \left\lceil \frac{right - left + 1}{2} \right\rceil \quad (1)$$

For the group given above, the left is rank 0 and the right is 9, therefore the center is 5, which implies the node (4,2). The root node not only transmits the multicast message, but also the new partition's subset information,

D_{subset} , to the center node. Using the same example, at the end of the first step the root node will have the subset

$$D_{subset_root} = \{(2,2), (2,4), (3,1), (3,3), (4,1)\}$$

with the values of left and right being rank 0 and 4, respectively. The node (4,2) will have the subset

$$D_{subset_(4,2)} = \{(4,2), (4,4), (1,1), (1,2), (1,3)\}$$

with the values of left and right being again 0 and 4.

In the second step, the original root and the (4,2) node both act as root nodes, partitioning their respective subsets into two and sending the multicast message to their subset's center node, along with the new partition's D_{subset} information. This process continues recursively until all destination nodes have received the message.

S-torus

S-torus, a path-based multicast routing algorithm, was defined by Robinson *et al.* [9] for wormhole-routed torus networks. It is a single-phase communication algorithm and the destination nodes are ranked and ordered to form a Hamiltonian cycle. A Hamiltonian cycle is a closed circuit that starts and ends at the source node, where every other node is listed only once. For any given network, more than one Hamiltonian cycle may exist. The Hamiltonian cycle that S-torus uses is based on a ranking order of nodes, which is calculated with the formula given in Eq. 2 for a k -ary 2D torus.

$$l(u) = \left[(\sigma_0(u) + \sigma_1(u)) \bmod k \right] + k \left[\sigma_1(u) \right] \quad (2)$$

Here, $l(u)$ represents the Hamiltonian ranking of a node u , with the coordinates given as $(\sigma_0(u), \sigma_1(u))$. More detailed information about Hamiltonian node rankings can be found in [9]. Following this step, the ordered Hamiltonian cycle Θ , is rotated around to place the root at the beginning. This new set is named as Θ' .

The root node then issues a multi-destination worm which visits each destination node one after another following the Θ' ordered set. At each destination node, the header is truncated to remove the visited destination address and the worm is re-routed to the next destination. The algorithm continues until the last destination node receives the message. Robinson *et al.* also proved that S-torus routing is deadlock-free [9].

Figure 1(c) illustrates the S-torus algorithm for the same example presented previously, where we assume a network without wormhole routing. The Hamiltonian rankings are noted in the superscript labels of each node, where Θ and Θ' are obtained as given below:

$$\begin{aligned}\Theta &= \{^4(1,3), ^6(1,1), ^7(1,2), ^8(2,2), ^{10}(2,4), ^{12}(3,1), \\ &\quad ^{14}(3,3), ^{16}(4,4), ^{17}(4,1), ^{18}(4,2)\} \\ \Theta' &= \{^8(2,2), ^{10}(2,4), ^{12}(3,1), ^{14}(3,3), ^{16}(4,4), ^{17}(4,1), \\ &\quad ^{18}(4,2), ^4(1,3), ^6(1,1), ^7(1,2)\}\end{aligned}$$

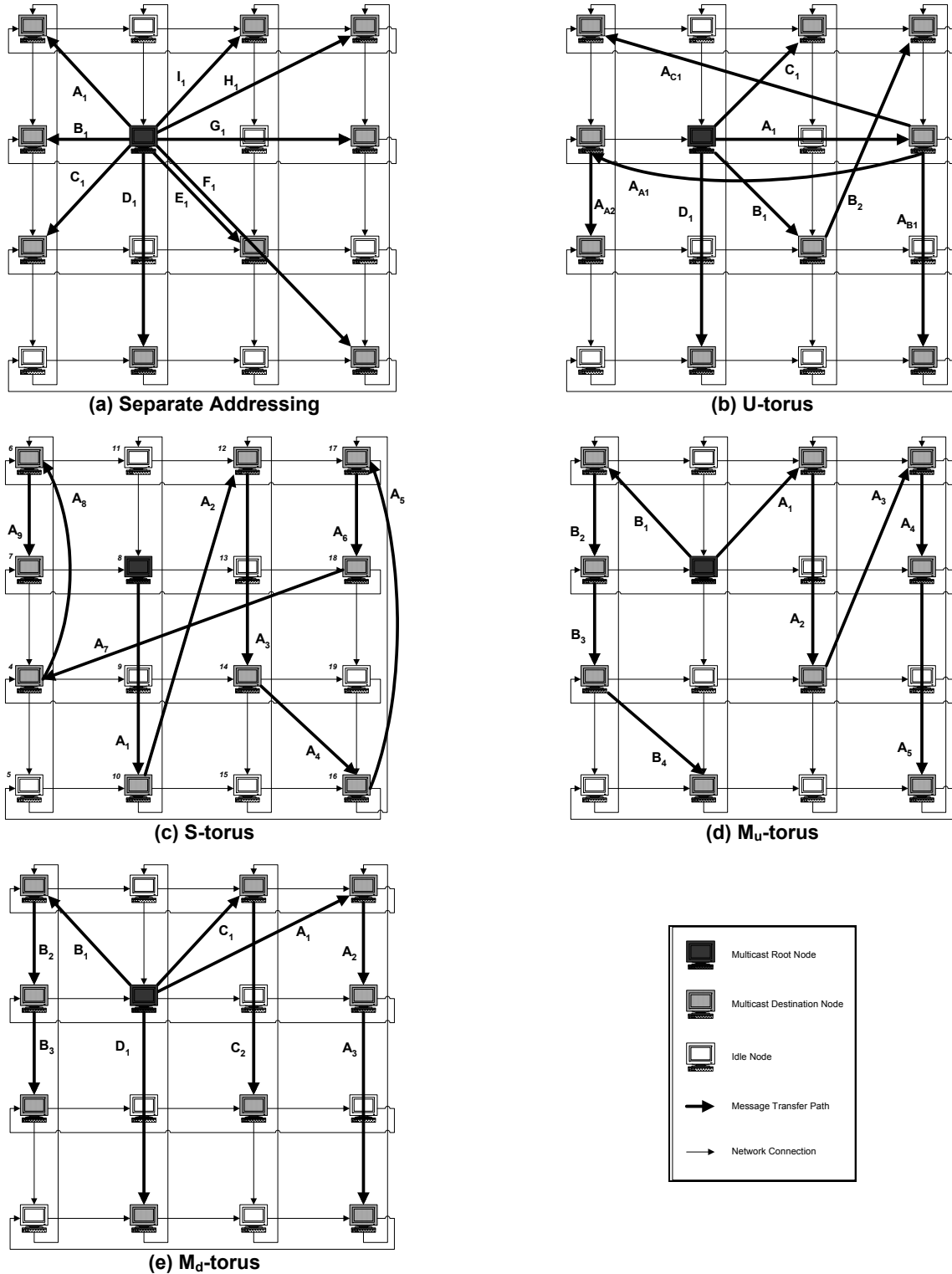


Figure 1: Separate addressing (a), U-torus (b), S-torus (c), M_u -torus (d), and M_d -torus (e) multicast algorithms for a typical multicast scenario with a group size of 10. Individual message paths are marked alphabetically, and the numerical labels represent the logical communication steps for each message path.

M-torus

Belying its simplicity, single-phase communication is known for large latency variations for a large set of destination nodes [24]. Therefore, to further improve the S-torus algorithm, Robinson *et al.* proposed the multi-phase multicast routing algorithm: M-torus [9]. The idea was to shorten the path lengths of the multi-destination worms to stabilize the latency variations and to achieve better performance by partitioning the multicast group. Furthermore, they introduced two variations of the M-torus algorithm, M_d -torus and M_u -torus. M_d -torus uses a dimensional partitioning method, whereas M_u -torus uses a uniform partitioning mechanism. In both of these algorithms, the root node separately transmits the message to each partition and the message is then further relayed inside the subsets using multi-destination worms. The M_d -torus algorithm partitions the nodes based on their respective sub-torus dimensions, therefore eliminating costly dimension-switching overhead. For example, in a 3D torus, the algorithm will first partition the group into subsets of 2D planes of the network, and then into ringlets for each plane.

For a k -ary, N -dim torus network, where k^N is the total number of nodes, the M_d -torus algorithm needs N steps to complete the multicast operation. On the other hand, the M_u -torus algorithm tries to minimize and equalize the path length of each worm by applying a uniform partitioning. M_u -torus is parameterized by the partitioning size, denoted by r . For a group size of d , the M_u -torus algorithm with a partitioning size of r requires $\lceil \log_r(d) \rceil$ steps to complete the multicast operation. For the same example presented previously, Figures 1(d) and (e) illustrate M_u -torus and M_d -torus, again assuming a network without wormhole routing.

4. Case study

To comparatively evaluate the performance of the selected algorithms, an experimental case study is conducted. The following subsections explain the details of the experiments as well as the results obtained.

4.1. Description

The case study is performed on a 16-node system, each with dual 1GHz Intel Pentium-III processors, 256MB of PC133 SDRAM, ServerSet III LE (rev 6) chipset, and a 133MHz system bus. An SCI network is used as the high-speed interconnect, where each node has PCI64/66 SCI NICs with 5.3 Gb/s link speed using Redhat Linux 7.2 with kernel version 2.4.7-10smp, mtrr patched, and write-combining enabled. The nodes are interconnected to form a 4×4 unidirectional SCI torus.

For all of the selected algorithms, the polling notification method is used. Polling notification is an approach to lower the latencies by simply putting the host CPU into a constant check-state for completion flags. Although it is known to be effective for achieving low latencies, oftentimes it results in higher CPU loads, especially if the polling process runs for extended periods. Moreover, to have a fair comparison among the algorithms and to decrease the completion latencies even further, the multicast-tree creation process is removed from the critical path, and the trees are generated at the beginning of each algorithm in every node that participates in the group. Additionally, a messaging mechanism that guarantees the delivery is used to decrease the number of required retransmissions and link contentions. The routing among the destination nodes is performed by Scali's built-in `SCA_ROUTE_MAXCY` function. `SCA_ROUTE_MAXCY` is a fault-tolerant routing mechanism that, when all the nodes are fault-free, behaves as a dimension-order routing algorithm [10].

Throughout the case study, modified versions of the three path-based algorithms, S-torus, M_d -torus, and M_u -torus are used. These algorithms were originally designed to multicast using multi-destination worms. However, as with most high-speed interconnects for cluster computing, our testbed does not support multi-destination worms. Therefore, these three algorithms are modified as follows. As each destination on a given path is visited by a multi-destination message, the message is saved by that destination node and a new message is generated that proceeds to the next destination on the same path.

As mentioned previously, the M_d -torus algorithm uses a dimensional partitioning method which partitions the destination nodes up to their lowest dimension possible. Consequently, on our 4-ary 2-D torus testbed, M_d -torus applies a 1st-order dimensional partitioning where the maximum partition lengths are naturally set as 4. To have a fair comparison between the M_d -torus and the M_u -torus algorithms, the partition length r of 4 is chosen for M_u -torus algorithm.

The U-torus algorithm transfers not only the multicast message from source to destination nodes, but also the D_{subset} information at each communication step. Throughout the case study, the D_{subset} information is embedded in the relayed multicast message at each step.

Separate addressing is also a unicast-based algorithm like U-torus, but it exhibits no algorithmic concurrency. However, with a simple modification to provide network pipelining, which enables multiple message transfers to occur in an overlapping parallel fashion, it is possible to provide concurrency with separate addressing. Consequently, for our case study we use a network-pipelining version of the separate addressing algorithm. Overall, network pipelining combined with the

guaranteed message transfer mechanism mentioned previously results in a non-blocking protocol for separate addressing with high achievable concurrency.

The case study experiments with the algorithms are performed for group sizes of 4, 6, 8, 10, 12, 14, and 16 and for message sizes of 2B and 512KB. Each algorithm is evaluated for each message size and group size 100 times, where each execution had 50 repetitions. Four different sets of experiments are performed to analyze the various aspects of each algorithm in depth, and they are: multicast completion latency, user-level CPU utilization, multicast startup and tree-creation latency, and link utilization and concurrency. For each algorithm, the various latencies are probed and measured separately. The maximum user-level CPU utilization of the root node is measured using Linux’s built-in `sar` utility. Finally, the link utilization and concurrency of each algorithm are calculated for each group size based on the communication pattern observed throughout the experiments.

4.2. Multicast completion latency

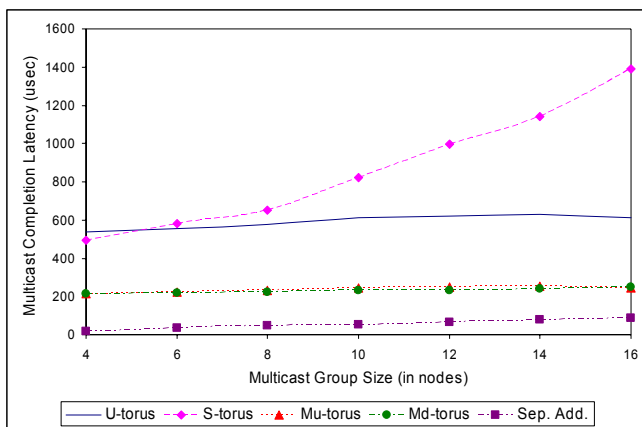
Completion latency is an important metric for evaluating and comparing different multicast algorithms, as it reveals how suitable an algorithm is for a given network. Two different sets of experiments for multicast completion latency are performed in this case study, one for a small message size of 2B, and the other for a large message size of 512KB. Figure 2(a) illustrates the multicast completion latency versus group size for small messages, while Figure 2(b) presents the same for large

messages. It is evident that, for both small and large messages, S-torus has the worst performance. Moreover, S-torus shows a linear increase in multicast completion latency with respect to the increasing group size, as it exhibits no parallelism in message transfers.

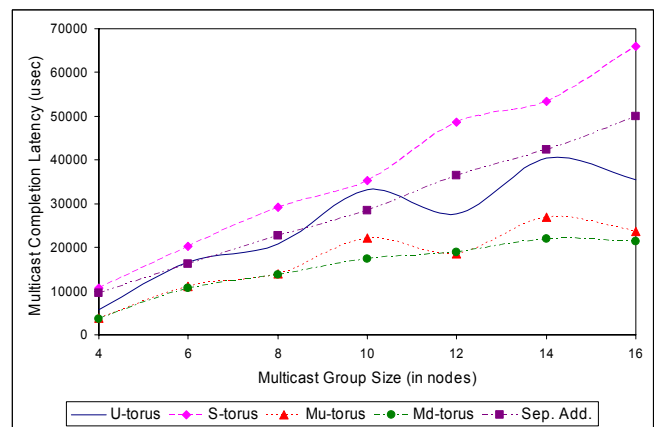
By contrast, the separate addressing algorithm has a higher level of parallelism (investigated further in Section 4.5) and, as a result, performs best for small messages. However, separate addressing is also known for linearly increasing completion latencies with increasing message or group size, and this phenomenon can also be easily seen from Figure 2(b).

The M_d -torus and M_u -torus algorithms exhibit nearly identical performance for both small and large messages, as they are basically two versions of the same multi-phase communication algorithm. The difference between these two becomes more distinctive at certain data points, such as 10- and 14- nodes for large messages. The M_u -torus algorithm is evaluated using a partition length of 4. For group sizes of 10 and 14 this parameter does not provide perfectly balanced partitions. This imbalance results in higher multicast completion latencies at these points, as can be seen from Figure 2(b). For large message sizes, the M_d -torus algorithm outperforms the rest due to its balanced partitioning.

Finally, U-torus has nearly flat latency values for small messages. For large messages, it exhibits similar behavior to M_u -torus, as its low link utilization becomes a bottleneck for some group sizes, such as 10 and 14 (investigated further in Section 4.5).

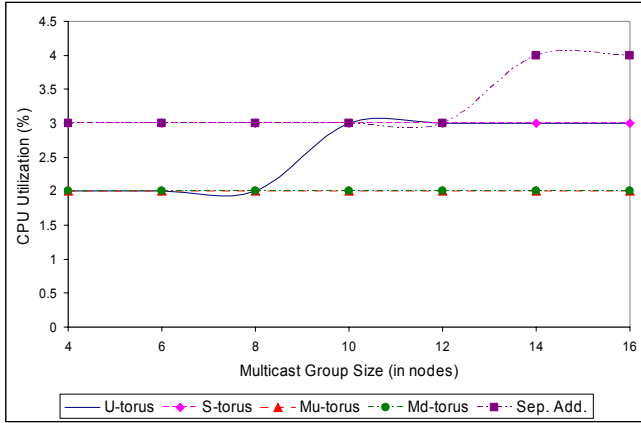


(a) 2B messages

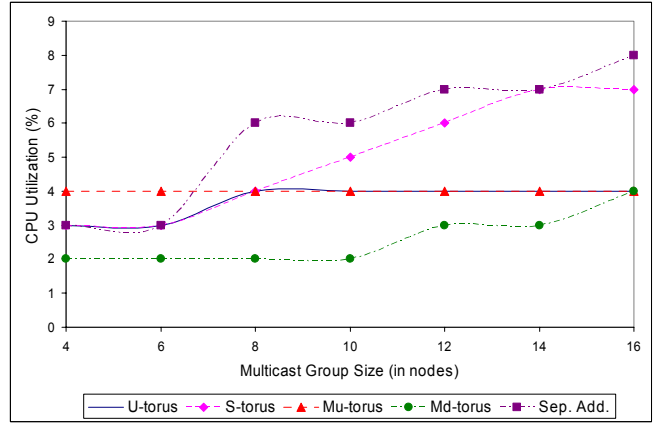


(b) 512KB messages

Figure 2: Completion latency vs. group size for small (a) and large (b) messages.



(a) 2B messages



(b) 512KB messages

Figure 3: User-level CPU utilization vs. group size for small (a) and large (b) messages.

4.3. User-level CPU utilization

Host processor load is another useful metric to assess the quality of a multicast protocol. Figures 3(a) and (b) present the maximum CPU utilization for the root node of each algorithm. The results are obtained for various group sizes and for both small and large message sizes.

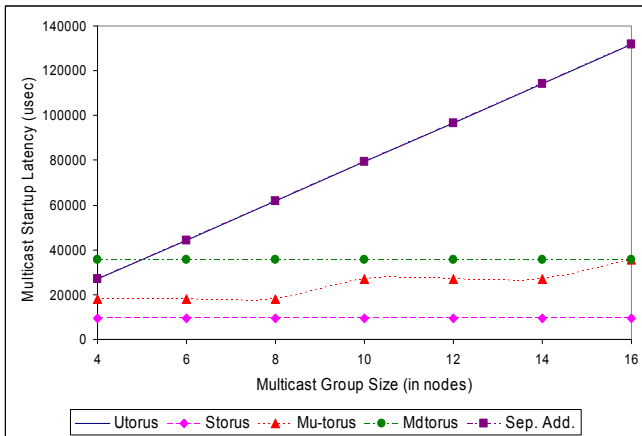
It is observed that S-torus exhibits constant CPU load for the small message size independent of the group size. However, for large messages, as the group size increases the completion latency also linearly increases as illustrated in Figure 2(b), and the extra polling involved results in higher CPU utilization for the root node. This effect is clearly seen in Figure 3(b).

In the separate addressing algorithm, the root node iteratively performs all message transfers to the destination nodes. As expected, this behavior causes a nearly linear increase in CPU load with increasing group size, which can be observed in Figure 3(b).

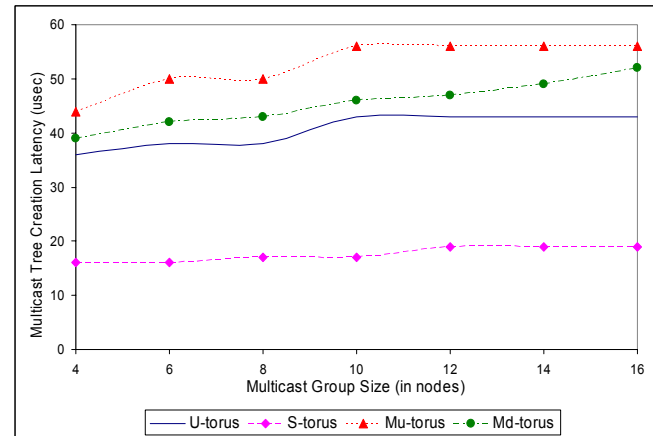
By contrast, since the number of message transmissions for the root node stays constant, M_d -torus provides a nearly constant CPU overhead for small messages for every group size. For large messages and small group sizes, M_d -torus performs similarly but for group sizes greater than 10, the CPU utilization tends to increase due to variations in the path lengths causing extended polling durations. Although these variations are the same for both small and the large messages, the effect is more visible for the large message size.

M_u -torus exhibits behavior identical to M_d -torus for small messages. Moreover, for large messages, M_u -torus also provides higher but constant CPU utilization.

For U-torus, the number of communication steps required to cover all destination nodes is given in Section 3. It is observed that at certain group sizes, such as 4, 8, and 16, the number of these steps increases, therefore the CPU load also increases. This behavior of U-torus can be clearly seen at Figures 3(a) and (b).



(a)



(b)

Figure 4: Startup latency (a) and tree-creation latency (b) vs. group size.

4.4. Startup latency

The multicast startup latency of the SCI API may also be an important metric since, for small message sizes, this factor might impede the overall communication performance. In addition, multicast tree creation latencies exhibit a similar effect. Figures 4(a) and (b) present these two variables versus group size.

The U-torus and separate addressing algorithms have unbounded fan-out numbers and, as clearly illustrated in Figure 4(a), the startup latencies for these two algorithms are identical and linearly increasing with group size. By contrast, the S-torus and M_d -torus algorithms have constant startup latencies because of their fixed fan-out numbers. Figure 4(b) presents the multicast tree creation latencies for the four algorithms that use a tree-like group formation for message delivery. The M_u -torus and M_d -torus algorithms only differ in their partitioning methods as described before and both methods are quite complex compared to the other algorithms. This complexity is seen in Figure 4(b) as they exhibit the highest multicast tree-creation latencies.

U-torus has a simple and distributed partitioning process and, compared to the two M-torus algorithms, it has lower tree-creation latency. However, unlike the other tree-based algorithms, S-torus does not perform any partitioning and it only orders the destination nodes as described previously. Therefore, S-torus exhibits the lowest and a very-slowly and linearly increasing latency, due to the simplicity of its tree formation.

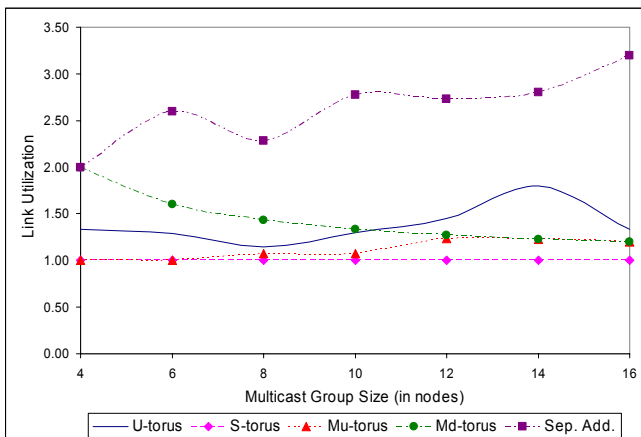
4.5. Link utilization and concurrency

Link utilization and concurrency are also metrics that are used to evaluate the selected routing algorithms. Link

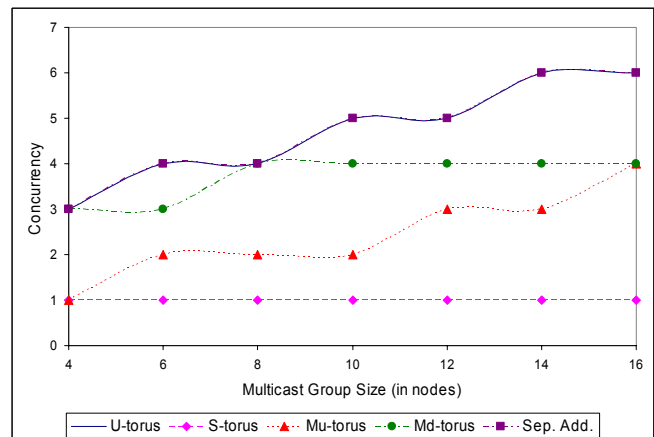
utilization can be divided into two components as number of *link visits* and number of *used links*. *Link visits* is defined as the cumulative number of links used during the entire communication process, while *used links* is defined as the number of individual links used. Link utilization, which is the ratio of the number of link visits to the number of used links, given in Figure 5(a) may not alone reveal useful insight. However, when combined with the concurrency presented in Figure 5(b), it illustrates the degree of communication balance for each algorithm. When analyzing the results in Figure 5, it is important to note the relationship between them. For example, a case where an algorithm has high link utilization and low concurrency reveals a possible contention problem. On the other hand, a high link-utilization with a high concurrency indicates that the algorithm is using the network effectively.

S-torus is a simple chained communication and there is only one active message transfer in the network at any given time. Therefore, S-torus has the lowest and a constant link utilization and concurrency compared to other algorithms, as can be seen in Figure 5. By contrast, due to the high parallelism provided by the recursive doubling approach, the U-torus algorithm has the highest concurrency. Separate addressing exhibits an identical degree of concurrency to the U-torus, because of the multiple message transfers overlapping at the same time due to the network pipelining. Also notable is that the high link utilization of separate addressing is the result of its link visits increasing more rapidly compared to the number of used links.

As can be seen from Figure 5(a), the M_d -torus has inversely proportional link utilization versus increasing group size. The explanation for this behavior can be



(a)



(b)

Figure 5: Link utilization (a) and concurrency (b) vs. group size.

obtained from an analysis of the M_d -torus algorithm. In M_d -torus, the root node first sends the message to the destination header nodes, and they relay it to their child nodes. As the number of dimensional header nodes is constant (k in a k -ary torus), with the increasing group size each new child node added to the group will increase the number of available links. Moreover, due to the communication structure of the M_d -torus, the number of links used increases much more rapidly compared to the number of link visits with the increasing group size, which overall will asymptotically limit the decreasing link utilization at 1. Additionally, as each dimensional header relays the message using separate ringlets, the concurrency of M_d -torus is upper bounded by the value of the k , which in our experiments is 4. This behavior can be observed from Figure 5(b) for data points greater than 8 nodes.

The M_u -torus algorithm has low link utilization as can be seen in Figure 5(a) for all group sizes. The reason is that M_u -torus multicasts the message to the partitioned destination nodes over a limited number of individual paths as shown in Figure 1(d), where in each path there is only a single link used at a time. By contrast, for a given partition length of constant size, an increase in the group size results not only in an increase in the number of partitions but also in the number of individual paths. Overall, this trait results in more messages being concurrently transferred at any given time over the entire network, as seen in Figure 5(b).

5. Conclusions

In this study, five different multicast algorithms for high-performance torus networks are evaluated. These algorithms are analyzed on direct SCI networks and their performances are examined under different configurations using various metrics for the case study.

As the separate addressing protocol uses network pipelining to hide the sender overhead, it appears to be the best choice for small messages and group sizes from the perspective of multicast completion latency. On the other hand, for large messages, the separate addressing algorithm has a nearly linear increase in completion latency with increasing group size. Also, as expected, the separate addressing protocol has a nearly linear increase in CPU utilization with increasing group size.

From the perspective of completion latency, for large messages and group sizes, the M_d -torus algorithm performs best because of the balance provided by its use of dimensional partitioning. In addition, the M_d -torus algorithm incurs a very low CPU overhead and achieves high concurrency for all the message and group sizes considered. As group size increases, the number of used

links increases more rapidly, and thus the M_d -torus algorithm achieves a more balanced communication.

It is also observed that the U-torus and M_u -torus algorithms perform better when the individual multicast path depths are approximately equal. Furthermore, the M_u -torus algorithm exhibits best performance when group size is an exact multiple of the partition length r . The U-torus and M_u -torus algorithms have nearly constant CPU utilizations for both small and large message sizes, whereas the U-torus algorithm has the highest concurrency among all evaluated algorithms due to the high parallelism provided by the recursive-doubling method.

From the perspective of completion latency and CPU utilization, S-torus is always the worst performer because of its zero concurrency and extensive communication overhead. As expected, due to the polling notification, with increasing group size the S-torus protocol has a nearly linear increase in completion latency and CPU utilization for large messages.

The results of this research make it clear that no single multicast algorithm is best in all cases for all metrics. For example, as the number of dimensions in the network increases, the M_d -torus algorithm becomes dominant. By contrast, for networks with fewer dimensions supporting a large number of nodes, the M_u -torus and the U-torus algorithms are expected to be the most effective. For small-scale systems, separate addressing appears to be an efficient and cost-effective choice. Finally, the S-torus algorithm is determined to be inefficient as compared to the alternatives in all the cases considered. This inefficiency is due to the extensive length of the paths used to multicast, which in turn leads to long and widely varying completion latencies, as well as a high degree of CPU utilization at the root node.

There are several possibilities for future research, one of which is analytical modeling of the selected algorithms to analyze system sizes beyond our existing testbed. Another possible direction is evaluating the performance of the selected algorithms for higher communication layers, such as MPI. This approach will allow us to obtain additional performance characteristics for high-speed torus networks, as the higher communication layers compared to the user-level API are more widely used. Yet another possible direction is to extend and integrate our SAN-based research with a MAN network, such as Gigabit Ethernet. This integration is expected to accurately mimic the real-world infrastructures of grids, in which the communication is structured in hierarchical network layers, from SANs to LANs, MANs, and WANs.

6. Acknowledgments

This research was supported in part by the U.S. Department of Defense, by matching funds from the University of Florida for the iVDGL project supported by the National Science Foundation, and by equipment support of Dolphin Interconnect Solutions Inc. and Scali Computer AS.

7. References

- [1] P.K. McKinley, H.Xu, A.H. Esfahanian, and L.M. Ni, "Unicast-Based Multicast Communication in Wormhole-Routed Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 12, pp. 1252-1265, Dec. 1994.
- [2] P.K. McKinley, Y.Tsai, and D.F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers," *IEEE Computer*, Vol. 28, No. 2, pp. 39-50, Dec. 1995.
- [3] R.F. DeMara and D.I. Moldovan, "Performance Indices for Parallel Marker-Propagation," *Proc. of 1991 Intl. Conference on Parallel Processing*, pp. 658-659, Aug. 1991.
- [4] V. Kumar and V. Singh, "Scalability of Parallel Algorithms for the All-Pairs Shortest Path Problem," *Technical Report, ACT-OODS-058-90, MCC*, Jan. 1991.
- [5] Y. Tseng, D.K. Panda, and T Lai, "A Trip-Based Multicasting Model in Wormhole-Routed Networks with Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 2, pp. 138-150, Feb. 1996.
- [6] R. Kesavan and D.K. Panda, "Multicasting on Switch-Based Irregular Networks using Multi-Drop Path-Based Multi-Destination Worms," *Proc. of Parallel Computing, Routing, and Communication Workshop (PCRCW'97)*, pp. 179-192, 1997.
- [7] R. Sivaram, D.K. Panda, and C.B. Stunkel, "Multicasting in Irregular Networks with Cut-Through Switches using Tree-Based Multi-Destination Worms," *Proc. of Parallel Computing, Routing, and Communication Workshop (PCRCW'97)*, pp. 35-48, 1997.
- [8] D.F. Robinson, P.K. McKinley, and B.H.C. Cheng, "Optimal Multicast Communication in Wormhole-Routed Torus Networks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 10, pp. 1029-1042, Oct. 1995.
- [9] D.F. Robinson, P.K. McKinley, and B.H.C. Cheng, "Path-Based Multicast Communication in Wormhole-Routed Unidirectional Torus Networks," *Journal of Parallel and Distributed Computing*, Vol. 45, No. 2, pp. 104-121, Sep. 1997.
- [10] Scali Comp. AS, *Scali System Guide Version 2.0, White Paper*, Scali Computer AS, Oslo, Norway, 2000.
- [11] "SCI: Scalable Coherent Interface," *IEEE Approved Standard 1596-1992*, 1992.
- [12] Scali Comp. AS, Web Site, <http://www.scali.com>.
- [13] X. Lin and L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks," *Proc. of International Symposium on Computer Architecture*, pp. 116-124, 1991.
- [14] P.K. McKinley and D.F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers," *IEEE Computer*, pp. 39-50, Vol. 28, No. 12, Dec. 1995.
- [15] W.J. Dally, "Virtual Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 2, pp. 194-205, Mar. 1992.
- [16] L.M. Ni, Y. Gui, and S. Moore, "Performance Evaluation of Switch-Based Wormhole Networks," *Proc. of International Conference On Parallel Processing*, pp. 32-40, 1995.
- [17] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques In Direct Networks," *IEEE Computer*, Vol. 26, No. 2, pp. 62-76, Feb. 1993.
- [18] K. Omang and B. Parady, "Performance Of Low-Cost Ultraspare Multiprocessors Connected By SCI," *Technical Report*, Department of Informatics, University of Oslo, Norway, 1996.
- [19] M. Ibel, K.E. Schauer, C.J. Scheiman and M. Weis, "High-Performance Cluster Computing using SCI," *Proc. of Hot Interconnects Symposium V*, 1997.
- [20] M. Sarwar and A. George, "Simulative Performance Analysis of Distributed Switching Fabrics for SCI-Based Systems," *Microprocessors and Microsystems*, Vol. 24, No. 1, pp. 1-11, Mar. 2000.
- [21] D. Gonzalez, A. George, and M. Chidester, "Performance Modeling and Evaluation of Topologies for Low-Latency SCI Systems," *Microprocessor and Microsystems*, Vol. 25, No. 7, pp. 343-356, Oct. 2001.
- [22] H. Bugge, "Affordable Scalability using Multicubes," in: H. Hellwagner, A. Reinfeld (Eds.), *SCI: Scalable Coherent Interface*, LNCS State-of-the-Art Survey, Springer, pp. 167-174, Berlin, 1999.
- [23] L.P. Huse, "Collective Communication on Dedicated Clusters Of Workstations," *Proc. of EuroPVM/MPI '99*, pp. 469-476, 1999.
- [24] H. Wang, and D.M. Blough, "Tree-Based Multicast in Wormhole-Routed Torus Networks," *Proc. PDPTA'98*, 1998.