A Simulation Study of Hardware Parameters for Future GPU-based HPC Platforms

Saptarshi Bhowmik*, Nikhil Jain‡, Xin Yuan*, Abhinav Bhatele†

*Department of Computer Science, Florida State University
‡NVIDIA, Inc.
†Department of Computer Science, University of Maryland, College Park
Email: bhowmik@cs.fsu.edu, nikhijain@nvidia.com, xyuan@cs.fsu.edu, bhatele@cs.umd.edu

***Abstract*—Compute nodes on high performance computing (HPC) platforms are increasingly equipped with multiple GPUs. This results in increased computational capacity per node, and reduction in the total number of nodes or endpoints in the system. This trend changes the computation and communication balance in comparison to pre-GPU era HPC platforms, which warrants a new study of hardware architectural parameters. In this work, we leverage the end-to-end system simulation capabilities of TraceR-CODES and study the impact of several hardware design parameters on the performance of realistic HPC workloads. We focus on three crucial hardware parameters: (1) number of GPUs per node, (2) network link bandwidth, and (3) network interface controller (NIC) scheduling policies, in the context of two popular network topologies – fat-tree and dragonfly.**

***Index Terms*—interconnect, GPUs per node, dragonfly, fat-tree**

I. Introduction

Large-scale systems that are expected to deliver over 1 Exaflop/s of sustained performance are being built. Unlike the systems that dominated the HPC industry a decade ago, most of the current and next-generation systems have a relatively modest number of nodes. For example, Sequoia at LLNL [1], the 2012 fastest supercomputer in the Top500 list, deployed 96K nodes to provide 20 Petaflop/s of peak performance while Summit at ORNL [2], one of the fastest supercomputers in June 2020, has only ∼4600 nodes but delivers 200 Petaflop/s of peak performance. Despite having fewer nodes, the communication performance that applications can obtain on a system continues to have a major impact on its overall performance [3].

The driving force behind the reduction in the number of nodes is compute acceleration devices such as GPUs [4]. For example, an NVIDIA Volta V100 can perform 7 Teraflop/s worth of double-precision computation in comparison to 200 Gigaflop/s for a Blue Gene/Q node. However, such a significant computing capability increase has not been matched by a similar increase in network capability. Further, the deployment of multiple acceleration devices per node has also contributed to a lower network bandwidth to flop/s ratio. Hence, even though communication requirements typically grow slower than computation requirements (e.g. as the growth of surface area versus volume), it is important to identify the best way to use the available communication capability as well the ideal computation/communication capability balance.

In this work, we use end-to-end system simulations to explore the performance impact of various hardware design and parameter choices for GPU-based systems. To drive the simulations, we instrument and trace representative applications using Score-P [5]. Information such as the amount of time spent in computation, communication patterns, and program flow is extracted from these captures. Based on these applications, realistic multi-job workloads are built and simulated using TraceR-CODES [6], [7], a scalable simulation framework for predicting application performance on HPC systems.

The key question we seek to answer is: does a system with fewer nodes each with more computing capability perform better than a system with more nodes each with lesser computing capability? To explore this space, our simulations compare the performance of the same traces of multi-job workloads on different systems that represent different compute capability per network endpoint.

Next, we evaluate the relationship between the available network bandwidth and overall application and workload performance. Specifically, we perform a sensitivity study of the overall performance with respect to a decrease or increase in the available network bandwidth. We hope that this study of variation in network bandwidth combined with the changing compute capability per node will guide us toward the optimal computation/communication capability ratio.

Finally, we explore the impact of message scheduling on performance. Multiple processes or control flows exist on the node and compete for scarce network resources. The message scheduling scheme that decides the distribution of injection and network bandwidth for each control flow can play an important role in determining the progress made in the individual control flows. At the same time, this scheme determines the skew of traffic load on the network and can affect the network communication performance. Hence, exploration of this design choice can help improve communication performance.

For all of the above studies, we have used two network topologies: fat-tree and dragonfly [8]. We chose these two topologies because of their prevalence in HPC. The fat-tree topology [9]–[11] is used in several large HPC systems

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

currently in use, e.g. Sierra at Lawrence Livermore National Laboratory (LLNL) [12] and Summit at Oak Ridge National Laboratory [2]. Similarly, the dragonfly topology is used in several existing as well as upcoming systems, e.g. Cori at NERSC [13], Theta at ANL [14].

The primary contributions of this work are:

- We conduct cross-platform and cross-network validation of the TraceR-CODES simulation work for multi-job workloads.
- We analyze the impact of the increase of per node computational capacity and the reduction of network endpoints on the performance of multi-job workloads.
- We quantify the impact of increasing network bandwidth for various compute capability equivalent systems.
- We also study the impact of changing the NIC-scheduling at the network endpoints for these systems.

II. Background

The simulation framework that we use in this study is TraceR-CODES [6], [7]. TraceR-CODES is a trace replaying simulator that uses ROSS's parallel event-driven simulator [15] underneath for network simulation. TraceR-CODES replays control and communication flow present in MPI application traces in the Open Trace Format 2 (OTF2) [16]. It can replay workloads that consist of multiple jobs in the system. TraceR-CODES supports unique features including:

- Ability to simulate multiple jobs differing in size, type, and location simultaneously.
- Computation scaling: the computation proportion of an application can be decreased or increased.
- Task mapping: the application processes can be mapped to specific compute nodes in the system.
- Ability to specify the number of iterations for the trace loop of an application.

Figure 1 shows the workflow of simulating a multi-job workload in TraceR-CODES. We first collect traces of applications with different job sizes using Score-P [5]. A multi-job workload is created from the applications of different sizes. To replay a multi-job workload on a specific interconnection network, TraceR-CODES takes as inputs, (1) the workload configuration that includes the job size for each job in the workload, the job mapping, and job iteration count, and (2) the network configuration that specifies the interconnection network parameters such as topology, bandwidth, and routing. Based on the workload and network configurations, TraceR-CODES replays the traces of the applications in the workload on the system with the specified interconnection network and reports the execution time of each job in the workload.

III. Modifications to TraceR-CODES

Although TraceR-CODES is powerful, it does not support all functions required by our study. We added a new graph-based network model to CODES, which gives us the flexibility to perform simulations on realistic fat-tree networks. We also added message scheduling mechanisms that are used in this study. Further, we modified the Tracer-CODES simulator

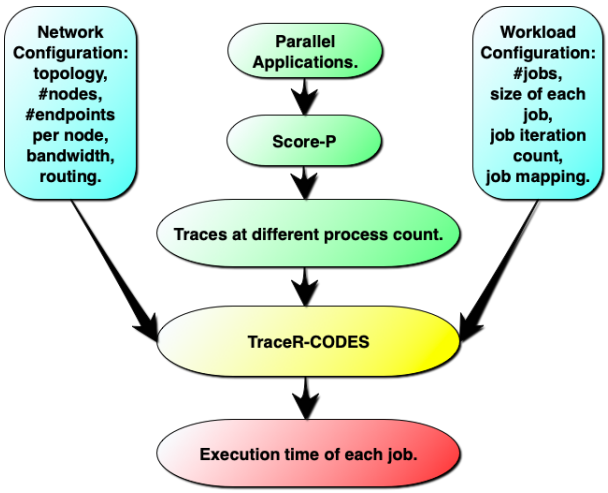


Fig. 1: Workflow of multi-job workload simulations using the TraceR-CODES simulation framework.

so that it can track communication-computation overlap in applications.

A. Simulating an arbitrary network topology

TraceR-CODES provides many built-in interconnection networks, including fat-tree and dragonfly. However, the connectivity within these built-in networks is rigid, and may not exactly match networks deployed in production supercomputers. Therefore, we implement a new network model, which we call the *arbitrary graph model*, that allows users to explicitly define their network topology.

B. Simulating message scheduling in the NIC

The network interface controller (NIC) in contemporary HPC systems is responsible for scheduling and packetizing messages.

FCFS scheduling: Messages are inserted at the back of the scheduling queue, as and when they arrive. During the packetization process, the scheduler keeps creating packets from the top of the queue until the entire message is packetized before it packetizes the next message in the queue.

RR scheduling: Messages are inserted into the scheduling queue of the network interface, as and when they arrive. During the packetization process, the scheduler creates one packet for a message and then moves to the next message: all messages are considered in a round-robin manner. RR not only allows concurrent communication progress for several communicating-pairs, but may also help the network in better utilizing multiple communication paths. While desirable, such a scheme is difficult to implement in the hardware as the number of concurrent messages can be very large.

RR-N scheduling: In this scheme, N is a parameter. RR-N is similar to RR, except that instead of packetizing every message in the scheduling queue in a round-robin manner, the scheduler packetizes the top N messages in the scheduling

queue. For example, in RR-2, the scheduler only packetizes the first 2 messages for communication. This newly added scheme simulates the real world scenario where a limited number of hardware queues are available at a NIC, which are used to keep multiple message in-flight concurrently.

C. Simulating multi-GPU nodes in TraceR-CODES

In TraceR, computation is simulated by fast-forwarding the simulation time by a desired amount. Thus, to emulate GPU execution in place of a CPU, we reduce the time spent in computational regions of an application by a certain factor. For each application, this factor is based on the speed up observed by us on Sierra supercomputer in comparison to the CPU cluster used to collect the application traces. Further, to simulate the impact of high-bandwidth links among the GPUs in a node (e.g. NV links), we added dedicated communication routes among the GPUs in a node.

IV. Validation

TraceR-CODES has been previously validated with micro-benchmarks and stand alone applications including pF3D, 3D Stencil, ping-pong, all-to-all, etc [3]. These validation studies were done for fat-tree networks and it was found that TraceR-CODES predicts the absolute value as well as the trends in the execution time with less than 15% error [3], [17]. However, these validation studies have been done with single job simulations. Further, these studies did not validate cross-platform and cross-network projections, i.e. traces were collected and projections were done for the same system.

To gain confidence in TraceR-CODES' prediction for cross-platform and cross-network multi-job workloads as well as in our new additions to TraceR-CODES, we validate TraceR-CODES with three random multi-job workloads. The validation is done by 1) randomly creating three workloads that consist of representative HPC benchmarks with different communication and computation characteristics, 2) running the workloads on the Quartz supercomputer [18] at LLNL, 3) simulating the workloads using TraceR-CODES with the system parameters set to the values for Quartz, and 4) comparing the predicted job execution times from the simulations with the measured times on Quartz.

The three workloads are formed by selecting jobs from two communication intensive benchmarks (Stencil4d and Subcomm3d) and two computation intensive applications (Kripke and Laghos). More information about the four applications is given in Section V.A.

We ran the three workloads in a dedicated access time (DAT) on Quartz at LLNL, during which period no other jobs ran on the machine. We used linear mapping of job ranks to nodes and measured the execution time of each job in the workloads. For simulation with the TraceR-CODES framework, we use the exact system settings as Quartz: (1) we create the exact fat-tree topology as Quartz using the arbitrary graph model; (2) we set the values of the network parameters to the corresponding values on Quartz: 11.9 GB/s peak link bandwidth, 8 packets buffer size, 4096 bytes packet size, and so on; and (3) the jobs and processes in each workload are mapped to compute nodes exactly in the same way as they ran on Quartz.

The traces for driving the simulation were collected on Vulcan [19], a 5D-torus based Blue Gene/Q system. Since, the computational capabilities of Vulcan are different from Quartz, we measure the relative compute scaling factor between Vulcan and Quartz, and scale the computation regions of simulations accordingly. This set up helps us evaluate the projections when the network (5D-torus vs fat-tree) as well as computational capability (IBM PowerPC vs Intel Xeon) of the traced system are different from the target system.

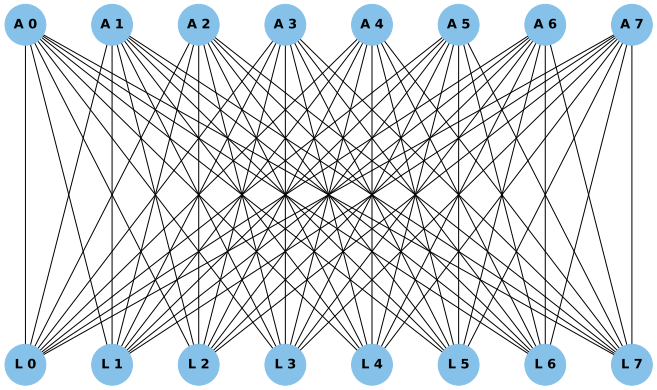


Fig. 2: A Quartz pod with eight aggregate and eight leaf switches, and all links.

Quartz Topology: The Quartz system deploys a 3-level fat-tree, with a 2:1 tapering at each of its 84 leaf switches. There are 84 aggregate switches and 32 core switches. Each switch has a radix of 48 and each leaf level switch is connected to 32 compute nodes. Note that some ports in the aggregate switches and core switches are left unused. The 84 leaf level switches are divided among 11 pods. Figure 2 shows a Quartz supercomputer pod. Each pod consists of 8 leaf switches and 8 aggregate switches, which are connected in an all-to-all bipartite graph. Each arc drawn here represents two physical links. In contrast, a standard 2:1 tapered fat-tree would have 16 leaf switches in each pod, which are connected to 16 aggregate switches using one physical link each. We give these details of the Quartz topology to highlight that Quartz' fat-tree is different from the standard, symmetric fat-tree topology, as are the networks in most production systems. These differences are the main driver for our development of the *arbitrary graph model*.

Figure 3 shows the results of the validation. On the horizontal axis, we have each application and their corresponding job size used in various workloads. Each blue dot represents the average of the error percentage between the predicted runtime and the measured runtime for various instances of the given application-job size pair that appear across the three workloads. For example, since Subcomm3d jobs with a process count of 128 appears two times across the three workloads, we compute their average error percentage to be -

7.88% and present it here. We can see that for all cases except 32-ranks Stencil4d, the prediction error is within 20%; and for all except 3 cases (32-rank Stencil4d, 32-ranks Kripke, and 64-ranks Kripke), the error is within 15%. These results suggest that TraceR-CODES predictions reasonably approximate the actual runtime on real systems for multi-job workloads even when the computational capability and underlying network are different.

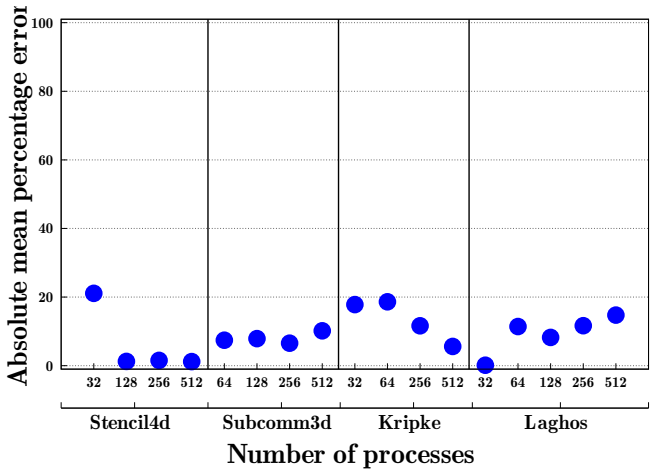


Fig. 3: Validation of TraceR-CODES (mean percentage error in predicted runtime compared to the actual runtime).

V. Simulation Methodology

In this section, we describe the applications and workloads used in this paper, and then discuss the hardware parameters varied in the simulations.

A. Applications and Workloads

We selected six applications of different computation and communication characteristics to create realistic HPC workloads. The applications include two communication-heavy kernels, Stencil4d [20] and Subcomm3d [20], two compute-intensive applications, Kripke [21] and Laghos [22], and two applications with a balanced communication-to-computation ratio, AMG [23] and SW4lite [24] (see Figure 4, left). The traces used in the study were collected using Score-P [5] on Vulcan, a Blue Gene/Q installation and Quartz, an Intel Xeon cluster at Lawrence Livermore National Laboratory (LLNL). The traces contain information about all MPI events executed on each MPI process, along with their timestamps. In addtion, they also record user annotations such as loop begin and end for the main compute loop. Following is a brief description of the six applications:

- **Stencil4d**: MPI benchmark with 8-point near-neighbor communication in a 4D virtual process grid.
- **Subcomm3d**: MPI benchmark with all-to-all communication within subsets of processes in a 3D virtual process grid.
- **Kripke**: 3D S^n deterministic particle transport code, which runs an MPI-based parallel sweep algorithm.
- **Laghos**: Proxy application that solves time-dependent Euler equations with MPI-based domain decomposition.
- **AMG**: Parallel algebraic multigrid solver.
- **SW4lite**: Proxy application for SW4 [24], a 3D seismic modeling code.

Figure 4 (left) presents the fraction of total execution time these applications spend in communication and computation when running with 32 processes. Computation is denoted by the red color, and non-overlapped communication is shown in green. At 32 processes, Stencil4d and Subcomm3d are dominated by communication. We tuned the communication-computation ratios in Stencil4d and Subcomm3d such that they replicate the runtime profiles of representative communication-intensive applications. Kripke and Laghos are dominated by computation with both nding more than 95% of time in computation. AMG and SW4lite spend ∼80% of their time in computation and the rest in communication. As described in Section III-C, suitable computation scaling factors are used to alter the behavior of these traces to emulate running the computation on GPUs. Figure 4 (right) shows how the computation-to-communication ratios change as we apply these scaling factors. Stencil4d and Subcomm3d spend most of their time in communication after compute scaling and the other applications now spend between 25-65% in communication.

The workloads in our study are created using the six HPC applications mentioned above at different process counts – 32, 64, 128, 256, and 512. In our study, the system supports up to 2048 processes. Thus, the sum of process counts in each of the workloads is exactly 2048. Each workload is obtained by iteratively randomly selecting an application and a job size until the total workload size has reached 2048. As a result, each workload has many jobs of different sizes, resembling the capacity workload of supercomputing centers [7]. Our experiments use 20 such random workloads. To ensure that the reported performance of each job size of each application is representative, we ensure that each job size of each application appears at least four times in the 20 workloads. This ensures that each job size of each application has been executed under different conditions in the experiments.

B. Hardware Design Parameters

Network topology: In our experiments, the impact of the hardware design parameters are studied in the context of two widely used interconnect topologies: fat-tree and 1D dragonfly.

(1) 1D Dragonfly – 1D Dragonfly [8] is a two-level direct network topology: switches form groups with a fully connected intra-group topology and groups are connected with an inter-group topology. The topology has three important parameters [8]: the number of compute nodes in each switch (p), the number of links in each switch that connect to other switches in the same group (a), the number of links in each switch that connect to other groups (h). A balanced dragonfly in general requires $a = 2p = 2h$. In our experiments, we set $p = h = 8$ and $a = 16$. Each group has 16 switches and 128

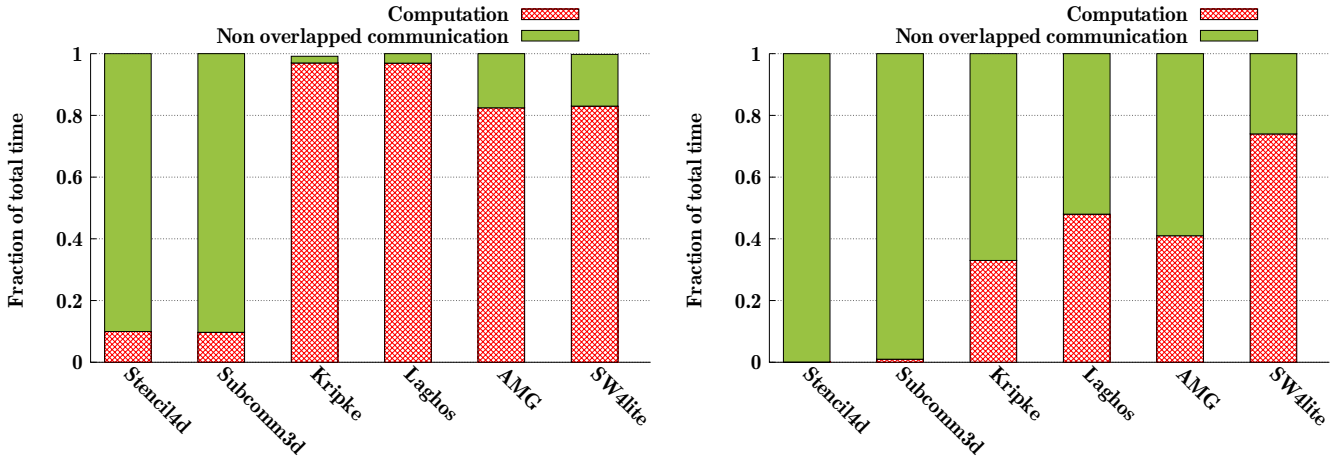


Fig. 4: Computation and communication characteristics of all applications without scaling (left) and with scaling for GPUs (right) running on 32 processes.

compute nodes. The global link connectivity between group follows the per-router arrangement [25]. The routing algorithm used is the progressive adaptive routing (PAR) [8], [25].

(2) Fat-tree – The other topology is a 3-level full bisection bandwidth fat-tree. In a 3-level full bisection bandwidth fat-tree, there are three types of switches: 1) core switches which are at the top layer to connect pods, 2) aggregate switches, which connects the leaf switches and form a pod, and 3) the leaf switches, which are connected to the compute nodes. In a 3-level full bisection bandwidth fat-tree, the number of uplinks in the aggregate and leaf switches is the same as the number of downlinks. For our study, the 3-level fat-tree is built using 32 radix switches. Each leaf switch connects to 16 compute nodes and 16 aggregate switches. Each pod has 16 aggregate switches, 16 edge switches, and 256 compute nodes.

Number of GPUs per node: In our study, we vary the number of GPUs in each compute node from 1 to 8 to analyze the impact of the increased computation density and the reduction of network endpoints on the system performance. Each GPU is assigned to one MPI process; to simulate different number of GPUs per node, multiple MPI process are assigned to a node. The GPUs inside a node are connected in an all-to-all connection topology resembling the intra node connectivity of Sierra system with NVlink. The bandwidth between GPUs within a node is set to be twice the network link bandwidth, so that it replicates that of Sierra supercomputer. The default setting for GPUs per node is 1 GPU per node. This is the default GPU per node setting whose performance is used to normalize other results.

In the experiments, when we increase the number of GPUs per node, we proportionally reduce the number of network endpoints, i.e. we make sure that for all network configurations, the total GPU count, as well the total MPI processes, is 2048. This is done to ensure that we compare systems that are of computationally equal capability as is often the case in the real world. Secondly, we make sure that each workload covers the entire network and no node is left empty during the simulation. Table I summarizes the network sizes used for each GPU per node setting, with the default setting being that of 1 GPU per node.

TABLE I: Network sizes for different GPUs per node.

GPUs per node	1D Dragonfly	Fat-tree
1	16 Groups	8 Pods
2	8 Groups	4 Pods
4	4 Groups	2 Pods
8	2 Groups	1 Pods

Network link bandwidth: We set our baseline link bandwidth as x=11.9 GB/s, which is the peak achieved link bandwidth on Mellanox EDR networks such as the Quartz supercomputer at LLNL. To analyze the sensitivity of various compute capability equivalent systems to communication capability, we vary the bandwidth from x/16 (16 times slow down of the baseline) to 16x (16 times speedup of the baseline). In the rest of the paper, we will use x to represent the base bandwidth, and will denote the network speed as x/16, x/8, x/4, x/2, x, 2x, 4x, 8x, and 16x.

Message scheduling: As the computation and communication density on the compute node increases, message scheduling performed by the NIC may have an impact on communication performance. In particular, scheduling schemes that alleviate head-of-line blocking may have significant benefits, especially when the link bandwidth is very high. In addition to head-of-line blocking, which is often mitigated by the use of virtual channels, message scheduling also affects congestion management and network utilization. Scheduling schemes that expose packets from multiple communicating-pairs to the network may perform better as it provides the network with the flexibility to use multiple network paths concurrently. To investigate the effect of message scheduling on a system with different network and different node compute capability,

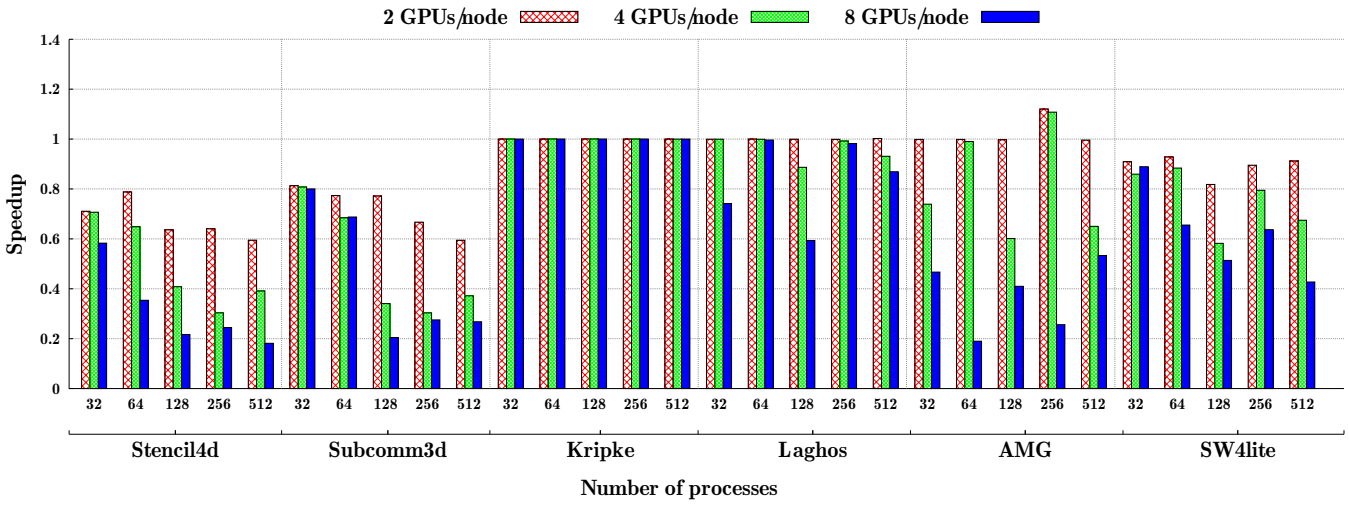


Fig. 5: Speedup on fat-tree for various numbers of GPUs per node settings with respect to 1 GPU/node configuration.

we compare the performance of FCFS, RR, and RR-N with different values of N on systems with different configurations.

VI. Simulation Results

We now present the results of simulation studies that vary different architectural parameters presented in Section V-B.

A. Impact of the number of GPUs per node

The number of GPUs per node determines the balance of computation to communication capacity of a system and thus is an important configuration choice in GPU-based HPC clusters. We study the impact of this parameter on different types of HPC applications.

Figure 5 presents the relative performance of the applications running on fat-tree based systems with different number of GPUs per node. The speedup in the figure is computed relative to the performance when running with 1 GPU per node. For example, in Figure 5, Stencil4d with 32 processes has a speedup of 0.71 in the 2 GPUs per node mode. This implies that the performance of Stencil4d with 2 GPUs per node is 71% of the Stencil4d performance with 1 GPU per node. Other configuration parameters are held constant at their default values (1x network link bandwidth, FCFS message scheduling, etc.) The reported performance is the average across all occurrences of an application and a given job size in the 20 workloads. Note that across the different GPUs per node configurations, each application and job size combination gets exactly the same computing resources. More GPUs per node does not imply more computing power for a given application and job size combination; it simply implies that the computing resources are available in a more condensed manner on fewer, more powerful nodes.

In Figure 5, we see that for communication-heavy applications (Stencil4d and Subcomm3d), as the number of GPUs per node increases, application performance drops for most job sizes. This is because as more GPUs are placed per node, the effective communication resources available for each GPU reduce. However, the performance drop is not linear w.r.t. the effective communication resources because the mapping of multiple MPI processes to node results in some of the data being communicated within node. This data can make use of the high-bandwidth intra-node GPU links.

An opposite effect is observed in the simulations of the 1D dragonfly topology in Figure 6. In some cases, such as Subcomm3d on 32 and 64 nodes, a significant amount of traffic is converted to intra-node when using 8 GPUs/node, which results in performance improvement of the application. Another factor that impacts performance is that when all processes in a job are mapped to a single switch, the job is less susceptible to inter-job network interference than when the processes in a job are mapped to multiple switches in the interconnect. With 4 GPUs per node, a 32-process job is mapped to 8 nodes and a 64-process job is mapped to 16 nodes. With 8 GPUs per node, a 32-process job is mapped to 4 nodes and a 64-process job is mapped to 8 nodes. Each switch in the fat-tree connects to 16 nodes and each switch in 1D dragonfly connects to 8 nodes: there are chances for the 32-process and 64-process jobs to be mapped completely within one switch and achieve higher performance.

For the next two applications (Kripke and Laghos), we observe a noticeably different impact of changing the balance of communication to computation capability. In the case of Kripke, more GPUs per node do not impact its performance. This is because the overall communication volume is low, and GPUs are often waiting on other GPUs to finish their computation. For Laghos, we observe a slowdown primarily with 8 GPUs per node. This indicates that having these many GPUs per node shifts the communication-computation balance and also the performance characteristics of the application.

Finally, for the last two applications (AMG and SW4lite), we observe a gradual slow down when more GPUs are incorporated per node, on both network topologies. While this performance drop is not as high as the communication-heavy

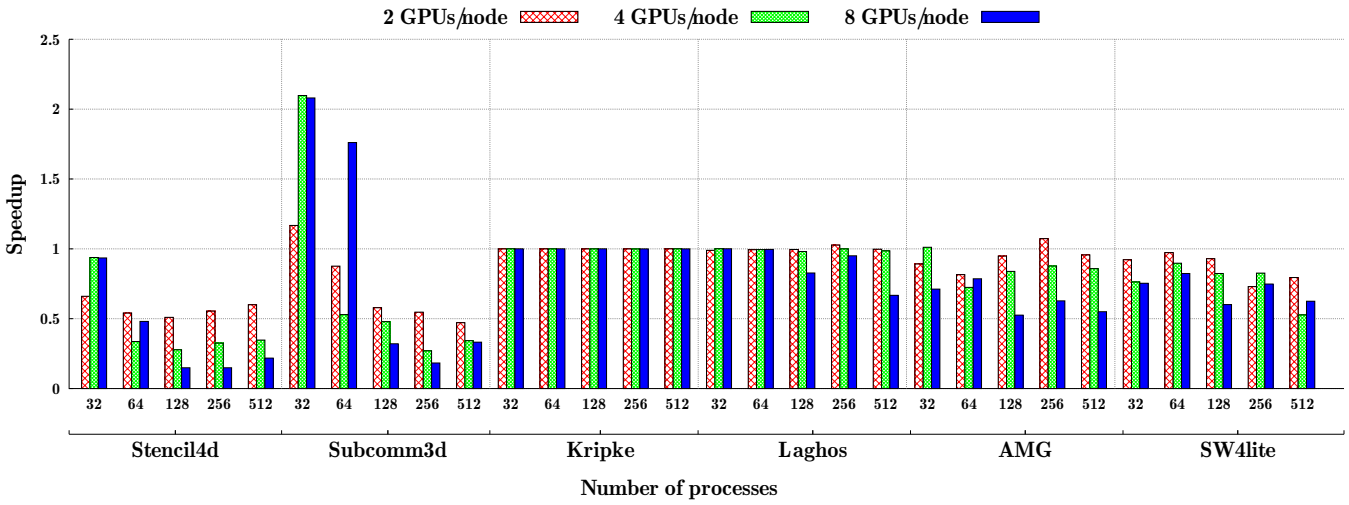


Fig. 6: Speedup on 1D dragonfly for various numbers of GPUs per node settings with respect to 1 GPU/node configuration.

applications, it is noticeable for the 4 and 8 GPUs per node configurations. We also find that for most applications that are sensitive to network performance, several factors including the communication pattern of the application, job mapping, and inter-job interference impact the execution time. For example, AMG and Laghos, experience higher slowdown in 8 GPUs per node configuration in workloads in which they are placed adjacent to communication-heavy applications. The typical reason for this slowdown is that communication-sensitive applications when mapped adjacent to similar applications contend for network resources, thus impacting the performance.

Overall Observation*: Most applications run slower with four or more GPUs per network endpoint.*
In our experiments, all but one application (Kripke, which is not sensitive to network capabilities) slow down noticeably with four or more GPUs per network endpoint. Although part of the communication volume may be restricted to intra-node communication with more GPUs per node, this benefit is typically overshadowed by performance loss due to the reduction of the node communication to computation ratio.

B. Impact of network bandwidth

In the previous section, we saw that as the number of GPUs per node increases, the default 1x network bandwidth becomes a performance bottleneck for many cases. Thus, we next study the impact of varying network bandwidth along with number of GPUs/nodes on application performance.

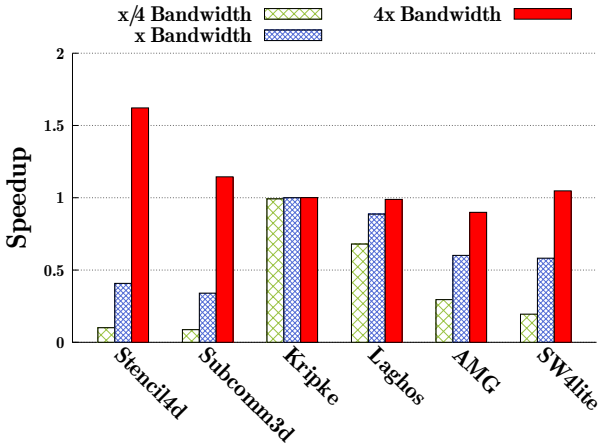
In our simulation experiments, we observed that the impact of network bandwidth on jobs of different sizes shares similar trends. Hence, we present data only for a job size of 128 processes. Figure 7a shows the performance for the 4 GPUs per node configuration with varying network bandwidth relative to 1 GPU per node, 1x network bandwidth configuration on fat-tree network. We find that the network bandwidth has a significant impact on most applications. The gains are highest for communication-heavy applications such as Stencil4d and Subcomm3d. Conversely, the impact of reducing the network bandwidth is also highest for those. A similar trend is observed for the 1D dragonfly topology as shown in Figure 7b.

Table II presents the minimum bandwidth required for each application and a given job size to achieve 90% of the performance of the default setting for the fat-tree topology. As expected, different types of applications have different bandwidth requirements. In general, communication-intensive applications require larger bandwidth to sustain the increased number of GPUs per node while computation-intensive applications have less bandwidth requirement. For example, for the 8 GPUs per node case with 512 processes job size, Stencil4d needs 8x network bandwidth to achieve 90% of the performance from the default setting; AMG and SW4lite need more than 4x bandwidth while Kripke only needs x/8 bandwidth.

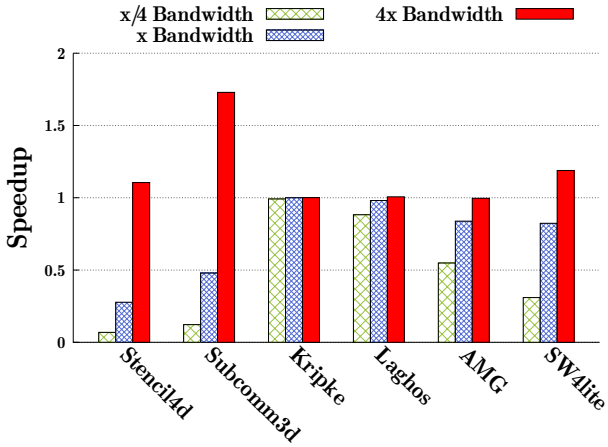
TABLE II: Minimum bandwidth required to achieve 90% of the performance of the default 1 GPU/node configuration for fat-tree

Applications	32 processes		512 processes	
	4 GPUs/node	8 GPUs/node	4 GPUs/node	8 GPUs/node
Stencil4d	1x	1x	4x	8x
Subcomm3d	x/2	x/2	4x	4x
Kripke	x/16	x/16	x/8	x/8
Laghos	x/2	2x	x	2x
AMG	4x	8x	4x	8x
SW4lite	2x	2x	2x	4x

Further, application requirement is also affected by the job size and the placement with other jobs. For example, 32-process Laghos ran slower in some workloads when mapped in the 8 GPUs per node configuration, which is why here we need double bandwidth to get more than 90% speedup. We also see that sometimes communication-intensive applications such as Stencil4d and Subcomm3d require less bandwidth in 8 GPUs per node configuration than 4 GPUs per node configuration to reach 90% of the performance for 32 processes and 64 processes. This is mainly due to the fact that, with a larger



(a) Results for fat-tree



(b) Results for 1D dragonfly

Fig. 7: Speedup for the 4 GPUs/node configuration over 1 GPU/node, 1x network bandwidth configuration. Data is shown only for job sizes of 128 GPUs; similar trends observed for other job sizes.

number GPUs per node, a significant fraction of the communication happens within the same node. This indicates that future GPU-based platforms must consider its workloads to decide important networking hardware parameters. The results for 1D dragonfly, which has a similar trend as that in fat-tree.

Overall Observation*: Bandwidth requirement to sustain high performance depends on GPU density and job sizes.*
Our results show that each type of application has a sweet-spot for them to perform effectively. Hence, the design of a future GPU cluster should take its applications into consideration in order to achieve the maximum performance-cost ratio.

C. Impact of message scheduling in the NIC

The impact of message scheduling on system performance has not received sufficient attention in the community. To our knowledge, this is the first time that the impact of message scheduling on system and application performance is being studied systematically. Similar to the impact of the number of GPUs per node and network link bandwidth, the impact of message scheduling is similar for both fat-tree and 1D dragonfly. Thus, we only discuss results for the 1D dragonfly in detail.

Figure 8 shows the speedup for 64 and 512 processes (GPUs) of Stencil4d relative to the default case with 1 GPU per node and FCFS scheduling (network bandwidth is fixed at 1x for all configurations). For the 1 GPU per node cases, the scheduling significantly affects the performance: the larger the number of messages the scheduler considers for packetization concurrently, the higher the performance. The RR scheduler reaches a speed-up of 1.45 for the 64-process job and 1.93 for the 512-process job in comparison to the default FCFS scheduler. A similar trend is observed for the 8 GPUs per node cases: the RR scheduler improves the speed up from

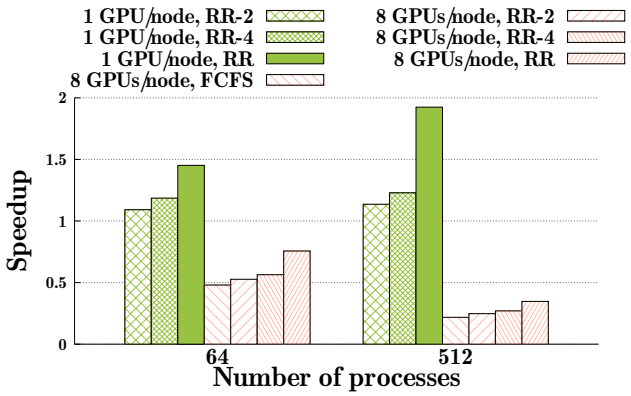


Fig. 8: Results for Stencil4d (64 processes and 512 processes on 1D dragonfly)

0.48 with the FCFS scheduler to 0.76 for the 64-process job, and from 0.22 to 0.35 for the 512-process job.

Figure 9 shows the speedup for 64 and 512 processes of Subcomm3d. For 1 GPU per node, RR scheduler performs better than FCFS. However, RR is only slightly better than RR-2 and RR-4 and achieves a 1.3 speed-up for the 64-process job and 1.7 speed-up for the 512-process job over FCFS. For 8 GPUs per node cases, all schedulers have similar performance with FCFS being slightly better than other scheduling schemes. Although both Stencil4d and Subcomm3d are communication-intensive, the impact of message scheduling is different. This is because the communication characteristics in these two applications are different.

Message scheduling has no impact on Kripke as Kripke is not sensitive to communication as seen earlier. Figure 10 shows the speedup for 64-process and 512-process simulations of Laghos. For 1 GPU per node cases, all schedulers have the same performance. For 8 GPUs per node cases, all schedulers

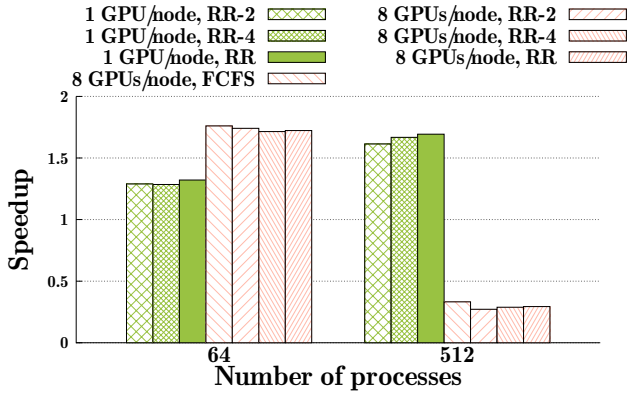


Fig. 9: Results for Subcomm3d (64 processes and 512 processes on 1D dragonfly)

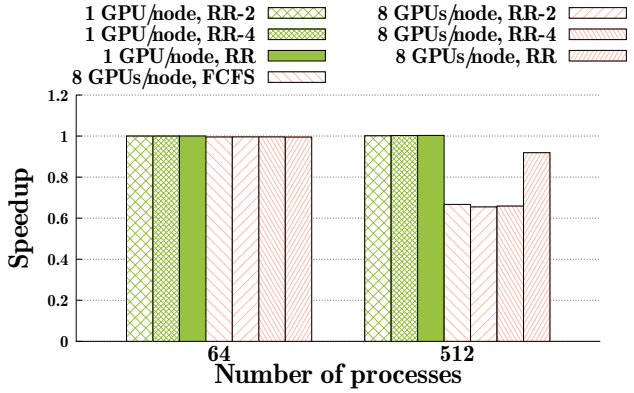


Fig. 10: Results for Laghos (64 processes and 512 processes on 1D dragonfly)

have the same performance for the 64-process job, but RR has a significantly better performance than others for the 512-process job. As shown in Figure 6, for 512 processes (and 256 processes and 128 processes), Laghos is affected by communication only in the 8 GPUs per node setting.

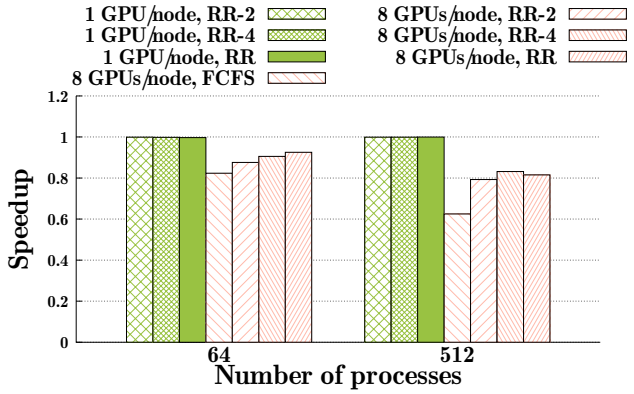


Fig. 11: Results for SW4lite (64 processes and 512 processes on 1D dragonfly)

Figures 11 show the results for SW4lite. Message scheduling has no impact on the 1 GPU per node cases, but affects the performance significantly for 8 GPUs per node cases for both applications and the two job sizes. The impact, however, depends on both the application and job sizes. Simillar results are seen for AMG application.

Overall Observation*: For most applications, some degree of round robin in NIC scheduling is effective. However the exact degree is application dependent – no single scheduling scheme can achieve the best performance across applications.*

Message scheduling can impact performance only when there are many concurrent communicating pairs. For the 1 GPU per node cases, it thus only affects the communication heavy applications such as Stencil4d, and has virtually no impact on the other applications in our study. As the number of GPUs per node increases, so does the number of communication sources and the number of concurrent communications. Thus, with 8 GPUs per node, message scheduling makes a difference in all applications, except Kripke. The magnitude of the impact, however, depends on the application as well as the job size: Round-robin (RR) is the most effective scheduling in many cases. However, each of the scheduling schemes that we use in our study achieves the best performance in some cases. For example, FCFS is the best for AMG with 64 processes and 8 GPUs per node; RR-4 is slightly better than other scheduling policies for SW4lite with 512 processes and 8 GPUs per node. We conclude that the effectiveness of message scheduling depends on both application and the network parameters, and needs to be further studied by examining more applications as well as system configurations.

VII. Related Work

Application simulations are widely used to evaluate an HPC network. Jain and Bhatele in their past studies have exploited the full system simulation capability of TraceR-CODES simulator to perform comprehensive simulation studies of various systems. Bhatele et al. performed a holistic study on how three different scalable topology - dragonfly, express mesh, and fat-tree performed under different conditions [26]. Here he tried to explore different network design aspects such as the number of nodes, routers, and links, all of which in turn, influence dollar costs. These studies are very important in system design, installation and procurement. Jain et al. have previously used multi-job workloads with different communication characteristic and communication-computation ratios to study the effects of various configuration parameters like link bandwidth, number of rails, number of planes, and tapering, for fat-tree [3]. Our work goes beyond the previous study by systematically investigating two different topologies (fat-tree and 1D dragonfly), the impact of the number of GPUs per node, its interaction with network link bandwidth, and the message scheduling scheme. A lot of previous studies have been conducted on designing and evaluating network topologies and routing methods. For example, Rahman et al. have conducted studies on topology custom routing across various dragonfly

topologies [27]. Here they showed how by customizing the paths used in UGAL routing in dragonfly topology, a better communication performance can be achieved. Zaid et al. have done similar studies with Jellyfish topologies [28].

Such studies, however, do not evaluate the overall system and application performance as we do in this paper.

VIII. Conclusion

We validate the TraceR-CODES simulation framework that is able to produce performance results similar to measurement results in real execution, and use the framework to study the impact of hardware parameters for future GPU-based HPC systems on the popular fat-tree and dragonfly topologies. The hardware parameters studied include the number of GPUs per node, the network bandwidth, and the message scheduling scheme. Our results indicate that as more GPUs are equipped on each compute node, more applications will be sensitive to the communication performance; and thus, the network bandwidth as well as the message scheduling method will play a more important role in the system performance. In general, higher network bandwidth is necessary for maintaining the performance of communication intensive applications on systems with more GPUs per node; some degree of round-robin fair scheduling works well for most applications. The exact impact of these hardware parameters, however, is application dependent. We conclude that to build a performance-cost effective GPU cluster, the applications to be executed on the cluster must be studied along with the system parameters. In summary, the general trend of shifting to fatter nodes does not always give optimal performance of an application. An investigation similar to ours can be used to determine the most cost-effective hardware parameters.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grants CICI-1738912, CRI-1822737, and SHF-2007827. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. This work used the XSEDE Bridges resource at the Pittsburgh Supercomputing Center (PSC) through allocations ECS190004 and CCR200042.

References

[1] LLNL, "Sequoia," https://asc.llnl.gov/computers/historic-decommissioned-machines/sequoia-and-vulcan, 2020.

[2] ORNL, "Summit," https://www.olcf.ornl.gov/summit/, 2020.

[3] N. Jain, A. Bhatele, L. H. Howell, D. Böhme, I. Karlin, E. A. León, M. Mubarak, N. Wolfe, T. Gamblin, and M. L. Leininger, "Predicting the performance impact of different fat-tree configurations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126967

[4] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "Gpu computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[5] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony *et al.*, "Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir," in *Tools for High Performance Computing 2011*. Springer, 2012, pp. 79–91.

[6] B. Acun, N. Jain, A. Bhatele, M. Mubarak, C. D. Carothers, and L. V. Kale, "Preliminary evaluation of a parallel trace replay tool for hpc network simulations," in *European Conference on Parallel Processing*. Springer, 2015, pp. 417–429.

[7] N. Jain, A. Bhatele, S. White, T. Gamblin, and L. V. Kale, "Evaluating hpc networks via simulation of parallel workloads," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 154–165.

[8] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. USA: IEEE Computer Society, 2008, pp. 77–88. [Online]. Available: https://doi.org/10.1109/ISCA.2008.19

[9] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[10] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem, "Oblivious routing in fat-tree based system area networks with uncertain traffic demands," *IEEE/ACM Transactions on Networking*, vol. 17, no. 5, pp. 1439–1452, 2009.

[11] X. Yuan, "On nonblocking folded-clos networks in computer communication environments," in *2011 IEEE International Parallel Distributed Processing Symposium*, 2011, pp. 188–196.

[12] LLNL, "Sierra," https://hpc.llnl.gov/hardware/platforms/sierra, 2020.

[13] NERSC, "Cori," https://docs.nersc.gov/systems/cori/interconnect/, 2020.

[14] ALCF, "Theta," https://www.alcf.anl.gov/alcf-resources/theta, 2020.

[15] 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.

[16] D. Eschweiler, M. Wagner, M. Geimer, A. Knüpfer, W. E. Nagel, and F. Wolf, "Open trace format 2: The next generation of scalable trace formats and support libraries." in *PARCO*, vol. 22, 2011, pp. 481–490.

[17] B. Acun, N. Jain, A. Bhatele, M. Mubarak, C. D. Carothers, and L. V. Kale, "Preliminary evaluation of a parallel trace replay tool for hpc network simulations," in *Euro-Par 2015: Parallel Processing Workshops*, S. Hunold, A. Costan, D. Giménez, A. Iosup, L. Ricci, M. E. Gómez Requena, V. Scarano, A. L. Varbanescu, S. L. Scott, S. Lankes, J. Weidendorfer, and M. Alexander, Eds. Cham: Springer International Publishing, 2015, pp. 417–429.

[18] LLNL, "Quartz," https://hpc.llnl.gov/hardware/platforms/Quartz, 2020.

[19] LLNL, "Vulcan," https://asc.llnl.gov/computers/historicdecommissioned-machines/vulcan, 2020.

[20] A. Bhatele, "Evaluating trade-offs in potential exascale interconnect topologies," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2018.

[21] LLNL, "Kripke," https://computing.llnl.gov/projects/co-design/kripke, 2020.

[22] LLNL, "Laghos," https://computing.llnl.gov/projects/co-design/laghos, 2020.

[23] E. Proxy, "Amg," https://proxyapps.exascaleproject.org/app/amg/, 2020.

[24] B. Sjogreen, "Sw4 final report for icoe," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2018.

[25] Z. S. A. Alzaid, S. Bhowmik, X. Yuan, and M. Lang, "Global link arrangement for practical dragonfly," in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3392717.3392756

[26] A. Bhatele, N. Jain, M. Mubarak, and T. Gamblin, "Analyzing cost-performance tradeoffs of hpc network designs under different constraints using simulations," in *Proceedings of the 2019 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2019, pp. 1–12.

[27] M. S. Rahman, S. Bhowmik, Y. Ryasnianskiy, X. Yuan, and M. Lang, "Topology-custom ugal routing on dragonfly," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–15.

[28] Z. ALzaid, S. Bhowmik, and X. Yuan, "Multi-path routing in the jellyfish network," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 832–841.