LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-756549

Evaluating Trade-offs in Potential Exascale Interconnect Topologies

A. Bhatele

August 16, 2018

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

EXASCALE COMPUTING PROJECT

ECP-U-2017-XXX

Evaluating Trade-offs in Potential Exascale Interconnect Technologies

WBS 2.4.2.01, Milestone **ECP-XX-XXXX**

ECP Hardware Evaluation Interconnect Working Group

August 21, 2018

U.S. DEPARTMENT OF ENERGY | Office of Science

National Nuclear Security Administration

ECP-U-2017-XXX

ECP Milestone Report
Evaluating Trade-offs in Potential Exascale Interconnect Technologies
WBS 2.4.2.01, Milestone ECP-XX-XXXX

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

August 21, 2018

ECP Milestone Report
Evaluating Trade-offs in Potential Exascale Interconnect Technologies
WBS 2.4.2.01, Milestone ECP-XX-XXXX

APPROVALS

Submitted by:

K. Scott Hemmert, Ray Bair, Abhinav Bhatele, Taylor Groves, Nikhil Jain, Cannada Lewis, Misbah Mubarak, Scott Pakin, Robert Ross, Jeremy Wilke ECP-XX-XXXX	Date

Concurrence:

Simon D. Hammond ECP Hardware Evaluation	Date

Approval:

Dan Hoag Federal Project Administrator	Date

REVISION LOG

Version	Creation Date	Description	Approval Date
1.0	August 21, 2018	Original	
2.0	August 21, 2018	New revision	

EXECUTIVE SUMMARY

This report details work to study trade-offs in topology and network bandwidth for potential interconnects in the exascale (2021-2022) timeframe. The work was done using multiple interconnect models across two parallel discrete event simulators. Results from each independent simulator are shown and discussed and the areas of agreement and disagreement are explored.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This study looks at the high-level trade-offs for interconnect architectures that are likely to be available in HPC machines in the 2021-2022 timeframe. The study uses multiple simulation models to look at the effects of varying machine size, injection bandwidth and topology for a selected set of benchmark communication patterns. The studies show the sensitivity of various applications to these network parameters. This study can help inform decisions on node size versus number of nodes, which topology is most efficient for our applications, how much injection bandwidth to provision, etc. These simulations only look at interconnects with maximally configured global bandwidth, and studies of networks configured with less will be the subject of future milestones.

Overall, the different models have good agreement on the effects of various changes in the parameter space. Using multiple simulation environments can provide added confidence in the results, as well providing added information for refining and understanding the models.

The report is structured as follows: Section 2 discusses the network parameters that are the focus of this study. Section 3 describes the three simulation environments used to study the trade-off space. Section 4 describes the proxies used to study various communication patterns. The results for the three simulators, along with observations for those simulations are presented in Section 5 and overall conclusions are presented in Section 6.

2. SYSTEM CONFIGURATIONS

The study explores the design space parameterized in four areas: topology, system size (node count), node injection bandwidth and node allocation. The results presented in this report reflect only performance comparisons and do not take into account any cost models (financial, power, etc.). The performance results will need to be weighed against anticipated costs to draw final conclusions as to the best machine configuration. The parameter values studied for each of these areas is discussed below.

2.1 TOPOLOGIES

All of the topologies are highly connected and have a significant amount of path diversity between endpoints. The topologies are fat-tree, dragonfly, megafly (sometimes called dragonfly+ depending on the vendor) and hyperX (also called a flattened butterfly). Because of the path diversity, most of these topologies will use adaptive routing algorithms to more effectively utilize the available bandwidth for all communication patterns. The choice of routing algorithm is one of the most significant differences between simulation models. A brief description of these topologies are given below.

2.1.1 Fat-tree

The fat-tree topology is a tree-based topology in which bandwidth of edges increases near the top (root) of the tree [1]. Practical deployments of fat-tree in most supercomputers resemble the folded-Clos topology. In this set up, many routers of the same radix are grouped together to form *core* switches (logical equivalent to higher levels of a tree) and provide high bandwidth. The default fat-tree configuration is a *full* fat-tree: the total bandwidth within a level does not decrease as we move from nodes connected to the leaf switches toward higher levels. In order to reduce the cost of the network, tapering can be deployed to connect more nodes per leaf switch. This reduces the total bandwidth at higher levels but also lowers the number of switches and links required to connect the same number of nodes in comparison to the full fat-tree. All the fat-tree systems used in this study are full fat-trees and provide a well recognized baseline for further comparisons.

2.1.2 Dragonfly

The dragonfly topology is a hierarchical network that consists of groups of routers and nodes connected together in a all-to-all manner (i.e. every group has at least one direct connection to every other group). The topology of the group can differ depending on the packaging decisions made by vendors. The original dragonfly described in [2] uses a one-dimensional fully connected group, where every router is connected to every other router in the same group. Each router is, in turn, connected to the same number of nodes. The

Aries interconnect provided by Cray, Inc. was the first commercially available dragonfly topology and used a 16 by 6 flattened butterfly where every router was connected to four hosts as the group topology.

We believe dragonfly networks in the exascale timeframe will have a 1-D fully connected group. There are multiple ways to describe the topology, but for the purposes of this report, the specific dragonfly topology will be described by four parameters:

- *Nodes per router*: The number of nodes connected to each router in the group.
- *Routers per group*: Number of routers in each group. Each router will have one link to every other router in the group.
- *Number of links between groups*: Number of links between each pair of groups. The number of links between groups will determine the total global bandwidth.
- *Number of groups*: Specifies the total number of groups in the system.

In a dragonfly with full global bandwidth, each group will have the same number of global links as there are host links. Some configurations will not allow this to be done in a balanced way, so a dragonfly with maximally configured global bandwidth will get as close to this number as possible, while still having the same number of links to each other group.

2.1.3 Dragonfly+/Megafly

The dragonfly+/megafly topology [3] is a dragonfly which uses a two-level fat-tree network in each group. This locally defined network is also known as a complete bipartite graph: a graph with two sub-groups where all nodes within one subgroup are connected to all nodes in the other subgroup but zero connections within a subgroup.

The two levels in each local group contain routers that will be referred to as *Leaf or Node-level switches*, those that have terminal/compute node connections but no global connections, and *Spine or Global switches*, those that have global connections to other groups but no terminal/compute node connections. This hierarchical organization is meant to help with load balancing by giving specific routers different responsibilities based on what level of the fat-tree they are.

Maximally configured global bandwidth for dragonfly+ is accomplished in the same as for the dragonfly described above.

2.1.4 HyperX/Flattened Butterfly

HyperX is a regular, multi-dimensional topology. Routers are identified by their index in each dimension and each router is connected to every other router that shares all but one location index. For example, routers in a 3 dimensional hyperX would be identified by their x, y and z coordinates.

A specific instance of a hyperX can be described by the number of nodes connected to each router, the number of routers in each dimension, and the number of links between each router in each dimension. For example, a 3-D hyperX with 16 routers in the x-dimension, 8 routers in the y- and z-dimensions would have a shape parameter of 16x8x8. Further if there were one link between each router in the x-dimension and 2 links between the routers in each of the y- and z-dimensions, the width parameter would be 1x2x2.

A hyperX with full global bandwidth will have the same number of links to other routers in each dimension as it has endpoints connected to it. Some configurations will not be able to accomplish this, so maximally configured global bandwidth will get as close to this as possible while still having the same number of links to each other router in a given dimension (the number of links can differ for different dimensions).

2.2 SYSTEM SIZES

The simulations study three different system sizes: 8k (8,192), 16k (16,384) and 32k (32,768) nodes. Each simulation uses the same number of total MPI ranks. The total rank count was set to 32,768, giving 4 ranks per node for 8k nodes, 2 ranks per node for 16k nodes and 1 rank per node for 32k nodes. The study assumes accelerated nodes with a total system compute of 1.3 EF/s for all three system sizes. This gives approximately 40 TF/s per rank (160 TF/s, 80 TF/s and 40 TF/s per node, for 8k, 16k and 32k nodes, respectively).

All of the topologies assumed the availability of a 64 port router. The configurations used for the simulations are shown in Table 1.

Topology	Nodes	Topology Parameters
Fat-tree	8k	Three levels, full bisection, 32 nodes per leaf switch, 32 leaf switches
	16k	Three levels, full bisection, 32 nodes per leaf switch, 64 leaf switches
	32k	Three levels, full bisection, 32 nodes per leaf switch, 128 leaf switches
HyperX	8k	16 nodes per router, Shape: 8x8x8, Link width: 2x2x2
	16k	16 nodes per router, Shape: 16x8x8, Link width: 1x2x2
	32k	16 nodes per router, Shape: 16x16x8, Link width: 1x1x2
Dragonfly	8k	16 nodes per router, 32 routers per group, 16 groups, 32 links between each group
	16k	16 nodes per router, 32 routers per group, 32 groups, 16 links between each group
	32k	16 nodes per router, 32 routers per group, 64 groups, 8 links between each group
Dragonfly+	8k	32 nodes per leaf switch, 32 leaf switches per group, 8 groups, 128 links between each group
	16k	32 nodes per leaf switch, 32 leaf switches per group, 16 groups, 64 links between each group
	32k	32 nodes per leaf switch, 32 leaf switches per group, 32 groups, 32 links between each group

Table 1: Topology parameters used for all simulations.

2.3 NODE INJECTION BANDWIDTH

In the 2021-2022 timeframe, the most likely interconnect technologies will provide 200 Gbps to 400 Gbps link bandwidths. The simulations done for this study used links of 200, 400 and 800 Gbps (25, 50 and 100 GB/s). The studies generally used a single link of these bandwidths, but they are representative of networks that will provide multiple planes of bandwidth. The likely physical configurations represented by these bandwidths are shown in Table 2.

200 Gbps	1 rail @ 200 Gbps
400 Gbps	1 rail @ 400 Gbps 2 rails @ 200 Gbps
800 Gbps	2 rails @ 400 Gbps 4 rails @ 200 Gbps

Table 2: Possible physical instantiations of the studied injection bandwidths.

2.4 NODE ALLOCATION

Two types of node allocation were simulated as part of this study: linear and random. For both cases, ranks assigned to a single node are allocated in sequential order. For example, in the four rank per node, logical ranks $4n$, $4n+1$, $4n+2$ and $4n+3$ will always be assigned to the same logical node. For the linear allocation, the logical nodes are assigned in sequential order to the physical nodes. For the random case, logical nodes are assigned randomly to the physical nodes. The linear case resembles the allocations that result from a system that was empty when a job started. Random allocation approximates a worse case that could arise as the machine allocates jobs over time and no longer has all contiguous nodes when allocating a job.

3. SIMULATION MODELS

This section describes the simulation models used to study the exascale interconnect architectures. The studies used two different simulator frameworks: TraceR/CODES and SST. Additionally, the SST simulations used two different network models: Macro (referred to as SST/Macro) and Merlin/Ember (referred to as SST/Merlin). Each of these models is described below.

The models were originally each designed for different purposes and use different fidelity models for the endpoint and routers.

3.1 TRACER-CODES

The TraceR-CODES framework [4, 5] provides a multi-level trace-driven infrastructure for packet-level simulations of traffic flow on HPC networks. It is built upon ROSS [6], a parallel discrete event simulation (PDES) engine, and has been successfully deployed for studying multi-job workloads on HPC networks [7, 8, 9]. The optimistic nature of the ROSS PDES engine drives the scalability of the TraceR-CODES framework and enables fast simulation using large core counts. The network simulation related components of this framework are developed by a collaboration among the Argonne National Laboratory, Lawrence Livermore National Laboratory, and Rensselaer Polytechnic Institute.

CODES provides a unified API for simulating traffic flow on many HPC networks, e.g. dragonfly, torus, hyperX, express mesh, and fat-tree. The network being simulated can be selected at runtime along with other parameters such as the topology dimensions, link bandwidth, etc. Switches are modeled using explicit input/output buffers, and loss-less interaction between connected ports is managed using buffer occupancies and token-based arbitration. For each network model, different routing schemes are implemented depending on the best known algorithms. Different methods of driving communication and control flow have been implemented on top of CODES including trace-driven, synthetic traffic patterns, and skeleton-like applications.

TraceR is a parallel execution trace simulator that replays the control and communication flow of HPC applications on top of CODES primarily using OTF2 traces. For multi-job workloads, each application is concurrently simulated and shares network resources with other applications. Users can also specify the job placement and task mapping for each application. For compute regions of the application, TraceR allows replacing and scaling the execution time recorded in the traces. This enables predictions of likely scenarios on future systems with different computational power. MPI semantics such as message matching, point-to-point protocol options such as Eager and Rendezvous, and collective communication algorithms are implemented in this layer.

3.2 SST

The Structural Simulation Toolkit (SST) [10, 11] is a simulation framework providing a parallel discrete event core. The framework allows different components to be connected together and provides a large library of ready-to-use components to model different aspects of high performance computing systems (Figure 1). SST is widely used by both industry and academic researchers.

For this work, the focus is on the capabilities of SST to model large scale HPC interconnection networks, for which SST has been widely used in the past [12, 13, 14]. The SST core uses a conservative parallelization algorithm for synchronizing components across threads and MPI ranks. The algorithm works well for large scale network simulations and allows SST to easily scale network simulations to dozens of nodes and hundreds of cores and larger.

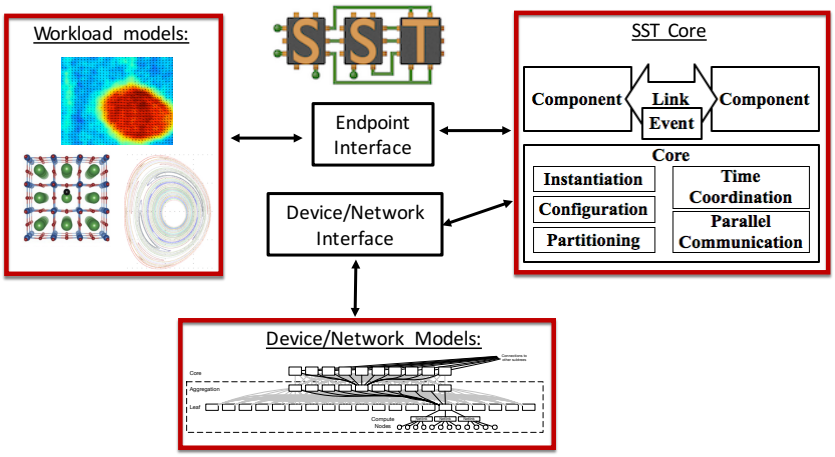

Figure 1: High level organization of the Structural Simulation Toolkit.

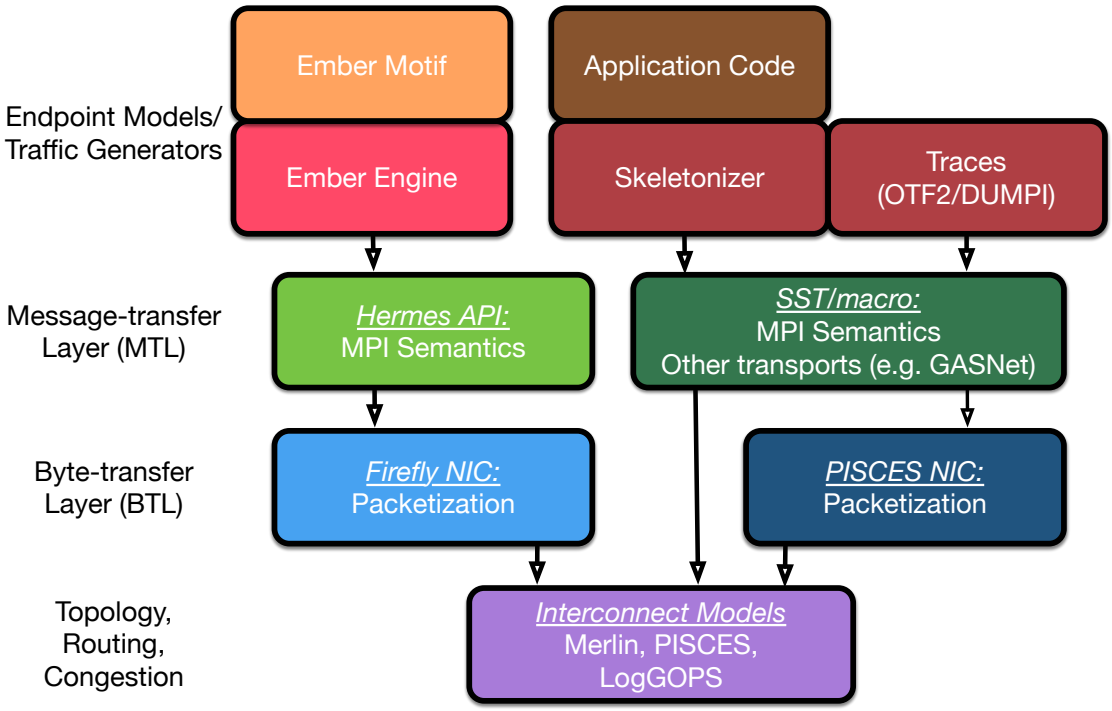

Figure 2: Overview of software stack showing traffic generation from SST endpoint models. MPI operations (e.g. send/recv or collectives) are generated by an Ember motif, trace, or skeleton app, converted into individual send/RDMA operations by an MPI semantics layer, and then packetized to inject traffic into the simulated interconnection network.

An SST interconnect simulation is characterized by its choice of endpoint model that generates network traffic and the hardware models that simulate the injected traffic (Figure 1). Endpoint models are primarily driven by traffic from message passing (MPI) applications. SST provides a software stack model beginning with individual MPI calls generated by an application; through MPI middleware that provides semantics and collective algorithms; a packetization of flows (MPI messages); and finally injection of packets into the network (Figure 2) The two main combinations of endpoint models used are Skeletons + SCULPIN in SST-macro and Ember + Merlin in the SST-elements library.

3.3 SST/MACRO

3.3.1 Skeletonization

The main endpoint model generating traffic for interconnection simulation with SST/macro are skeleton applications derived from MPI source code. An SST compiler wrapper performs a source-to-source transformation of the source code, eliminating large memory allocations and replacing expensive compute regions with estimated delays. Full details of skeletonization can be found in a recent publication [15]. An overview of the process is given in Figure 3. For simple applications, the compiler can skeletonize fully automatically. This was the case for the Sweep3D, Halo, and Sub-a2a benchmarks. For more complicated applications, pragmas are required to guide skeletonization.

3.3.2 Network Model Details

The main network model used with SST/macro is called SCULPIN. The model is designed to be coarse-grained, capturing the most important congestion effects and topology trends but with some tradeoff in accuracy for efficiency. SCULPIN simulates individual packet contention on switch ports. The SCULPIN model is

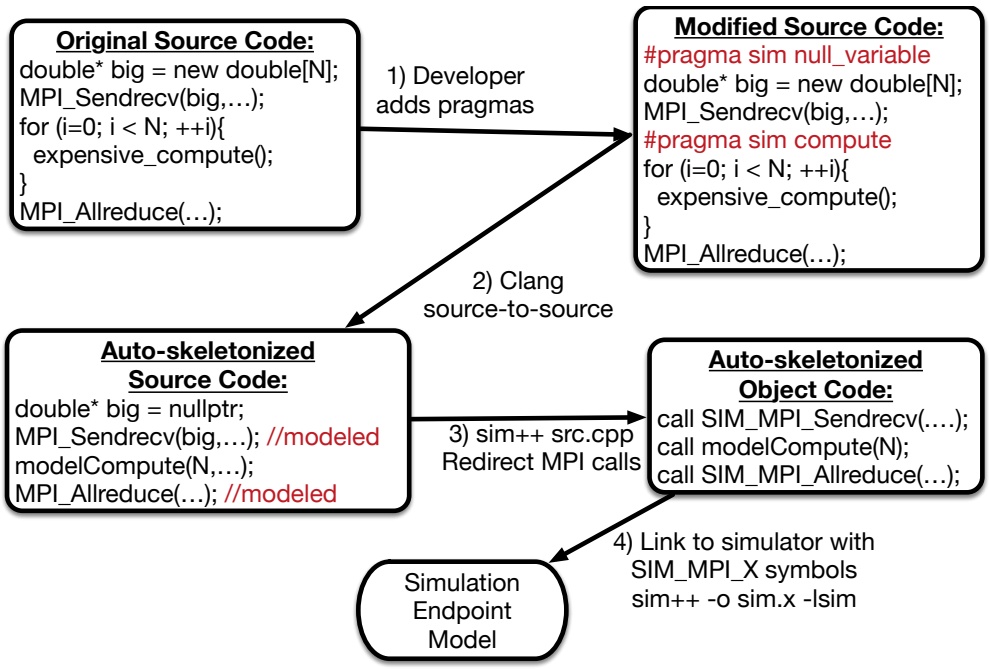

Figure 3: Source-to-source transformation of an MPI application to a skeleton application using the SST compiler. Skeletonization can be automatic or guided by pragma hints. Skeletonization removes large memory allocations and expensive compute regions.

relatively simple, making it less prone to implementation bugs or erroneous results for corner cases. PISCES (see below) is the most accurate, but most complex model. It fully models buffer occupancies, credits, and crossbar arbitration for each packet. PISCES should be the most accurate, but is also the most likely to exhibit "false positives" for severe congestion due to implementation bugs or model failures. In general, The network models in SST/macro are designed to form a hierarchy of increasing accuracy and increasing complexity. To understand the approximations underlying SCULPIN, we introduce the detailed model PISCES and the corresponding approximations made to derive SCULPIN.

PISCES PISCES (Packet-flow Interconnect Simulation for Congestion at Extreme-Scale) models individual packets moving through the network. Flits (flow-control units) are approximately modeled using flow-like approximations. Packets can have partial occupancies in several different buffers, approximating wormhole routing. However, arbitration is modeled on whole packets, not individual flits.

1. A message (flow) is broken up into packets. Depending on available space in the Tx buffer, a limited number of packets may be able to queue up in the buffer. If credits are available in the Rx buffer for the link and the link is idle, the packet moves into the next Rx buffer after a computed delay.
2. The router selects a path for the packet and the packet requests to the crossbar to transmit to the corresponding output port. If credits are available for the Rx buffer, the crossbar may select the packet in arbitration and move it to the output buffer. After moving, the Rx buffer returns credits to the previous Tx buffer for that packet.
3. Step 1 is repeated for the next Rx buffer, waiting for credits and link availability.
4. Repeat Step 2
5. Repeat Step 3
6. Packet arrives in NIC Rx queue and queues waiting to inject into local memory. After injection, the Rx buffer returns credits to the corresponding Tx buffer.

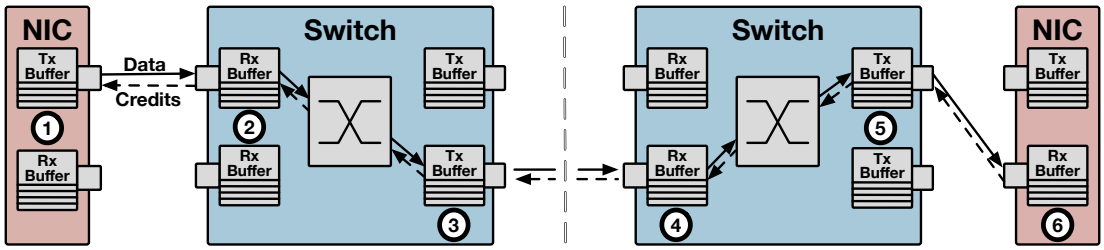

Figure 4: PISCES (Packet-flow Interconnect Simulation for Congestion at Extreme-Scale) models individual packets moving through the network. Flits (flow-control units) are approximately modeled using flow-like approximations. For details on numbered steps, see text.

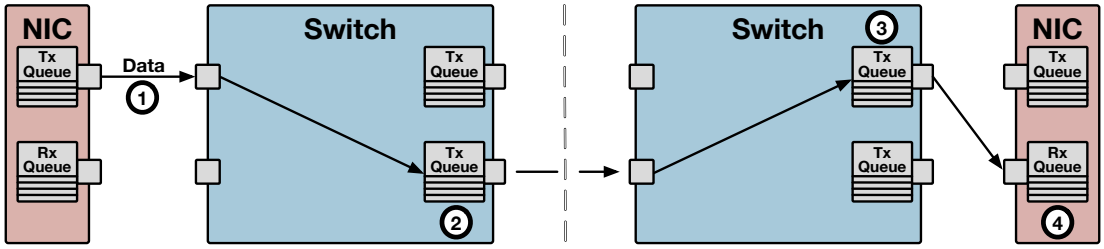

Figure 5: SCULPIN (Simple Congestion Unbuffered Latency Packet Interconnection Network) models the main source of contention in today's networks occurring on the output port ser/des. For details on numbered steps, see text.

SCULPIN Under current architectural trends, switches have ample buffer space and crossbar bandwidth, making the mostly likely bottleneck edge bandwidth through the output ports. SCULPIN (Simple Congestion Unbuffered Latency Packet Interconnection Network) models the main source of contention in today's networks occurring on the output port ser/des. Unlike PISCES, individual flits are not able to wormhole route across links interspersed with flits from other packets.

1. A message (flow) is broken up into packets. Each packet waits in the queue to send based on link availability and QoS.
2. After being selected, the packets are forwarded to the switch. Packets are immediately routed to the correct output port, skipping crossbar arbitration. Packets wait in unbounded queues, thereby assuming sufficient buffer space is always available.
3. Repeat Step 1. Packet waits in queue until link becomes available based on QoS. Packet is immediately forwarded to next output port, skipping arbitration
4. Repeat Step 1.
5. Packet arrives in NIC Rx queue (no credits, buffer assumed to always have space). Packets queue waiting to inject into local memory.

Adaptive Routing SST/macro provides numerous different adaptive routing methods for different topologies. The main routing methods are:

- Minimal (all topologies)
- UGAL: Universal globally adaptive load-balanced (Dragonfly, Dragonfly+)

- PAR: Progressive adaptive routing (Dragonfly, Dragonfly+, HyperX)
- Scatter/D-mod-K (Fat-tree)

UGAL and PAR are capable of "misrouting" off the minimal path to avoid congested links. Both methods misroute to a random intermediate switch. Minimal paths area taken from source to intermediate and again from intermediate to destination. UGAL makes non-minimal routing decisions at the source switch only and cannot dynamically misroute along the way. PAR, in contrast, can misroute from the minimal path at each step. PAR overcomes the major drawback of UGAL, namely that congestion detected via backpressure is often not evident at the source and is only detected after one or more minimal steps. This is particularly critical for dragonfly topologies. In general, UGAL requires fewer virtual channels than PAR since misrouting only occurs at the source. Other UGAL-based routing methods such as piggybacking congestion information on packets to provide more accurate congestion estimates at the source have been described, but are not explored here. Instead, PAR is taken as a proxy for adaptive routing with "accurate congestion detection."

For fat-tree, all routing decisions occur at the source. Dynamic congestion information is not taken into account. However, at the injection switch, traffic is scattered across all up paths to avoid congestion. Scattering occurs across all ports regardless of src-dst pair. The SST/macro fat-tree models, despite using only "minimal" paths does not assume a fixed, static routing table.

3.4 SST - MERLIN/EMBER

As discussed above, Merlin provides a set of components that can be used to model a detailed network fabric while Ember provides traffic generation. Merlin has a large set of tunable parameters and supports a variety of topologies while Ember has a large library of traffic motifs.

The Merlin and Ember SST models have been used to study a wide range of architectural and system features involving the network from the impact of global link organization in a dragonfly network on scheduling algorithms [14], to the possibility of power saving in a large scale (110,575 node) dragonfly [16].

The primary Merlin component for this simulation is a high radix router model called hr_router. It models a single lossless high radix switch. Input and output buffers are explicitly modeled with fixed depths, and the crossbar is modeled on a cycle by cycle basis. The topology is determined by loading a subcomponent which implements the routing algorithm for the simulation. Congestion information is communicated to the topology object so it can make adaptive routing decisions. The congestion information is available as either queue occupancies or remaining credits in the output buffer. The queue occupancy information can be configured along two perpendicular axes. First, it can supply the occupancy of just the output buffer in the current router, or it can be supplied as an estimate of the current router output buffer plus the next routers input buffers. Second, the occupancy can be supplied per virtual channel or aggregate across the entire port. The dragonfly model currently uses remaining credits for adaptive routing. The hyperX model uses occupancy as its routing metric, all was configured to use local information per virtual channel.

The endpoints are modeled using the Ember and Firefly libraries. Ember provides a state machine model of the communications to be modeled (called a motif) and Firefly models the communication software stack and network interface. The ember motifs have varying complexities from simple patterns to more complete computations. The motifs are composable so more complex applications and workflows can be built up from simple building blocks.

4. BENCHMARK COMMUNICATION PATTERNS

This study looked at three benchmark communication patterns to help gain a deeper understanding of how the various network parameters will impact applications that use these patterns. The three patterns used, all of which are implemented in the MPI programming model, are described below. As noted, the three simulation models use different forms of the benchmarks: MPI traces for TraceR/CODES, skeletonized codes for SST/Macro and motifs for SST/Merlin.

4.1 NEAREST NEIGHBOR (HALO3D)

The nearest neighbor communication pattern is used in stencil codes. For this study, a 27-point 3D stencil was used. The stencil is formed of a 3x3x3 neighborhood and each process communicates with the neighbors

on the face of the cube formed around it as the center process. The specific instance used is Halo3D. The per process grid size used for the simulations was nx=128, ny=128, nz=128, with nvar = 10 (variables per grid point). The source code used by each simulator is shown in Table 3.

Tracer/CODES	`https://github.com/sstsimulator/ember/tree/master/mpi/halo3d-26`
SST/Macro	`https://github.com/jjwilke/ember` forked from `https://github.com/sstsimulator/ember`
SST/Merlin	`https://github.com/sstsimulator/sst-elements/tree/master/src/sst/elements/ember/mpi/motifs`

Table 3: Links to source code used by each simulator for Halo3D benchmark.

4.2 WAVEFRONT (SWEEP3D)

Wavefront communication patterns are seen in some particle transport codes. The pattern starts at a corner of the grid and "sweeps" out in a wavefront, in a pipelined manner. The benchmark used for this pattern was Sweep3D configured with the following per process parameters: nx=40, ny=40, nz=1000, KBA=10. The source code used by each simulator is shown in Table 4.

Tracer/CODES	`https://github.com/sstsimulator/ember/tree/master/mpi/sweep3d`
SST/Macro	`https://github.com/jjwilke/ember` forked from `https://github.com/sstsimulator/ember`
SST/Merlin	`https://github.com/sstsimulator/sst-elements/tree/master/src/sst/elements/ember/mpi/motifs`

Table 4: Link to source code used by each simulator for Sweep3D benchmark.

4.3 SUBCOMMUNICATOR ALL-TO-ALL (SUBA2A/FFT3D)

In this pattern, a 3D grid is decomposed along the X and Y dimensions, and subcommunicators are formed along a 1D line in both X and Y. Each process communicates in an all-to-all manner within each of it's subcommunicators, one dimension at a time. This pattern represents the shuffles that happen in a multidimensional FFT. Two different benchmarks were used to represent this pattern:

1. SubA2A: all-to-all messages within subcommunicators created along X and Y dimensions in a 3D grid of MPI processes as observed in distributed multi-dimension FFT computations. Message size exchanged between every pair within the all-to-alls: 32 KB, each sub-communicator contains 32 processes. Used by TraceR/CODES and SST/Macro.

2. FFT3D: motif to mimic the communication pattern of a 3D FFT. The motif decomposes the 3D space into a 2D square. For 32k nodes, this square is 181x181, for the other two sizes, the grid is 180x180. The reason for the difference is that the motif will not partially allocate a node. The pattern does four all-to-all messages with subcommunicators, plus a barrier. If first does the all-to-all with rows, then columns, then a barrier, then another all-to-all in columns then in rows. This motif is used by SST/Merlin.

   The source code used by each simulator is shown in Table 5.

Tracer/CODES	`https://github.com/LLNL/chatterbug/tree/master/subcom3d-a2a`
SST/Macro	`https://github.com/jjwilke/ember` forked from `https://github.com/sstsimulator/ember`
SST/Merlin	`https://github.com/sstsimulator/sst-elements/tree/master/src/sst/elements/ember/mpi/motifs`

Table 5: Links to source code used by each simulator for SubA2A/FFT3D benchmarks.

5. RESULTS

Each of the simulators was used to sweep across the parameter space described above. The one exception is that SST/Merlin does not currently support the Megafly topolgy. The results are shown as bar graphs of normalized time for executing a single interation of the benchmark. Results are shown for both linear and random placement and times are normalized to the fat-tree configuration with 8k nodes, 200 Gpbs injection bandwidth and linear allocation.

The results for and conclusions from the runs from each simulation model are shown independently in the following sections. Overall conclusions across simulators are in Section 6.

5.1 TRACER-CODES

Since the choice of routing scheme can have a significant impact on the observed performance, we list the routing scheme used by CODES:

- HyperX: minimal-path adaptive routing; no detours used.
- Dragonfly: progressive adaptive routing; hybrid of minimal and non-minimal paths.
- Dragonfly+: progressive adaptive routing; hybrid of minimal and non-minimal paths.
- Fat-tree: adaptive routing; least congested path for upward-downward traversal.

For each benchmark, two placements have been tested: 1) a randomized placement in which MPI ranks/processes are mapped to nodes in an arbitrary fashion; note that all processes mapped to a node are assigned consecutive ranks. 2) a linear placement is which processes are assigned consecutively to near-by nodes based on network hierarchy.

5.1.1 Halo3D

The simulation results for Halo3D are shown in Figure 6.

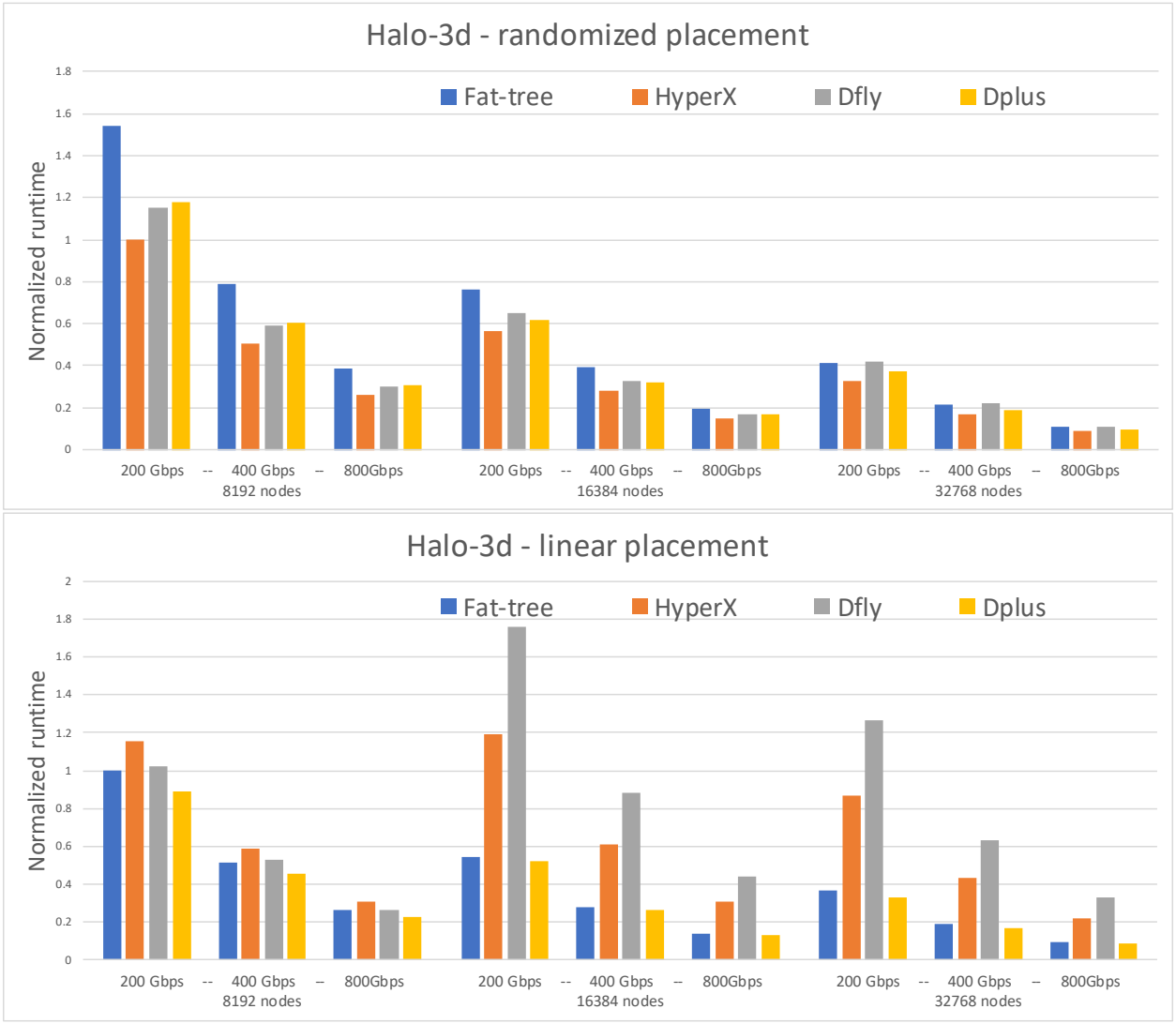

Figure 6: TraceR-CODES results for the halo3d26 motif using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

The data shows a significant positive impact of higher total network bandwidth/capacity for Halo3D. This manifests in two ways: first, increased network bandwidth for a given system size shows a large performance gain; second, increased system size for the same injection bandwidth shows a moderate performance gain. This second effect is likely due to an increase in network bandwidth per compute ratio. These trends hold for both linear and random placement.

The various topologies also show differences in performance compared to each other. The hyperX topology shows the best performance for random placement, but performs poorly for linear placement. This is likely due to the fact that minimal routing is used, which doesn't allow enough path diversity to route around congestion caused by the neighborhood traffic. Though similar in structure, dragonfly+ outperforms dragonfly when using linear placement. This may be due to the choice of adaptive routing algorithms for the dragonfly and will be investigated as part of future work. Fat-tree's adaptive routing fails to get the best performance in worst-case like random placement scenario (static routing does even worse).

5.1.2 Sweep3D

The simulation results for Sweep3D are shown in Figure 7.

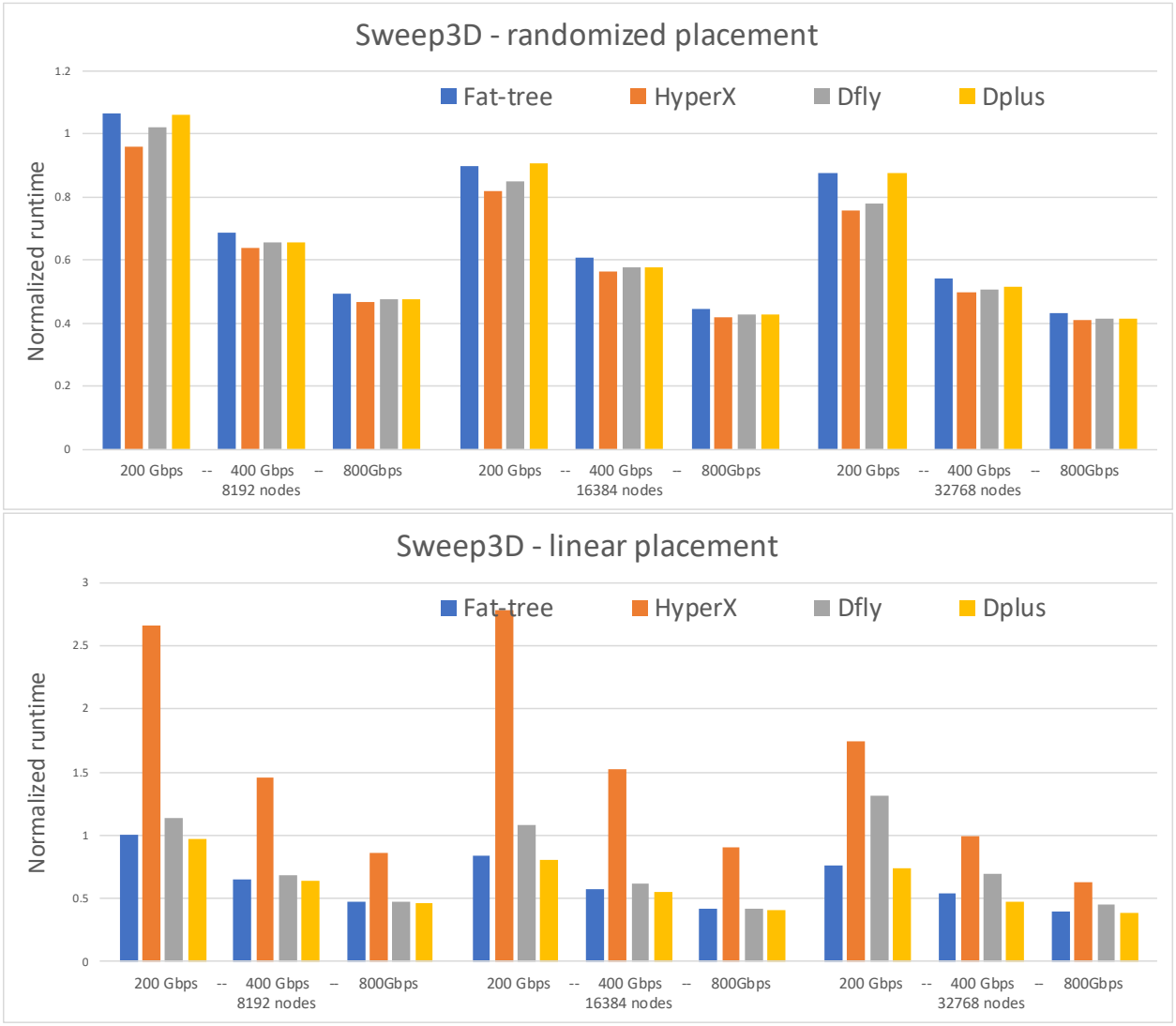

Figure 7: TraceR-CODES results for the halo3d26 motif using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

Compared to Halo3D, Sweep3D is much less sensitive to network parameters, though some difference can be noted. First, the different topologies have very similar performance characteristics for a give machine size and injection bandwidth. This applies for both linear and random allocation. The one exception is the hyperX with linear placement, but, as explained above, this is likely due to the use of minimal only routing.

Using higher link bandwidths does show improved performance but the gains are not proportional to the increase in the bandwidth. For a given link bandwidth, higher node counts have minimal impact on performance.

5.1.3 SubA2A

The simulation results for SubA2A are shown in Figure 8.

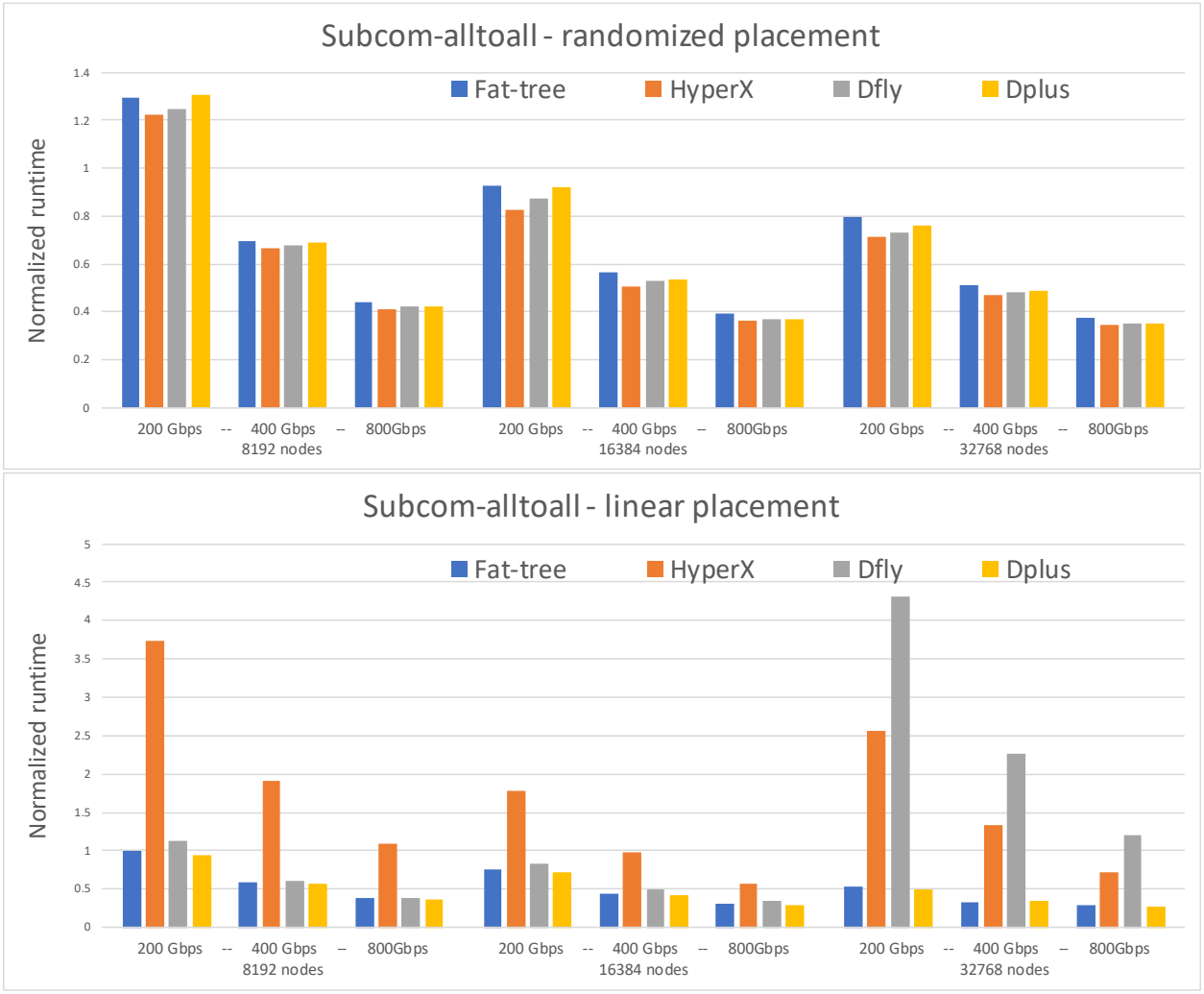

Figure 8: TraceR-CODES results for the halo3d26 motif using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

The SubA2A benchmark shows mixed sensitivities to network parameters. First, there is little difference between the performance provided by the various topologies, especially in the random case. However, substantial variation is seen in the linear case for hyperX at all sizes and dragonfly at 32k nodes. The extra network bandwidth from increased node count at the same link bandwidth provides some benefit, particularly at lower link bandwidths.

5.1.4 *Overall Observations*

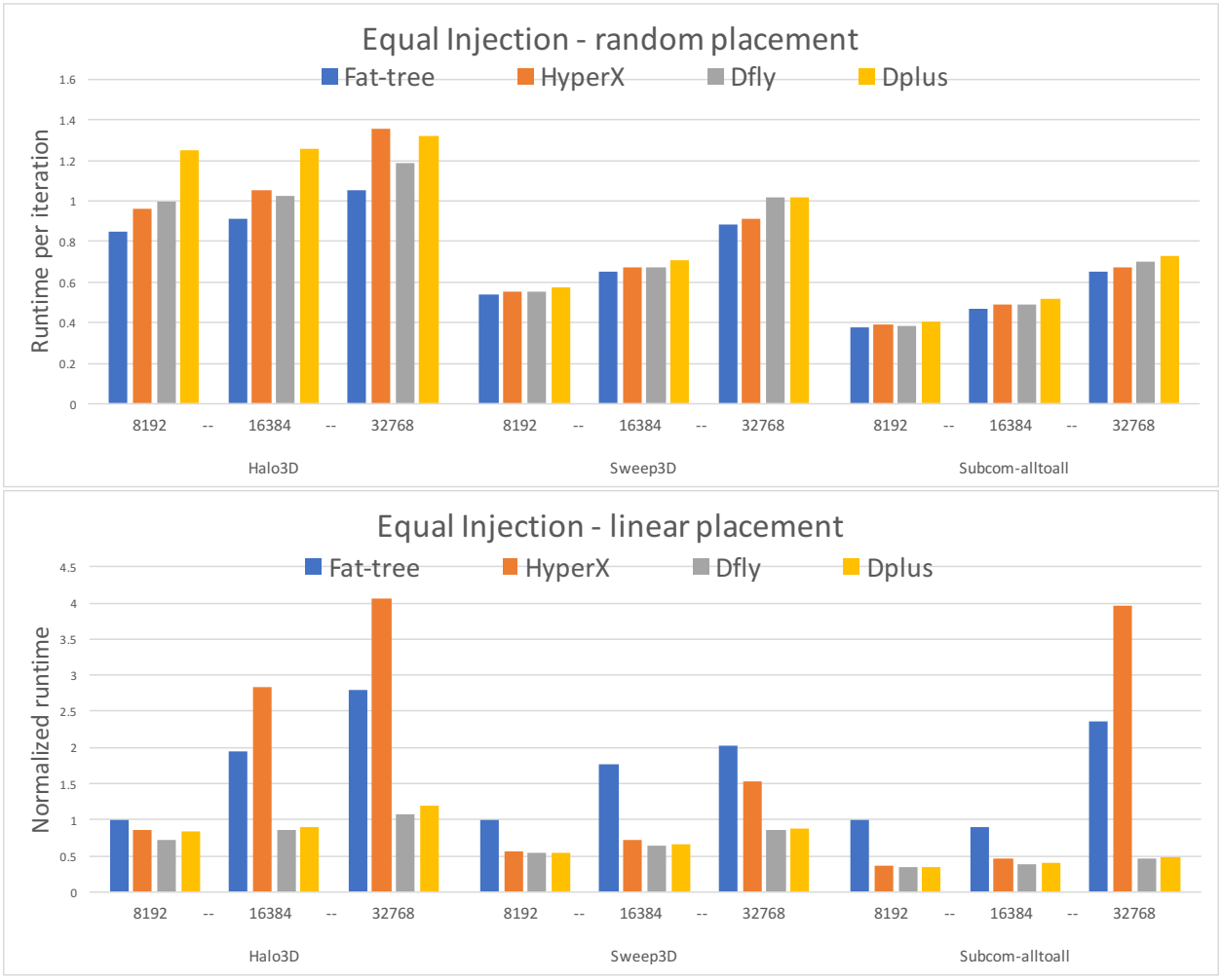

Figure 9: Tracer/CODES results comparing configurations with equal injection bandwidth per rank (8k/800 Gbps, 16k/400 Gbps and 32k/200 Gbps) for each of the benchmarks.

Overall, we make the following observations from the TraceR/CODES results:

1. For fat-tree, adaptive routing is a must to exploit its capabilities, especially if linear-like placement cannot be guaranteed.
2. Dragonfly+ consistently outperforms dragonfly for the scenarios tested.
3. For HyperX, minimal adaptive routing is unable to address high congestion scenarios. Need for evaluation of hybrid adaptive routing such as those of dragonfly and dragonfly+.
4. Figure 9 compares the performance for the three benchmark for approximately iso-bandwidth systems: 8K nodes with 800 Gbps links, 16K nodes with 400 Gbps nodes, and 32 K nodes with 200 Gbps links. The results suggest that use of a higher node count negatively impacts performance vis-a-vis use of higher bandwidth links at a smaller node count. The effect is consistent for randomized placement and shows similar trend for all benchmarks and topologies. For linear placement, the fat-tree and hyperX topology is impacted significantly more.

5.2 SST - MACRO

In addition to the parameters outlined in Section 2 the results were also run with two different classes of routing algorithms: minimal and adaptive. For adaptive routing results, progress adaptive routing is used for HyperX, Dragonfly, and Dragonfly+ while an oblivious routing scheme is used for fat-tree to spread traffic across up paths. For the minimal routing results, minimal routing is used for HyperX, Dragonfly, and Dragonfly+ while an oblivious routing scheme is used for fat-tree to spread traffic across up paths.

5.2.1 Adaptive Routing

The first set of results in Figures 10-12 show the performance results using adaptive routing for Halo3D, subcommunicator all-to-all, and Sweep3D benchmarks.

Halo3D In general, little performance variation is observed across the topologies for Halo3D regardless of linear or random placement. The notable exception is Dragonfly, for which performance significantly degrades at 16K and 32K configurations. At 8K, each individual switch is "fully connected" having at least one global link directly connected to all other groups. At 16K, each switch no longer has a sufficient number of ports to provide a global link to every other group. This dramatically increases the amount of *intra-group* traffic as most *inter-group* traffic is required to hop within a group. Improvements to the progressive adaptive routing (PAR) used for Dragonfly are likely required for these topologies. In particularly, the selection of "gateway" routers for group-to-group traffic likely needs to be better scattered to avoid inter-group traffic overwhelming a particular gateway.

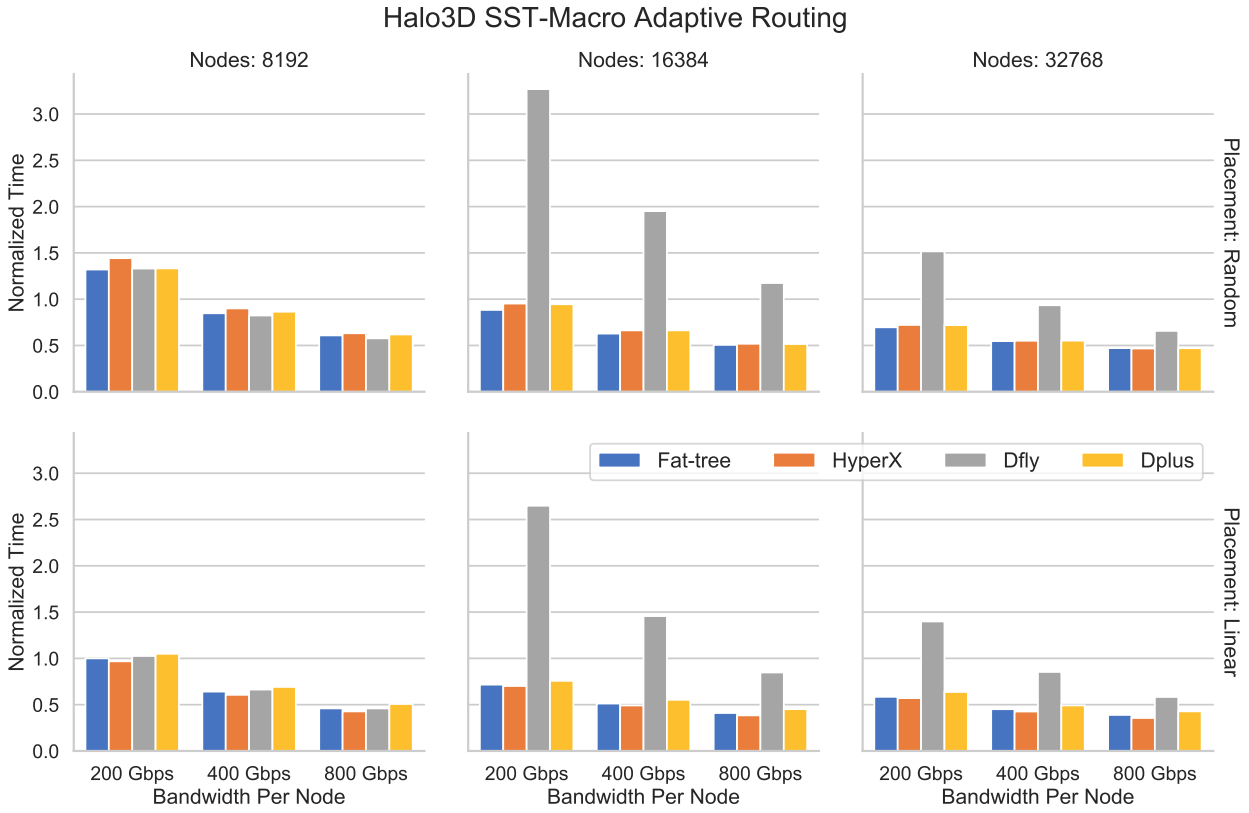

Figure 10: SST/macro results using adaptive routing for the Halo3D skeleton app using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

SubA2A For the subcommunicator all-to-all, fat-tree provides the best performance, although Dragonfly+ is competitive. Dragonfly suffers the same performance problems as in Halo3D. HyperX, which matched or slightly exceeded fat-tree performance for many Halo3D cases, now shows significantly worse performance - particularly for random indexing. Improvements to the adaptive routing for HyperX may be possible, although the SST/macro routing scheme already allows a mis-routing in each dimension.

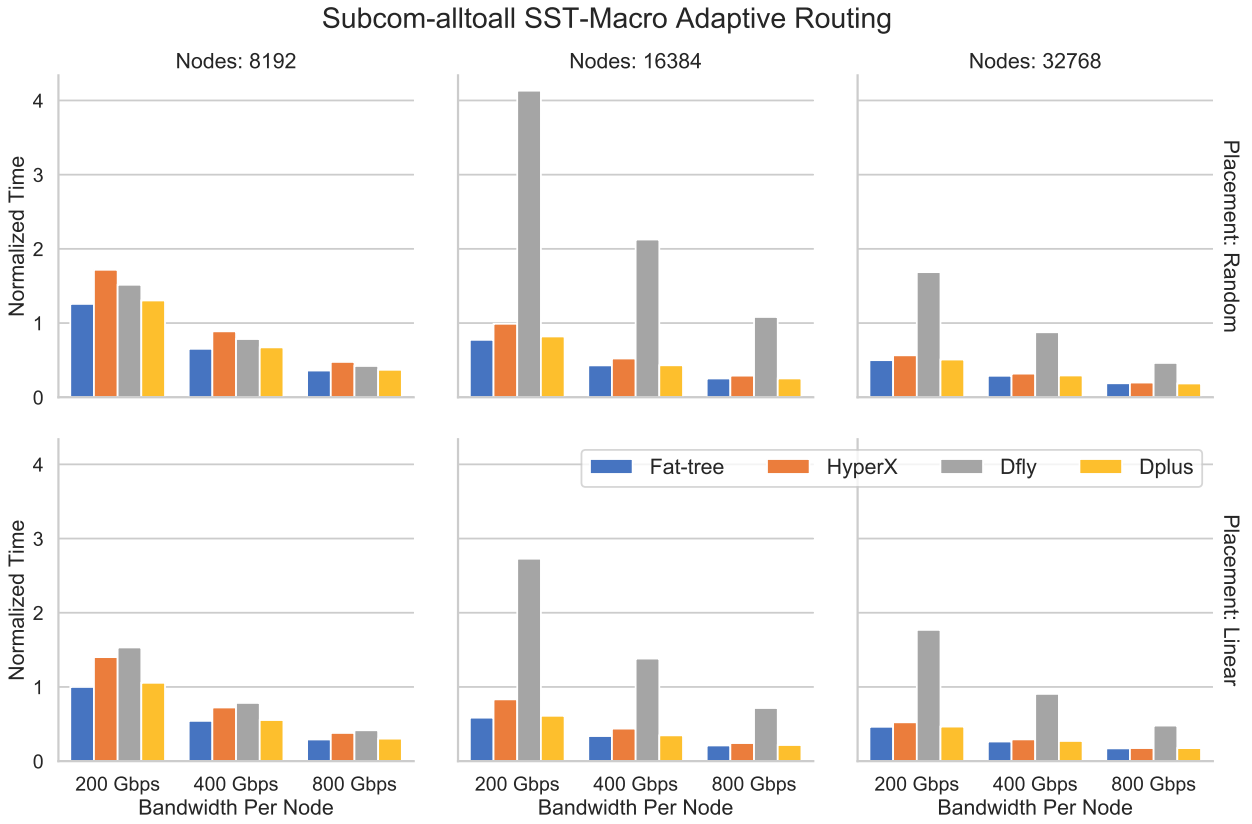

Figure 11: SST/macro results for the subcommunicator all-to-all skeleton app using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

Sweep3D Sweep3D more closely resembles Halo3D. Little performance variation is seen, with the exception of Dragonfly. HyperX, which showed much worse performance for subcomm-a2a, shows slightly improved performance relative to fat-tree.

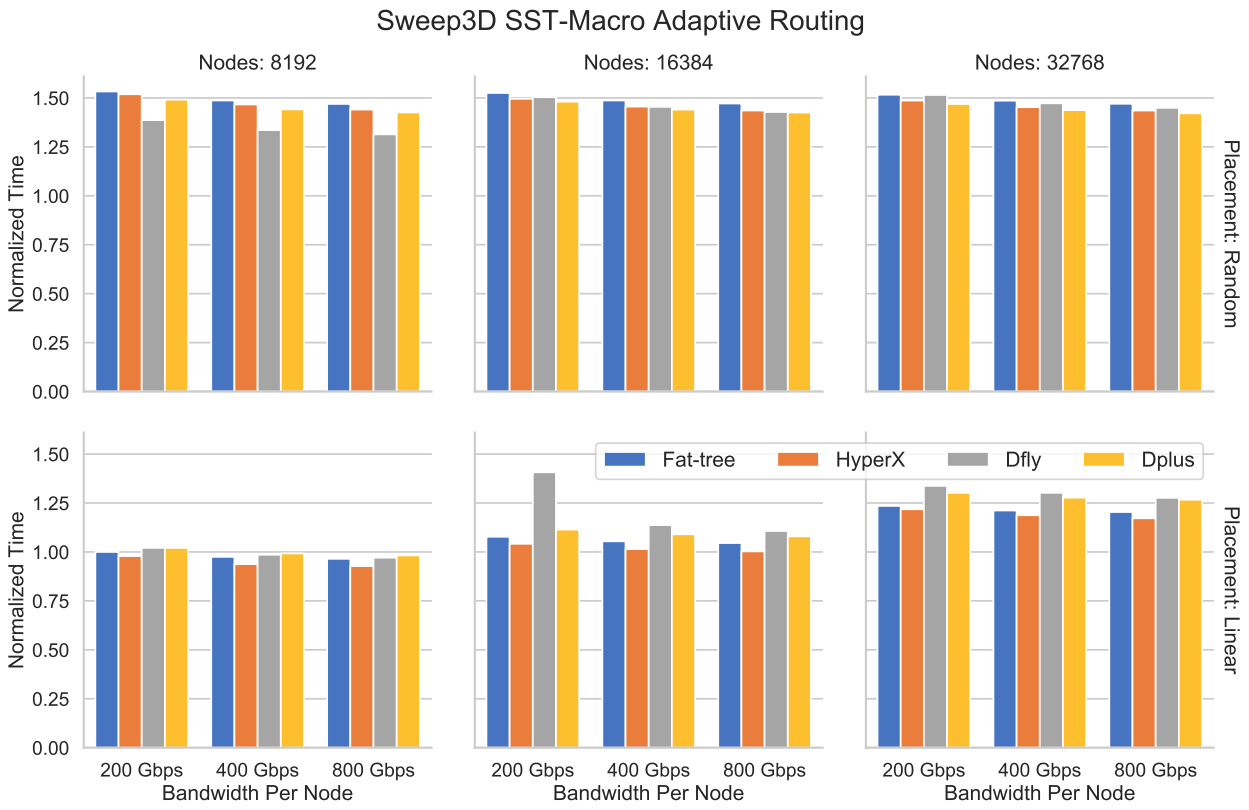

Figure 12: SST/macro results using adaptive routing for the Sweep3D skeleton app using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

5.2.2 Minimal Routing

To demonstrate performance with commodity equipment that may not support adaptive routing (and for comparison to the other simulators which only support minimal routing), we repeat the results for the benchmarks enforcing minimal routing. Fat-tree again uses a scattering scheme across all available paths and thus demonstrates an "optimistic" case relative to the other topologies which are not as easily able to exploit path diversity.

Halo3D For Halo3D, many of the topologies perform surprisingly well with only minimal routing - particularly Dragonfly+. However, all of the topologies (except fat-tree) have "pathological" cases where performance seriously degrades. For example, Dragonfly+ performance suffers at 32K when only minimal routing is enabled. HyperX performance suffers most at 8K and 16K.

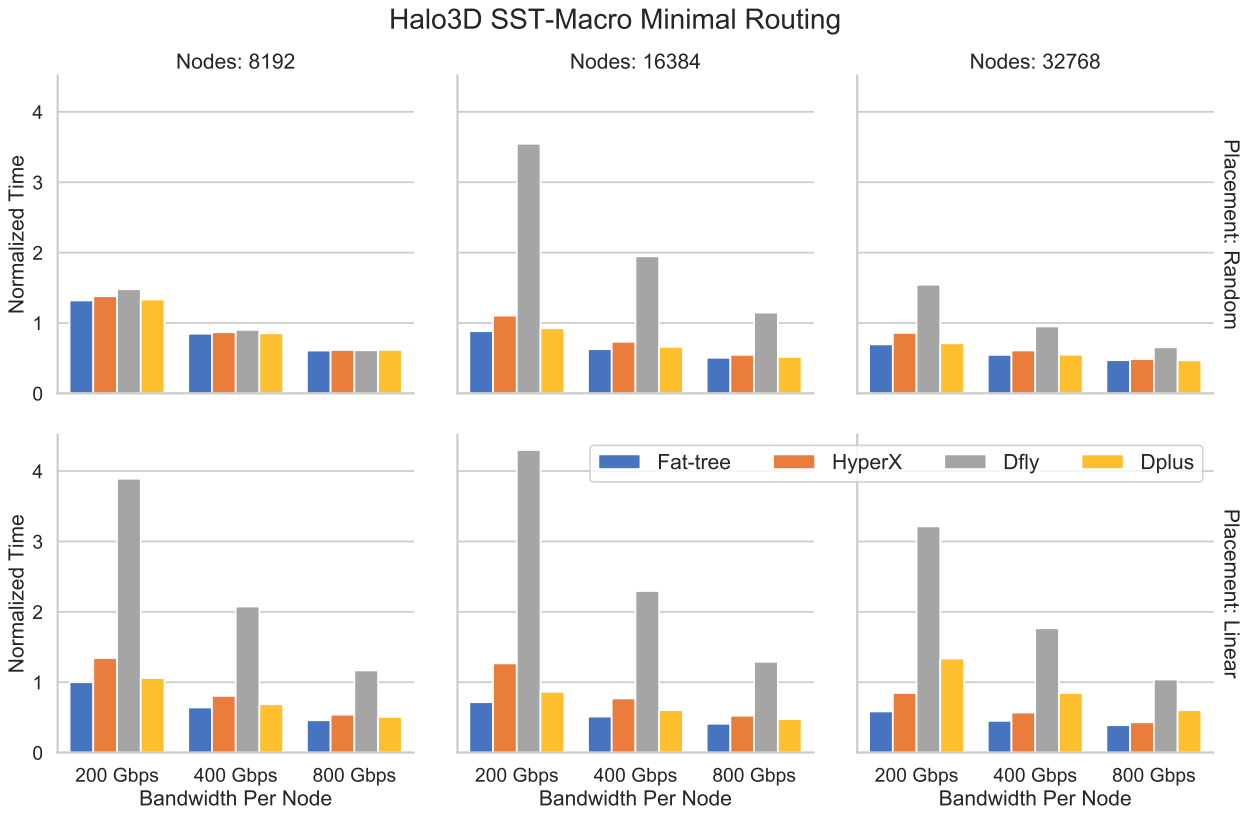

Figure 13: SST/macro results using minimal routing for the Halo3D skeleton app using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

SubA2A/Sweep3D This story largely repeats for the subcommunicator all-to-all and Sweep3D apps. HyperX with minimal routing is notably poor for the 8K all-to-all benchmark. Dragonfly+, while not fully matching fat-tree for performance, is often only slightly worse despite only using minimal paths.

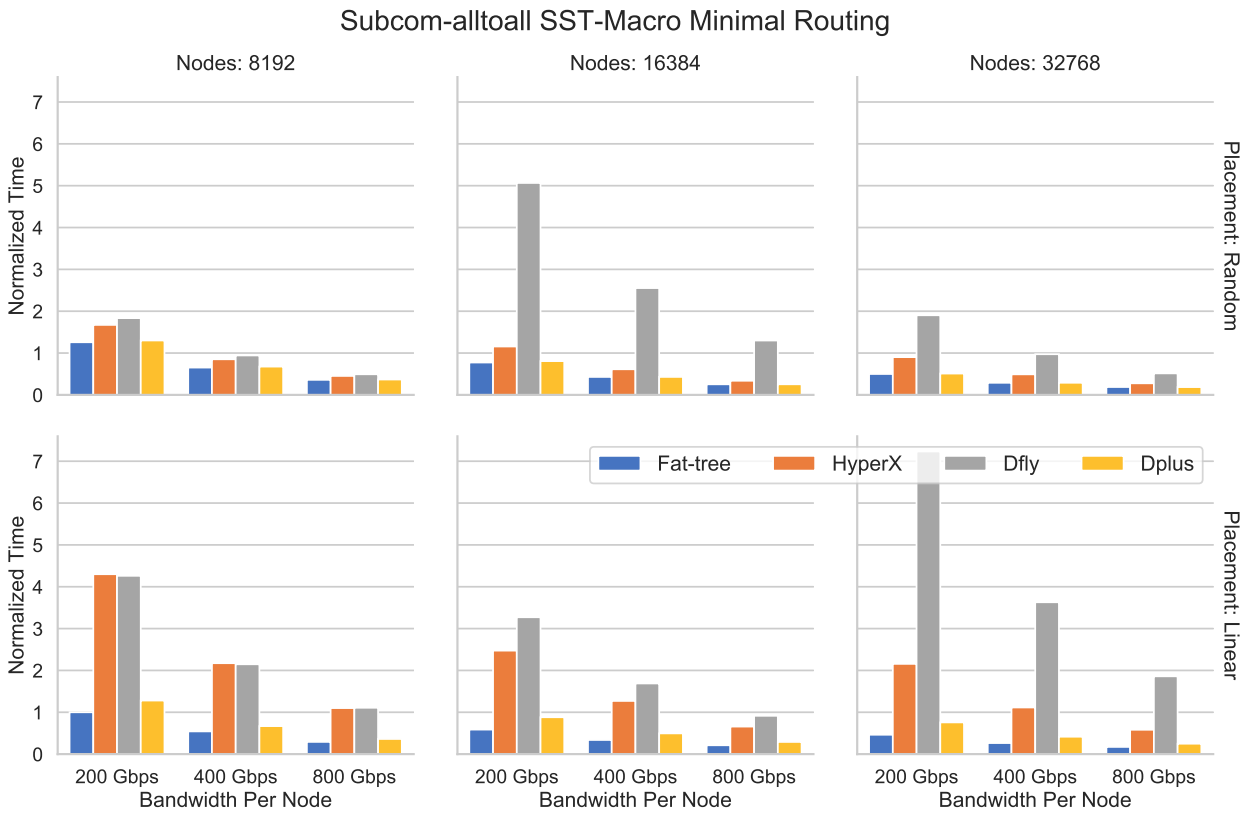

Figure 14: SST/macro results for the subcommunicator all-to-all skeleton app using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

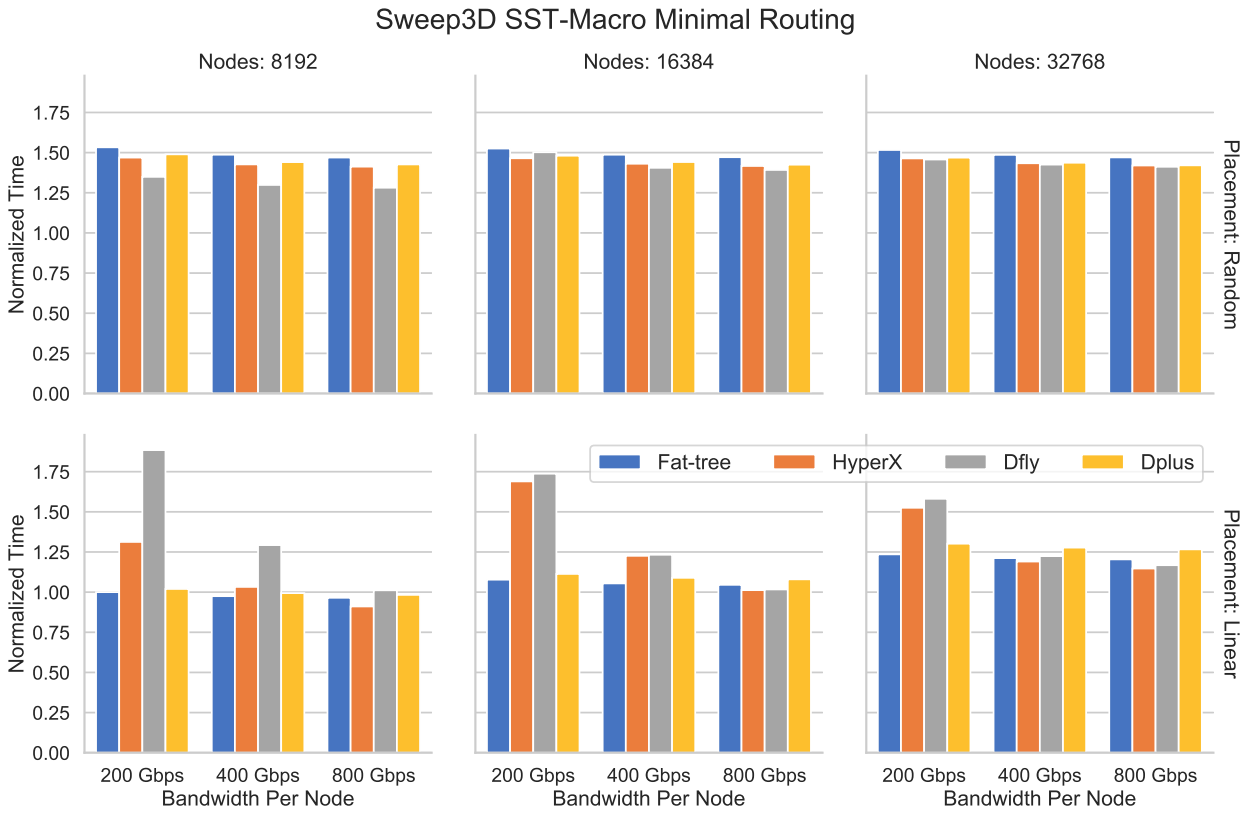

Figure 15: SST/macro results using minimal routing for the Sweep3D skeleton app using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

5.2.3 Network Size

Three different network sizes have been considered from 8K to 32K nodes. Although quad-rail (800Gbps effective link bandwidth) to single rail (200Gbps) results are shown for each topology, a quad-rail network for 8K nodes is "equivalent" to a single-rail network for 32K nodes. For a given topology, these configurations will have the same number of ports and switches and therefore an equal injection bandwidth *per MPI rank*, since the 8K system is assumed to have "fat nodes" with 4x compute relative to the 32K system. The 8K and 16K system have multiple MPI ranks per node and the benefit of multiple rails. Given such networks of the same total size (port count/switch count), Figure 16 shows performance for the different configurations. The inter-node traffic reduction and multiple rails improves performance, with the 8K configuration outperforming both 16K and 32K configurations. Some topologies and apps degrade more in performance as size increases. Dragonfly performance is very poor for 16K and 32K for the reasons outlined in 5.2.1 Fat-tree, e.g., degrades significantly at 32K for the subcomm all-to-all benchmark.

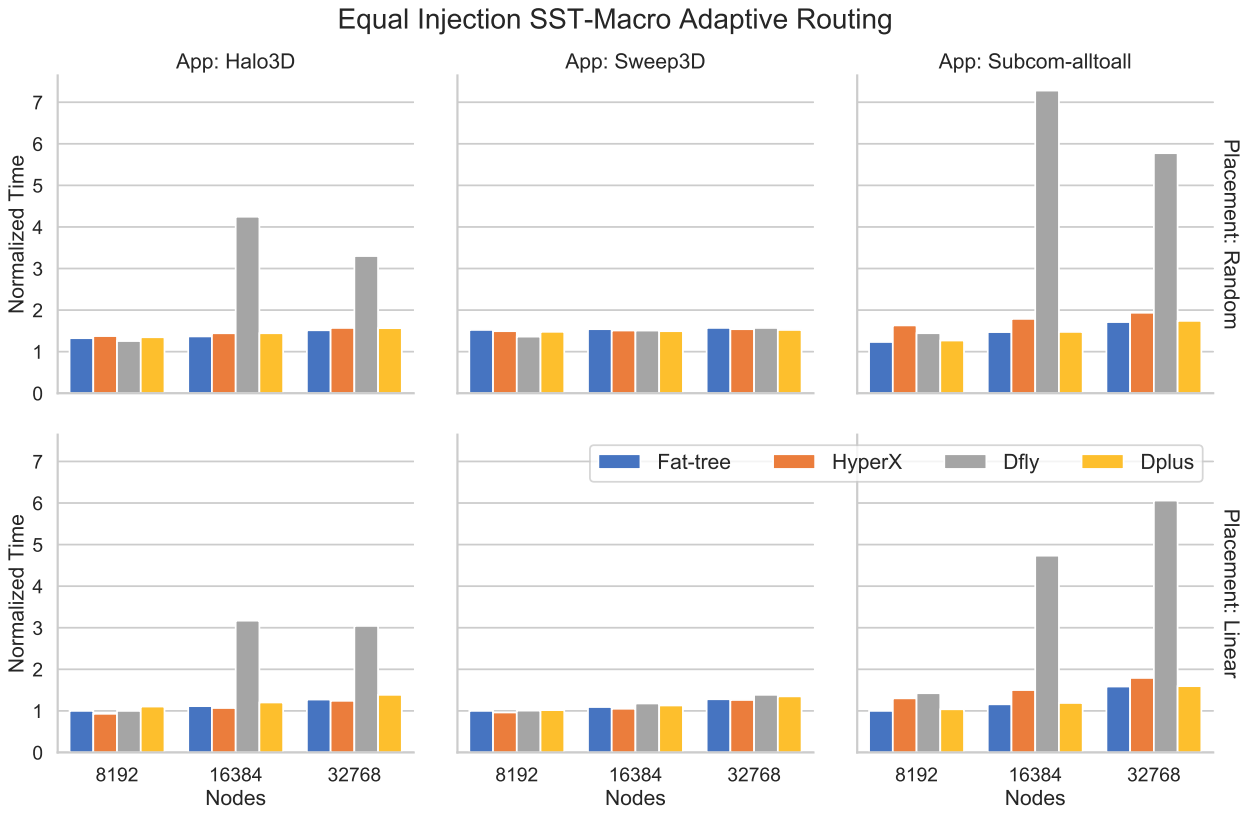

Figure 16: Comparison of network performance for different network sizes normalized to equal injection bandwidth per MPI rank. Quad-rail 8192, dual rail 16384, and single rail 32768 topologies are shown. Each individual topology is normalized across the sizes to have the same number of ports/switches and total bandwidth.

5.3 SST - MERLIN/EMBER

The SST merlin models provide multiple routing algorithms for both the hyperX and dragonfly topologies, but, currently, only a single deterministic routing algorithm for fat-tree. Also note that megafly/dragonfly+ is not yet supported in SST, but the models are currently under development. The results in this section use the following algorithms:

- HyperX: dimension ordered adaptive routing (DOAL): packets traverse the network in dimension order, but may take one adaptive route per dimension if congestion is encountered. Requires 2 virtual channels for deadlock free routing.
- Dragonfly: uses a version of UGAL routing: a packet takes either a minimal route or a valiant route (to a random group) as determined by the congestion in the first router. Requires 3 virtual channels for deadlock free routing.
- Fat-tree: deterministic routing: each packet takes a deterministic route from source to destination, no adaptation. Requires 1 virtual channel for deadlock free routing.

Each benchmark was run with both linear and random placement. For linear placement, the MPI ranks are mapped linearly onto the physical nodes. For random placement, ranks within a node are placed linearly, but nodes are placed randomly on the endpoints in the topology.

5.3.1 *Halo3D*

The results for the Halo3D benchmark are shown in Figure 17.

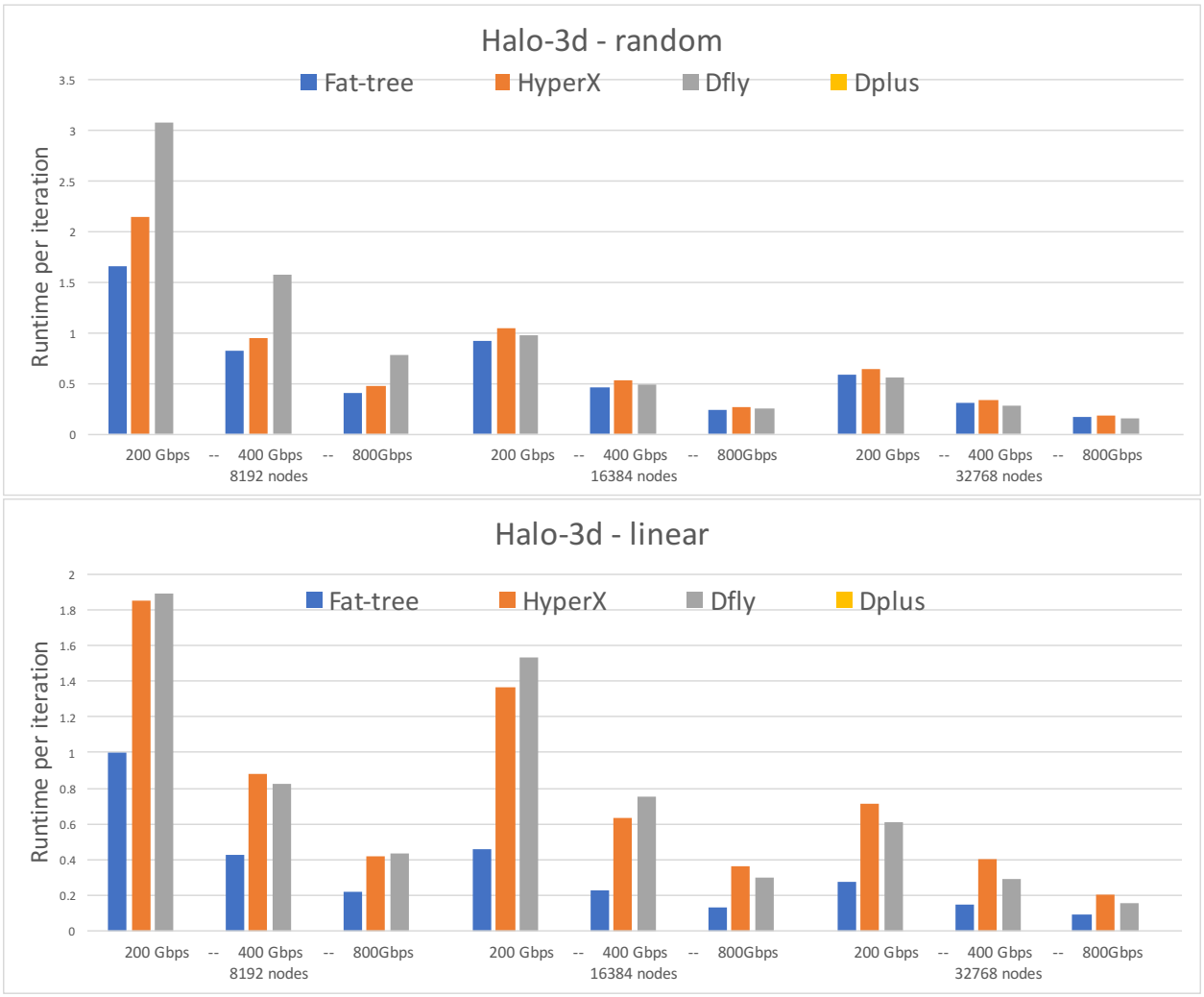

Figure 17: SST/merlin results for the halo3d26 motif using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

The results show that applications doing halo exchanges are extremely sensitive to network parameters. There are two major trends that can be noted from the data for both linear and random placement. First, higher link bandwidth translates into better performance. Second, more endpoints, even at the same total system compute power and rank count, gives better performance at the same link bandwidth. This is likely due to there being more total available bandwidth in the network due to having more network links.

For random placement, the choice of topology made little difference, except for dragonfly at 8192 nodes, where there is a large variation. For linear placement, the fat-tree topology has a clear advantage at all network sizes and injection bandwidths. It's interesting to note that the effect of random versus linear placement on performance varies for the different topologies. The fat-tree always does better with linear placement, while hyperX and dragonfly more often do better with random placement (with the notable exception of a dragonfly at 8192 nodes). We believe this is because random placement spreads out the traffic on the network allowing for more packets to take direct instead of indirect routes. For linear traffic, much of the traffic from physically close nodes targets another set of physically close nodes, so, much of the traffic is contending for the same set of links, resulting in more non-minimal traffic as adaptive routing kicks in.

5.4 SWEEP3D

The results for the Sweep3D benchmark are shows in Figure 18.

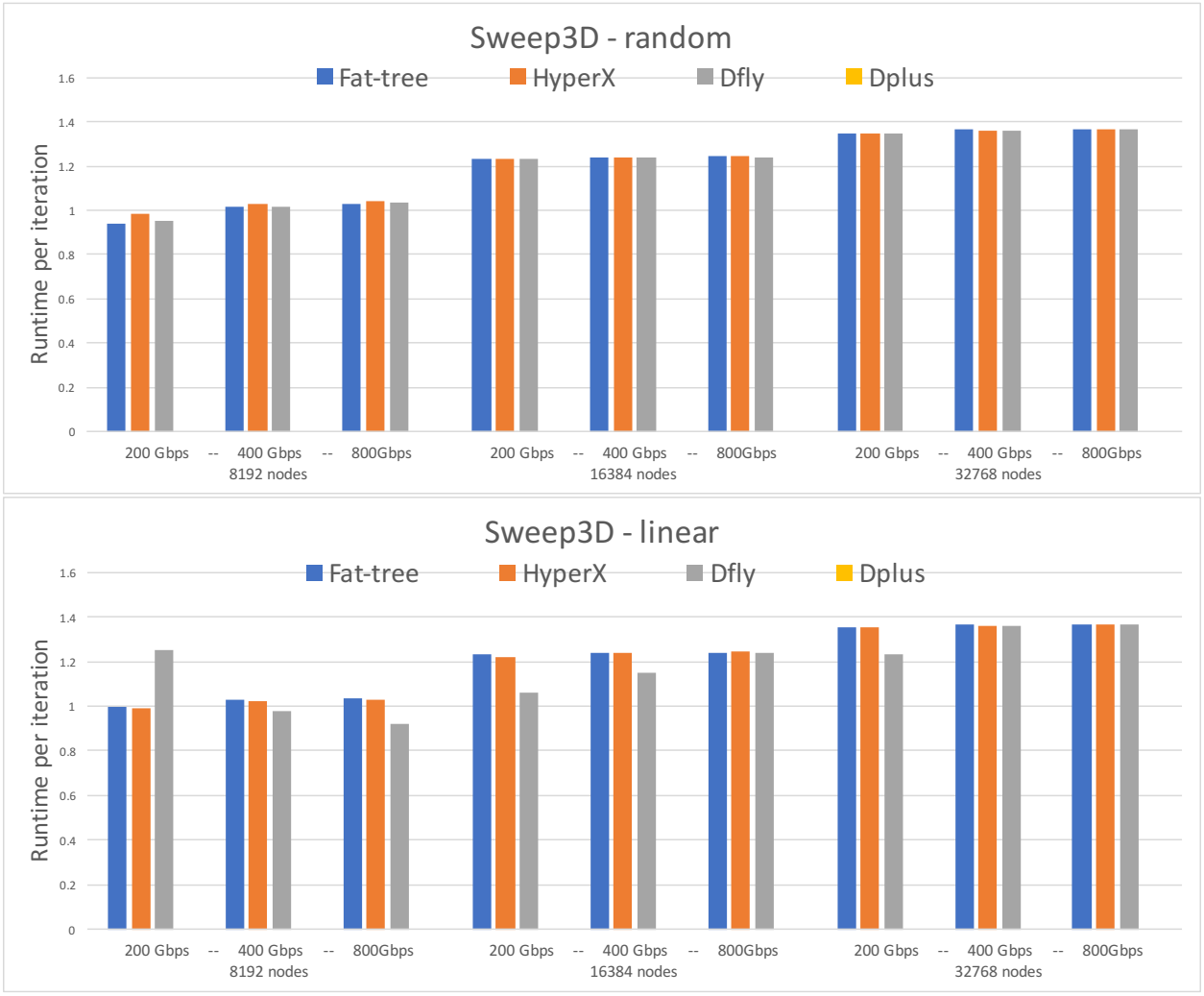

Figure 18: SST/merlin results for the sweep3d motif using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

Sweep3D appears to be relatively insensitive to most network parameters. Within a given system size, there is little impact to varying the link bandwidth, topology or allocation strategy. However, there is an impact of network size. The data shows a clear trend of lower performance as network size increases across both random and linear placement. The impact is an almost 40% increase in runtime from 8192 to 32768 nodes for both linear and random placements.

5.5 FFT3D

The results for the FFT3D benchmark are shown in Figure 19.

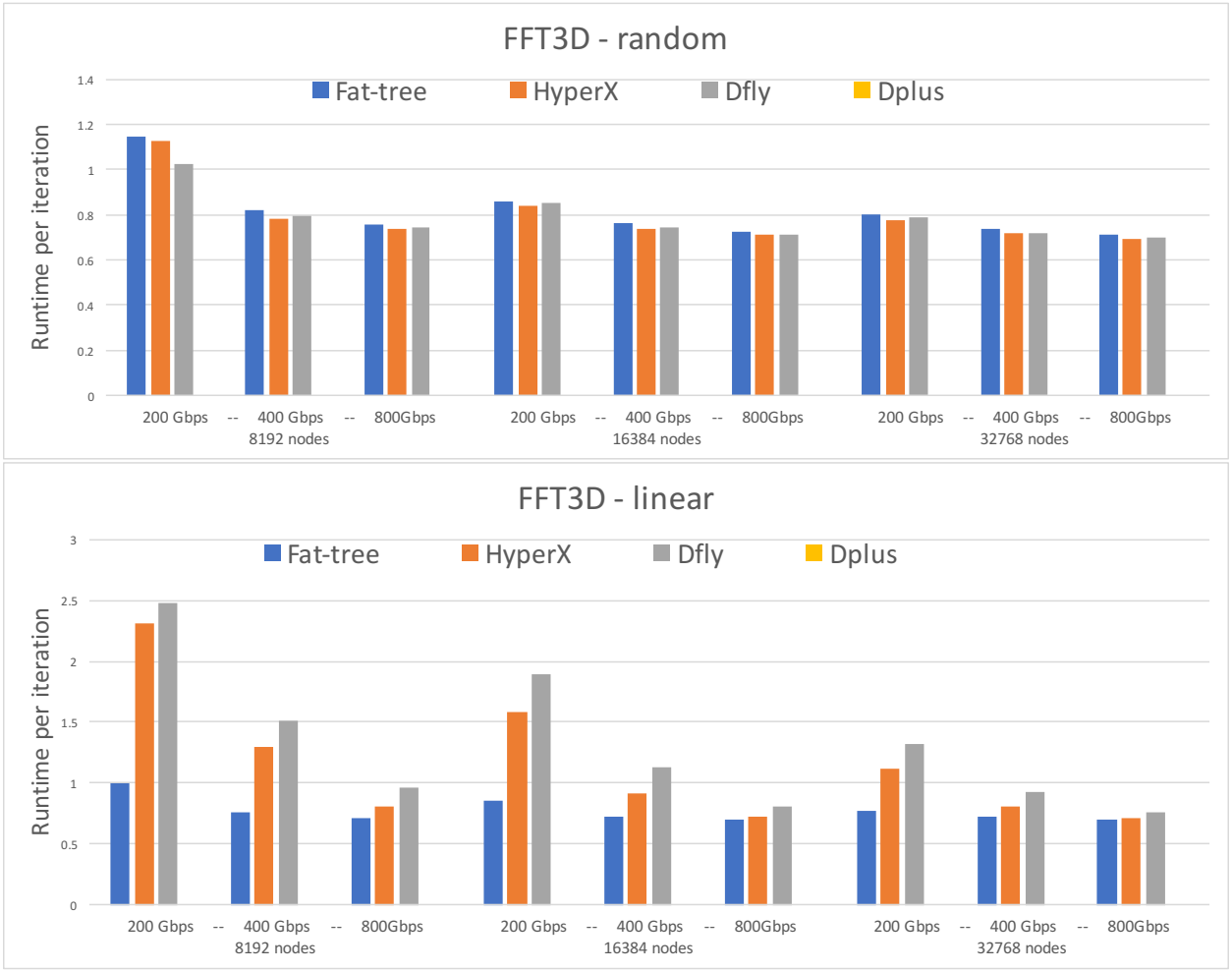

Figure 19: SST/merlin results for the FFT3D motif using both linear and random placement of ranks. Results are normalized to the 8192 node fat-tree with 200Gbps injection bandwidth results.

The FFT runs show sensitivity to injection bandwidth for some configurations, but not others. For example, linear placement shows that higher node count configurations have less performance impact across the various bandwidths for all topologies. For lower node counts, higher injection bandwidth can provide a substantially higher performance for hyperX and dragonfly, by the impact on fat-tree is still minimal.

Random placement leads to worse performance on the fat-tree, but better performance for both hyperX and dragonfly. This is likely due to the fact that the subcommunicator pattern limits the number of one node communications and the random placement allows hyperX and dragonfly to load balance the traffic with fewer indirect routes. For random placment, the performance differences seen across topologies are neglible.

5.5.1 *Network Size*

Figure 20 compares different machine sizes using the same injection bandwidth per MPI rank. With the exception of Halo3D, the choice of topology has only a minimal impact on performance. Most of the differences in the topologies could likely be mitigated with the choice of routing algorithms. Future work will verify the algorithms used with those being developed by the various vendors.

Random placement has a negative impact on the performance of the halo communication pattern, but has much less impact on the other two benchmarks, and in some cases slightly helps hyperX and dragonfly by naturally spreading out communications and relying less on non-minimal paths.

Also of note that fewer nodes a preferred over more nodes at the same bandwidth per compute, though the difference is minimal in most cases. This is likely due to more neighbor communications happening in shared memory for the more powerful nodes. It is important to note that fewer more powerful nodes does not allow you to reduce the injection bandwidth per compute.

Figure 20: SST/merlin results comparing configurations with equal injection bandwidth per rank (8k/800 Gbps, 16k/400 Gbps and 32k/200 Gbps) for each of the benchmarks.

6. CONCLUSIONS

For many of the studied configurations, the different models showed good agreement on the effects of changing the various parameters. Collating the data across the models show several trends.

For a system of a given size, increasing bandwidth positively impacts both the Halo3d and SubA2A communications, while Sweep3D was relatively insensitive to injection bandwidths.

Generally, linear allocation is better or as good as random allocation. In some cases, such as Halo3D, the penalty for random placement can be as large as 30-50% (see Figures 9, 16 and 20).

There were some large difference in results when comparing specific topologies across the three simulation models. We believe most of this difference can be attributed to the different routing algorithms used, pointing to the importance of routing algorithm choice.

Comparing across the system configurations that represent the same injection bandwidth per MPI rank (8k/800 Gbps, 16k/400 Gbps and 32k/200 Gbps) shows that there is little sensitivity to total node count, though the 8k system size does get slightly better performance than the larger system sizes. However, using fewer more nodes does not make up for a lower injection bandwidth per compute ratio. One of the biggest determining factors for performance is the bandwidth to compute ratio.

This performance trade-offs noted here will need to be measured against network cost before a final determination of the best network parameters for a system can be made.

7. FUTURE WORK

The studies in this milestone report were limited to single applications running across the entire machine on networks with fully configured global bandwidth. One piece of future work will look at the possible interference that could be seen when the machine has multiple applications running. Another study will look at the effects of reducing global/bisection bandwidth on the applications. We will also be adding new benchmark applications to the studies to help us understand the impact of interconnect architecture on a broader range of workloads.

Additionally, future work will attempt to understand the differences in the various simulation models. This could lead to improvements in the models, but could also simply help us understand in which regimes each simulation model functions best.

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-TR-756549).

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA0003525.

This work was supported by the US Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract DEAC52-06NA25396).

REFERENCES

[1] C.E. Leiserson. Fat-trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, 34(10), October 1985.

[2] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 77–88, Washington, DC, USA, 2008. IEEE Computer Society.

[3] Mario Flajslik, Eric Borch, and Mike A Parker. Megafly: A topology for exascale systems. In *International Conference on High Performance Computing*, pages 289–310. Springer, 2018.

[4] Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns. Enabling parallel simulation of large-scale HPC network systems. *IEEE Trans. Parallel Distrib. Syst.*, 2016.

[5] Bilge Acun, Nikhil Jain, Abhinav Bhatele, Misbah Mubarak, Christopher D. Carothers, and Laxmikant V. Kale. Preliminary evaluation of a parallel trace replay tool for hpc network simulations. In *Proceedings of the 3rd Workshop on Parallel and Distributed Agent-Based Simulations*, PADABS '15, August 2015. LLNL-CONF-667225.

[6] David W. Bauer Jr., Christopher D. Carothers, and Akintayo Holder. Scalable time warp on blue gene supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, PADS '09, Washington, DC, USA, 2009. IEEE Computer Society.

[7] Nikhil Jain, Abhinav Bhatele, Samuel T. White, Todd Gamblin, and Laxmikant V. Kale. Evaluating HPC networks via simulation of parallel workloads. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16. IEEE Computer Society, November 2016. LLNL-CONF-690662.

[8] Nikhil Jain, Abhinav Bhatele, Louis Howell, David Böhme, Ian Karlin, Edgar Leon, Misbah Mubarak, Noah Wolfe, Todd Gamblin, and Matthew Leininger. Predicting the performance impact of different fat-tree configurations. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17. IEEE Computer Society, November 2017. LLNL-CONF-736289.

[9] Xu Yang, John Jenkins, Misbah Mubarak, Robert B. Ross, and Zhiling Lan. Watch out for the bully! Job interference study on dragonfly network. In *Supercomputing*, November 2016.

[10] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, March 2011.

[11] Arun Rodrigues, Elliot Cooper-Balis, Keren Bergman, Kurt Ferreira, David Bunde, and K. Scott Hemmert. Improvements to the structural simulation toolkit. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 190–195, ICST, Brussels, Belgium, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[12] K. Scott Hemmert, Brian Barrett, and Keith D. Underwood. Using triggered operations to offload collective communication operations. In Rainer Keller, Edgar Gabriel, Michael Resch, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface*, pages 249–256, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[13] Brian W. Barrett, Ron Brightwell, K. Scott Hemmert, Kyle B. Wheeler, and Keith D. Underwood. Using triggered operations to offload rendezvous messages. In Yiannis Cotronis, Anthony Danalis, Dimitrios S. Nikolopoulos, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface*, pages 120–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[14] F. Kaplan, O. Tuncer, V. J. Leung, S. K. Hemmert, and A. K. Coskun. Unveiling the interplay between global link arrangements and network management algorithms on dragonfly networks. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 325–334, May 2017.

[15] Jeremiah J. Wilke, Joseph P. Kenny, Samuel Knight, and Sebastien Rumley. Compiler-assisted source-to-source skeletonization of application models for system simulation. pages 123–143, 2018.

[16] T. Groves, R. E. Grant, S. Hemmert, S. Hammond, M. Levenhagen, and D. C. Arnold. (sai) stalled, active and idle: Characterizing power and performance of large-scale dragonfly networks. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 50–59, Sept 2016.