TRAM: Optimizing Fine-grained Communication with Topological Routing and Aggregation of Messages

Lukasz Wesolowski*, Ramprasad Venkataraman†*, Abhishek Gupta*, Jae-Seung Yeom‡, Keith Bisset‡, Yanhua Sun*, Pritish Jetley*, Thomas R. Quinn§, Laxmikant V. Kalé*

* Department of Computer Science, University of Illinois at Urbana-Champaign
{wesolwsk, gupta59, sun51, pjetley, kale}@illinois.edu
† Google Inc. vram@google.com
‡ Department of Computer Science, Virginia Bioinformatics Institute, Virginia Tech {jyeom, kbisset}@vbi.vt.edu
§ Department of Astronomy, University of Washington trq@astro.washington.edu

***Abstract*—Fine-grained communication in supercomputing applications often limits performance through high communication overhead and poor utilization of network bandwidth. This paper presents Topological Routing and Aggregation Module (TRAM), a library that optimizes fine-grained communication performance by routing and dynamically combining short messages. TRAM collects units of fine-grained communication from the application and combines them into aggregated messages with a common intermediate destination. It routes these messages along a virtual mesh topology mapped onto the physical topology of the network. TRAM improves network bandwidth utilization and reduces communication overhead. It is particularly effective in optimizing patterns with global communication and large message counts, such as all-to-all and many-to-many, as well as sparse, irregular, dynamic or data dependent patterns. We demonstrate how TRAM improves performance through theoretical analysis and experimental verification using benchmarks and scientific applications. We present speedups on petascale systems of 6x for communication benchmarks and up to 4x for applications.**

***Keywords*-Communication Optimization, Message Aggregation, Interconnection Networks**

I. Introduction

Architectural trends in capability-class supercomputing systems point to a rapid increase in the number of processing elements, while the available memory or data movement capacities are not increasing as fast [1]. These trends, and the computational requirements of grand challenges in many domains, indicate that parallel applications will have to optimize for the strong scaling regime. Additionally, application classes with unstructured, data-dependent, and fine-grained communication patterns are becoming increasingly prominent. Hence, large-scale parallel execution will be increasingly fine-grained, and both compute and communication grain sizes will become smaller.

Programming models that are evolving to harness such extreme scale concurrency are also encouraging the increased expression of parallelism. Consequently, techniques like overdecomposition and lightweight work or control units are resulting in increasingly fine-grained communication patterns. Even in established parallel programming systems like MPI, many small control messages and acknowledgements are sent by the underlying messaging library to orchestrate data movement. As we scale to tens and hundreds of thousands of nodes, the number of such messages can become significant. For these reasons, it is essential to optimize fine-grained communication and to provide abstractions that protect applications from this burden.

In this paper we present Topological Routing and Aggregation Module (TRAM), a library for improving communication performance of fine-grained and/or bandwidth-heavy parallel applications. At the core of TRAM is the idea of aggregation and routing of *data items*, or small units of communication, over a virtual N-dimensional mesh mapped onto the processes of a parallel application. For fine-grained communication, each application-level message can typically be represented by a single data item. By combining data items, which are typically on the order of tens of bytes, into larger units, TRAM reduces the impact of per message overhead and the amount of bandwidth consumed by the message header, improving communication throughput. TRAM is a *streaming* library. Rather than send using a synchronized schedule and deliver the full payload for each destination all at once, TRAM sends and delivers data gradually. This allows for a significant overlap of communication and computation. Using TRAM requires only minor changes to application code.

The main contributions of this paper are: (1) the introduction of TRAM, a generic streaming library for topological routing and aggregation over arbitrary virtual mesh topologies, (2) an analysis of the underlying message aggregation and routing approach to determine how it improves performance, (3) automatic selection of TRAM aggregation buffer size and virtual topology specification, and (4) speedups in scientific applications of up to 4x using TRAM.

In the rest of the paper, section III presents a description of our library and the underlying parallel runtime system. Used properly, TRAM can greatly improve communication

performance, but there are computation and communication costs associated with using it which are important to understand. To further this goal, section IV presents a theoretical study of message aggregation as used in our library, while section V presents analysis and experimental results showing how virtual topology affects message aggregation, TRAM memory footprint, and congestion. We conclude the paper with experimental results for two scientific applications, demonstrating speedups of up to 4x with TRAM.

II. Related Work

The key aspects of our approach, mainly *a*) *message aggregation using a library*, *b*) *software routing*, the use of a *c*) *generalized mesh virtual topology*, and *d*) *streaming* have each been explored in isolation in a number of contexts. Here we will compare to work that most closely resembles key elements of our approach.

Aggregation and software routing of messages has been studied most prolifically for collective communication, particularly all-to-all, going back at least 20 years to work on indirect mesh algorithms by Thakur and Choudhary [2].

Over time, the approach was generalized to other virtual topologies and a wider class of collectives. Past work from our group demonstrated improved performance of all-to-all and many-to-many personalized communication using 2D and 3D virtual meshes for routing and aggregation [3]. Kumar also presented analysis of the reduction in the number of messages by using 2D and 3D virtual topologies to aggregate messages for all-to-all personalized communication [4]. Most of these algorithms did not employ streaming, however. Kumar's work did include a streaming library that used a routing and aggregation approach within a two dimensional virtual mesh approximating a square, but its topology was not configurable and the work did not analyze the importance of a good match between virtual and physical topologies.

A more recent example of aggregation and routing of messages over virtual topologies using a streaming library is Active Pebbles [5]. In contrast to our approach, this work did not involve matching the virtual topology to the physical network topology to ensure minimal routing.

Other research efforts have focused on application and machine-specific uses of the aggregation and routing approach. Garg and Sabharwal demonstrated dramatic improvements in performance of the HPC Challenge Random Access benchmark on Blue Gene systems [6]. Kumar et al. used topological routing and aggregation to improve performance of All-to-All and FFT on the Blue Gene/L [7].

Message aggregation and routing over virtual topologies have also each been applied in isolation to reduce runtime system overhead. For ARMCI on the Cray XT5, Yu et al. proposed virtual topologies and routing protocols to control resource management and contention [8]. For Infiniband networks, Koop et al. described a scheme for MVAPICH to dynamically coalesce messages for the same destination, with the effect of reducing memory usage and increasing the message rate [9]. We believe a scheme like TRAM, that combines aggregation with routing over a virtual topology, could further improve the effectiveness of these approaches.

In our performance analysis we consider the impact of network topology and congestion on communication performance. Other research on communication over mesh and torus topologies which took into account issues of network congestion includes work on MPI collectives [10], topology-aware mapping [11], and network saturation on Blue Gene/P [12]. Our approach to analyzing message aggregation and determining a good message size also shares common elements with work on message strip-mining [13].

While MPI collectives are typically meticulously optimized, they often artificially limit communication-communication overlap through the use of a static communication schedule. Neighborhood collectives, which allow for expression of a larger range of collective operations and generation of schedules based on link load information, offer an improvement, but are still limited in dynamic scenarios by the use of a static communication schedule [14]. TRAM routes messages over a static communication graph, but it does not follow a prescribed schedule. Instead, it streams data according to its availability using a parameterized communication grain size. As such, we believe TRAM is particularly well suited for implementation of collectives for sparse, irregular, or dynamic communication patterns. We hope that the work and application examples in this paper motivate the need for streaming extensions of collective operations in a future version of the MPI standard.

III. Topological Routing and Aggregation Module

Topological Routing and Aggregation Module [15] is a library for optimizing fine-grained communication patterns in parallel applications. It is implemented as a library in CHARM++, a mature parallel runtime system for distributed object-oriented parallel programming [16].

A. Background

For simplicity of discussion, the CHARM++ runtime system can be assumed to consist of a process on every core involved in the parallel run. These processes, called Processing Elements (PEs), are globally ranked. Expressing parallelism in this system typically involves creating collections of globally accessible objects called *groups* and *arrays*. Groups map a single runtime-system-managed object to every core in a parallel run. Arrays, on the other hand, may contain an arbitrary number of such objects, which are assigned to physical cores by the runtime system based on predefined or custom mappings. A parallel program begins from one or more objects marked as being initial, which in turn create groups and arrays and invoke methods on these objects to continue the parallel program. Functions on individual member objects of groups and arrays can be invoked from any core using globally unique identifiers. If the invoked object is not local to the core where the call is made, an asynchronous message is sent by the runtime system to the appropriate destination, where the message is delivered to the scheduler

for the local instance of the runtime system. Functions invoked by the scheduler are non-preemptible. When a scheduler picks a message from its queue and calls the corresponding function, it becomes inactive until the function it called returns. TRAM is implemented as a group, so an *instance* of TRAM has one library object on every PE used in the run. We use the term *local instance* to denote a member of the TRAM group on a particular PE.

Most collective communication patterns involve sending linear arrays of a single data type. In order to more efficiently aggregate and process network data, TRAM restricts data sent using the library to a single data type specified by the user through a template parameter for the library. We use the term *data item* to denote a single instance of this data type submitted to the library for sending. While the library is active (i.e. after initialization and before termination), an arbitrary number of data items can be submitted to the library at each PE.

B. Routing

TRAM performs aggregation in the context of a virtual mesh topology comprising the processes involved in the parallel run. The number of dimensions in the topology and their sizes are specified when constructing an instance of the library. Let the variables j and k denote PEs within the N-dimensional virtual topology of PEs and x denote a dimension of the mesh. We represent the coordinates of j and k within the mesh as $(j_0, j_1, \ldots, j_{N-1})$ and $(k_0, k_1, \ldots, k_{N-1})$. Also, let

$$f(x, j, k) = \begin{cases} 0, & \text{if } j_x = k_x \\ 1, & \text{if } j_x \neq k_x \end{cases}$$

j and k are *peers* if

$$\sum_{d=0}^{N-1} f(d, j, k) = 1. \tag{1}$$

When using TRAM, PEs communicate directly only with their peers. Sending to a PE which is not a peer is handled inside the library by routing the data through one or more *intermediate destinations* along the route to the *final destination*.

Suppose a data item destined for PE k is submitted to the library at PE j. If k is a peer of j, the data item will be sent directly to k, possibly along with other data items for which k is the final or intermediate destination. If k is not a peer of j, the data item will be sent to an intermediate destination m along the route to k whose index is $(j_0, j_1, \ldots, j_{i-1}, k_i, j_{i+1}, \ldots, j_{N-1})$, where i is the greatest value of x for which $f(x, j, k) = 1$.

Note that in obtaining the coordinates of m from j, exactly one of the coordinates of j which differs from the coordinates of k is made to agree with k. It follows that m is a peer of j, and that using this routing process at m and every subsequent intermediate destination along the route eventually leads to the data item being received at k. Consequently, the number of messages $F(j, k)$ that will carry the data item to the destination is

$$F(j, k) = \sum_{d=0}^{N-1} f(d, j, k). \tag{2}$$

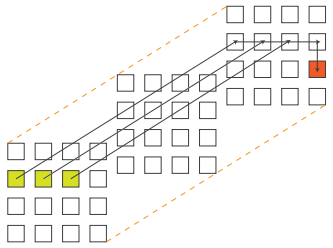


Fig. 1: TRAM routes messages along the dimensions of a virtual topology, using intermediate destinations for increased aggregation. The three separate messages in this example are combined at an intermediate destination and delivered in a single message to the destination.

C. Aggregation

TRAM amortizes communication overhead by aggregating data items at the source and at every intermediate destination. Every local instance of the TRAM group buffers the data items that have been submitted locally or received from another PE for forwarding. Because only peers communicate directly in the virtual mesh, it suffices to have a single buffer for every peer of a given PE. For a dimension d within the virtual topology, let s_d denote its *size*. Consequently, each local instance allocates up to $s_d - 1$ buffers per dimension, for a total of $\sum_{d=0}^{N-1}(s_d - 1)$ buffers. Buffers are of a constant size. Users can directly control TRAM memory footprint by specifying individual buffer sizes or total buffer space.

Sending with TRAM is done by submitting a data item and a destination identifier, either PE or array index, using a function call to the local instance. The library uses the previously described algorithm to identify the peer that will be the final or intermediate destination, and places the data item in the buffer destined to the resulting PE. Buffers are allocated lazily only when at least one data item needs to be sent to a peer. They are sent out immediately upon filling up. When a message is received at a destination, the contained data items are distributed into the appropriate buffers for further routing to their final destinations, or delivered to the user if the recipient is the final destination. Figure 1 shows an example of aggregation of messages by TRAM in a 3D topology.

D. Periodic Dispatch and Termination

Data dependent sending patterns, where the delivery of a data item is required to generate further data item sends, can suffer deadlocks if the original data item is held up in an intermediate buffer due to aggregation. This creates a situation where it is potentially necessary to trade some of the performance benefit from aggregation in order to guarantee global progress. TRAM provides a message dispatch mechanism for

such situations. It periodically checks for progress in the library and sends all buffers out if no sending took place since the last time the check was done. The period is configurable, but should be infrequent enough to permit aggregation.

Termination of a communication step occurs through an ordered dispatch of messages along dimensions from highest to lowest. Termination requires each data sender in a communication step to specify that it has finished submitting data items. The underlying runtime system can be queried for the number of objects sending data on each PE, allowing the activation of the termination mechanism when all senders on a given PE signal completion of sending. During termination, each local TRAM instance ensures that it has finished receiving messages along a higher dimension before sending out its buffers for the next dimension. This check is made by comparing the number of messages received along a given dimension with the sum of the total message counts that were sent out by the peers which finished sending along that dimension. These counts are a small and amortized overhead in TRAM messages.

IV. Message Aggregation Analysis

The topological routing and aggregation approach employed in TRAM can significantly improve performance, but it also carries nontrivial costs in resource utilization and computation time. This section presents analysis and benchmark results that demonstrate the potential benefits and costs of topological routing and aggregation. Based on these results, we show how to select TRAM parameters to increase the effectiveness of the library while controlling the costs.

A. Experimental Methodology

We used three types of supercomputers for experimental results: *a*) *Intrepid and Surveyor* - Blue Gene/P systems at Argonne Leadership Computing Facility, 3D network topology, rectangular job partitions *b*) *Vesta and Vulcan* - Blue Gene/Q systems at Argonne and Lawrence Livermore National Laboratories, 5D network topology, rectangular job partitions *c*) *Blue Waters* - Cray XE6/XK7 system at National Center for Supercomputing Applications, 3D network topology, irregular job partitions. Benchmark results represent an average of 1000 iterations.

B. Nearest Neighbor Communication

The simple case of network communication between processes located on neighboring nodes in the network is sufficient to demonstrate the main benefits of aggregation and determine bounds on message sizes that may benefit from it.

Figure 2 shows effective bandwidth utilization for single message sends of various sizes on our test systems. Results in the plot are represented as percentages of the theoretical network link bandwidth between the nodes. Bandwidth utilization is as low as 0.1% for small message sizes, and rises with increasing message size until coming close to saturating the link bandwidth. For our purposes, we will define the saturation threshold as 75% of the peak *observed* bandwidth, which is about 60% of the theoretical maximum bandwidth

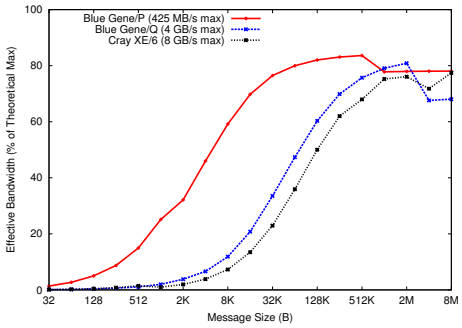


Fig. 2: Effective bandwidth of nearest-neighbor communication relative to peak link bandwidth on three supercomputing systems. Individual sends of small to medium-sized messages utilize at most a few percent of the network bandwidth.

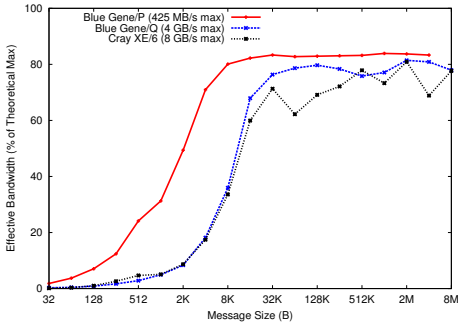


Fig. 3: Effective bandwidth for pipelined sends. On each of the three systems, messages of 16 to 32 KB come close to saturating the link bandwidth.

on our test systems. This point is reached at message sizes of 8 KB, 128 KB, and 256 KB, on Blue Gene/P, Blue Gene/Q, and Cray XE/6 systems, respectively. Aggregating messages at or above this size will lead to little improvement in bandwidth utilization. In fact, very large messages may utilize less bandwidth. We believe this to be a result of cache overflow, when memory, rather than network bandwidth, may be a bottleneck. This behavior is apparent for message sizes above 1 MB on Blue Gene/P and above 4 MB on Blue Gene/Q.

A single send, taken in isolation, does not represent the typical state of a network during execution of a large application. In order to more confidently bound the region of messages that will benefit from aggregation in practice, we repeated the benchmark by sending a stream of messages of a given size between the nodes. The expectation is that this should lead to better network utilization for small to medium-sized messages due to concurrent occupancy of network resources by data for multiple messages and pipelining of communication with the send and receive work in the runtime system. The experiment also more closely approximates the send behavior of TRAM as it streams messages using aggregation buffers of a constant size. The results for this test are shown in Figure 3. Compared to isolated sends, the saturation threshold was reached at lower message sizes when streaming the messages: 4 KB on Blue

Gene/P and 16 KB on Blue Gene/Q and Cray XE/6. Note that despite the streaming, sends of messages below 128 bytes on Blue Gene/P and below 2 KB on Blue Gene/Q and Cray XE/6 still attain less than 10% of the available network bandwidth. Aggregating these messages into buffers of size 8 or 16 KB could provide dramatically higher bandwidth utilization and higher performance for bandwidth-limited applications.

C. Performance Bound Analysis

The previous experiments show that aggregation can significantly improve network bandwidth utilization. We next examine the trade-offs involved in using TRAM, and define bounds on how TRAM affects various performance characteristics. We will use the following parameters in our analysis:

- m: data item payload size in bytes
- e: size of the message header in bytes
- α: constant overhead per message
- β: inverse of network bandwidth
- g: aggregation buffer size (in units of data items)
- r: rate at which items are submitted to TRAM
- N: number of dimensions in virtual topology
- l: average number of links traversed by a message
- z: total number of data items to send

Message Latency It takes time for an aggregation buffer to fill as data items are generated by the application, submitted to the library, and copied into the correct buffer. Buffering can be treated as directly adding to the latency of each data item, or the time from submission to receipt of the data item at the destination. In addition, an aggregate message will generally take longer to arrive at the destination after being sent out compared to a single item. As a result, average item latency can be expected to significantly increase when using TRAM. This must be taken into account when selecting the data item types to be aggregated. In particular, latency-sensitive messages along the critical path should not be aggregated.

Bytes Injected/Sent on the Network TRAM typically reduces the volume of data sent on the network by decreasing the aggregate header data sent. It is important for the routing component not to dilate the path of each data item compared to a direct send, as a message that travels over multiple links consumes bandwidth on each link along the route. We will later see how this can be ensured through careful selection of the virtual topology. In contrast, TRAM will usually increase the aggregate data *injected* onto the network, as it delivers and later re-injects the same data for every intermediate destination along the route of an item.

Message Count Aggregation using a multi-dimensional approach with intermediate destinations reduces the total number of messages by a factor of g/N. The most important consequence of the reduced message count is a corresponding reduction in aggregate message processing overhead.

Table I summarizes the effects of TRAM on the above performance parameters with lower and upper bounds for each quantity. The analysis assumes buffers are filled to capacity. Lower bounds correspond to the most optimistic scenario. For example, for item latency, the lower bound is an estimate for the final item inserted into a buffer before it is sent. For the number of messages and bytes injected, the lower bound assumes all data items are delivered to a peer of the source process (i.e. after a single send). The upper bounds are closer to what may be expected in practice.

	Direct Sends	**TRAM L Bound**	**TRAM U Bound**
Item Latency	$\alpha + \beta(m + e)$	$\alpha + \beta(gm + e)$	$N[\alpha + \beta(gm + e) + g/r]$
Agg. Link Usage	$lz(m + e)$	$lz(m + e/g)$	$lz(m + e/g)$
Injected Bytes	$z(m + e)$	$z(m + e/g)$	$Nz(m + e/g)$
Message Count	z	z/g	Nz/g

TABLE I: Comparison of various performance parameters for direct sends vs. TRAM aggregation

Topology	**F(j,k)**	**# Nodes**	**% Nodes**
32 x 32 x 32	0	1	3.1e-3
	1	93	.28
	2	2883	8.3
	3	29791	90.9
16 x 16 x 16 x 16	0	1	1.5e-3
	1	60	9.2e-2
	2	1350	2.05
	3	13500	20.6
	4	50625	77.2

TABLE II: Distribution of the number of messages required to deliver a data item using TRAM on two symmetric topologies

Note that latency, bytes injected, and total number of messages for TRAM are all affected by the number of dimensions in the virtual topology, which determines the maximum number of messages needed to deliver a data item from the source to its destination. The next section explores what the gains are in return for this significant overhead.

V. Virtual Topology Analysis

Topological aggregation with delivery at intermediate destinations involves significant overhead. However, this overhead is offset by an improved potential for aggregation and reduced memory usage. Here we seek to determine when multi-dimensional aggregation is worth the additional overhead compared to a direct one-dimensional aggregation mechanism.

A. Aggregation Memory Overhead

In the direct aggregation approach, a separate buffer is allocated for each destination PE, leading to a high memory overhead for applications with dense communication graphs. By contrast, the maximum number of buffers per local TRAM instance never exceeds its number of peers. For a given buffer size g, data item size m, and topology dimensions $s_d, d \in 0...N-1$, the memory footprint for the buffer space is $m \times g \times \sum_{d=0}^{N-1}(s_d - 1)$ per local instance of the library. As an example, each local TRAM instance for a 16 x 16 x 16 x 16 virtual topology will allocate up to $15 * 4 = 60$ buffers, which should easily fit in main memory and possibly in cache for buffer sizes which we had experimentally determined to be sufficient for good performance (e.g. 8 KB). This offers a strong reason to prefer TRAM to the direct aggregation approach. In the example above, the direct approach would

lead to $65535 \times 8KB = 512MB$ per core, which may be too high to fit in the main memory of a compute node.

B. Message Counts and Buffer Fill Rate

Multi-dimensional aggregation has a dual effect on communication performance. On one hand, peer-based buffering may lead to a faster buffer fill rate by aggregating into a smaller number of buffers. On the other hand, using intermediate destinations increases the number of messages needed to deliver items to most destinations. To quantify the overall increase in message counts, consider an instance of TRAM operating over an N-dimensional virtual topology. A data item submitted at a PE j to be delivered to PE k will be communicated using a series of messages whose number is specified by $F(j,k)$ from Equation 2. Table II shows the distribution of this function for a fixed source PE j^* for two example topologies, a 32 x 32 x 32 3D topology and a 16 x 16 x 16 x 16 4D topology. For an N-dimensional mesh where each dimension has size d, the number of values of k for which $F(j^*,k) = a$ can be counted using the expression

$$\binom{N}{a} \times (d-1)^a. \tag{3}$$

Sending a message within a PE does not involve network communication, so the message count for this case is 0.

Results show that a data item sent to a randomly selected PE within these topologies will typically use N messages along the route to its final destination. The implications of this are that for all-to-all communication, the average value of $F(j,k)$ will be close to N. On the other hand, the location of the destination in relation to the source in practical communication scenarios is rarely arbitrary, provided that some care is taken in mapping application-level objects to processors in a topology-aware fashion. For example, for applications with mostly nearest-neighbor communication, $F(j,k)$ will be 1 for most pairs of communicating PEs.

The overall performance effect of multi-dimensional aggregation will hence depend on a number of factors, including the total number of items sent, the rate at which items are submitted, the communication pattern, and the improvement in cache locality due to the tighter buffer space.

C. Virtual to Physical Topology Mapping

While the choice of a virtual topology to be used for TRAM is left to the user of the library, we believe that *matching* the physical topology typically leads to the best performance. In this section, we explain why this is the case.

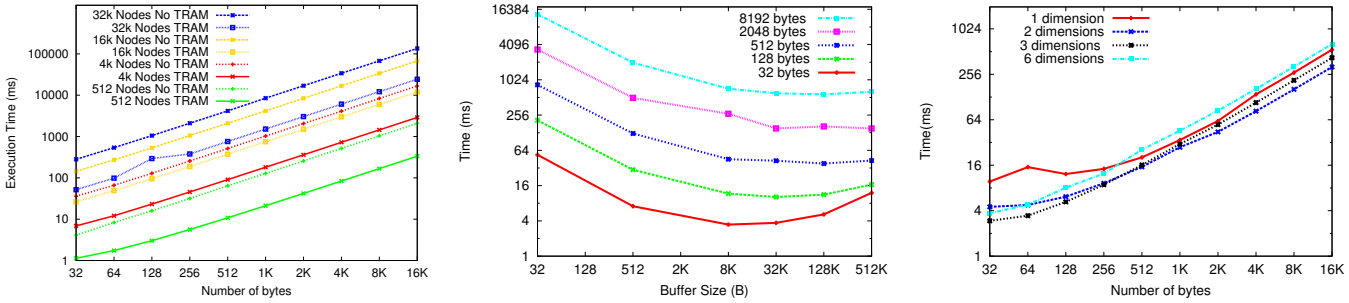
Consider a data item sent over an N-dimensional mesh or torus using TRAM, and let $a = F(j,k)$ from Eq. 2 be the number of messages required to deliver it to its destination. As noted in Sec. III, every intermediate message along the route makes positive progress toward the destination along a single dimension of the virtual topology. If the virtual topology is identical to the physical topology, then as long as the network's dynamic routing mechanism does not make an intermediate message take a non-minimal path, the route to the destination determined by TRAM will be minimal.

It is also possible to reduce the number of dimensions in a virtual topology that matches a physical topology while preserving minimal routing. This can be done by merging any two dimensions that are consecutive in the order of routing. For example, a 4 x 4 x 8 topology can be reduced to a 16 x 8 topology by merging the first two dimensions. A data item sent within the reduced topology obtained in this way will follow the same route as in the full topology while skipping over some intermediate destinations. As we have seen, intermediate destinations add overhead and require re-injecting data onto the network, but in return allow aggregation of data items which would otherwise be sent separately. In cases where the costs outweigh the benefits, a lower dimensional virtual topology that preserves minimal routing will perform better than a higher-dimensional one.

In contrast to matched or properly reduced virtual topologies, topologies which do not maintain minimal routing can severely degrade performance. For example, assume a random mapping between the PEs in the virtual topology and the physical topology, and let s_d be the size of dimension d in the physical topology. On average, each intermediate message will travel the average distance between pairs of PEs in the physical topology, which is $\sum_{d=0}^{N-1} \frac{s_d}{4}$ for a torus and $\sum_{d=0}^{N-1} \frac{s_d}{3}$ for a mesh. In other words, an intermediate message when using a random mapping will on average travel as many hops as a full route would when using a virtual topology that matches the physical topology. Path dilation due to non-minimal routing is clearly undesirable, as it wastes link bandwidth and can increase network congestion. This effect grows in proportion to the size of the physical topology, which affects the number of hops traveled by each intermediate message, and also in proportion to the number of dimensions in the virtual topology, which affects the average number of intermediate messages required to deliver a data item.

For runs with multiple PEs per compute node, a natural extension of the approach of matching the physical topology is to use an additional dimension in the TRAM virtual topology for the PEs within a node. As an example, we often used a 6D topology on Blue Gene/Q, corresponding to the 5 dimensions of the network and the PEs within a node as the 6th dimension. The intranode dimension, when specified as the final dimension of the topology, provides the benefit of some level of intranode aggregation in the initial routing step before data is sent on the network.

For some network types, network switches form a topology that is distinct from the compute node topology. A common example are Clos networks, such as Infiniband networks. These networks have multiple levels of switches, where only switches at the lowest level are directly connected to compute nodes. Matching the virtual topology to the network topology will not work on these systems. Our approach on these systems has been to use a 2D topology, with compute nodes as one dimension and PEs within a node as the second. Techniques that further optimize TRAM for this important class of systems



(a) All-to-all benchmark on Blue Gene/P (b) Effect of aggregation buffer size (c) Effect of virtual topology dimensionality

Fig. 4: All-to-all benchmark: (a) on Blue Gene/P (b,c) Effect of various parameters on Blue Gene/Q with TRAM

are part of our future work.

D. Experiments with Fine-grained All-to-all

To verify the effectiveness of TRAM in improving fine-grained communication performance, we ran tests of an all-to-all benchmark on the ALCF Intrepid Blue Gene/P system with the following partition topologies: *a*) 512 (8 x 8 x 8), *b*) 4096 (8 x 16 x 32), *c*) 16384 (16 x 32 x 32), and *d*) 32768 (32 x 32 x 32) nodes. We used a single PE per compute node in these tests. To simulate high-volume fine-grained communication, the benchmark was performed in rounds by looping over all the destinations and sending 32 bytes to each destination per round, without synchronizing between rounds. For TRAM runs we used data items of size 32 bytes, set the size of aggregation buffers to 2 KB and used a 3D virtual topology that matched the physical topology. To reduce congestion, we used a standard optimization of randomizing the order in which sends are performed for both versions of the benchmark. Results are shown in Figure 4a. Using TRAM led to speedups of between 3.5 and 6.25, indicating a clear performance advantage for fine-grained communication.

We ran further tests of this benchmark on the ALCF Vesta Blue Gene/Q system to measure how buffer size and virtual topology affect performance. Figure 4b shows the results using TRAM on 64 nodes with various aggregation buffer sizes. The same 3D virtual topology was used for all runs. The benefits of the multi-dimensional aggregation approach, which aggregates items into fewer buffers, can be seen in the test with 32 B per destination, which improved in execution time up to a buffer size of 8 KB. For most cases, performance improved with increased buffer size up to the experimentally determined saturation threshold of 32 KB, and leveled off for buffer sizes higher than that. However, in all-to-all tests of size up to 128 bytes where there was little data to aggregate, using very large buffers hurt performance. Although TRAM sends only the valid items in a partially filled buffer, per-item book-keeping data is stored separately at the head of the pre-allocated message buffer to reduce alignment padding and is hence sent even for the unfilled portion of the buffer. This overhead degraded performance in pathological cases with very large buffers and little data to send.

The Blue Gene/Q 5D network topology makes it particularly suitable for testing the effects of virtual topology. Figure 4c presents all-to-all results on a 64-node partition for a set of virtual topologies that preserve minimal routing. The difference in execution time between the best and worst topology was as high as 4.4x, demonstrating the importance of taking some care in selecting a virtual topology when using TRAM. The 3D virtual topology was best for low total payload sizes, while a 2D virtual topology was better for higher total payload size. For higher payload sizes, there was sufficient data available for aggregation when using lower dimensional virtual topologies, so that a 3D or higher topology was not worth the overhead of additional intermediate destinations. The 6D topology, matching the five dimensions of the network topology and cores per node as the sixth dimension, further reinforced this conclusion, generally performing worse than the 3D topology at this node count.

Additional benchmark performance results using TRAM are available in the CHARM++ submission to the HPC Challenge Competition [17].

E. Automatic Selection of Library Parameters

In order to automate the process of selecting TRAM configuration parameters, we employed the Performance-Analysis-Based Introspective Control System (PICS) [18], which dynamically tunes performance parameters within a search space specified by the user. Our tests of TRAM using the Control System show it to be effective in selecting aggregation buffer size and virtual topology specifications which maximize performance. Figure 5 plots the performance of runs of the all-to-all benchmark on 64 nodes of Blue Gene/Q. The Control System converged to a proper buffer size within 5 decisions and to a good virtual topology within 25 decisions.

VI. Application Results

We will now take a look at two production-level applications whose performance improved when we used TRAM to aggregate fine-grained messages.

For each of these applications, using TRAM required making relatively minor code changes to instantiate, initialize and terminate the library, process data items, and replace sends with submission of data items to TRAM. These changes amounted to an additional 141 lines of code for EpiSimdemics and 171 lines of code for ChaNGa. For each application, less than 0.5% of the application code was affected.

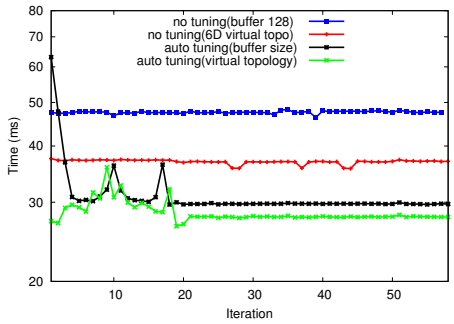


Fig. 5: Automatically tuning buffer size and virtual topology for the all-to-all benchmark on Blue Gene/Q

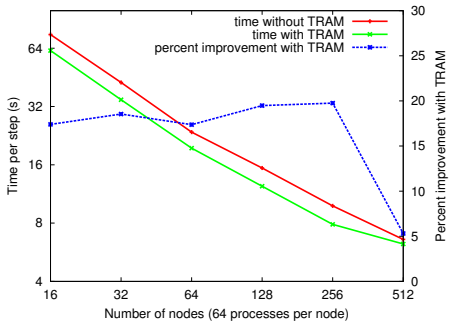


Fig. 6: ChaNGa performance on Blue Gene/Q

A. N-Body Simulations: ChaNGa

ChaNGa [19], [20] is a code for performing collisionless N-body simulations. It simulates cosmological phenomena with periodic boundary conditions in comoving coordinates or isolated stellar systems. It uses a Barnes-Hut tree to calculate gravity, with hexadecapole expansion of nodes and Ewald summation for periodic forces. The same tree is also used for neighbor finding required by the hydrodynamics component that uses the Smooth Particle Hydrodynamics (SPH) algorithm. Timestepping is done with a leapfrog integrator with individual time steps for each particle.

Gravity is a long-range force, so ChaNGa simulations involve heavy network communication. Most of this communication is not fine grained. Although individual particles are only 40 bytes in size, a hierarchy of abstractions in the code groups the particles into structures of increasing size to control the grain size for computational and communication purposes. While the messages which communicate the data are typically not fine-grained, the messages which request the data for particles or groups of particles are very fine-grained, having just 16 bytes of payload data. These messages, typically representing 30 - 50% of the total number of messages in a run, are a good match for aggregation using TRAM.

Figure 6 shows average iteration time with and without TRAM for a 50 million particle simulation of a dwarf galaxy formation, using 10-iteration runs on Vesta with 64 processes per node. We found that using TRAM to aggregate the request messages improved execution time for the gravity-calculation phase of the application by 20 - 25%, leading to a 15 - 20% overall performance improvement for most node counts. At 512 nodes with this dataset, the application reached its scaling limit, where the portion of the time spent in the gravity phase was smaller, and the impact of using TRAM less noticeable as a result. For the TRAM runs, we used a 3D virtual topology and a buffering memory footprint of just 4096 data items of size 16 bytes, reinforcing the idea that a relatively small aggregation buffer space provides a large performance improvement.

B. Contagion Simulations: EpiSimdemics

EpiSimdemics [21], [22] is an agent-based application which simulates the spread of contagion over extremely large interaction networks. Contagion refers to transmitted phenomena such as diseases, opinions, malware propagation in computer networks, spread of social movements, etc. Here, we consider the spread of a disease over the contact network of a population. Person agents send visit messages to locations they plan to visit. When a location receives all the visit messages, interactions between spatially and temporally colocated people are computed and infection messages are sent to newly infected agents. Once an agent receives all the infection messages destined for it, its health state is updated.

For this simulation, the primary communication pattern is the person to location communication. In the CHARM++ implementation, there are two types of object arrays called *LocationManager* and *PersonManager* to handle collections of location and person objects. The PersonManager to LocationManager communication can be expressed as a bipartite graph. Since PersonManager contains many persons and each person can visit many locations, this communication pattern is many to many. In a typical simulation scenario, visit messages have size 36 bytes and account for more than 99% of the total communication volume.

Manual buffering vs. TRAM The initial implementation of EpiSimdemics used *manual application-specific buffering* for reducing communication overhead. This manual buffering operates at the object level, and combines the messages destined to the same LocationManager object. There is one fixed-size buffer for each pair of PersonManager-LocationManager objects, making the total number of buffers $M * N$ if there are M PersonManager objects and N LocationManager objects. The buffer size parameter is controlled through the configuration file. The application aware nature of manual buffering allows some optimizations. Since all the messages originating from an object are combined and sent to the same destination object, some fields, such as the index of the source object, need to be sent only once. In our case, this resulted in an extra 12 bytes (hence 48 bytes total) overhead per data item when using TRAM compared to application-specific buffering.

However, TRAM provides multiple benefits compared to application-specific buffering: (1) it saves programming effort since it is a library as opposed to application-specific implementations; (2) it operates at processor level, allowing it to perform more aggregation than the object-level manual buffering; (3) it is topology-aware; (4) it generally uses less

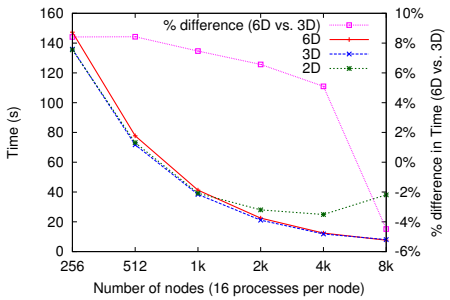


Fig. 7: Effect of topology on EpiSimdemics performance for CA dataset using TRAM

#Nodes	Estimated CPU Overhead (s) No Aggr	Estimated CPU Overhead (s) TRAM	Estimated Communication (s) No Aggr	Estimated Communication (s) TRAM	Observed TRAM Improvement (s)	Observed TRAM Final Time (s)
8	6900	646	6900	122	8470	4220
16	3450	324	3610	67.3	4200	2120
32	1720	161	1700	35.9	1960	1080
64	862	80.8	954	19.5	974	542
128	431	40.4	470	11.6	478	283
256	215	20.2	155	9.14	236	147
512	108	10.1	73.1	4.71	113	77.7
1024	53.9	5.06	42.6	2.52	49.0	41.3
2048	26.9	2.52	23.3	1.51	25.6	22.5
4096	13.5	1.26	12.6	.924	14.8	12.4
8192	6.73	.630	9.88	.462	7.13	7.69

TABLE III: Estimated overhead due to message processing and communication for EpiSimdemics on Blue Gene/Q (CA dataset) when not using aggregation, compared to the observed improvement in execution time from using TRAM

memory for buffering. Manual buffering allocates a fixed buffer for each object pair, while TRAM can share the buffer space over multiple source and destination objects. The maximum memory required per process for manual buffering in our simulations is $(36 \times PerChareBufferSize \times P)$ bytes whereas for TRAM it is $(48 \times PerProcessBufferSize)$. Hence, per process memory requirement for manual buffering grows as θ(P), making it unscalable.

Results We evaluated the performance using TRAM on the Vulcan Blue Gene/Q system. EpiSimdemics was run using the CHARM++ PAMI BlueGeneQ SMP machine layer [23]. For TRAM, we experimented with various topologies – 6D, 3D and 2D. Figure 7 shows the execution time for a contagion simulation on population data of the state of California for various node counts using a buffer size of 64 items. At medium scale, 3D topology marginally outperformed 6D, whereas for 8K nodes, 6D was marginally better.

Figure 8 compares the application speedup for three aggregation scenarios: TRAM using 6D topology, manual buffering, and direct sending without aggregation, for a contagion simulation over the populations of three different states. A buffer size of 64 items was used for TRAM and manual buffering. The speedup is calculated with respect to a base run. The base run refers to sequential execution, provided that the memory footprint of the application fit in a single node. Otherwise, base execution time was approximated by running on 2 or 4 nodes and assuming ideal speedup up to that point. For small scale (1–128 nodes), TRAM performs almost as well as manual buffering and up to 4x better than without aggregation. At larger scale (beyond 256 nodes), TRAM outperforms manual buffering. For CA dataset, at 8K nodes the execution time using TRAM was 7.7s compared to 17.2s using manual buffering.

Understanding TRAM Performance In section IV, we identified bandwidth utilization improvement and communication overhead reduction as the two main mechanisms for performance improvement from using TRAM. Using our performance model along with experimentally determined performance parameters, we can separately approximate the benefit due to each of the two mechanisms. For a given dataset, the number of visit messages in our simulations was constant. By multiplying this number by an experimentally determined value for the constant per message runtime system overhead incurred at the sender and destination (about 5 μs each), we obtained an upper bound for the communication overhead incurred in an EpiSimdemics simulation. Likewise, we approximated network communication time using the total number of bytes sent on the network for a given simulation and an experimentally determined bandwidth utilization for an all-to-all communication pattern for each node count. We also approximated the corresponding network and processing time when using TRAM based on the model from section IV.

Table III compares the aggregate time spent for message processing and network communication when not using aggregation and when using TRAM to aggregate. For reference, the total observed improvement in execution time from using TRAM is also presented. Results suggest that the overall improvement from TRAM is due to a combination of reduction in CPU overhead and improved bandwidth utilization. As expected, the processing and network time largely overlap.

VII. Conclusion

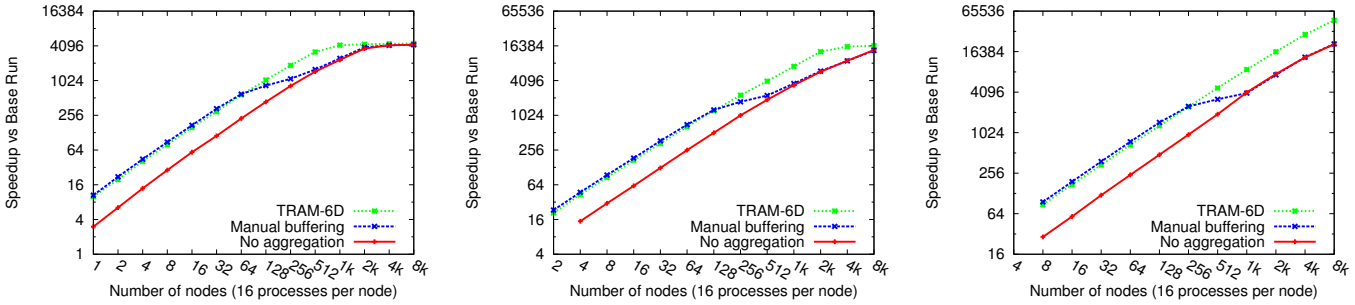
We have presented and analyzed TRAM, a streaming library for optimization of fine-grained communication using aggregation and routing of messages over virtual mesh topologies. Our experiments have shown TRAM to be an effective tool in improving performance in benchmarks and scientific applications (by up to 4x), while our analysis identified the key performance trade-offs of the approach. We also showed that parameter selection for the library can be automatically optimized. Our future plans are to investigate alternative virtual topologies and aggregation techniques.

VIII. Acknowledgments

The authors would like to thank Gengbin Zheng and Eric Bohm for their helpful suggestions.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number OCI 07-25070) and the state of



(a) AR (small dataset) (b) MI (medium-sized dataset) (c) CA (large dataset)

Fig. 8: Scaling EpiSimdemics up to 8k nodes (128k cores) on Blue Gene/Q using message aggregation

Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

This work is also part of the "Collaborative Research: Simulation of Contagion on Very Large Social Networks with Blue Waters" PRAC allocation supported by the National Science Foundation (OCI 0832603).

REFERENCES

[1] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems," 2008.

[2] R. Thakur and A. Choudhary, "All-to-all communication on meshes with wormhole routing," in *Parallel Processing Symposium, 1994. Proceedings., Eighth International*, 1994, pp. 561–565.

[3] L. V. Kale, S. Kumar, and K. Vardarajan, "A Framework for Collective Personalized Communication," in *Proceedings of IPDPS'03*, Nice, France, April 2003.

[4] S. Kumar, "Optimizing communication for massively parallel processing," Ph.D. dissertation, University of Illinois at Urbana-Champaign, May 2005.

[5] J. J. Willcock, T. Hoefler, N. G. Edmonds, and A. Lumsdaine, "Active pebbles: parallel programming for data-driven applications," in *Proceedings of the international conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 235–244.

[6] R. Garg and Y. Sabharwal, "Software routing and aggregation of messages to optimize the performance of hpcc randomaccess benchmark," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[7] S. Kumar, Y. Sabharwal, R. Garg, and P. Heidelberger, "Optimization of all-to-all communication on the blue gene/l supercomputer," in *Proceedings of the 2008 37th International Conference on Parallel Processing*, ser. ICPP '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 320–329.

[8] W. Yu, V. Tipparaju, X. Que, and J. Vetter, "Virtual topologies for scalable resource management and contention attenuation in a global address space model on the cray xt5," in *Parallel Processing (ICPP), 2011 International Conference on*, 2011, pp. 235–244.

[9] M. Koop, T. Jones, and D. Panda, "Reducing connection memory requirements of mpi for infiniband clusters: A message coalescing approach," in *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, 2007, pp. 495–504.

[10] P. Sack and W. Gropp, "Faster topology-aware collective algorithms through non-minimal communication," in *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '12. New York, NY, USA: ACM, 2012, pp. 45–54.

[11] A. Bhatele, E. Bohm, and L. V. Kale, "Optimizing communication for charm++ applications by reducing network contention," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 211–222, 2011.

[12] P. Balaji, H. Naik, and N. Desai, "Understanding network saturation behavior on large-scale blue gene/p systems," in *Proceedings of the 2009 15th International Conference on Parallel and Distributed Systems*, ser. ICPADS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 586–593.

[13] C. Iancu, P. Husbands, and W. Chen, "Message strip-mining heuristics for high speed networks," in *Proceedings of the 6th international conference on High Performance Computing for Computational Science*, ser. VECPAR'04. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 424–437.

[14] T. Hoefler and T. Schneider, "Optimization principles for collective neighborhood communications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 98:1–98:10.

[15] L. Wesolowski, *TRAM*, Parallel Programming Laboratory, Department of Computer Science, University of Illinois, Urbana, IL, http://charm.cs.illinois.edu/manuals/html/libraries/manual.html.

[16] L. Kalé and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," in *Proceedings of OOPSLA'93*, A. Paepcke, Ed. ACM Press, September 1993, pp. 91–108.

[17] L. Kale, A. Arya, N. Jain, A. Langer, J. Lifflander, H. Menon, X. Ni, Y. Sun, E. Totoni, R. Venkataraman, and L. Wesolowski, "Migratable Objects + Active Messages + Adaptive Runtime = Productivity + Performance A Submission to 2012 HPC Class II Challenge," Parallel Programming Laboratory, Tech. Rep. 12-47, November 2012.

[18] Y. Sun, J. Lifflander, and L. V. Kale, "PICS: A Performance-Analysis-Based Introspective Control System to Steer Parallel Applications," in *ACM Proceedings of 4th International Workshop on Runtime and Operating Systems for Supercomputers ROSS 2014*, Munich, Germany, June 2014.

[19] F. Gioachin, A. Sharma, S. Chakravorty, C. Mendes, L. V. Kale, and T. R. Quinn, "Scalable cosmology simulations on parallel machines," in *VECPAR 2006, LNCS 4395, pp. 476-489*, 2007.

[20] P. Jetley, L. Wesolowski, F. Gioachin, L. V. Kalé, and T. R. Quinn, "Scaling hierarchical n-body simulations on gpu clusters," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010.

[21] C. Barrett, K. Bisset, S. Eubank, X. Feng, and M. Marathe, "Episimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks," in *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC)*. IEEE Press, 2008.

[22] J.-S. Yeom, A. Bhatele, K. R. Bisset, E. Bohm, A. Gupta, L. V. Kale, M. Marathe, D. S. Nikolopoulos, M. Schulz, and L. Wesolowski, "Overcoming the scalability challenges of epidemic simulations on blue waters," in *Proceedings of the IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '14. IEEE Computer Society, May 2014.

[23] S. Kumar, Y. Sun, and L. V. Kale, "Acceleration of an Asynchronous Message Driven Programming Paradigm on IBM Blue Gene/Q," in *Proceedings of 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Boston, USA, May 2013.