Virtual Topologies for Scalable Resource Management and Contention Attenuation in a Global Address Space Model on the Cray XT5

Weikuan Yu[†‡] Vinod Tipparaju[‡] Xinyu Que[†] Jeffrey S. Vetter[‡]
Department of Computer Science [†] Computer Science & Mathematics[‡]
Auburn University Oak Ridge National Laboratory
{*wkyu,xque*}*@auburn.edu* {*tipparajuv,vetter*}*@ornl.gov*

***Abstract*—Global Address Space (GAS) programming models enable a convenient, shared-memory style addressing model, and support completely asynchronous data movement. Their underlying runtime systems face critical challenges in (1) scalably managing resources (such as memory for communication buffers), and (2) gracefully handling unpredictable communication patterns and any associated contention. In this research, we investigate these challenges for a popular GAS runtime library, Aggregate Remote Memory Copy Interface (ARMCI) on, large-scale Cray XT5 systems. We represent the management of communication resources as directed graphs, and propose two new scalable virtual topologies, Meshed Fully Connected Graphs (MFCG) and Cubic Fully Connected Graphs (CFCG), for scalable resource management and contention attenuation. To ensure deadlock-free communication in these multi-dimensional topologies, we design and develop *lowest dimension first* forwarding to support fully- or partially-populated MFCG and CFCG on any number of nodes. We have extensively evaluated the benefits of these virtual topologies on the petascale Jaguar Cray XT5 system at Oak Ridge National Laboratory. Our experimental results demonstrate MFCG as the most suitable virtual topology because of its benefits in resource management, contention mitigation, and the resulting benefit to scientific applications.**

***Keywords*- GAS; ARMCI; Virtual Topology; Contention.**

I. Introduction

Several supercomputing sites have deployed systems with extreme amounts of computational power [1]. The Jaguar Cray XT5 system at the Oak Ridge National Laboratory (ORNL), the IBM Roadrunner system at Los Alamos National Laboratory (LANL), and China's GPGPU-based Tianhe-1A system can perform to the order of 10^{15} floating point operations per second (petaflop). While supercomputing systems grow to unprecedented number of processors (with LLNL Sequoia [2] system and NCSA Blue Waters [3] system in the near future), scientific applications continue to face many challenges such as programming productivity, application scalability, and efficiency. Message passing has been the *de facto* model to achieve scalability and efficiency, however, Global Address Space (GAS) or Partitioned Global Address Space (PGAS) models are emerging as scalable alternatives because they have the ability to alleviate programming burden by supporting data access to both local and remote memory through a simple shared-memory addressing model.

PGAS languages such as Unified Parallel C (UPC) [4], Co-Array Fortran (CAF) [5], and X10 [6], and GAS libraries such as Global Arrays (GA) Toolkit [7] are becoming increasingly popular. These GAS languages and libraries use the services of an underlying communication library (which we refer to as the GAS runtime) for serving their communication needs. GAS languages normally use this runtime as a compilation target to do the data transfers on distributed memory architectures. They have a translation layer that translates a memory access to a corresponding data transfer operation on the underlying system. ARMCI (Aggregated Remote Memory Copy Interface) [8] is a popular GAS runtime that has been used to implement both PGAS languages (such as Co-Array Fortran) and GAS libraries (such as GA). While some MPI applications have reached a sustained petaflop performance and beyond, NWChem [9] computation chemistry code is a GAS model based application and is one of the three applications to have crossed the petaflop barrier in terms of sustained performance [10] on Jaguar. This was made possible by the porting of Global Arrays toolkit, and more specifically, its GAS runtime, ARMCI [11].

Unfortunately, running a GAS model and its underlying GAS runtime in the context of a real scientific application at a scale similar to Jaguar (200,000+ cores) has brought forth a few staggering challenges. These challenges are a result of the characteristics and asyn-

chronous one-sided features of the GAS runtime. The first is that of resource management, incurred by unpredictable communication patterns and communication resources (such as buffers) that need to be allocated to support it. The second challenge is that of network contention – allowing any process to access the address space of any other process and supporting load balancing at the same time create an environment that is prone to contention.

In this paper we describe a powerful *virtual topology* approach to attenuating contention and efficiently managing communication resources in ARMCI (and any GAS/PGAS runtime in general) at petascale and beyond. We represent the allocation of communication resources as directed graphs. While the original model can be depicted as a fully connected graph (FCG), we introduce two new scalable virtual topologies, Meshed FCGs (MFCG) and Cubic FCGs (CFCG), for scalable resource management and contention attenuation. We systematically examine communication characteristics of MFCG, CFCG, and Hypercube. To ensure deadlock-free communication in these multi-dimensional topologies, we design and develop *lowest dimension first* (LDF) forwarding to support fully- and partially-populated MFCG and CFCG on any number of nodes.

We have successfully implemented these virtual topologies and LDF in ARMCI on Jaguar, and conducted experiments to evaluate these topologies using microbenchmarks and real large-scale applications. Our results demonstrate MFCG as the most suitable virtual topology because of its benefits in resource management, contention mitigation, and the resulting benefit for scientific applications.

The rest of the paper is organized as follows. Section II discusses background and motivation. Section III defines several virtual topologies. Section IV describes the implementation of LDF. Experimental results are provided in Sections V and VI, followed by related work in Section VII. We conclude the paper in Section VIII.

II. Background and Motivation

ARMCI guarantees that its one-sided operations are fully unilateral (i.e., may complete regardless of the actions taken by the remote process). In particular, polling the application by remote process (implicitly when making a library call, or explicitly by calling provided polling interface) is not required for communication progress. This is realized by introducing a communication helper thread at each compute node. This communication helper thread (CHT), also called communication server, is created by the lowest ranked process (*master*) on a node. An area of shared memory is allocated for these processes. The CHT handles remote one-sided requests on behalf of all local processes, and exchanges data with them through the shared memory.

Typically, communication resources are allocated in ARMCI (and similarly in other GAS/PGAS runtime systems that we are aware of) to support asynchronous one-sided communication primitives, despite any lack of corresponding communication mechanisms in the underlying network. Hence, in order to realize one-sided implementation for all operations (particularly for lock, unlock, accumulate, and noncontiguous data transfer operations), a process needs to have a way to initiate an operation without the involvement of the targeted process. To this aim, a CHT pre-allocates buffers and other communication structures for requests from remote processes.

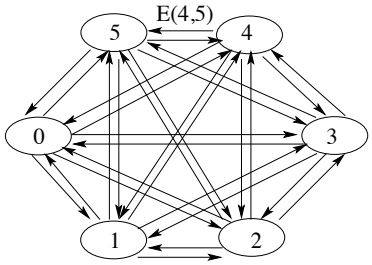


Figure 1: A Directed Graph Representing Resource Allocation for One-Sided Requests

We define *virtual topology* as a means to represent the graph of resource allocation. In the case of memory buffers for communication, a directed graph can represent the resource allocation of buffers amongst all nodes. A graph G: (V, E) consists of a set of vertices V and a set of edges E. A vertex i represents all processes and the CHT on a single node i. A directed edge E(i, j) from i to j denotes the fact that there is a set of request buffers allocated on node i for processes on node j. For an ARMCI application running on N nodes, this representation of buffer allocation forms a FCG with $N*(N-1)$ directed edges. There are $(N-1)$ outgoing edges at each vertex (node), representing $N-1$ sets of buffers from $N-1$ remote nodes. Figure 1 shows the resource allocation graph for a 6-node case.

A. Critical Challenges for GAS Runtime

The directed graph representation of resource allocation in Figure 1 reveals two critical challenges that a GAS model (in our case, Global Arrays) poses to its underlying GAS runtime (in our case, ARMCI).

Resource Management – The first challenge is on the allocation of resources for communication. Consider an example of the targeted systems for this work, the Cray XT5. Cray XT5 has Seastar2+ interconnect and

uses the connection-less Portals messaging library as the lowest level communication protocol. To support the connection-less Portals interface, Seastar2+ allows for 256 simultaneous message streams. When additional streams need to be initiated (or in case of resource exhaustion), the Cray BEER (Basic End to End Reliability) protocol does the necessary flow control and handles reliability. This means that the resource allocation problem for ARMCI communication buffers (where a set of buffers needs to be allocated for every incoming edge as shown in Figure 1) maps to parallel message streams in Portals but at a different scale. The total request buffer requirement in ARMCI for the FCG would be roughly $N * B * M$, where N is the total number of processes (actually slightly smaller than N due to local processes), B the buffer size, and M the set of buffers per process. With only two 16-KB buffers per process, it would require 1,024 MB per CHT to support parallel programs with 32,000 processes, and 32 GB per CHT on a future system with a million processes.

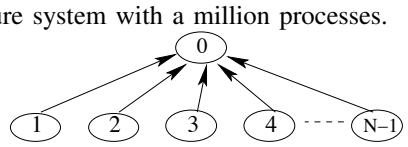


Figure 2: A Flat-Tree Representation of Contention among Communication Requests

Contention – Another challenge revealed by the FCG model is the potential contention that could be caused by many concurrent requests to a single node. Because all nodes (vertices) are directly connected, the paths for requests from all nodes to traverse a virtual FCG and reach one node can be represented as a flat tree of depth 1. Figure 2 shows a tree representation of request traversal paths to Node 0. Such a flat tree is very vulnerable to transient hot-spot accesses, such as when thousands of processes simultaneously accessing one data element in an address space. These scenarios create a severe contention problem in addition to the resource allocation problem described above. Under such scenarios, significant burden is placed on the physical network, which will be forced to adopt some throttling mechanisms, typically causing serious slowdown of the entire communication and jeopardizing the system productivity.

III. Virtual Topologies for Global Arrays Runtime

As discussed in Section II, the default resource allocation in ARMCI leads to a serious scalability challenge. More importantly, its resource dependence relationship (irrespective of any underlying physical network topology) can cause contention when some processes become hot-spots to the communication requests. A virtual topology FCG can precisely reflect the state of resource allocation and contention. It also suggests that alternative virtual topologies may offer a solution for scalable resource management and contention attenuation. To this aim, we introduce two new virtual topologies: MFCG and CFCG. We examine the features of these two topologies, along with Hypercube.

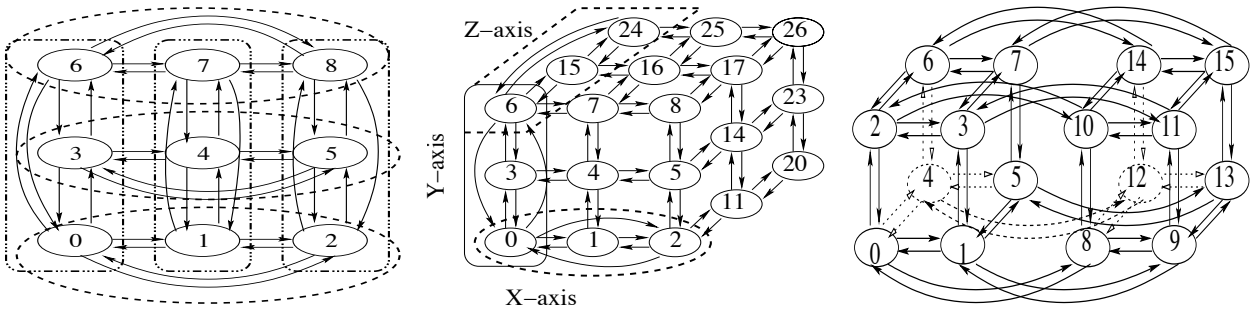
A. MFCG

The first virtual topology is called Meshed Fully Connected Graphs (MFCG for short). Figure 3 (a) shows an example of MFCG, in which all nodes are virtualized as vertices in a $X \times Y$ mesh (in this case, $X = 3$ and $Y = 3$). Nodes with the same Y-offset are fully connected. That is to say, they all dedicate request buffers to each other. The same policy is applied to nodes with the same X-offset. Thus, for an arbitrary $X \times Y$ MFCG, an individual node has $(X - 1)$ outgoing edges on X-dimension and $(Y - 1)$ outgoing edges on Y-dimension. A request forwarding mechanism is provided to exchange requests between a pair of nodes that are not directly connected. Therefore, using MFCG, the number of request buffers on each node decreases to $O(\sqrt{N})$, instead of $O(N)$ in FCG.

MFCG is also beneficial in alleviating contention. Figure 4 (a) shows request paths for nodes in a 3×3 MFCG to reach Node 0. Two types of request paths are possible: the first type is used by the nodes that are directly connected to Node 0; and the second type is used by the nodes that are not directly connected. These paths form a tree of height 2 and rooted at Node 0. Compared to the flat tree as shown in Figure 2, the contention is reduced to $O(\sqrt{N})$. One may rightfully argue that contention as depicted in Figure 4(a) does not reflect the actual contention in the physical network. The purpose of scalable virtual topology is to offer a convenient tool that can cope with network contention at a software level, instead of leaving the contention issues completely to the network hardware.

B. CFCG

Another virtual topology we introduced is Cubic Fully Connected Graphs (CFCG). Figure 3 (b) shows an example of CFCG, in which all nodes are virtualized as vertices in a $X \times Y \times Z$ cube (in this case, $X = 3$, $Y = 3$, and $Z = 3$). The nodes with the same offsets on two dimensions are fully connected as an FCG. For an arbitrary $X \times Y \times Z$ CFCG, an individual node have $(X-1)$ outgoing edges on X-dimension, $(Y-1)$ outgoing edges on Y-dimension, and $(Z-1)$ outgoing edges on Z-dimension (to clarify, not all vertices/edges are shown



(a) Meshed FCGs (b) Cubic FCGs (c) Hypercube

Figure 3: Three Virtual Topologies (For clarity, not all vertices/edges are shown in CFCG)

for CFCG). Using CFCG, the number of request buffers on one node scales in the order of $O(\sqrt[3]{N})$, instead of $O(N)$ with FCG. A request may have to be forwarded maximally two times before reaching its destination.

Figure 4(b) shows the tree representation of request paths for nodes in a $3 \times 3 \times 3$ CFCG to reach Node 0. These directed paths form a trinomial tree of height 3 and rooted at Node 0. For a system with N nodes, the tree of request paths rooted at an arbitrary node will be k-nomial tree where k = $\sqrt[3]{N}$. Compared to the flat tree in Figure 2, network contention is then reduced by an order of $O(\sqrt[3]{N})$, at the expense of up to 2 forwarding steps to deliver a request.

C. Hypercube

As discussed above, CFCG is more scalable in resource allocation than MFCG and FCG, despite more steps for request forwarding. One may wonder if a virtual topology of even higher dimension could be a worthy solution. So we investigate the third virtual topology, Hypercube. Figure 3(c) shows 16 nodes that are connected as a Hypercube. Each node is directly connected to $\log_2 N$ nodes (4 in this case). Figure 4(c) provides a tree representation of request paths from all nodes to Node 0. For N nodes, it is essentially a binomial tree of depth $\log_2 N$. Using Hypercube, the number of request buffers required on one node scales in the order of $O(\log_2 N)$. Two nodes may be separated by up to $\log_2 N$ dimensions apart. Therefore, up to $(\log_2 N - 1)$ forwarding steps are needed for a request to reach its destination. On the other hand, at each depth of a request path tree, contention is reduced by an order of $O(\log_2 N)$.

IV. Implementation

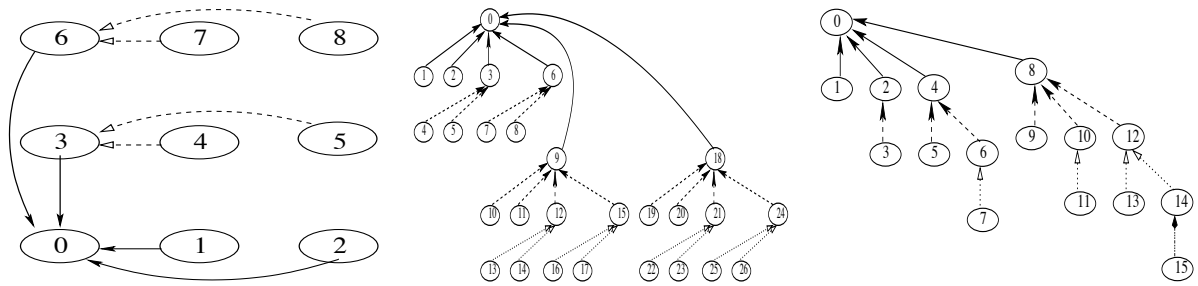
We have implemented MFCG, CFCG, and Hypercube in ARMCI on Jaguar. The support for request forwarding is the key to realizing these virtual topologies. Communication servers, i.e., CHTs, on intermediate nodes are used to forward a request from the original process to the target server. Upon the arrival of a request, the target sends a response (or acknowledgment) directly to the original process. If an intermediate server (or the target) detects that the request is forwarded from an upstream server, it sends an acknowledgment to the upstream server. To support multidimensional topologies such as MFCG, CFCG, and Hypercube, our implementation also allows a request to be forwarded multiple steps.

For correct request forwarding, the actual implementation of virtual topologies requires proper handling of two important issues: (a) how to determine the order of forwarding; and (b) how to enable virtual topologies, MFCG and CFCG, when the number of nodes can only be configured as partially-populated topologies (mesh or cube), e.g., a prime number that cannot be evenly divided. As mentioned in Section III-C, we include Hypercube only to examine its tradeoff in resource management and contention, compared to MFCG and CFCG. For the investigative purpose, we only support hypercube when the number of nodes is a power of 2.

A. Lowest-Dimension-First Forwarding

Multiple communication steps are needed for an ARMCI request to properly reach its destination, in multi-dimensional virtual topologies such as MFCG, CFCG and Hypercube. Each step corresponds to a relationship in which an upstream node is dependent on the availability of request buffer at the downstream node. If the forwarding of requests were to happen arbitrarily, it would create cyclic dependences and lead to deadlocks in a multi-dimensional virtual topology.

We develop a lowest-dimension-first (LDF) protocol to ensure deadlock-free forwarding in virtual topologies. Algorithm 1 illustrates the selection of next node for request forwarding in LDF. For two nodes $S = (s_0, s_1, ..., s_{k-1})$ and $T = (t_0, t_1, ..., t_{k-1})$ on a virtual topology with k dimensions, LDF always chooses the lowest dimension i on which S and T differ. A request is then forwarded to the next destination D, which is a number



(a) Tree for MFCG (b) Trinomial (or K-nomial) Tree for CFCG (c) Binomial Tree for Hypercube

Figure 4: Tree Representations of Request Paths in Virtual Topologies

Algorithm 1 Lowest Dimension First Forwarding

{Dimension: k}
{Current Node: $S = (s_0, s_1, ..., s_{k-1})$}
{Destination Node: $T = (t_0, t_1, ..., t_{k-1})$}
$D \leftarrow S$ {Initialize D as the next node}
$i \leftarrow 0$
while $(D \neq T)$ **do**
  if $s_i \neq t_i$ **then**
    $D \leftarrow (s_0, s_1, ..., s_{i-1}, t_i, s_{i+1}, ..., s_{k-1})$
    {Forward the request to the next node, D}
  end if
  $i \leftarrow i + 1$
end while

derived by replacing s_i of S with t_i. Since the order of forwarding is established in a monotonic dimension order, breaking any cyclic dependence. Therefore LDF is deadlock-free. When the number of nodes allows virtual topologies to be fully populated as meshes, cubes, or hypercubes, LDF as shown in Algorithm 1 works perfectly.

B. Forwarding on Virtual Topologies with Any Number of Nodes

Forwarding on virtual topologies is similar to routing in physical interconnects. In the case of a fully populated two-dimensional MFCG, LDF can be reduced to the classic turn model [12] that was designed for 2-D meshes. However, the key difference is that a virtual topology is very dynamic and often partially populated. For this reason, each node frequently changes its position from one topology to another. It is important that deadlock-free forwarding be enabled on virtual topologies (MFCG and CFCG) with any number of nodes.

We achieve that by strictly ordering all nodes in a lowest dimension first manner. For a virtual topology G with dimension k, the lower order dimensions are first populated with available nodes. Only the highest dimension, $k-1$, is allowed to be partially populated. Assume that a virtual topology G has M as its highest ranked node, where $M = (M_0, M_1, ..., M_{k-1})$. With all nodes ordered this way, we extend the LDF algorithm slightly. It allows forwarding only when the next destination D is a number smaller than or equal to M. An extra condition, "if $(D \leq M)$", is introduced to Algorithm 1 before a request is forwarded. With this extension, if forwarding paths of a set of requests did not violate this extra condition, there would not be a deadlock because their forwarding paths are determined by Algorithm 1. For a possible deadlock to occur, one request must have violated this condition once in its path. This is not possible because the nodes are strictly ordered and no node can have a rank higher than M (by definition). Therefore, it prevents any circle in request forwarding. The listing of the extended LDF algorithm is not included here, due to the simplicity of this addition.

V. Analysis of Resource Management and Contention Attenuation

We conduct experiments on Jaguar at ORNL to evaluate the impact of virtual topologies on resource management and contention attenuation.

A. Scalable Resource Management

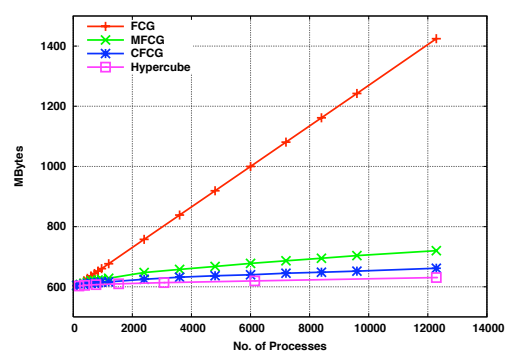


Figure 5: Scalability Virtual Topologies for Memory Management

Jaguar runs the Compute Node Linux operating system. On each node, the /proc file system reports the

memory footprint of all processes as the resident working set size (VmRSS). We create an ARMCI program that reports VmRSS from all processes. This number represents the total memory consumed by an ARMCI process at runtime before any additional application-level memory consumption.

We measure the impact of virtual topologies on memory resources. Our experiments are conducted with 12 processes per node. All processes start with a memory consumption of about 612 MBytes. However, due to the allocation of request buffers by the internal CHT, a master process requires more memory for an increasing number of remote processes. The size of each buffer in CHT is 16KB; and the number of buffers per process is 4. Figure 5 shows the memory consumption of master processes, using different virtual topologies. As expected, the memory requirement of FCG increases linearly. On 12,288 processes, FCG has a memory consumption of 1,424 MBytes, an increment of 812 MBytes, on top of 612 MBytes that is needed to run a few processes. The other three virtual topologies provide much better scalability in terms of memory resources. Compared to FCG, MFCG, CFCG, and hypercube cut down the increment in memory consumption significantly, by 7.5, 16.6, and 45 times, respectively.

B. Contention Attenuation

Virtual topologies are also designed to address the other critical challenge, hot-spot contention in the GAS runtime. We evaluate contention for all one-sided ARMCI operations, and observe that virtual topologies are beneficial to the contention caused by lock, accumulate, noncontiguous data transfer, and atomic operations. Herein presented are results for two representative operations, noncontiguous vector data transfer and atomic Fetch-&-Add operations.

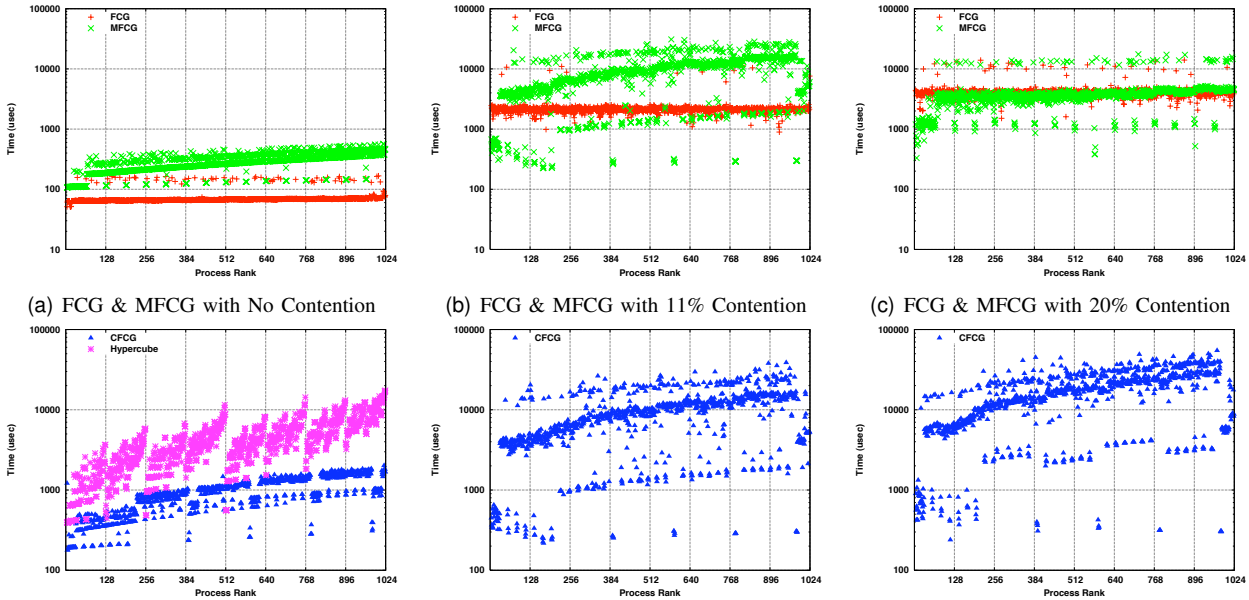
1) Description of Contention Experiments: We define hot-spot contention as the percentage of processes in a program that are contending for communication to a single process, or access to a single data element. It is understood that such contention can arise from sources outside of a program, e.g., from other programs or system services. But, for practical purposes, we consider those beyond the scope of this study, and focus on hot-spot contention within a program.

We use programs with 1,024 processes for contention assessment, 4 processes per node across 256 nodes. These numbers provide a reasonable balance between the need of many nodes to exhibit contention and the need of clarity in visualizing all data points of the results. In these programs, each process (except those on the same node with Rank 0), prepares its data as needed (vectored or strided data in the case of noncontiguous data transfer operations), and then performs one or more one-sided operations to Rank 0. This is then repeated for 20 iterations. The average time for these iterations is taken as the time to complete an operation between the respective process and Rank 0.

Measurements are collected under three different contention scenarios. In the first scenario, each process sequentially performs its own one-sided operations to Rank 0, repeats for 20 iterations, and records the time. At the same time, all other processes are idle in a barrier. This effectively measures the performance of one-sided operations between Rank 0 and all other processes, without any contention. In the second scenario, each process sequentially performs the same number of operations to Rank 0, for the same number of iterations. However, in the meantime, one in every nine processes performs the same operations to Rank 0, while the remaining processes are idle in a barrier. Therefore this corresponds to 11% contention. The third scenario is very similar to the second one, except that one in every five processes concurrently invokes one-sided operations to Rank 0. This then corresponds to 20% contention.

2) Noncontiguous Data Transfer Operations: We conduct experiments to measure the performance of vectored put and get operations as representatives of noncontiguous data transfer functions. Figure 6 shows the time of vectored put operations from all remote processes to Rank 0. Comparisons are provided among varying levels of contention (no contention, 11% contention, and 20% contention).

Figures 6 (a) and (d) show the comparisons under no contention. Several performance behaviors are revealed by this figure. First, the use of MFCG, CFCG and Hypercube increases the time to complete noncontiguous data transfer operations between Rank 0 and other processes. Second, even though all processes are one step away from Rank 0 in FCG, the time to complete noncontiguous data transfers gradually increases with the process rank. This suggests that the distance between a processes and Rank 0 in the underlying physical topology would play a role and contribute to the increased performance. This increment of time is magnified by the use of MFCG, CFCG and Hypercube. In particular, the results from Hypercube indicate that using a topology with very high dimensions for minimal memory consumption does not provide a good tradeoff to the performance. Third, with MFCG, the performance numbers from all processes form several distinct curves, representing differences in their (virtual-) topological relationship with respect to Rank 0. The same can be observed for CFCG and



(a) FCG & MFCG with No Contention (b) FCG & MFCG with 11% Contention (c) FCG & MFCG with 20% Contention

(d) CFCG & Hypercube with No Contention (e) CFCG with 11% Contention (f) CFCG with 20% Contention

Figure 6: Vectored Data Transfer Operations Under Varying Levels of Contention

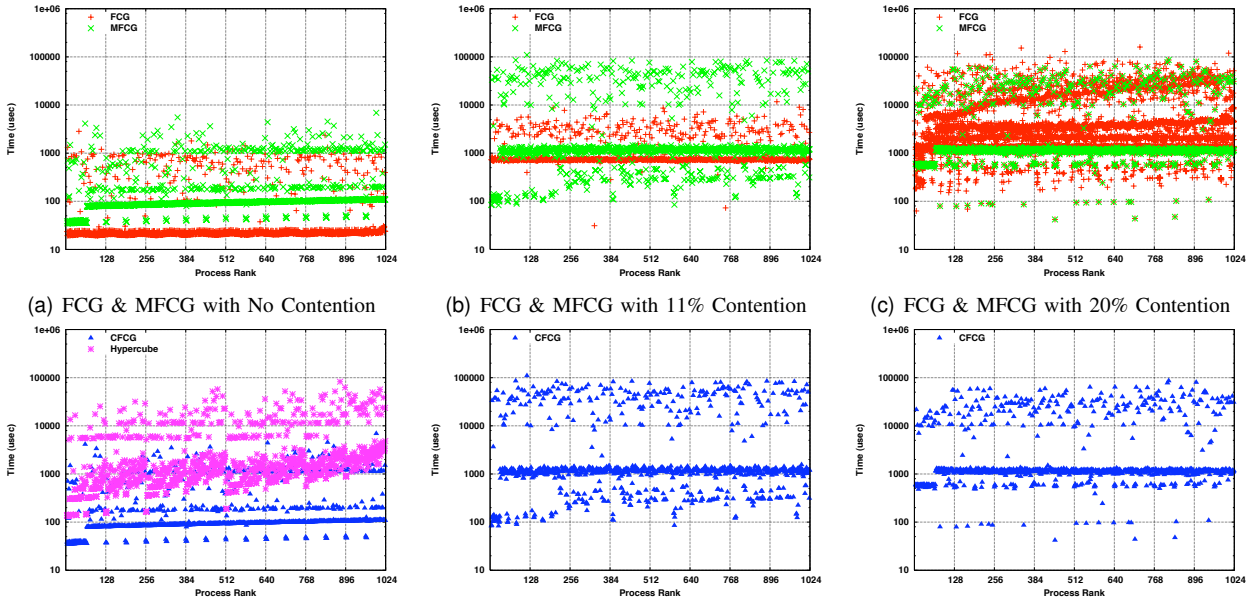
Hypercube as shown in Figure 6 (d).

Figures 6 (b), (c), (e), and (f) show performance comparisons with increased contention. Hypercube is not included in (e) and (f) because it takes too long to get a complete set of numbers. While contention increases the time to complete noncontiguous data transfer operations for all cases, it is evident that all virtual topologies exhibit contention resilience. While the performance of vectored put operations is degraded by nearly two orders of magnitude due to contention inside FCG. With 20% contention, it becomes faster to complete noncontiguous data transfer operations for nearly all processes when using MFCG, compared to FCG. Comparing Figures 6 (b) and (c) it is interesting to note that MFCG also reduces the variations among all processes at higher hot-spot contention. The operation time for the group of processes in the middle has been brought down. This counterintuitive observation is because of the execution behavior of ARMCI CHT. When more processes are actively forwarding requests, they stay in the polling mode for handling requests and therefore have better response time in average. In summary, these results demonstrate that virtual topologies, such as MFCG and CFCG, can attenuate the pressure of many contending noncontiguous data transfer operations, and lead to graceful resilience to contention.

3) Atomic Fetch-&-Add Operations: We measure the performance of fetch-&-add as a representative of atomic operations. Figure 7 shows the time for fetch-&-add operations from all remote processes to Rank 0. Comparisons are provided among different virtual topologies, and among varying levels of contention (no contention, 11% contention, and 20% contention).

Figures 7 (a) and (d) show the comparisons under no contention. Similar observations can be made for atomic operations as revealed by Figures 6 (a) and (d). To be brief, these include (1) the use of MFCG, CFCG and Hypercube increases the time to finish atomic operations under no contention; (2) the time of an atomic operation increases with a higher ranked process, suggesting a correspondence to the distance between the process and Rank 0 in the underlying physical topology; and (3) the performance numbers of atomic operations from all processes form several distinct groups, representing their relationship in the virtual topologies.

Figures 7 (b), (c), (e), and (f) show comparisons with increased contention. Again, hypercube was not included in (e) and (f). While contention increases the time to complete atomic operations for all cases, it is also evident that all virtual topologies exhibit contention resilience. With 20% contention, it becomes faster to complete atomic operations for nearly all processes using MFCG than FCG. Under the same level of contention, even with CFCG, the time for fetch-&-add is shorter for a majority of processes compared to the same with FCG. These results again demonstrate that virtual topologies, such as MFCG and CFCG, can greatly attenuate the pressure of contending atomic operations.



(a) FCG & MFCG with No Contention (b) FCG & MFCG with 11% Contention (c) FCG & MFCG with 20% Contention

(d) CFCG & Hypercube with No Contention (e) CFCG with 11% Contention (f) CFCG with 20% Contention

Figure 7: Fetch-&-Add Operations Under Varying Levels of Contention

VI. Performance Evaluation with Applications

We have shown that virtual topologies can be very beneficial to reduce memory footprint and attenuate contention that would occur to hot-spot processes. It is important to find out how and when these features can be beneficial to real applications. To this aim, we conduct experiments on Jaguar at ORNL to evaluate virtual topologies with real applications.

A. NAS LU Application

The LU application in the NAS parallel benchmark suite [13] has been ported to the ARMCI runtime. It can scale to hundreds or a couple of thousand processes. We evaluate the performance impact of virtual topologies to LU at this scale. Figure 8 shows the performance of LU using all four virtual topologies on a varying number of processes. As shown in the figure, virtual topologies (MFCG, CFCG, and Hypercube) perform better or similar to FCG. At a lower number of processes, the benefit of virtual topologies is slightly higher. Two observations can be made about these results. First, the LU application does not suffer much from hot-spot contention. Second, the reduction in memory footprint does not directly lead to the reduction in execution time, which is quite reasonable. On the other hand, these results are encouraging because they demonstrate that, despite the additional forwarding steps on ARMCI operations such as non-contiguous data transfer and atomic accumulation, virtual topologies still bring comparable or better performance for applications such as LU.

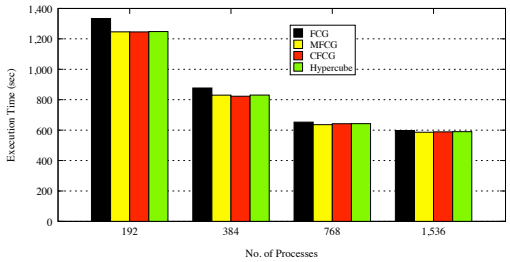


Figure 8: The Performance of NAS LU

B. A Large-Scale Application: NWChem

We evaluate virtual topologies using two electronic structure methods in a large-scale application NWChem [9]: the SiOSi3 method for Density Functional Theory (DFT) and the water model of Coupled Cluster (CC) in its CCSD(T) incarnation. Figure 9 shows the performance of NWChem with different virtual topologies. The performance of SiOSi3 on all virtual topologies is shown in Figure 9(a). Among the four topologies, MFCG and CFCG clearly performs better than the default FCG. MFCG reduces the total execution time by as much as 48%. Hypercube, because of its extra number of forwarding steps, actually leads to worse performance than FCG. These results suggest that SiOSi3 is very prone to hot-spot contention, in which case MFCG is the best virtual topology to mitigate the impact.

Figure 9(b) shows the performance of CCSD(T) water

model when using FCG and MFCG. FCG generally performs better than MFCG, except in one case at 10,000 cores. This result suggests that the total execution time for the water model does not benefit from virtual topologies such as MFCG. The primary benefit of MFCG is the ability to significantly reduce memory consumption of ARMCI low-level runtime library (as detailed in Section V-A). This spares much more memory to be used by applications and help them achieve better scaling.

These application evaluation results demonstrate that MFCG can achieve the best balance of memory consumption, the need of request forwarding, and contention attenuation for the GAS runtime. With much reduced memory consumption at the runtime level, MFCG in general performs comparably to the default FCG. Particularly when an application is experiencing hot-spot contention, MFCG can mitigate the impact of contention and lead to significantly reduced total execution time.

VII. Related Work

Topologies for communication networks have been well documented in the textbooks [14], [15]. Exploiting scalable topologies for high performance communication networks has also been studied extensively in the literature, such as those in [16], [17], [18]. Our research represents an innovative use of classic topologies. By imposing mesh and cube topologies on top of small fully connected graphs (FCG), we introduced meshed FCGs (MFCG) and cubic FCGs (CFCG) to formalize challenging issues faced by today's petascale programming models.

Numerous algorithms were investigated to support deadlock-free message routing in interconnection networks. In their classic paper, Dally et al. [19] proposed deadlock-free message routing algorithms, such as dimension-order routing, for multiprocessor interconnection networks using the concept of virtual channels. Duato et al. [20] investigated deadlock-free adaptive multicast routing algorithms on worm-hole networks using a path-based routing model. Lin and Lionel [21] compared different multicast worm-hole routing algorithms, such as dual-path routing and multi-path routing, for multicomputers with 2D-mesh and hypercube topologies. Our work builds on top of the dimension-order routing algorithm, and proposes the deadlock-free LDF (lowest dimension first) algorithm. LDF only needs to forward an ARMCI request once per dimension in MFCG, CFCG, and Hypercube. In addition, it allows partially populated MFCG and CFCG on any number of network nodes.

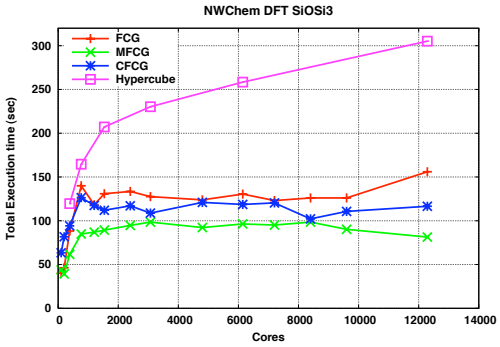
Many efforts studied the scalability of resource management for other contemporary programming models. Sur *et al.* [22] examined the memory scalability of various MPI implementations on the InfiniBand network. Koop *et al.* [23] exploited the use of message coalescing to reduce the memory requirements for MPI on InfiniBand clusters. Chen *et al.* [24] optimized the communication for UPC applications through a combination of techniques including redundancy elimination, split-phase communication, and communication coalescing. Our work differs from these earlier studies by introducing new virtual topologies to reveal the challenges of resource management and contention in the ARMCI Global Address Space runtime system. To the best of our knowledge, this is the first in literature to exploit the concept of virtual topology for systematic investigation of scalability and contention issues in Global Address Space programming models.

VIII. Conclusion

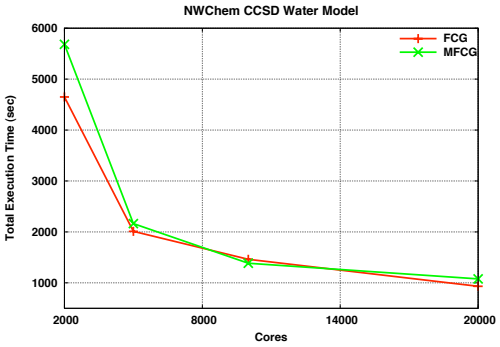
In conclusion, we have systematically studied the resource management and contention issues in a GAS run-time system, ARMCI, on the petascale Jaguar Cray XT5 system at ORNL. First, we introduce the concept of virtual topology to represent the management of communication resources in ARMCI as directed graphs, and substantiate it with two new virtual topologies, MFCG and CFCG, as well as Hypercube. Second, we design and develop *lowest dimension first* forwarding to ensure deadlock-free communication in ARMCI using MFCG and CFCG on any number of nodes. Third, we investigate and extensively evaluate the performance of all three virtual topologies, MFCG, CFCG, and Hypercube, and demonstrate that MFCG is the best choice in accomplishing scalable resource management and contention attenuation.

In the future, we look forward to further optimization of the GA model and ARMCI on petascale systems. We are investigating large-scale applications that can leverage more memory at the application level for better total execution time. We also plan to study the applicability of virtual topologies on other petascale platforms with different physical topologies, e.g., BlueGene/P [25], [26]. Furthermore, we plan to investigate the benefits of virtual topologies in the context of PGAS languages such as UPC [4] and Co-Array Fortran [5].

Acknowledgments
This work was funded in part by NSF award CNS-1059376, by UT-Battelle grant (UT-B-4000087151), and by National Center for Computational Sciences. This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S.



(a) SiOSi3



(b) CCSD(T) $(H_2O)_{11}$

Figure 9: NWChem Execution Time

Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] "Top 500 supercomputing sites," http://www.top500.org.

[2] LLNL, "ASC Sequoia," https://asc.llnl.gov/computing_resources/sequoia/.

[3] NCSA, "Blue Waters: Sustained Petascale Computing," http://www.ncsa.illinois.edu/BlueWaters/.

[4] "Upc specifications, v1.2," http://www.gwu.edu/∼upc/publications/LBNL-59208.pdf.

[5] Y. Dotsenko, C. Coarfa, and J. Mellor-Crummey, "A multi-platform co-array fortran compiler," Sept.-3 Oct. 2004, pp. 29–40.

[6] "Report on experimental language X10," 2008, http://dist.codehaus.org/x10/documentation/languagespec/x10-170.pdf.

[7] "Global arrays toolkit," http://www.emsl.pnl.gov/docs/global.

[8] J. Nieplocha, V. Tipparaju, M. Krishnan, and D. K. Panda, "High Performance Remote Memory Access Communication: The Armci Approach," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 233–253, 2006. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/20/2/233

[9] R. A. Kendall, E. Aprà, D. E. Bernholdt, E. J. Bylaska, M. Dupuis, G. I. Fann, R. J. Harrison, J. Ju, J. A. Nichols, J. Nieplocha, T. P. Straatsma, T. L. Windus, and A. T. Wong, "High performance computational chemistry: An overview of NWChem a distributed parallel application," *Computer Physics Communications*, vol. 128, no. 1-2, pp. 260–283, June 2000. [Online]. Available: http://dx.doi.org/10.1016/S0010-4655(00)00065-5

[10] E. Apra, R. J. Harrison, W. de Jong, A. Rendell, S. Tipparaju, V. an d Xantheas, and R. Olsen, "Liquid water: Obtaining the right answer for the right reasons," in *Supercomputing, 2009. SC '09. Proceedings of the ACM/IEEE SC 2009 Conference*, 2009.

[11] V. Tipparaju, E. Apra, W. Yu, and J. S. Vetter, "Enabling a highly-scalable global address space model for petascale computing," in *Computing Frontiers '09*, 2010.

[12] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *SIGARCH Comput. Archit. News*, vol. 20, no. 2, pp. 278–287, 1992.

[13] D. H. Bailey, L. Dagum, E. Barszcz, and H. D. Simon, "Nas parallel benchmark results," in *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, pp. 386–393.

[14] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, 1st ed. Morgan Kaufmann Publishers, Inc. , 1991.

[15] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. The IEEE Computer Society Press, 1997.

[16] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Comput.*, vol. 39, no. 6, pp. 775–785, 1990.

[17] D. K. Panda, "Fast barrier synchronization in wormhole k-ary n-cube networks with multidestination worms," in *HPCA '95: Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 1995, p. 200.

[18] F. Petrini and M. Vanneschi, "k -ary n -trees: High performance networks for massively parallel architectures," *Parallel Processing Symposium, International*, vol. 0, p. 87, 1997.

[19] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.

[20] J. Duato, "A theory of deadlock-free adaptive multicast routing in wormhole networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 9, pp. 976–987, 1995.

[21] X. Lin and L. M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," *SIGARCH Comput. Archit. News*, vol. 19, no. 3, pp. 116–125, 1991.

[22] S. Sur, M. J. Koop, and D. K. Panda, "High-performance and scalable mpi over infiniband with reduced memory usage: an in-depth performance analysis," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2006, p. 105.

[23] M. J. Koop, T. Jones, and D. K. Panda, "Reducing connection memory requirements of mpi for infiniband clusters: A message coalescing approach," in *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 495–504.

[24] W.-Y. Chen, C. Iancu, and K. Yelick, "Communication optimizations for fine-grained upc applications," in *PACT '05: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 267–278.

[25] IBM BG/P Team, "Overview of the IBM Blue Gene/P project," *IBM Journal of Research and Development*, vol. 52, no. 1/2, pp. 199–220, Jan. 2008.

[26] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. S. Vetter, P. Worley, and W. Yu, "Early evaluation of ibm bluegene/p," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008.