Scalable Interconnection Network Models for Rapid Performance Prediction of HPC Applications

Kishwar Ahmed, Jason Liu
Florida International University
Email: {kahme006,liux}@cis.fiu.edu

Stephan Eidenbenz, Joe Zerr
Los Alamos National Laboratory
Email: {eidenben,rzerr}@lanl.gov

***Abstract*—Performance Prediction Toolkit (PPT) is a simulator mainly developed at Los Alamos National Laboratory to facilitate rapid and accurate performance prediction of large-scale scientific applications on existing and future HPC architectures. In this paper, we present three interconnect models for performance prediction of large-scale HPC applications. They are based on interconnect topologies widely used in HPC systems: torus, dragonfly, and fat-tree. We conduct extensive validation tests of our interconnect models, in particular, using configurations of existing HPC systems. Results show that our models provide good accuracy for predicting the network behavior. We also present a performance study of a parallel computational physics application to show that our model can accurately predict the parallel behavior of large-scale applications.**

***Index Terms*—Modeling and simulation; High-performance computing; Interconnection network; Performance evaluation**

I. Introduction

Recent years have witnessed dramatic changes in High-performance Computing (HPC) to accommodate the increasing computational demand of scientific applications. New architectural changes, including the rapid growth of multi-core and many-core systems, deeper memory hierarchies, complex interconnection fabrics that facilitate more efficient data movement for massive-scale scientific applications, have complicated the design and implementation of the HPC applications. Translating architectural advances to application performance improvement may involve delicate changes to sophisticated algorithms, to include new programming structures, different data layouts, more efficient buffer management and cache-effective methods, and alternative parallel strategies, which typically require highly skilled software architects and domain scientists.

Modeling and simulation plays a significant role, in identifying performance issues, evaluating design choices, performing parameter tuning, and answering what-if questions. It is thus not surprising that there exists today a large body of literature in HPC modeling and simulation, ranging from coarse-level models of full-scale systems, to cycle-accurate simulations of individual components (such as processors, cache, memory, networks, and I/O systems), to analytical approaches. We note, however, that none of the existing methods is capable of modeling a full-scale HPC architecture running large scientific applications in detail.

To do so would be both unrealistic and unnecessary. Today's supercomputers are rapidly approaching exascale. Modeling and simulation needs to address important questions related to the performance of parallel applications on existing and future HPC systems at similar scale. Although a cycle-accurate model may render good fidelity for a specific component of the system (such as a multi-core processor) and a specific time scale (such as within a microsecond), the model cannot be naturally extended to handle arbitrarily larger systems or longer time durations. Partially this is due to the computational complexity of the models (both spatial and temporal). More importantly, no existing models are known capable of capturing the entire system's dynamics in detail. HPC applications are written in specific programming languages; they interact with other software modules, libraries and operating systems, which in turn interact with underlying resources for processing, data access, and I/O. Any uncertainties involved with the aforementioned hardware and software components (e.g., a compiler-specific library) can introduce significant modeling errors, which may undermine the fidelity achieved by the cycle-accurate models for each specific component.

George Box, a statistician, once said: *"All models are wrong but some are useful."* In order to support full-system simulation, we must raise the level of modeling abstractions. Conceptually, we can adopt an approach, called *"selective refinement codesign modeling"*, where we begin with both architecture and application models at coarse level, gradually refine the models with potential performance bottlenecks, and eventually stop at models sufficient to answer the specific research questions. This iterative process is based on the assumption that we can identify performance issues from the models in a timely manner. To do so, we need to develop methods that facilitate rapid and yet accurate assessment and performance prediction of large-scale scientific applications on current and future HPC architectures.

We set out to design and develop a simulator, called the *Performance Prediction Toolkit (PPT)*. Four major aspects distinguish our effort from other existing approaches. First, our simulator needs to easily integrate large-scale applications (especially, computational physics code) with full-scale architecture models (processors, memory/cache, interconnect, and so on). Second, our simulator must be able to combine selected models of various components, potentially at different levels of modeling abstraction, providing a trade-off between the computational demand of the simulator and the accuracy of the models. Third, the simulator needs to adopt a minimalistic approach in order to achieve a short development cycle. It

is important that new models can be easily incorporated in the simulator; the simulator needs to keep up with the fast refresh rate of HPC systems. Last, the simulator must be able to achieve scalability and high performance; it needs to be capable of handling extremely large-scale models, e.g., using advanced parallel discrete-event simulation techniques.

In this paper, we focus on PPT's interconnection network models. There have already been many interconnection network models in the literature (for example, [1], [2], [3], [4], [5]); some of them have also shown capable of achieving simulation of large systems with both accuracy and efficiency. PPT's interconnection network models are different in several aspects.

First, PPT's interconnection network models include widely used interconnect topologies with emphasis on production networks (both existing and planned interconnection networks). In today's top-ranked HPC systems, we see three common network types: torus (e.g., Cray's Gemini and IBM's Blue Gene/Q), dragonfly (Cray's Aries), and fat-tree (Infiniband). They constitute a majority of the production network topologies. Our survey on the latest supercomputers (http://www.top500.org, June 2016) shows that the three topologies account for 54% of the 500 fastest supercomputers in the world (44% for fast ethernet and 2% proprietary). Among the top 100 supercomputers, the three topologies grow up to 82%. 14 of the top 15 ranked supercomputers are interconnected by the three types. In PPT, separate interconnection network models have been developed and carefully parameterized in PPT to capture various production interconnection networks. Previously, we presented a sufficiently detailed interconnect model for Cray's Gemini 3D torus network [6]. In this paper, we add the other two interconnection network models.

Second, PPT's interconnection network models are packet-level models, where network transactions (e.g., for MPI send/receive and for collective operations) are modeled as discrete events representing individual packets (typically, around 64 bytes in size) being transferred by the network switches and compute nodes. This is a conscious design decision. Our hypothesis is that in most scenarios, packet-level simulation should be sufficient to capture major network behaviors (throughput, delay, loss, and network congestion) with sufficient accuracy, and as such, should be able to identify potential performance bottlenecks at the interconnection networks while running large-scale scientific applications. Compared to more detailed models, such as those implemented at the phit level (virtual channels), packet-level simulation can easily outperform detailed models by several orders of magnitude. Our preliminary experiments (discussed more later) suggest that our packet-level models can provide sufficient accuracy.

Last, PPT's interconnection network models can be easily incorporated with the application models. Our interconnection network models interface with the Message Passing Interface (MPI) model. MPI is the most commonly used parallel programming tools for scientific applications on modern HPC platforms. Our MPI model provides convenient methods for deploying the parallel applications and performing communications on the target parallel platform. We have implemented all common MPI functions, including point-to-point communications (both blocking and asynchronous methods) and collective operations (such as gather/scatter, barriers, broadcast, reduce, and all-to-all). In addition, we implemented MPI groups and communicators so that collective communications can take place among an arbitrary subset of processes. As a result, most scientific applications can be simulated directly using the communication functions provided by the MPI model.

The rest of this paper is organized as follows. Section II describes related work and compares our approach with the existing methods. Section III describes the overall design and implementation of our Performance Prediction Toolkit, in particular, focusing on the interconnection network models. We provide the details of our torus, dragonfly and fat-tree interconnection network models (along with validations) in Section IV, Section V and Section VI, respectively. Section VII presents a trace-based simulation study to demonstrate the capability of our model for incorporating realistic applications. Section VIII describes a performance study of a parallel application (computational physics) using our interconnection network model and shows that our model can accurately predict the strong-scaling trends of the application. Finally, we conclude the paper and outline future work in Section IX.

II. Related Work

Many HPC simulators exist. Here we focus on those that provide interconnection network models. Some of these simulators aim at full-system simulation, where parallel applications are simulated to their behavior on the target architecture. BigSim [1] falls into this category. BigSim is built on Charm++ for scalable performance, which is an object-based and message-driven parallel programming system [7]. The interconnection network model implemented in BigSim, however, is relatively simple. For example, it does not consider network congestion in detail [8].

To study the performance of large-scale MPI applications, $\mu\pi$ is an MPI simulator based on an efficient conservatively-synchronized parallel simulator that features a process-oriented world-view [9]. Experiments show that the simulator is capable of simulating hundreds of millions of MPI ranks running on parallel machines. However, $\mu\pi$ does not have any reasonably detailed interconnection network model. The same can be said about the Extreme-scale Simulator (xSim) developed at the Oak Ridge National Laboratory [2]. xSim supports execution of millions of virtual MPI ranks running a very simple MPI program, using a lightweight parallel discrete-event simulation (PDES) engine.

The CODES simulator [10] is a comprehensive simulation platform that can model various large-scale HPC systems, including storage systems, interconnection networks, HPC and data center applications. CODES is built on ROSS, a parallel discrete-event simulation engine using reverse computation [11]. CODES provides detailed models for various interconnect topologies, including torus [4], dragonfly [5], and fat-tree [3]. CODES has shown capable of simulating large-scale interconnect configurations (with millions of nodes).

Although CODES is complementary to our work, there are three major differences. First, the interconnection network models in CODES are phit-level models that can capture more detailed transactions related to virtual channels than the corresponding packet-level models in PPT. While conceptually, more detailed models may render higher simulation fidelity, the computational demand would be much higher (by as much as several orders of magnitude). As such, a performance study using CODES typically would only focus on simple operations (for example, a random send/receive pattern or one collective call) and at a much smaller time scale, while using PPT we can study more complex application behaviors with greater efficiency and flexibility. In terms of accuracy, our experiments show that PPT's interconnection models can reasonably produce performance results that match from other empirical studies.

Second, CODES does not have a full-fledged MPI model. On the contrary, the interconnection network models in PPT are fully integrated with the MPI implementation from design. In this way, one can easily model complex application behaviors in PPT.

Last, the interconnection network models in PPT are designed to reflect real implementations (e.g., Cray's Gemini, Aries, IBM Blue Gene/Q, and Infiniband). In doing so, we can study the performance of applications over various interconnection networks of real (either existing or planned) HPC systems.

III. Performance Prediction Toolkit (PPT)

In this section, we provide a brief overview of PPT and existing components of interconnection network models.

A. Overall Design

The Performance Prediction Toolkit (PPT) is designed specifically to allow rapid assessment and performance prediction of large-scale scientific applications on existing and future high-performance computing platforms. More specifically, PPT is a library of models of computational physics applications, middleware, and hardware that allows users to predict execution time by running stylized pseudo-code implementations of physics applications.

PPT models are highly parameterized for applications, middleware, and hardware models, allowing parameter scans to optimize parameter values for hardware-middleware-software pairings. PPT does not yield cycle-accurate performance metrics. Instead, the results from PPT are used to examine underlying algorithmic trends and seek bottlenecks to on-node performance and scaling on HPC platforms. The conclusions of such analysis may range from optimization of current methods to further investigation of more substantial algorithmic variations.

An application is a stylized version of the actual application that captures the loop structure of important numerical kernels. Not all elements of the code are included in a PPT application, and it does not predict numerical accuracy of an algorithm, but rather predicts the execution time of a given job instance.

Middleware models in PPT currently include only MPI. It interfaces with the PPT application models and implements the communication logic in the loop structure of the actual applications.

Hardware models exist for interconnection networks and compute nodes. PPT's interconnection models are fully integrated with the MPI model. The interconnection models implement different network topologies and can be set up with different configurations. The library consists of configurations for various common production interconnection networks.

The node models in PPT use hardware parameters clock-speed, cache-level access times, memory bandwidth, etc. Application processes can advance simulated execution time by calling a compute-function with a task list as input, which consists of a set of commands to be executed by the hardware, including, for example, the number of integer operations, the number of floating-point operations, the number of memory accesses, etc. The hardware model uses this information to predict the execution time for retrieving data from memory, performing ALU operations, and storing results.

B. Simian

PPT is developed based on Simian, which is an open-source, process-oriented parallel discrete-event simulation (PDES) engine [12]. Simian has two independent implementations written in interpreted languages, Python and Lua, respectively.

Simian has several unique features. First, Simian adopts a minimalistic design. At its core, Simian consists of only approximately 500 lines of code. It thus requires low effort to understand the code and it is easy for model development and debugging. Second, Simian features a very simplistic application programming interface (API). The simulator consists of only three main modules and a handful of methods. To maximize portability, Simian requires minimal dependency on third-party libraries. Third, Simian takes advantage of just-in-time (JIT) compilation for interpreted languages. For certain models, Simian has demonstrated capable of outperforming C/C++-based simulation engine.

Simian supports process-oriented world view. It uses lightweight threads to implement the simulation processes—greenlets in Python and coroutines in Lua. In PPT, each parallel instance of an application is naturally implemented as a process. Thus, the processes can be blocked for sending and receiving messages using MPI.

C. MPI model

The Message Passing Interface (MPI) model is commonly used by parallel applications. We have previously implemented MPI model, which performs communication between compute nodes over the underlying interconnection network [6].

We have implemented all common communication functions in MPI. These functions can be divided into three groups. The first group consists of functions for point-to-point communications, which include: `MPI_Send` (blocking send), `MPI_Recv` (blocking receive), `MPI_Sendrecv`, `MPI_Isend` (nonblocking

send), MPI_Irecv (nonblocking receive), MPI_Wait, and MPI_Waitall. The second group are collective operations, which include: MPI_Reduce (reduction), MPI_Allreduce, MPI_Bcast (broadcast), MPI_Barrier, MPI_Gather (gathering data), MPI_Allgather, MPI_Scatter (scattering data), MPI_Alltoall (pairwise communications), and MPI_Alltoallv. The third group of functions deal with groups and communicators. They include: MPI_Comm_split (create new sub-communicators), MPI_Comm_dup, MPI_Comm_free, MPI_Comm_group (new group), MPI_Group_size, MPI_Group_rank, MPI_Group_incl (new subgroups), MPI_Group_excl, MPI_Group_free, MPI_Cart_create (new cartesian communicator), MPI_Cart_coords, MPI_Cart_rank and MPI_Cart_shift.

The underlying interconnection network determines the details of the MPI implementation. For example, Cray's XC series network uses Aries dragonfly interconnect [13]. MPI uses Fast Memory Access (FMA) for message passing. Messaging are performed as either GET or PUT operations (depending on the size of the message). A PUT operation initiates data flow from the source to the target node. When a packet reaches destination, a response from the destination is returned to the source. FMA allows a maximum of 64 bytes of data transfer for each network transaction—larger messages must be broken down into individual 64-byte transactions. A PUT message consists of a 14-phit request packet (i.e., 42 bytes, where each phit is 24 bits). Each request packet is followed by a 1-phit response packet (3 bytes) from destination to source. A GET transaction consists of a 3-phit request packet (9 bytes), followed by a 12-phit response packet (36 bytes) with 64 bytes of data.

Packets are handled individually in the interconnection network, as they follow their routes visiting individual network switches in sequence from the source node to the destination node. In Simian, we use separate processes inside the compute nodes and switch nodes for handling packet buffering and forwarding. A reliable data transfer scheme (with acknowledgment and retransmission) is implemented at the compute nodes.

IV. Torus Model

In this section, we present implementation of torus-based interconnects. First, we briefly outline the 3-D torus-based Gemini interconnect. Next, we describe our implementation of 5-D torus-based Blue Gene/Q interconnect architecture, along with a validation of our implementation of Blue Gene/Q interconnect.

A. Cray's Gemini Interconnect

Previously, we designed and implemented a detailed model for the Gemini interconnection network [6]. Gemini is a part of the Cray's XE6 architecture, where each compute node contains two processors with its own memory and a communication interface. The switch nodes each connect with two compute nodes; they are interconnected as a 3-D torus. Each switch node gives ten torus connection: two connections per direction in the "X" and "Z" dimension and one connection per direction in the "Y" direction. To model potential congestions in the network, we implemented a detailed queuing model to capture the interactions of network transactions (packet send, receive, and buffering). Our interconnection network model supports multiple routing algorithms, such as deterministic, hashed, and adaptive, which can be selected during configuration.

We conducted extensive validation study of the Gemini interconnection network model [6]. We measured the model-predicted MPI performance (bandwidth and latency for point-to-point and collective communications) and compared the results with published results in the literature. We also conducted a trace-driven simulation of communication calls from real-life scientific applications. Results have all shown that our Gemini interconnect model provides good accuracy.

B. Generic Torus Model

We extended the Gemini model and developed a generic model for the torus topologies for all dimensions (i.e., 5-D torus, 6-D torus etc.). In this paper, we specifically focus on the interconnect for the Blue Gene/Q architecture, which is a 5-D torus topology.

1) Blue Gene/Q Interconnect: IBM's Blue Gene/Q is a system currently used by many large-scale high-performance systems (e.g., Sequoia at Lawrence Livermore National Laboratory (LLNL), Mira at Argonne National Laboratory (ANL), Vulcan at LLNL). Blue Gene/Q is a 5-D torus-based interconnect architecture, where each switch node connects to ten neighboring switches (two in each direction in five dimensions). The network is optimized for both point-to-point and collective MPI communications [14]. The message unit (MU) in Blue Gene/Q supports both direct PUT and remote GET, where messages are packetized for transmission [15]. Data portion of packets increments in chunks of 32 bytes, up to 512 bytes [15]. Packets contain 32-byte header: 12 bytes for the network and 20 bytes for the MU.

2) Validation: Now we present a validation of our 5-D torus-based Blue Gene/Q interconnect model. We considered a real HPC system, IBM Sequoia supercomputer, that deploys Blue Gene/Q interconnect architecture. IBM Sequoia was built by IBM and is maintained by Lawrence Livermore National Laboratory (LLNL). Sequoia consists of 96 racks containing 98,304 compute nodes connected via the 5-D torus topology of dimensions $16 \times 12 \times 16 \times 16 \times 2$ [16]. The bandwidth along the links is 2 GB/s [15]. The link delay is set to be 40 ns [16]. We measured the end-to-end latency between two end nodes for a Blue Gene/Q system (to compare with the latency values reported in [15]). The end-to-end latency is measured by the propagation delay and the number of hops between the two end nodes. In this latency test, an MPI process sent an 8-byte data to all other MPI processes (mapped on different compute nodes). The measured delays are between 700 ns and 1300 ns. The results are consistent with the data reported in [15] (where the Blue Gene/Q system end-to-end latency is reported to be between 718 ns and 1264 ns).

V. Dragonfly Model

In this section, we first describe our dragonfly interconnect model and then focus specifically on Cray's Aries interconnect that has been applied in many real HPC systems. We conducted validation experiments of our Aries interconnection network model, the results of which are presented at the end of this section.

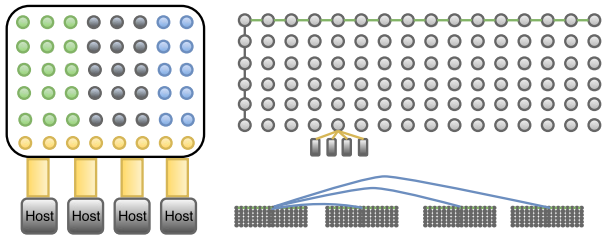
A. Dragonfly Topology

Dragonfly topology was first proposed in [17]. It is a cost-efficient topology, which reduces network cost through exploiting the economical, optical signaling technologies and high-radix (virtual) routers. Dragonfly topology has a three-tier network architecture [17]. At first level, each router is connected to p nodes, usually through backplane printed circuit boards (PCBs) links. At second level, a group is formed through connecting a routers to each other. The local connections used in this level are referred to as *intra-group* connections. These connections are typically built using short-length electrical cables. At the last level, each router has h *inter-group* connections to routers in other groups. These global connections are usually built using longer optical cables. The maximum network size of such dragonfly topology is $ap(ah+1)$ nodes.

In this paper, we consider a dragonfly topology with *local link arrangement* to be completely-connected (i.e., a 1-D flattened butterfly). Therefore, each router has $a-1$ local connections to the other routers in the group. *Global link arrangement* specifies how switch in a group is connected to switch of the other group. Several alternatives for global link arrangements are defined and evaluated against each other in the literature (e.g., consecutive, palmtree, circulant-based) [18]. In this paper, we consider the *consecutive* arrangement of global links, which was also considered in the paper where dragonfly was initially proposed [17], [18]. The consecutive arrangement connects routers in each group consecutively, where groups are also numbered consecutively.

In our dragonfly implementation, we support two types of routing: minimal (MIN) and non-minimal (VAL). MIN routing is ideal for benign traffic patterns (e.g., uniform random traffic). Since we consider completely-connected local channels, any packet can reach destination in at most three hops: one hop within the source group to reach the switch with global connection to the destination group, one hop to traverse the global link and one hop within the destination group to reach the destination node. VAL routing, on the other hand, is suitable for adversarial traffic patterns. Following the principal of this algorithm, we route a packet to a randomly chosen intermediate group first and then route to the final destination. As a result of using 1-D local channels, a packet generally reaches destination through traversing two global channels and three local channels.

Another dragonfly routing variation is Universal Globally-Adaptive Load-balanced (UGAL) routing, which chooses between MIN and VAL on a packet-by-packet basis and sends packet to paths with least queuing delay to alleviate congestion. Since for a large-scale system, it is infeasible to know the queue information on all other queues (UGAL-G), the switching between MIN and VAL can be performed based on local queue information (UGAL-L). We do not consider adaptive routing at all in our current implementation. Our logics behind such decision are the followings: 1) We expect adaptive routing to play a minor role in the overall performance prediction of large-scale HPC applications compared to other factors; and 2) The additional overhead of message communication necessary for implementing adaptive routing would significantly impact scalability of our simulation.



(a) (b)

Fig. 1. Cray's Aries block diagram. (a) Aries ASIC, (b) Aries connections.

B. Cray's Aries Interconnect

Cray's Aries network (developed as part of Defense Advanced Research Projects Agency's or DARPA's program) uses dragonfly topology [19], [13]. Aries contains a 48-port router, four network interface controllers (NICs) connecting four nodes and a multiplexer known as Netlink. The system is built from four-node Aries blade (where each blade contains a single Application-specific Integrated Circuit or ASIC). A simplified block diagram for Aries ASIC is shown in Fig. 1(a). Aries network consists of Cascade cabinets, where each pair of cabinets can house up to 384 nodes. There are three chassis per cabinet, each chassis contains 16 Aries blades. Two such cabinets construct a group (i.e., each group in Aries contains six chassis or 96 Aries blades).

We implemented three types of connections in Aries interconnect. The *first dimension*, which is also termed as green dimension, connects each blade in a chassis to the other 15 blades in the same chassis through chassis backplane. The *second dimension* is known as black dimension. Each Aries blade has connections to 5 other peer nodes of the other chassis in the group through electrical cables. The *third dimension* consists of blue links. Each Aries blade contains 10 global links; each group contains 96 blades. Therefore, one could possibly support 960 groups in an Aries system. However, following an actual Aries system, we grouped four global links into one, and the maximum allowed system size of our designed Aries interconnect is 241 groups. We connected the groups using the consecutive arrangement scheme. The connections are shown as an example in Fig. 1(b).

Both MIN and VAL routings are supported in Aries architecture [20]; we implemented both in our model. There are two cases. For *intra-group routing,* MIN routing requires at most two hops, while VAL routing selects a random switch inside the group and thus requires up to four hops. For *inter-group routing,* MIN routing requires at most five hops to reach destination

(two local link traversals each for source and destination groups, and one global link traversal), where VAL routing selects a random intermediate group. For the latter, a packet needs to be routed to a random intermediate switch in each of the source group, destination group and intermediate group, thus requiring at most fourteen hops before a packet is reaching its destination.

C. Aries Validation

As described in the previous section, Aries has 96 switches per group, 4 hosts per switch. Each switch has 48 network ports: 40 of which are used to connect the switches together and 8 are used to connect the switch to processors. We set the inter-group link bandwidth to be 4.7 GB/s per direction and intra-group link bandwidth per direction to be 5.25 GB/s [21]. The bandwidth of the interface connecting a host to its router is set to be 16 GB/s [13]. The link latency is set to be 100 ns (in a quiet network, measured router-to-router latency is reported to be 100 ns [13]).

For validation, we considered a large-scale interconnect at a recently-developed HPC system for validation of Aries. Trinity is being built at Los Alamos National Laboratory by U.S. Department of Energy (DOE). Trinity uses a Cray XC40 system that consists of 9436 nodes [22] connected via the Aries dragonfly network. We measured average end-to-end latency for considered interconnect system as a function of transfer size. Fig. 2(a) shows measured MPI latencies from our simulator. Fig. 2(a) also compares our latencies with the empirical result reported in [13]. As shown in the figure, our measured MPI latencies closely resembles the published results.

We measured the MPI throughput between two different nodes for different message sizes and compared it with the empirical values also published in [13]. The results are demonstrated in Fig. 2(b). We considered two different types of traffic for throughput comparison: pingpong and unidirectional. Fig. 2(b) shows that, in case of pingpong traffic, the simulation closely resembles the empirical results until 4K data size and after 4K data size, we can observe a slight shift. For unidirectional traffic, a good match is observed above 16K data size. Overall, the model has a good prediction of the throughput in general. There are many factors that may affect the throughput, including buffer management at both the sender and receiver, and also system overheads that may not be included in our model.

We also conducted a latency measurement for MPI collective operations. For this experiment, we used the configuration of interconnect deployed at supercomputer Darter. Darter was built by National Institute of Computational Sciences (NICS) [23]. It is a Cray XC30 system that consists of 748 compute nodes and 10 service nodes in two groups. The nodes are connected using dragonfly network topology with Cray's Aries interconnect. Fig. 2(c) shows the result of measured time for `MPI_Allreduce`, as we vary the message size. As expected, the latency of collective operation increases with increase in message size. When message size becomes much higher, delay increases due to congestion in the network. We compared our results for `MPI_Allreduce` time with the one reported at [24], for a similar configuration (i.e., the Darter supercomputer). The compared results show close correspondence to each other.

VI. Fat-tree Model

In this section, we present our fat-tree topology model design and a validation of fat-tree model based on configuration used for Infiniband interconnect architecture.

A. Fat-tree Topology

Fat-tree is one of the most widely used topologies for current HPC clusters and also the dominant topology on Infiniband (IB) technology [25]. Besides, this interconnect has also received significant attraction in data center networks [26]. Some unique properties that make fat-tree much popular among HPC and data center networks are: deadlock avoidance without use of virtual channels, easier network fault-tolerance, full bisection bandwidth, etc.

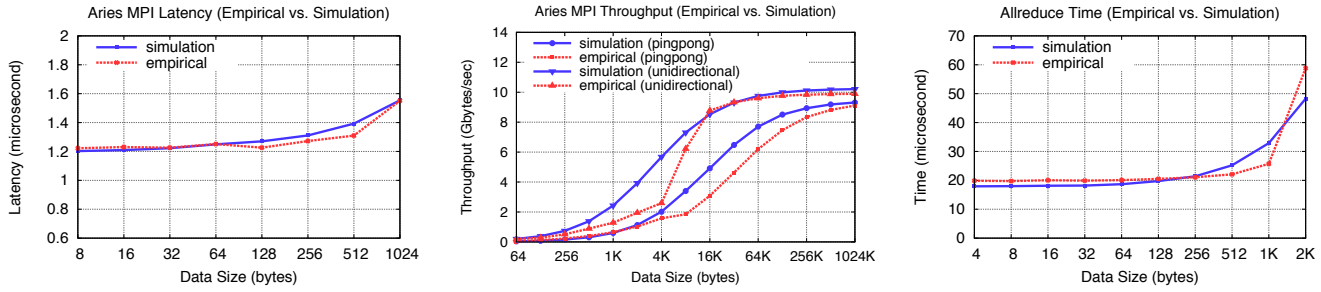
Since its first introduction, many variations of fat-tree have been proposed in the literature (e.g., [27], [28]). Among them, m-port n-tree fat-tree [27] and k-ary n-tree [28] are the most popular ones. In this work, we implement the m-port n-tree variations due to its wide popularity compared to other existing fat-tree variations [29]. As such, it has been considered in recent literature as the fat-tree topology variation for large-scale systems [3].

An m-port n-tree is a fixed-arity fat-tree consisting of $2(m/2)^n$ processing nodes and $(2n-1)(m/2)^{n-1}$ m-port switches [27]. The height of the tree is $n+1$. Each of the switches in an m-port n-tree have an unique identifier based on the level and value of m and n. The processing nodes are the nodes in the leaf and are also denoted uniquely. We used notation scheme outlined in [27] to denote both switches and processing nodes. We connected the switches to each other in both upward and downward directions and also to processing nodes based on the conditions specified in [27].

We implemented routing in the fat-tree network as two separate phases: upward phase and downward phase. In upward phase, the packet is forwarded from a source towards the direction of one of the root switches. In downward phase, the packet is forwarded downwards towards one of the leaf nodes as the destination. Transition between these two phases takes place at the lowest common ancestor (LCA) switch. The LCA switch can reach both the source and destination using downward ports of that switch. Many efforts exist to improve the routing performance in fat-tree network (e.g., Valiant algorithm [30], ECMP [31], Multiple Local Identifier (MLID) routing scheme [27]). We implemented the MLID routing scheme for Infiniband network, as presented in [27]. MLID routing scheme relieves the link congestion through exploiting multiple paths available in fat-tree topology.

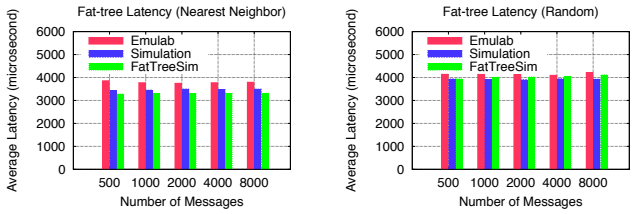
B. Fat-tree Validation

Now we present validation of our m-port n-tree fat-tree-based Infiniband interconnect system. As an example of HPC system deploying fat-tree interconnect, we consider Stampede



(a) (b) (c)

Fig. 2. Aries validation. (a) Comparison of Aries latencies in terms of message size, (b) Comparison of Aries MPI throughput in terms of message size, (c) Comparison of MPI Allreduce time.



(a) (b)

Fig. 3. Comparison with FatTreeSim and Emulab. (a) Nearest neighbor traffic, (b) Random destination traffic.

supercomputer specifications. Stampede is built by National Science Foundation (NSF) at the Texas Advanced Computing Center (TACC), U.S. Stampede consists of 6,400 nodes connected via fat-tree-based Infiniband FDR network [32]. The FDR Infiniband consists of 56 Gb/s Mellanox switches [32] and we use this configuration in both uplink and downlink bandwidth of our fat-tree interconnect model. We assign 0.7 μs as uplink and downlink latency for our considered fat-tree interconnect [33].

We compare our model with the output reported by a recently-proposed fat-tree simulator, FatTreeSim [3]. We also compare with Emulab (a network testbed) output reported at [3] for similar system setup. We consider a 4-port 3-tree fat-tree interconnect with total 16 processing nodes and 20 switches. We set the message size to 1,024 bytes. We consider two traffic patterns for comparison: nearest neighbor and random destination. We vary the number of messages from 500 to 8,000 for conducting similar comparison to data reported in [3]. Figs. 3(a) and 3(b) show comparison with Emulab and FatTreeSim for the nearest neighbor and random destination traffic, respectively. As evident from both figures, average latency calculated for each message in our model demonstrates close correspondence to the result from both Emulab and FatTreeSim and for both types of traffic with varying number of messages.

VII. Trace-based Simulation

We perform a brief comparison study of different topology performance based on real-life application communication traces provided by the National Energy Research Scientific Computing Center (NERSC) [34]. The traces provide parallel speedup performance and MPI communication operations after running various DOE mini-apps at large-scale communication facility. For each run of an application, a set of trace files (one for each MPI ranks) are provided. We collected and processed the trace files using the SST DUMPI toolkit [35]. The processed output contains the measured start and end time of the MPI call and parameters exclusive to the call (e.g., source or destination rank, data size, data type). We run the simulation by reading processed trace file for each rank, one entry at a time, and then calling the corresponding simulator MPI process. We run the trace for each of the interconnection network models we have presented (i.e., Aries, Fat-tree, Gemini, Blue Gene/Q). We use configurations of Trinity and Stampede interconnects to represent architectures of Aries and Fat-tree, respectively. We use interconnect configuration of Hopper (a supercomputer built by NERSC [36]) in our Gemini interconnect validation. Hopper contains 6,384 nodes connected via the Gemini interconnect at $17 \times 8 \times 24$. For Blue Gene/Q architecture, we use the interconnect configuration of Mira (a supercomputer at Argonne National Laboratory, which uses 5-D torus-based Blue Gene/Q at dimensions: $8 \times 12 \times 16 \times 16 \times 2$).

For this experiment, we collected the traces for DOE mini-app Big FFT (which solves 3-D FFT problem) on 100 processes from [34]. There are a total of 400 calls to `MPI_Alltoallv` and 500 calls to `MPI_Barrier`. The trace also contains a number of group and sub-communication MPI calls: 4000 calls to `MPI_Group_free`, 2000 calls to `MPI_Group_incl` and 2000 calls each to `MPI_Comm_create` and `MPI_Comm_group`. Running such trace-based simulation serves two purposes: 1) It demonstrates that our designed topology models are capable of supporting real-life communication applications, and 2) It provides a comparison among different interconnect topologies with respect to their effect on parallel applications.

Fig. 4 shows average number of hops traversed by each of the different topologies considered in this paper. As can be seen in the figure, Aries incurs the minimum number of hops in average, while Gemini has the maximum number of hops. Fig. 4 also shows that the simulation time differs in accordance with the number of hops: Aries incurs minimum simulation time, while Gemini takes the most simulation time, among all four types of topologies.

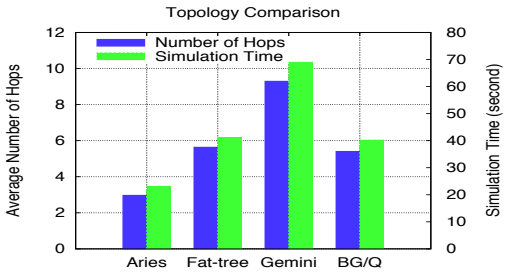


Fig. 4. Comparison of different architectures for trace-based run.

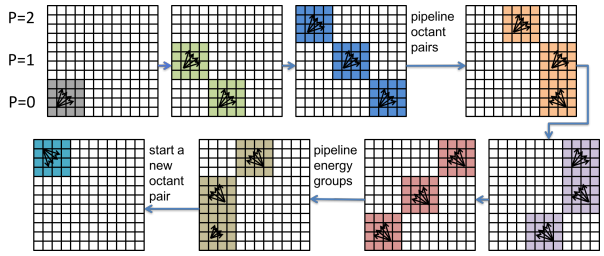


Fig. 5. A 2-D illustration of the parallel wavefront solution technique for radiation transport.

VIII. SNAP Performance Study

The SN Application Proxy (SNAP) [37] is a "mini-app" based on the production code PARTISN [38] at Los Alamos National Laboratory (LANL). PARTISN is a code for solving the radiation transport equation for neutron and gamma transport. The resulting solution is the distribution of these sub-atomic particles in space, direction of travel, particle speed, and time. These dimensions of space, direction, speed, and time form a phase space that is discretized to formulate a linear system of equations. Solving this system of equations in parallel in the most efficient manner may depend on the architecture of the supercomputer employed. Simulation capabilities provided by the PPT will allow faster exploration of the optimizations and variations necessary when considering different computing systems.

For problems of particular interest, the geometry is often modeled in three dimensions, *x*-*y*-*z*. A finite volume discretization creates a structured, Cartesian mesh of spatial mesh *cells*. In 3-D space, particle trajectory is defined by the projection of some vector of unit length to the three axes of the coordinate system. This scheme offers two degrees of freedom—i.e., projection to two axes of a unit vector forces the third projection to a unique value. For our transport solver, we apply the so-called *discrete ordinates* solution technique, whereby we compute the solution for a finite set of possible directions or "angles." Each angle is associated with a particular weight, and the solution for each angle may be performed individually. This scheme is known as "discrete ordinates," and we build the set of angles according to well-known numerical quadrature rules for integration. The two degrees of freedom manifest themselves in SNAP as a list of angles per octant in 3-D space and the eight octants themselves. Particle speed (or energy) is binned into *groups* that represent the sum of all particles in some range of energy values. Lastly, the time dimension is discretized with a finite difference solver. We have a system of equations in seven dimensions (three in space, two in angle, one in energy, and one in time).

The governing transport equation is hyperbolic in nature in the space-angle dimensions. Information flows from an upstream source to downstream destinations. The solution for any given direction in the discrete ordinates solution scheme thus requires one to add particle sources and subtract particle sinks while stepping logically through the spatial mesh cells. Colloquially, it is known as a "transport mesh sweep." A mesh sweep can further be thought in terms of a task graph. Such a graph for a mesh sweep on a structured mesh, as we have in SNAP, is known a priori and is the same for all directions. This allows scheduling and operation optimizations to be included explicitly in the code.

This system of equations offers multiple levels of parallelism which are exploited in SNAP [39]. The global spatial mesh is distributed across MPI ranks. Parallel solution over the spatial domain is achieved with a parallel wavefront solution technique specific to radiation transport, called the **KBA** method. An MPI process is assigned all spatial cells along one dimension in *x*-*y*-*z* space. This dimension itself is broken up into smaller spatial "work chunks." Mesh sweeps are done in octant-pairs— i.e., the $+/-x$-direction. A sweep begins in one octant for the sole chunk with no upstream dependence. Downstream, parallelism over space is achieved by solving for work chunks on the same diagonal plane, because such work chunks are independent. Octant pairs and energy groups are "pipelined" to extend the graph and maximize parallel efficiency. Different octant pairs are started at the completion of some previously scheduled pair. Fig. 5 shows an example of a 2-D mesh.

This solution process is performed for all angles of an octant and for all octants. The solution for each angle may be performed independently. This is achieved by aligning the data and using vector instructions to apply single-instruction-multiple-data parallelism to each operation that uses information from the angular domain. This process must be repeated for each different group. Shared memory and OpenMP threads are also employed in SNAP to parallelize the work over the energy domain, but this has not yet been modeled in the PPT.

We implemented an application model for the parallel wavefront solution technique of SNAP, called SNAPSim. SNAPSim uses similar input variables as described above to parameterize the problem to describe the size of the problem and the size of the individual tasks. Although SNAP requires iterations to formulate a solution, SNAPSim uses a fixed number of iterations, which is a sufficient abstraction for modeling the cost of instructions, data flow, and communications for an actual SNAP simulation. The sum of all work is broken up into work chunks, where each chunk represents the solution for some chunk of spatial cells, all directions captured by a single octant of a unit sphere, and all particles binned into a single energy group. The nature of the problem permits that some work chunks be performed concurrently, and other

chunks wait for these upstream tasks to be completed before progressing. The model captures this scheduling and estimates the compute time associated with a single chunk for different architectures. The time is computed per a hardware simulator that uses machine-specific details to estimate the computation time.

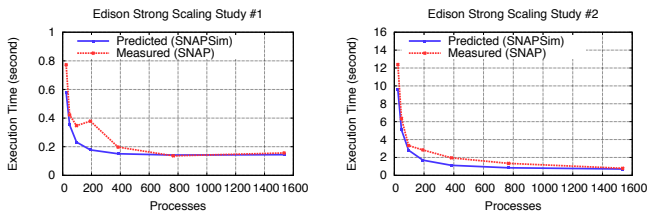
The SNAPSim model does not solve any transport equations, but it does estimate the cost to perform those operations. A function is used to compute the time necessary to handle some "tasklist." This tasklist comprises integer, floating point, and vector operations; data loads; and estimates of memory locality for costs of cache misses. SNAPSim uses two tasklists: one for the actual sweep operations and one for computing source terms that couple the solutions of different energy groups. Each tasklist is determined by the problem parameters. A SNAPSim instance is defined by the number of cells in x-y-z space, $nx \times ny \times nz$; the number of angles per octant, $nang$; the number of groups, ng; and the number of time steps, $nsteps$. The size of each chunk is also provided with the parameters $ichunk$, $jchunk$, and $kchunk$. Formulas using these parameters were prepared by counting instructions in an architecture-independent way with the Byfl application analysis code [40].

SNAPSim mimics the sweep process described above once. The functions for computing tasklist-associated times are also performed once. Simulated MPI calls using the MPI and interconnect simulators ensure the proper sweep order is maintained. When a task between MPI communications would normally be performed, SNAPSim instead simulates an MPI-sleep time equal to the pre-computed time according to the tasklist. The result is the approximate solution time for a single iteration, which we can replicate as many times as necessary. The final approximation can either be used for its own analysis or compared to actual SNAP runs for validation of the SNAPSim model, as we discuss below.

To test the SNAPSim model using the MPI and interconnect simulators provided by the PPT, we use the PPT hardware and interconnect model of NERSC's Edison supercomputer. Edison is a Cray XC30 system that uses the Aries interconnect with a dragonfly topology. Each node in Edison is composed of two sockets, each with 12 Intel Ivy Bridge cores and 32 GB of main memory.

Our interest to date has been to model parallel SNAPSim runs in a strong-scaling sense, by fixing the size of the problem and using additional resources (i.e., cores or MPI processes) to continually reduce the number of spatial cells assigned to any one core. We present the results of two strong-scaling studies that differ in the size of the problem considered in terms of the number of cells, angles, and groups.

The first problem uses a $32 \times 32 \times 48$ spatial mesh, 192 angles, and 8 groups. Chunk sizes for spatial parallelism contain 8 cells in the x-dimension and a number that ranges from four cells to one cell in both the y- and z-dimensions as the number of processes is increased. We always start from 24 cores or one compute node to focus on the effects of off-node communication on scaling. Fig. 6(a) shows that



(a) (b)

Fig. 6. SNAPSim vs. SNAP Edison strong scaling. (a) Study #1: smaller problem demonstrates scaling limitation, (b) Study #2: larger problem for continued scaling.

this problem has a low computational load at high core counts. The small computation-to-communication ratio leads to diminishing returns with increasing cores and perhaps even worse performance. While not in exact agreement, the simulator is capturing the trend from measured SNAP simulations quite well.

The second problem is larger with a $64 \times 32 \times 48$ spatial mesh, 384 angles, and 42 energy groups. Each chunk has 16 cells in the x-dimension and a number of cells in the other dimensions that varies with increasing processes. Fig. 6(b) shows how this problem maintains a much more consistent scaling trend due to the larger amount of computation between communications. And importantly, the PPT model again predicts the measured trend well.

The results show the PPT can accurately predict strong scaling trends for SNAP on modern hardware with a well-understood interconnect. Succeeding in this validation exercise permits future deployment of SNAPSim and the PPT for optimizing performance according the strong scaling properties without requiring a system allocation and extensive testing.

IX. Conclusions

In this paper, we presented interconnection network models for performance prediction of large-scale scientific applications on HPC system. Specifically, we presented interconnect models for three widely used topologies and corresponding interconnect architectures: dragonfly (Cray's Aries), fat-tree (Infiniband) and 5-D torus (IBM's Blue Gene/Q). We conducted extensive validation study of our interconnection network models, including a trace-driven simulation of real-life scientific application communication patterns. We also performed performance study of a computational physics-based parallel application using our interconnect model. All the results show that our interconnect models provide reasonably good accuracy. For future work, we plan to translate all our interconnect models to Simian Lua to study the parallel performance of our interconnect models on large-scale HPC systems.

Acknowledgments

We gratefully acknowledge the support of the U.S. Department of Energy through the LANL/LDRD Program for this work. We would also like to thank the anonymous reviewers for their comments and suggestions.

REFERENCES

[1] G. Zheng, G. Kakulapati, and L. V. Kalé, "Bigsim: A parallel simulator for performance prediction of extremely large parallel machines," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*. IEEE, 2004, p. 78.

[2] C. Engelmann and F. Lauer, "Facilitating co-design for extreme-scale systems through lightweight simulation," in *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–8.

[3] N. Liu, A. Haider, X.-H. Sun, and D. Jin, "FatTreeSim: Modeling large-scale fat-tree networks for HPC systems and data centers using parallel and discrete event simulation," in *Proceedings of the 3rd ACM Conference on SIGSIM-Principles of Advanced Discrete Simulation*. ACM, 2015, pp. 199–210.

[4] N. Liu, C. Carothers, J. Cope, P. Carns, and R. Ross, "Model and simulation of exascale communication networks," *Journal of Simulation*, vol. 6, no. 4, pp. 227–236, 2012.

[5] M. Mubarak, C. D. Carothers, R. Ross, and P. Carns, "Modeling a million-node dragonfly network using massively parallel discrete-event simulation," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*. IEEE, 2012, pp. 366–376.

[6] K. Ahmed, M. Obaida, J. Liu, S. Eidenbenz, N. Santhi, and G. Chapuis, "An integrated interconnection network model for large-scale performance prediction," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 177–187.

[7] 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.

[8] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, 2014.

[9] K. S. Perumalla, "$\mu\pi$: a scalable and transparent system for simulating MPI programs," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, p. 62.

[10] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "CODES: Enabling co-design of multilayer exascale storage architectures," in *Proceedings of the Workshop on Emerging Supercomputing Technologies*, 2011, pp. 303–312.

[11] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A high-performance, low-memory, modular Time Warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[12] N. Santhi, S. Eidenzenz, and J. Liu, "The Simian concept: parallel discrete event simulation with interpreted languages," in *Proceedings of the 2015 Winter Simulation Conference*, L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, Eds., 2015.

[13] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf, 2012.

[14] Institute for Development and Resources in Intensive Scientific Computing, "Turing, ibm blue gene/q: Hardware configuration," http://www.idris.fr/eng/turing/hw-turing-eng.html#architecture.

[15] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The ibm blue gene/q interconnection network and message unit," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 2011, pp. 1–10.

[16] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. Satterfield, B. Steinmacher-Burow, and J. Parker, "The ibm blue gene/q interconnection fabric," *IEEE Micro*, vol. 32, no. 1, pp. 32–43, 2012.

[17] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 77–88.

[18] C. Camarero, E. Vallejo, and R. Beivide, "Topological characterization of hamming and dragonfly networks and its implications on routing," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 4, p. 39, 2015.

[19] G. Faanes, A. Bataineh, D. Roweth, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, J. Reinhard *et al.*, "Cray cascade: a scalable hpc system based on a dragonfly network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 103.

[20] R. Alverson, "Cray high speed networking," in *Proceedings of the 20th Annual Symposium on High-Performance Interconnects (HOTI)*, 2012.

[21] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, and P.-T. Bremer, "Simulating and visualizing traffic on the dragonfly network," https://cug.org/proceedings/cug2016_proceedings/includes/files/pap155.pdf, 2015.

[22] N. Wichmann, C. Nuss, P. Carrier, R. Olson, S. Anderson, M. Davis, R. Baker, E. W. Draeger, S. Domino, A. Agelastos *et al.*, "Performance on trinity (a cray xc40) with acceptance-applications and benchmarks," *Memory*, vol. 2, p. 4, 2015.

[23] M. R. Fahey, R. Budiardja, L. Crosby, and S. McNally, "Deploying darter-a cray xc30 system," in *International Supercomputing Conference*. Springer, 2014, pp. 430–439.

[24] R. D. Budiardja, L. Crosby, and H. You, "Effect of rank placement on cray xc30 communication cost," in *The Cray User Group Meeting*, 2013.

[25] Mellanox Technologies, "Deploying hpc cluster with mellanox infiniband interconnect solutions," http://www.mellanox.com/related-docs/solutions/deploying-hpc-cluster-with-mellanox-infiniband-interconnect-solutions.pdf, June 2016.

[26] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[27] X.-Y. Lin, Y.-C. Chung, and T.-Y. Huang, "A multiple lid routing scheme for fat-tree-based infiniband networks," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, 2004, p. 11.

[28] F. Petrini and M. Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Parallel Processing Symposium, 1997. Proceedings., 11th International*. IEEE, 1997, pp. 87–93.

[29] B. Bogdaski, "Optimized routing for fat-tree topologies," PhD thesis. University of Oslo, Norway., 2014.

[30] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. ACM, 1981, pp. 263–277.

[31] D. Thaler, "Multipath issues in unicast and multicast next-hop selection," in *RFC 2991, Nov. 2000 [Online]. Available: http://tools. ietf. org/html/rfc2991*. Citeseer, 2000.

[32] Texas Advanced Computing Center, "Stampede user guide," https://portal.tacc.utexas.edu/user-guides/stampede.

[33] Mellanox Technologies, "Infiniband performance," http://www.mellanox.com/page/performance_infiniband.

[34] Department of Energy, "Design forward characterization of DOE miniapps," http://portal.nersc.gov/project/CAL/doe-miniapps.htm, Accessed June 15, 2016.

[35] "DUMPI: The MPI Trace Library," http://sst.sandia.gov/about_dumpi.html.

[36] National Energy Research Scientific Computing Center, "Hopper," https://www.nersc.gov/users/computational-systems/retired-systems/hopper/configuration/interconnect/.

[37] R. J. Zerr and R. S. Baker, "Snap: Sn (discrete ordinates) application proxy, version 1.01: user's manual," LANL, https://github.com/losalamos/SNAP, Accessed May 23, 2016, 2013.

[38] R. E. Alcouffe, R. S. Baker, J. D. Dahl, E. D. Fichtl, S. A. Turner, R. C. Ward, and R. J. Zerr, "Partisn: a time-dependent, parallel neutral particle transport code system," LANL, LA-UR-08-07258, last revised, December 2015, 2015.

[39] R. S. Baker, "An sn algorithm for modern architectures," in *Proceedings of ANS M&C 2015, Nashville, TN, USA, April 19–23, 2015*, 2015.

[40] S. Pakin, "Byfl: compiler-based application analysis," LANL, https://github.com/losalamos/Byfl, Accessed May 23, 2016, 2016.