

Efficient all-to-all Collective Communication Schedules for Direct-connect Topologies*

Prithwish Basu
RTX BBN Technologies
Cambridge, MA
prithwish.basu@rtx.com

Siddharth Pal
RTX BBN Technologies
Cambridge, MA
siddharth.pal@rtx.com

Liangyu Zhao
University of Washington
Seattle, WA
liangyu@cs.washington.edu

Arvind Krishnamurthy
University of Washington
Seattle, WA
arvind@cs.washington.edu

Jason Fantl
RTX BBN Technologies
Cambridge, MA
jason.fantl@rtx.com

Joud Khoury
RTX BBN Technologies
Cambridge, MA
joud.khoury@rtx.com

Abstract

The all-to-all collective communications primitive is widely used in machine learning (ML) and high performance computing (HPC) workloads, and optimizing its performance is of interest to both ML and HPC communities. All-to-all is a particularly challenging workload that can severely strain the underlying interconnect bandwidth at scale. This paper takes a holistic approach to optimize the performance of all-to-all collective communications on supercomputer-scale direct-connect interconnects. We address several algorithmic and practical challenges in developing efficient and bandwidth-optimal all-to-all schedules for any topology and lowering the schedules to various runtimes and interconnect technologies. We also propose a novel topology that delivers near-optimal all-to-all performance.

CCS Concepts: • **Computing methodologies** → **Distributed algorithms**; • **Computer systems organization** → **Interconnection architectures**.

Keywords: Interconnection, Network, Topology, Collective Communication, all-to-all, Scale, GPU, CPU

ACM Reference Format:

Prithwish Basu, Liangyu Zhao, Jason Fantl, Siddharth Pal, Arvind Krishnamurthy, and Joud Khoury. 2024. Efficient all-to-all Collective Communication Schedules for Direct-connect Topologies. In *International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24)*, June 3–7, 2024, Pisa, Italy. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3625549.3658656>

*Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

HPDC '24, June 3–7, 2024, Pisa, Italy

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0413-0/24/06

<https://doi.org/10.1145/3625549.3658656>

1 Introduction

Collective communications have received significant attention in both high performance computing (HPC) and machine learning (ML) disciplines. The all-to-all collective, in particular, is used in several HPC workloads such as with the 3D Fast Fourier Transform (FFT) [40] used in molecular dynamics [4, 12] and direct numerical simulations [35]. It is also used in ML workloads, for example, to exchange large embeddings in the widely deployed Deep Learning Recommendation Model (DLRM) [36, 37], and in the mixture-of-experts (MoE) models [30]. All-to-all collective communication is often a bottleneck at scale in these workloads [10, 16, 43].

An emerging approach to meet these challenging demands has been to employ various forms of optical circuit switching to achieve higher bandwidths at reasonable capital expenditure and energy costs [24, 27, 31, 32, 53, 55, 61]. Hosts communicate using a limited number of optical circuits that may be reconfigured at timescales appropriate for the hardware (see §2.1), thus exposing network topology as a configurable component. We refer to this setting as *direct-connect* with circuits that are configured and fixed for an appropriate duration. Direct-connect fabrics and topologies such as mesh, Tori, DragonFly [28], and SlimFly [13] have been well studied in the HPC community and deployed across several supercomputers, such as with Google's TPUv4 [24, 31].

Computing bandwidth-optimal all-to-all schedules on a direct-connect topology with N nodes can be formulated using the Max Concurrent Multi-Commodity Flow problem, hereafter MCF, and solved in polynomial time using linear programming (LP) [47]. MCF, however, suffers from high time complexity even at modest scales since the number of flow variables in a bounded degree network scales as $O(N^3)$. At $N = 1000$, for example, even a state-of-the-art LP solver [8] is unable to generate a schedule on a fast machine within an entire day. For smaller $N (< 100)$, which is typical of ML applications, it takes tens of minutes to generate a schedule. This makes it hard for the algorithm to react quickly to changes in the topology, for example, due to topology reconfiguration or failures. *We enhance the scalability of the exact all-to-all MCF by decomposing it into a simpler*

master LP and a set of N children LPs that are parallelized for fast computation. We demonstrate a $O(\text{poly}(N))$ speed up in time complexity under decomposition and parallelization, reducing actual runtime on $N = 1000$ by orders of magnitude to 40 minutes instead. For N in the hundreds, it takes seconds to generate a schedule. Prior works [20, 26, 29, 47] try to improve computational complexity by trading off optimality using approximation schemes. These works still end up significantly underperforming our decomposed MCF in practice, both in terms of performance and complexity.

Another challenge lies in lowering the MCF solution to both ML accelerators and HPC runtimes and fabrics. These fabrics employ different topology, routing, and flow control mechanisms as they have historically been designed with different objectives [18]. We devise a general model of the underlying network, distinguishing between fabrics that support additional forwarding bandwidth (i.e., forwarding bandwidth at the Network Interface Card (NIC) is higher than the injection bandwidth at the host/accelerator) and those that do not. Additional forwarding bandwidth increases all-to-all performance in direct-connect settings as it compensates for the *bandwidth tax* [33] (since a node acts as a router and uses a significant fraction of its total link bandwidth to forward other node traffic). We develop an algorithmic toolchain for producing and lowering near bandwidth-optimal all-to-all collective communication schedules to arbitrary supercomputer-scale topologies and different interconnect technologies. On host or accelerator runtimes where data movement is “scheduled”, we devise a novel time-stepped version of the MCF problem. On fabrics with hardware “routing” and additional forwarding bandwidth, we develop scalable algorithms for computing static routes either by directly extracting the paths from the MCF solution or by employing path-based MCF formulations where flow variables are defined on paths instead of on links. We develop compilers and tools for lowering the schedules and the routes to the underlying runtime and interconnect, and we demonstrate near-optimal all-to-all performance on a range of topologies at different scales.

Finally, we establish an analytical lower bound for all-to-all performance on any topology, use it to compare different topologies and show the superiority of generalized Kautz graphs in terms of both performance and coverage. It is known that topologies with higher bisection bandwidth result in higher all-to-all throughput [13, 24]. Several works in the HPC community have investigated the all-to-all behavior of different topologies. Earlier works proposed specialized patterns for higher dimensional mesh, tori [51] and hypercubes [23], while later works proposed more complex topologies that have beneficial graph properties, e.g., high expansion coefficient [54], large spectral gap [58], and low diameter [13]. Many of the proposed topologies, however, do not have sufficient coverage in realizable graph sizes (N) and degree (k). We propose the class of generalized Kautz (GenKautz)

graphs [21], which are known for their expansion properties and can be constructed for any N and k .

2 Background and Terminology

2.1 Direct-Connect Fabrics for ML and HPC

Our work identifies topologies and schedules helpful for a broad range of direct-connect interconnects common to both HPC and ML accelerator fabrics. These include, for example, *switchless physical circuits* [3], *patch-panel optical circuits* [52], and *optical circuit switches* (OCS) [24]. These options differ in cost, scalability, and reconfigurability [56]. For example, commercially available OCSs can perform reconfigurations in ≈ 10 ms, are more expensive than patch panels, but scale to fewer ports (e.g., Polatis 3D-MEMS switch has 384 ports at \$520 per port [41]). With these reconfigurable fabrics, topology becomes a degree of freedom, and ongoing work is demonstrating how to exploit this degree of freedom for increased performance [24, 27, 55, 59, 61]. Despite supporting faster reconfigurations, OCSes still suffer from relatively high reconfiguration latency, precluding rewiring of the circuits during a typically-sized collective operation. Accordingly, collectives need to operate over a set of pre-configured circuits that remain unchanged for the duration of the collective operation. We refer to this setting as *direct-connect*, circuits (and topology) that are configured and remain static for the duration of the collective algorithm.

Our work additionally targets different interconnect technologies, broadly ML accelerator and HPC interconnects. These employ different topologies, routing, and flow control, as they have historically been designed with different objectives [18, 37]. Table 1 highlights high-level differences between the two fabrics. HPC interconnects have generally focused on reducing latency using low-diameter topologies with high bisection bandwidth and hardware routing with cut-through flow control. With hardware routing, where each node or NIC serves as a router, the total forwarding bandwidth may exceed the host injection bandwidth to accommodate for the forwarding bandwidth tax. ML accelerator interconnects, on the other hand, optimize for high link bandwidth as they are mostly focused on collectives, tend not to employ hardware routing, and use synchronized accelerator schedules with store-and-forward flow control.

2.2 All-to-all Schedules, and Throughput

The network topology is modeled as a directed graph, represented as the tuple $G = (V, E)$, where V denotes the set

Table 1: Comparison of HPC and ML accelerator fabrics.

	HPC	ML Accelerator
Schedules	Path-based	Link-based
Topology focus	Bisection bandwidth	Node bandwidth
Flow Control	Cut-through	Store-and-forward
Injection BW	B	B
Forwarding BW	$\geq B$	B

of nodes ($|V| = N$) and E denotes the set of directed edges. The direct-connect fabric imposes a constraint that all nodes have degree d , which is the number of links/ports on each host or accelerator and is ideally low and independent of N . The link bandwidth is b , and the node bandwidth is $B = db$.

Each node i has a data buffer B_i comprised of N contiguous and equally sized shards $B_{i,j}$ each of size m bytes, $0 \leq i, j < N$, $|B_i| = Nm$, $|B_{i,j}| = m$. The all-to-all collective *transposes* the buffers, i.e., each node i sends shard $B_{i,j}$ to node j .

Communication schedules can operate at a finer granularity than a shard. We define chunk $C_{i,j}$ to be a subset of shard $B_{i,j}$, both specified as *index sets* of elements in a shard with $B_{i,j}$ representing the entire shard. For example, the shard can be an interval $[0, 1]$, and $C_{i,j}$ be some subinterval. Chunks do not need to be the same size. Since each chunk $C_{i,j}$ has a known source node i and destination node j , we omit the indexes and simply use C to denote the chunk. An all-to-all *communication (comm) schedule* A for G with t_{\max} *comm steps* specifies which chunk is communicated over which link or route in any given *step*. Specifically, A is a set of tuples $(C, (u, w), t)$ with $u, w \in V$ and $t \in \{1, \dots, t_{\max}\}$. $(C, (u, w), t)$ denotes that node u sends chunk C to node w at comm step t . Chunking is performed during schedule compilation (§4).

Link-based Schedules: In fabrics without hardware routing, chunks only flow on directly connected edges $(u, w) \in E_G$.

Path-based Schedules: In fabrics with hardware routing, (u, w) may not correspond to an edge in G , i.e., chunks can flow on end-to-end paths between source and destination as determined by the routing function.

The *throughput* of an all-to-all schedule for a shard size m is $\frac{(N-1)m}{T}$, where T is the time to complete the all-to-all schedule (the time for each node to send $N - 1$ shards each of size m bytes).

Finally, *algorithm runtime* is the time taken by the *algorithm* to compute and lower the schedule for a given network.

2.3 Related work

In the theory community, optimization of the all-to-all collective has been formulated as a maximum concurrent multi-commodity flow problem (MCF) and solved in polynomial time using LP [47]. Although the MCF has polynomial time complexity, it can be hard to solve in practice for large problem sizes. As a result, several works have proposed fully polynomial time approximation schemes (FPTAS) [20, 26, 47]. The best known FPTAS schemes [26] have time complexity $O\left(\frac{N^3}{\epsilon^2} \log^{O(1)} N\right)$ and get to a factor of $(1 - \epsilon)$ of the optimal throughput. In this paper, we improve the tractability of LP-based solutions while not sacrificing optimality. We decompose the original MCF problem into a master LP and N simpler parallelizable child LPs. Since the former (which dominates the time complexity – see Fig. 7) has $O(N^2)$ variables, one can leverage recent LP solving techniques with time complexity $O(N^{2.37})$ [15] to solve the MCF in $O(N^{4.74})$ time. In practice, our master LP has lower time complexity

owing to its special structure, and MCF is significantly better in running time than the FPTAS schemes (for small values of ϵ) without sacrificing optimality even for moderate N (Fig. 7). Moreover, the *sequential* FPTAS schemes are unable to exploit the parallelism the way we do.

Early HPC works investigated efficient all-to-all collective communication on well-known topologies, e.g., hypercubes, meshes, and tori. Johnsson and Ho [23] proposed optimal all-to-all collectives for single-port and n -port models of hypercubes. Scott [45] proposed optimal all-to-all collectives on meshes. Suh et al. [51] and Yang et al. [57] proposed all-to-all collectives for mesh and tori that could leverage virtual cut-through and wormhole-switched networks.

More recent works have studied all-to-all communication on topologies that have beneficial graph properties for supporting datacenter communications. The bisection bandwidth of a network (χ) is known to be related to all-to-all throughput in the sense that the latter is bounded from above by $\frac{4\chi}{N^2}$. Prior works have therefore used χ as a proxy for all-to-all throughput [13, 48, 54], and as a result, expander graphs got significant interest due to their low modularity and hence high χ . Xpander [54] routes all-to-all traffic along K -shortest paths on expander graphs with multi-path TCP congestion control [44] to yield good throughput in switch-based datacenter settings. The all-to-all problem has been formulated as an MCF in such contexts [25, 42], and it has been shown that multiple expanders have nearly identical performance for all-to-all traffic. However, ours is the first study that applies multiple forms of MCF constructs (link- and path-based) to optimize all-to-all collective communications on a diverse set of HPC and ML fabrics and topologies at scale.

Recently, Cai et al. [14] proposed an SMT-logic-based approach (SCCL) for synthesizing optimal collectives in a topology-agnostic manner for GPU fabrics. However, their approach is computationally expensive due to the NP-hard nature of the SMT formulation. Followup work TACCL [46] relies on integer programming and suffers from similar computational bottlenecks, as we show in §5. Recently proposed TE-CCL [9] improves upon TACCL’s performance by combining multi-commodity flow with Mixed Integer Linear Programming (MILP) and A* search. Their models focus on link-driven latency, which can be important at small sub-Megabyte buffer sizes. Our formulations, on the other hand, maximize network utilization for all-to-all under large buffer sizes, and we observe that MCF solutions in general attempt to take short paths through the network anyway. Our approach is significantly more scalable, generating efficient schedules for 1K+ nodes in much less time than what TE-CCL reports it takes to solve all-to-all on 128 node networks. Finally, the work in [60] optimizes the all-reduce collective.

3 Multi-commodity Flow-based Algorithms

Fig. 1 shows a summary flowchart of the algorithms we employ for generating all-to-all schedules for direct-connect

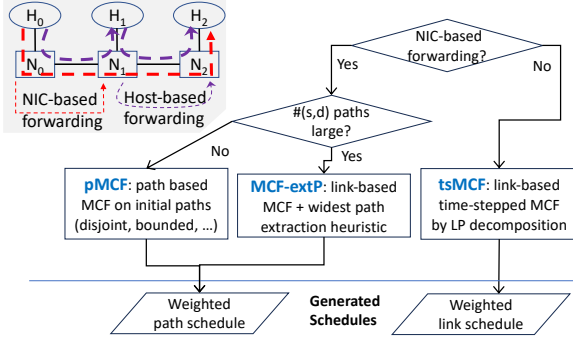


Figure 1: Generating link- and path-based schedules. Top left example shows difference between NIC-based and host-based forwarding of flow from host H_0 to H_2 .

fabrics. For ML-style fabrics with host/GPU-based forwarding, we generate weighted link-based schedules by solving the time-stepped version of the MCF. An MCF solution defines what chunks of data corresponding to a certain (s, d) pair (or commodity) should be transmitted by an intermediate node u over each of its outgoing links (u, v) at time step t . A naive solution involves solving a linear program (LP) on variables defined for each commodity, link, and time step—in the worst case, the total number of variables grows as $N(N-1) \times O(N) \times O(N) = O(N^4)$ where N is the network size (bounded degree networks have $O(N)$ links and the number of time steps, $l_{max} \geq$ the diameter, which can be $O(N)$). We propose to *decompose* this LP into a *master* source-only LP that first computes aggregate optimal flow rates leaving each source s and then uses this solution to compute optimal flow rates for each (s, d) pair. This enables scaling to networks with thousands of nodes.

For HPC-style fabrics with NIC-based forwarding, we generate path-based schedules that constitute a set of paths $\mathcal{P}_{s,d}$ for each (s, d) pair and weights w_{p_i} associated with each path $p_i \in \mathcal{P}_{s,d}$ controlling the fraction of traffic that should be sent along p_i . Optimal path-based schedules can be computed by solving the path-based version of the MCF, which is a natural *dual* of the link-based version mentioned earlier. However, this involves defining optimization variables for every possible (s, d) path, which is prohibitive for many topologies, even if we restrict the path set to include only shortest paths. We use good heuristics like sampling good path sets of small cardinality (e.g., edge-disjoint paths) to mitigate this problem. We also propose another radically different approach that instead solves the link-based MCF, and then applies an iterative “widest path” extraction algorithm to greedily extract high-flow (s, d) paths from the optimal per-link flows. Although potentially suboptimal, this approach is tractable and has good performance on the topologies we study.

3.1 Problem formulation

3.1.1 Link variable based MCF formulation. Given a network $G = (V, E, cap : E \rightarrow \mathbb{R}^+)$, where cap denotes link capacities, the problem of maximizing all-to-all throughput

can be modeled as a *maximum concurrent multi-commodity flow* (MCF) problem with $N(N-1)$ commodities of equal demand. This problem can be formulated using Linear Programming [20, 26, 29, 47]. We define variables $f_{(s,d),(u,v)}$ to denote the amount of flow of commodity $s \rightarrow d$ that should traverse link (u, v) and *concurrent demand* variable F (i.e., the common rate at which all commodities will flow concurrently), and solve the LP below.

Link-based max-concurrent MCF formulation:

$$\text{maximize } F \quad (1)$$

$$\text{subject to: } \sum_{s,d} f_{(s,d),(u,v)} \leq cap(u,v), \forall u, v \quad (2)$$

$$\sum_v f_{(s,d),(u,v)} \leq \sum_w f_{(s,d),(w,u)}, \forall s, d, u : s \neq u, d \neq u \quad (3)$$

$$\sum_w f_{(s,d),(w,d)} \geq F, \forall s, d \quad (4)$$

$$f_{(s,d),(u,v)} \geq 0, \forall s, d, u, v \quad (5)$$

The flow conservation constraint is modeled by inequality (3). This improves the speed of the LP solver; at the optimal solution, the inequality is enforced with no slack. Also, enforcing the demand constraint (4) only at the sink node d is sufficient since the combined flow conservation and demand constraints at the sink enforce the same at the source. If however, a flow $f_{(s,d)}$ with optimal F returned by the solver has extra flow near s (due to inequality (3)), a post-processing step from d to s is executed to ensure exact flow conservation. An optimal flow generally follows links along multiple paths over the network. This LP is solvable in polynomial time, albeit in high-order polynomial time. To improve solver efficiency, we use a compact formulation of the LP in which all the flow conservation and demand constraints are expressed by a single matrix-vector constraint that relates the product of the node-to-link incidence matrix and link-flow vector to the per-commodity demand matrix scaled by F . This eliminates the “pre-solve” canonicalization step.

A key disadvantage of the per-commodity based LP approach is that the number of link-flow variables for a k -regular graph is $kN^2(N-1)$, which gets intractable even at modest scales (hundreds of nodes).

3.1.2 Decomposing the MCF LP for scalability. Since the MCF LP approaches discussed above are computationally challenging, we decompose the problem of computing the optimal flow for $N(N-1)$ commodities by considering N groups of source-rooted flows (each delivered to $N-1$ destinations). Specifically, we follow the steps below.

(1) *Compute source-based grouped commodity flows:* Solve a master LP, defined in (6)-(9), for computing the optimal concurrent rates for N source-rooted grouped multicommodity flows. The source-based flow conservation (8) reflects the fact that the total amount of flow entering u has to be greater than the sum of the amount of flow leaving u and the amount sunk at u (which must equal the concurrent flow value F). Since we worry about only N groups of commodities (instead

of $N(N-1)$ point-to-point commodities), we only need kN^2 variables, which is tractable (thousands of nodes).

(2) *Compute optimal per-commodity flows*: Once the per-source optimal flow has been computed, solve N additional simpler Child LPs, one per source as defined in (10)-(14), to determine the flow values per link for each (s, d) commodity. Each such LP (say for source s) will set the link capacities to the flow values computed by the master LP, and will solve a standard maximum concurrent multicommodity flow (on a thusly capacity-adjusted graph) for $N-1$ commodities $\{s \rightarrow v | v \in V \setminus \{s\}\}$. These LPs can be run in parallel on multi-core processors, have $kN(N-1)$ flow variables to optimize, and are generally simpler in complexity than the original LP. Solving $N+1$ LPs with $O(N^2)$ variables each is much more tractable than solving a single LP with $O(N^3)$ variables since the computation complexity of LP is generally much higher than linear in the number of variables.

Decomposed link-based MCF (for scalability):

Master LP to compute source-based grouped commodity flows:

$$\text{maximize } F \quad (6)$$

$$\text{subject to: } \sum_s f'_{s,(u,v)} \leq \text{cap}_{(u,v)}, \forall u, v \quad (7)$$

$$F + \sum_v f'_{s,(u,v)} \leq \sum_w f'_{s,(w,u)}, \forall s, u : s \neq u \quad (8)$$

$$f'_{s,(u,v)} \geq 0, \forall s, u, v \quad (9)$$

Child LPs to extract link flows from source-based flows:

$$\text{minimize } \sum_{d,u,v} f_{(s,d),(u,v)} \quad (10)$$

$$\text{subject to: } \sum_d f_{(s,d),(u,v)} \leq f'_{s,(u,v)}, \forall u, v \quad (11)$$

$$\sum_v f_{(s,d),(u,v)} \leq \sum_w f_{(s,d),(w,u)}, \forall d, u : s \neq u, d \neq u \quad (12)$$

$$\sum_w f_{(s,d),(w,d)} \geq F, \forall d \quad (13)$$

$$f_{(s,d),(u,v)} \geq 0, \forall s, d, u, v \quad (14)$$

The decomposed LP approach yields the optimal MCF value F as the standard approach (in §3.1.1) although the actual flow values f returned may be different.

3.1.3 Time-stepped MCF (tsMCF) formulation. The MCF formulation presented in §3.1.1 yields the optimal rates at which each commodity should be transmitted by considering the flow of data to mimic that of infinitesimally divisible fluids. This is inadequate for ML network fabrics where accelerators send finite data chunks in a finite number of fixed-length time steps. This necessitates the generation of time-stepped schedules. To this end, we extend the notion of MCF to the temporal domain by computing flows on a time-expanded stacked graph [11] representation of G . It has $l_{\max} + 1$ time-indexed instances u_t of each node $u \in G$ and directed edges $u_t \rightarrow v_{t+1}$ with *capacity* = 1 whenever $(u, v) \in G$ as well as “self” edges $u_t \rightarrow u_{t+1}$ with *capacity* = ∞ denoting potential buffering at u over time. l_{\max} is set to a value $\geq \text{diameter}(G)$. We compute flows on

this time-expanded graph essentially following the procedure described in §3.1.1. The main difference is in the size of the input graph. In this case, it has $(l_{\max} + 1)|V|$ nodes and $l_{\max}(|V| + |E|)$ links, and hence the LPs take somewhat longer to solve. However, the time-expanded graphs are directed acyclic graphs and are hence less complex. This tends to help mitigate the running time issues of the LPs. Another key difference is that all nodes do not source/sink traffic; instead, the nodes u_0 source traffic for nodes $v_{l_{\max}+1}$ and non-zero flow along an infinite capacity “self” edge essentially simulates waiting for a time slot. The equivalent time-stepped LP formulation is given below.

tsMCF: for ML fabrics with host/GPU forwarding

$$\text{minimize } \sum_t U_t \quad (15)$$

$$\text{subject to: } \sum_{s,d} f_{(s,d),(u,v),t} \leq U_t, \forall u, v, t \quad (16)$$

$$\sum_{t' \leq t, v} f_{(s,d),(u,v),t'} \leq \sum_{t'' < t, w} f_{(s,d),(w,u),t''}, \forall s, d, u, t : s \neq u, d \neq u, t > 1 \quad (17)$$

$$\sum_{t',v} f_{(s,d),(u,v),t'} = \sum_{t'',w} f_{(s,d),(w,u),t''}, \forall s, d \quad (18)$$

$$\sum_{t',v} f_{(s,d),(s,v),t'} = \sum_{t'',w} f_{(s,d),(w,d),t''} = 1, \forall s, d \quad (19)$$

$$0 \leq f_{(s,d),(u,v),t} \leq 1, \forall s, d, u, v \quad (20)$$

The objective (15) minimizes the utilization of bandwidth at every time step, while the constraint (16) ensures that the total utilization at every edge is less than the bandwidth. The constraint (17) enforces the amount of data received by node u must be greater than or equal to the amount of data sent by node u from comm step 1 to every other comm step (the difference is reserved for future send). Constraint (18) enforces the total amount received by node u is equal to the total amount sent by node u . Constraint (19) enforces that the total amount sent by the source and received by the destination is equal to 1. One can add a multiplier to $f_{(s,d),(u,v),t}$ in the first constraint if link bandwidth is not uniform among all links. This time-stepped LP can be decomposed into a source-based LP + child LPs as described in §3.1.2.

3.1.4 Path-variable based MCF (pMCF) formulation.

In networks supporting multi-hop routing, we need to compute the optimal rates of flows along multiple paths from each source s to each target node d . We first compute a set of paths \mathcal{P} for each commodity/ (s, d) pair. Next, for each path $p \in \mathcal{P}$ and (s, d) pair, define flow variable $f_{(s,d),p} \in \mathbb{R}^+ \cup \{0\}$. As in the link-based formulation, we ensure that the link capacity constraints are obeyed at each link. The flow conservation constraints are automatically obeyed at each node since data will be flowing along simple paths (in-degree and out-degree are 1). The LP formulation is shown in (21)-(24).

If the set \mathcal{P} is allowed to consist of all paths of unbounded length, then path-based MCF is a natural dual of link-based

MCF and hence provides the same optimal MCF value. However, solving such a dual problem is impractical since $|\mathcal{P}|$ typically grows exponentially with N . We make the path-based MCF in practical scenarios tractable by curtailing the number of paths in \mathcal{P} to $O(\text{poly}(N))$. Restricting the path lengths to below l_{\max} drastically reduces $|\mathcal{P}|$. This approach works for many networks of interest (per our empirical observation), e.g., expander graphs like Generalized Kautz graphs, where $|\mathcal{P}|$ is polynomial in N, l_{\max} . However, in several other graphs that possess a high degree of symmetry, e.g., the torus, the number of paths of length l_{\max} grows exponentially in l_{\max} since the diameter is \sqrt{N} ; therefore $|\mathcal{P}| \geq \frac{1}{\sqrt{N+1}} \binom{2\sqrt{N}}{\sqrt{N}}$, which grows super-exponentially in N ; this makes the approach intractable for large tori. One tractable heuristic that we have empirically observed to achieve the optimal MCF solution is to choose \mathcal{P} to be a maximal set of link-disjoint (s, d) paths, which can be found efficiently and whose cardinality is upper bounded by $kN(N-1)$ for k -regular graphs.

pMCF: for fabrics with NIC forwarding
maximize F (21)

subject to: $\sum_{s,d} \sum_{p \ni e} f_{(s,d),p} \leq \text{cap}_e, \forall e \in E$ (22)

$\sum_{p \ni \mathcal{P}(s,d)} f_{(s,d),p} \geq F, \forall s, d$ (23)

$f_{(s,d),p} \geq 0, \forall s, d, u, v$ (24)

3.2 Applying MCF to the different fabrics

3.2.1 Source-routed fabrics. The paths along which (certain fractions of) each commodity would flow must be provided at the respective sources of the commodity. We propose two different approaches below based on whether a topology has low or high path diversity (see Fig. 1).

pMCF: Directly applying path-variable based MCF (low path diversity). The path-variable based MCF formulation described in (21)-(24) can be applied to source-routed fabrics for graphs in which the path diversity does not grow exponentially in N , as is the case with expander graphs. However, this approach is not tractable for graphs like the multi-dimensional torus where the number of paths (with length $\leq l_{\max}$) grows super-exponentially in N .

MCF-extP: Applying link-based MCF and extracting paths (high path diversity). We first solve the decomposed link-based MCF described in §3.1.2. However, for source-routed fabrics, we need to obtain the paths on which the different commodities should flow. Therefore, as a final step, we greedily extract paths from the link-based MCF solution.

Widest path extraction. Given the MCF flow values on each link corresponding to each (s, d) commodity, we construct a weighted subgraph DAG $G_{s,d}$ of the original graph G induced by the edges with non-zero (s, d) flow (weights = MCF flows). We then iteratively extract (s, d) paths from $G_{s,d}$ by greedily solving the *widest path* problem, by making minor modifications to Dijkstra's shortest path algorithm:

1. Find (s, d) path p in $G_{s,d}$ with maximum flow rate (r).

2. Subtract r from the capacities of all the links in p .
3. Repeat the two previous steps until s no longer has a non-zero capacity path to d .
4. Upon termination, the algorithm finds a set of (s, d) paths with decreasing flow rates, which are ready for lowering.

3.2.2 Non source-routed fabrics (tsMCF). When forwarding and flow control are performed by the host/GPU (no hardware routing), we use the time-stepped link-based MCF formulation described in §3.1.3 to obtain chunk schedules that exactly specify what data needs to be sent (or received) by each GPU at every time step.

Handling host-to-NIC bottlenecks In scenarios where the host-to-NIC bandwidth B_{host} is less than the net egress/ingress link bandwidth at the NIC, i.e., $B_{\text{host}} < d \cdot b$, the host-to-NIC bandwidth becomes a bottleneck. Figure 2 shows how to augment the original NIC topology to model this host-to-NIC bottleneck. We augment the graph in a manner that forces data to flow through the host even when the node is not the destination. The MCF computed between the host nodes on the augmented graph yields the optimal throughput. We apply §3.1.3's tsMCF formulation to generate schedules.

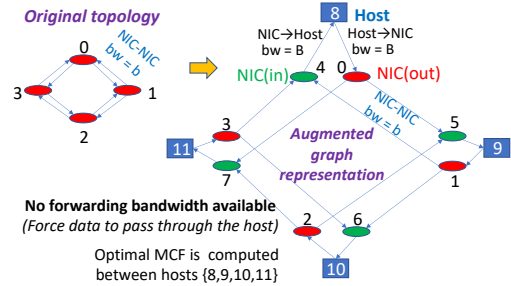


Figure 2: Topology augmentation to model host-to-NIC bottleneck

4 Schedule Compilation

We implement compilers and interpreters to lower the schedules and paths, and execute them on CPU and GPU runtimes. **Link-based Schedules:** The link-based MCF algorithm produces schedules that are chunked and lowered to both Microsoft's MSCCL [34] and Intel's oneCCL [22]. MSCCL is an open-source collective communication library that extends NCCL [38] and RCCL [7] with an interpreter providing the ability to program custom collectives on GPUs. Collective schedules are defined in XML as instructions (send/receive/reduce/copy) within GPU thread blocks that the interpreter executes. We additionally lower the same schedules to oneCCL+libfabric [22], an open-source collective communications library by Intel that supports CPUs.

We extended oneCCL with an interpreter, in a similar way that MSCCL extended NCCL, that executes the XMLs. The oneCCL XMLs similarly specify instructions (send, receive, reduce, copy, sync) and add scratch buffers for chunk forwarding. The primary challenge in creating both MSCCL and oneCCL schedules was chunking. The MCF solution

produces the fractional rates $f_{(s,d),(u,v),t}$ for each commodity (s, d) on each link (u, v) at each time step t . The lowering algorithm determines the smallest chunk size to support the lowest such rate, which guides how granularly a shard is chunked and how chunks are combined and forwarded by intermediate ranks. At each time step, a rank then sends the total outgoing flow (the chunks received in the previous time step), receives the total incoming flow (chunks to receive in the current time step), and performs synchronization. In both MSCCL and oneCCL, we have the ability to increase the number of channels by duplicating the schedule and running a parallel copy on different threads. All sends and receives are asynchronous with no data dependencies.

Path-based Schedules. Recall that the weighted path-based MCF algorithm produces a set of weighted paths for each commodity (shard) $B_{i,j}$ (source i , dest j). Weighted multi-path routing and flow control should ideally be performed by the hardware, to which we would lower the MCF schedules (the per-commodity routes and weights). In our testbeds, we use the Cerio (Rockport) NC1225 network card [3] (described in §5.1). While the current version of the Cerio card natively supports multi-path routing on user-specified routes, it does not expose the capability to program the weights per route. This means we cannot directly use the native multi-path capability, which we disable. We instead approximate the weighted paths MCF by: (1) lowering the routes for each commodity $B_{i,j}$ to the underlying fabric, (2) dividing each shard/commodity into a set of equal-sized chunks, and (3) steering chunks onto routes as defined in our schedule.

Specifically, the Cerio card exposes a utility for lowering our computed source routes to the hardware, where a route specifies the egress ports on the traversed links from source to destination as well as the layer identifier for the route; the layer identifier is used to assign routes to different virtual channels in order to eliminate deadlocks [50]. The card also allows us to steer flows to routes at the application layer. This is possible in ROCE v2 by setting the UDP source port when creating the RDMA Queue Pair (QP), such that the tuple (src port, src IP, dst IP, QP number) hashes to the desired route id. We implemented the scheduling and flow steering functionality in Open MPI+UCX [39]. The chunked schedule specification is lowered to an XML that is executed by our interpreter. The latter is implemented in OMPI as part of the tuned collectives component within the Modular Component Architecture (MCA). The schedule defines the chunks and path each should take, and the extended OMPI+UCX runtime creates the right number of QPs and performs the steering. A shard is divided into a set of equal-sized chunks as follows: we compute the highest common factor across all path weights in the MCF solution and use that as the base chunk size (call it c). Each shard of size m bytes is then divided into $\lceil m/c \rceil$ chunks, and the right number of chunks is assigned to each path based on the path weight. This approach ensures all chunks (flows) fairly share the bandwidth,

approximating the ideal MCF in practice on the Cerio fabric. We discuss the scalability limitations of this approach in §5.5.

5 Evaluation

We evaluate schedule performance and algorithm runtime at increasing scales on different hardware and on different direct-connect optical topologies, some of which are well-studied (complete bipartite, hypercube, twisted hypercube, Torus) while others are non-standard such as punctured tori with non-homogenous degree per node. We also present performance results at a large scale in simulation. We summarize our performance results next.

Performance of link-based schedules: Our lowered tsMCF schedules deliver near-optimal throughput performance on different topologies and scales, outperforming state-of-the-art baselines by up to $1.6\times$ (§5.2 Fig. 3).

Performance of path-based schedules: Our lowered MCF-extP and pMCF schedules deliver near-optimal throughput performance on different standard and non-standard topologies and scales, outperforming state-of-the-art scalable baselines by up to 30% (§5.2, Fig. 4 and Fig. 5), and the all-to-all speedups directly translate to speedups in the 3D FFT workload (§5.2, Fig. 6). MCF outperforms other scalable baselines at large scale in simulation (§5.3, Fig. 8, Fig 9).

Algorithm runtime. Through large-scale simulations going up to 1000 nodes, our MCF decomposition approach yields orders of magnitude improvement in algorithm runtime over the original MCF and all other baseline schedule generation approaches (§5.3, Fig 7).

Topology. We identify GenKautz as a family of expander topologies that have near-optimal all-to-all performance while also having complete coverage in N and d , outperforming other well-known expander topologies (§5.4, Fig. 10).

5.1 Direct-connect testbed and cluster

We evaluate the all-to-all schedules on two testbeds: an internal 8 server (1 NVIDIA A100 GPU [1] per server) testbed that supports topology reconfiguration, and an external 27 server (1 CPU per server) cluster at the Texas Advanced Computing Center (TACC) [2, 5] where topology is fixed to the Torus.

Cerio Card. In both testbeds, each server is equipped with a Cerio NC1225 network card [3]. The card supports source routing with multi-path and cut-through flow control, and stores up to 8 routes per destination. It offers up to 300 Gbps of total forwarding bandwidth using 12×25 Gbps links ($b = 25$ Gbps or 3.125 GB/s), and supports 100 Gbps of injection bandwidth from the host or GPU using x16 PCIe gen3. We can accordingly evaluate both link and path-based schedules on both testbeds.

Internal GPU Testbed. The network cards are directly connected via a Telescent optical patch panel [52]. Our testbed can realize different topologies by reconfiguring the patch panel. We limit our evaluation to bidirectional topologies, specifically hypercube and twisted hypercube both with

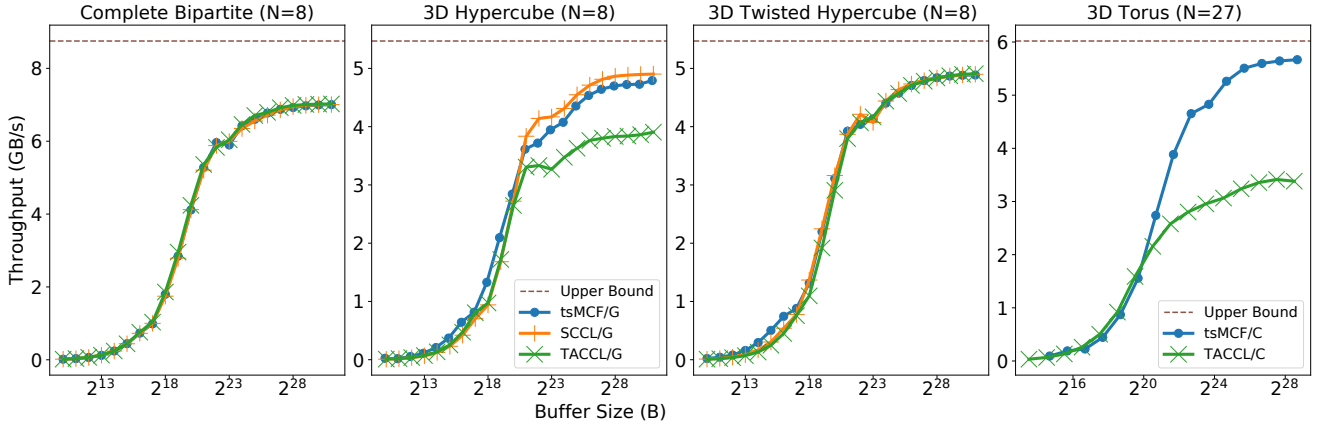


Figure 3: Throughput of link-based all-to-all schedules on different topologies and runtimes. Appended /G indicates the schedule is lowered to GPUs and the MSCCL [34] runtime, whereas /C means the schedule is lowered to CPUs and the oneCCL [22] runtime. Averaged over 20 iters.

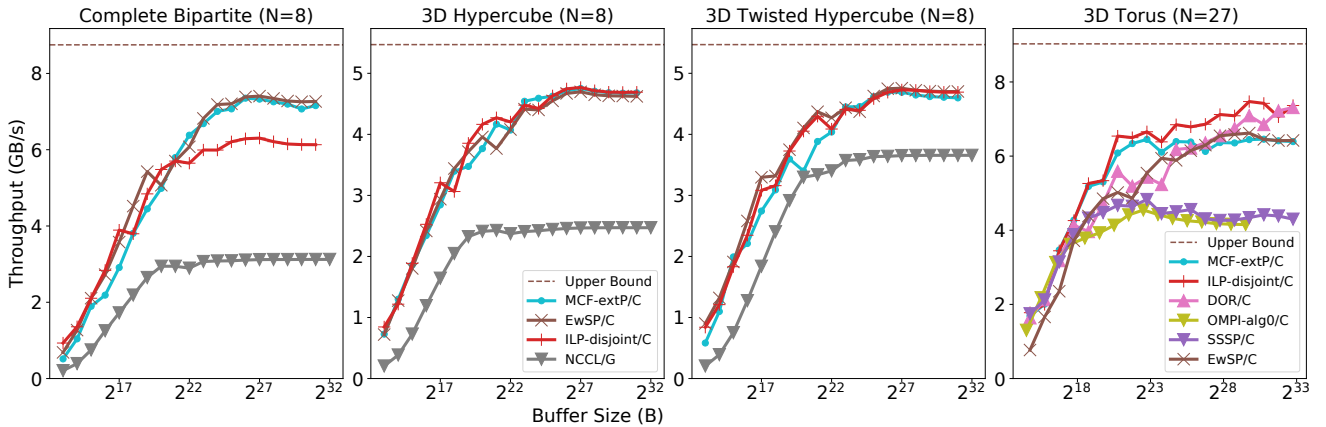


Figure 4: Throughput of route-based all-to-all schedules on different topologies and runtimes. Appended /G indicates the schedule is lowered to GPUs and the NCCL [34] runtime, whereas /C means the schedule is lowered to CPUs and the Open MPI [39] runtime. Averaged over 20 iters.

degree 3 (i.e., $B=75$ Gbps), and complete bipartite with degree 4 ($B=100$ Gbps). While unidirectional topologies (e.g., GenKautz) can be realized by configuring the patch panel in simplex mode, we cannot accurately evaluate their performance since the requisite overlay routing for the reverse path traffic (acks) is currently only supported using routing rules performed by the kernel leading to unpredictable RTTs. **TACC HPC Cluster.** A cluster of CPU servers at TACC are connected in a fixed torus topology using the Cerio fabric [2, 5]. We use a $3 \times 3 \times 3$ torus (27 nodes, degree=6) from the cluster to run our experiments. The host injection bandwidth of 100 Gbps is less than $B=150$ Gbps for degree 6; hence, we evaluate the benefits of path based schedules to exploit the extra forwarding bandwidth for all-to-all.

5.2 Evaluation of optimized collectives

Time-stepped tsMCF schedules: Fig. 3 shows our lowered link-based tsMCF schedules deliver near-optimal performance at large message sizes on different topologies and scales. State-of-the-art scheduling algorithms such as SCCL [14]

do not scale, failing to terminate even at a modest 27-node scale (Fig. 3, right). TACCL [46] schedules, on the other hand, underperform on the Hypercube by 22% and on the 3D Torus by up to $1.6 \times$ at large buffer sizes (Fig. 3, right). Since our GPU testbed is constrained to an 8-node scale, we resort to the CPU cluster to evaluate at larger 27-node scale. In Fig. 3, we append the algorithm name with /C to indicate that we lowered to oneCCL and ran on CPUs, whereas /G refers to lowering to MSCCL and running on the A100 GPUs. Recall the link-based schedule experiments in Fig. 3 do not use the routing capability from the underlying fabric; all transfers are on point-to-point links controlled by the scheduling thread(s) on the host GPU or CPU.

On the 3D Torus (Fig. 3, right), tsMCF uses the bottleneck model (§3.2.2 Fig. 2) to produce the schedule since the host injection bandwidth (100 Gbps) is lower than the NIC bandwidth (150 Gbps for degree 6). The flow value produced by MCF on this bottlenecked 3D Torus topology is $f = \frac{2}{27}$, which dictates the theoretical upper bound of $(N-1)fb = (26)(\frac{2}{27})(3.125) = 6.01$ GB/s. On the other

hand, the flow value in the non-bottlenecked setting, as we discuss shortly in Fig. 4 (right), is $\frac{1}{9}$ which is 57% higher. The theoretical *upper bound* on throughput is $(N - 1)fb$, where f is the optimal flow value given by MCF (each commodity gets a max flow value of f assuming link capacity is 1, so when link capacity is b , the max achievable flow out of a node sourcing $N - 1$ flows is $(N - 1)fb$).

In summary, our experiments show that *the optimized tsMCF schedules are ideal for both GPU and HPC fabrics where additional forwarding bandwidth is not available while being generalizable to a wide range of topologies.*

Path-based schedules utilizing forwarding bandwidth

We implement two link-load-minimizing single-path baselines. The first is based on Integer Linear Programming (ILP) which is tractable only at small scales. It selects a subset of a candidate set of (s, t) paths such that the maximum load on any link is minimized. Low maximum load leads to high all-to-all throughput. We experiment with both link-disjoint (ILP-disjoint) and shortest paths (ILP-shortest) in the candidate set. The second baseline is Single Source Shortest Path (SSSP) [19] heuristic that iteratively computes shortest paths through a graph whose link weights reflect the additional congestion caused due to each iteration.

Fig. 4 shows our lowered MCF-extP schedules deliver near-optimal bandwidth performance in practice as we see at small scale ($N=8$) for the complete bipartite topology (left), and both the hypercube and twisted hypercube topologies (middle). On the complete bipartite, we see MCF-extP outperforms ILP-disjoint, which matches the theoretical results since the latter, which is constrained to a single path per commodity, is not bandwidth optimal on this topology. On the 27 node 3D Torus (right) on the supercomputer, our MCF-extP slightly underperforms due to practical limitations with injection rate control in the current fabric (more in §5.5). Both dimension ordered routing (DOR) [17] and ILP-disjoint are theoretically bandwidth optimal on the 3D Torus and are strong baselines. However, DOR does not work on non-Tori topologies, and ILP-disjoint does not produce optimal solutions in general and becomes intractable for larger topologies. SSSP is both scalable and general, but it produces sub-optimal solutions and is more than 50% worse than MCF-extP on the 3D Torus at large buffers.

MCF-extP also far outperforms NCCL and OMPI’s native all-to-all algorithms up to 2.3× on Bipartite, and 55% on 3D Torus. NCCL/OMPI native schedules perform $N-1$ point-to-point send/rcv operations (flows) per rank. These flows utilize the deadlock-free routes underneath which are computed by the Cerio fabric [50]. We also implement and evaluate the Equal weight Shortest Path (EwSP) baseline that distributes each commodity equally on all the shortest paths between source and destination. While EwSP performs very well on all the four topologies in Fig. 4, this is not the case in general, as we show later in §5.3, Fig. 8.

Comparing path-based schedules of Fig. 4 to link-based schedules of Fig. 3 on the same topologies, we see that the bandwidth performance of MCF-extP is comparable to that of tsMCF at large buffer sizes for the Bipartite (degree=4), HyperCube and Twisted Hypercube (degree=3). This is expected on these low-degree topologies since there is no additional forwarding bandwidth that path-based MCF can exploit, and both approaches have the same theoretical upper bound. On the other hand, MCF-extP significantly outperforms tsMCF on the larger 27 node 3D Torus (by about 3.4× at 1MB, and 15% at larger buffers) since it is able to exploit the additional forwarding bandwidth in this case (degree=6, $B = 150 \text{ Gbps} \geq 100 \text{ Gbps}$ injection bandwidth)—here MCF-extP is unable to reach its theoretical expected performance due to practical limitations on the current fabric (more in §5.5). In general, we also see MCF-extP has a significant performance advantage at smaller buffers due to the superior latency performance of cut-through routing as compared to having to incur a global synchronization per timestep with tsMCF.

Performance on non-standard topologies We assess the topology-agnostic quality of MCF-extP on heterogeneous degree topologies produced by sampling sub-graphs from the 3D Torus. Specifically, we sample 10 different instances of the 3D Torus to create *edge-punctured* (3 random edges removed) and *node-punctured* (3 random nodes removed) Tori. Baselines such as DOR are not defined for such punctured Tori. Fig. 5 shows the throughput of MCF-extP compared to ILP-disjoint and SSSP. The results are consistent with those of Fig. 4, where MCF-extP significantly outperforms SSSP but underperforms ILP-disjoint due to practical limitations with injection rate control we discuss shortly in §5.5. The results also match the empirical results, where we observe similar maximum link load for both ILP-disjoint and MCF (which is $\sim 30\%$ lower than SSSP). Puncturing the Torus is a way to emulate failures of links and/or nodes, which would be expected in a cloud setting, and to show the superior performance of MCF in such scenarios. When combined with the superior algorithm runtime (Fig. 7), this shows that *our approach is able to more quickly react to failures in networks of hundreds of nodes without compromising on performance.*

Workload speedups We implement distributed 3D Fast Fourier Transform (FFT), and run it on the 27 node 3D Torus, and on the edge-punctured 3D Torus on up to 1296^3 grid size, corresponding to all-to-all buffer size up to 1.29 GB. Fig. 6 shows the speedups in the 3D FFT on the 3D Torus (top, corresponding to all-to-all speedups from Fig. 4, right) and the punctured 3D Torus (bottom, corresponding to all-to-all speedups from Fig. 5). We observe up to 20% (14.9%) total speedup in the FFT time using our MCF schedules compared to SSSP schedules on the 3D Torus (edge-punctured 3D Torus). The speedups are a direct result of the faster all-to-all MCF schedules, consistent with Fig. 4 and Fig. 5. We use the latest version of the FFTW library [6] with multi-threading and with OMPI running our all-to-all schedules. We use slab

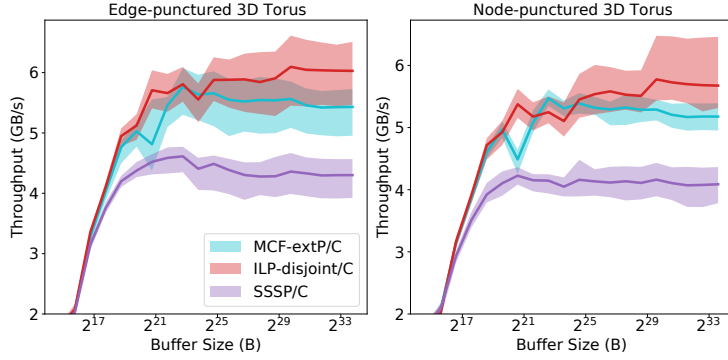


Figure 5: Performance on punctured 3D Torus, removing 3 links (left) or 3 edges (right) at random. Envelope min/max/average (line) over 10 instances (20 iters per)

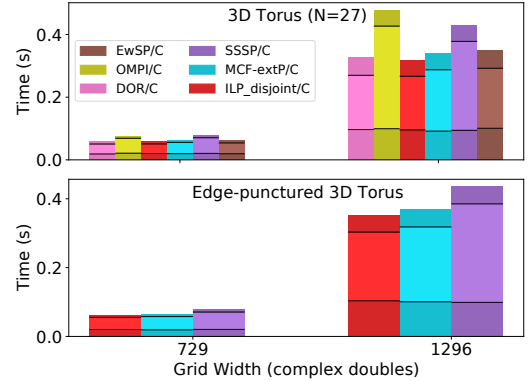


Figure 6: 3D FFT Times ($N=27$ processes, 32 threads each)

decomposition, where each process (1 multi-threaded process per node) performs three steps: first computes 2D FFTs on its slabs and packs the data, then runs all-to-all with all other processes, and finally unpacks and computes 1D FFTs to complete the 3D FFT. These three steps involved in the FFT computation are shown with bands in the bars of Fig. 6 with the first step corresponding to the bottom band.

5.3 Large scale numerical simulations

Benefits of MCF decomposition: Fig. 7 shows that our MCF decomposition approach (MCF-decomp) yields orders of magnitude improvement in algorithm runtime over the original MCF (MCF-original), and the other highly unscalable topology-agnostic schedule generation approaches such as SCCL [14], TACCL [46], Karakostas’ Fully Polynomial Time Approximation Scheme (FPTAS) [26], and ILP-disjoint. We show the computation time required for MCF-original (link-based) on N^3 variables and MCF-decomp on N^2 variables for the Generalized Kautz graph [21] (also see §5.4) for various choices of N . We observe that even at the scales of $N = 50, 100$, MCF-decomp is two orders of magnitude faster, while MCF-original fails to produce a solution for $N > 100$ nodes in a reasonable time. In contrast, SCCL [14]

is unable to generate all-to-all schedules for $N = 16$ even in 10^4 seconds. TACCL [46], a heuristic designed to be more scalable than SCCL, takes over 30 minutes to generate all-to-all schedules for even 32-node networks. Furthermore, for slightly larger networks, it is unable to produce even an approximate solution in 30 minutes. Even ILP-disjoint is only able to generate optimal schedules up to $N = 44$. While we evaluate on the GenKautz topology in Fig. 7, the scaling trends apply to other topologies (expanders, tori) as well.

The algorithm runtime of MCF-decomp follows a polynomial trend, and it is dominated by the time taken to solve the “Master LP”, which is a source-based MCF leading to a solution in 40 minutes for $N = 1000$ provided all “Child LPs” and subsequent “Widest path” extraction functions are run in parallel on N cores. The exact degree of the polynomial governing the speedup factor over original MCF is determined by that of the polynomial that determines the time complexity of solving such network flow LP problems. For example, if a state-of-the-art LP solver takes $O(N^{2.37})$ time to solve N -variable problems, the speedup factor can be estimated to be $O(N^{3 \times 2.37} / N^{2 \times 2.37}) = O(N^{2.37})$.

Fig. 7 also shows the runtime performance of a state-of-the-art FPTAS by Karakostas [26] at $\epsilon = 0.05$. While it shows a polynomial scaling trend unlike the other baseline schemes mentioned earlier, it significantly underperforms the MCF (decomposed) algorithm in running time even after sacrificing optimality, thus making it impractical for networks larger than a couple of hundred nodes. The FPTAS, being an inherently sequential algorithm, cannot exploit the high degree of parallelism that is exploited by decomposed MCF.

Performance of schedules that utilize NIC forwarding bandwidth: Fig. 8 (and Fig. 9) shows the *all-to-all time* (the time to concurrently transmit the workload assuming unit commodity demand and link capacities; it is equal to $1/MCF$ and the maximum link load) of various path-based schemes normalized by optimal link-based MCF. Comparing single path schemes, mainly ILP and SSSP, although SSSP is fast, it is up to $1.6\times$ worse than the theoretically optimal MCF solution.

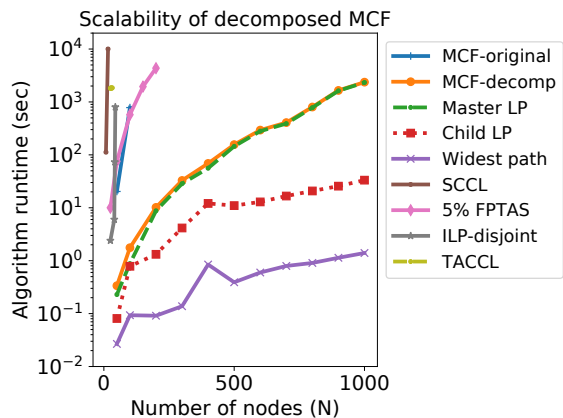


Figure 7: Scaling on GenKautz (degree=4); MCF-decomp is master LP + N parallel child LPs + widest path extraction heuristic

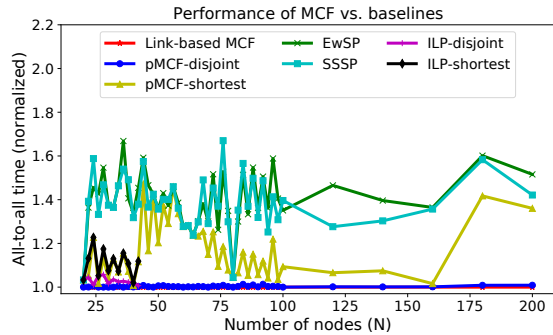


Figure 8: Performance on degree 4 Generalized Kautz graphs normalized by Link-based MCF

On the other hand, the ILP schemes, although performant, do not scale well (Fig. 7).

Although multipath schemes exploit the path diversity of the network well, naive multipath approaches such as Equal Weight Shortest Path that distribute the workload evenly along all available (s, t) shortest paths do not perform well. Its performance is similar to that of SSSP. However, *weighted* multipath schemes such as pMCF (Path-based MCF (disjoint)) and MCF-extP (Link-based MCF with path extraction) can achieve optimal or near-optimal performance. pMCF is optimal in theory if the number of (s, t) paths in the initial set is not restricted. However, this set can be extremely large, thus necessitating the development of decomposed MCF-extP, whose scalability has been illustrated in Fig. 7. However, in practice, if the (s, t) path set is restricted to link-disjoint paths, pMCF can almost match the optimal performance of pure link-based MCF. Interestingly, pMCF run with all shortest (s, t) paths exhibits suboptimal performance, especially for expander graphs (Fig. 8) since the latter do not have too many shortest paths. While pMCF (with shortest paths) performs well on topologies such as tori, the starting path set is often exponentially large in size, thus making the scheme impractical. In contrast, link-disjoint (s, t) paths can be computed in polynomial time for arbitrary topologies. Since there are at most d disjoint paths for any (s, t) pair, the number of LP variables is $O(N^2)$, making it comparable to decomposed Link-based MCF in terms of time complexity.

Fig. 9 illustrates the generality and good performance of MCF schemes (when compared to baselines like SSSP) since they perform optimally/near-optimally on heterogeneous degree-irregular subgraphs, e.g., formed by disabling random links. At the $N = 81$ scale considered ($d = 8$), interestingly, ILP-disjoint shows performance close to link-based MCF's if we allow it a tolerance factor ($\epsilon = 0.1$); but it does not scale to large N owing to it being NP-hard.

5.4 Near throughput-optimal all-to-all topologies

We ask *which topologies with N nodes and degree d yield the best all-to-all performance?* This is an important design

question for supercomputing clusters and is becoming more important with increased deployments of reconfigurable fabrics [24, 31, 55]. We derive an upper bound for all-to-all throughput (equivalently, lower bound the all-to-all collective time) for arbitrary d -regular graphs with N nodes, and then highlight an expander graph that achieves performance close to this bound. Since the bound is reasonably tight, it allows us to compare the performance of various topologies with respect to the theoretical optimal.

Theorem 1 (Lower bound on all-to-all time.). *The time taken to accomplish all-to-all communication in a d -regular graph G on N nodes scales as $\Omega(N \log_d N)$.*

Proof sketch. First, consider a single source node r and an arborescence (outgoing rooted directed tree) of G (denoted by $T_{d,N}$), which has N nodes and maximum out-degree d . It has d^k number of nodes at all levels k except when k is equal to its height. If r needs to send $N - 1$ flows of value f to each of the other $N - 1$ nodes along $T_{d,N}$, the minimum capacity needed is $f \times \sum_{u \in V_{T_{d,N}}} D(r, u)$, where $D(s, t)$ is the distance (hop count) between s and t along $T_{d,N}$. Thus, the minimum capacity needed for sending commodities from all nodes (along N respective rooted arborescences) is $N \times f \times \sum_{u \in V_{T_{d,N}}} D(r, u)$. Assuming that a d -regular di-graph has d outgoing unit capacity links per node, the total capacity available in the network is $d \times N$. It follows that $f \times \sum_{u \in V_{T_{d,N}}} D(r, u) \leq d$. The all-to-all workload completion time/latency is given by:

$$1/f \geq \sum_{u \in V_{T_{d,N}}} D(r, u)/d. \quad (25)$$

Also, in a d -regular graph where each arborescence is a full k layer tree, $N = \sum_{i=0}^{k-1} d^i = \frac{d^k - 1}{d - 1}$ and $\sum_{u \in V_{T_{d,N}}} D(r, u) = \sum_{i=0}^{k-1} i \times d^i = \frac{d^{k+1}(k-1) - d^k k + d}{(d-1)^2} = \Theta(k d^{k-1})$. The RHS of Eq. (25) then becomes $\Theta(k d^{k-2}) = \Theta(N \log_d N)$. This establishes the scaling law for the lower bound on all-to-all time in any d -regular N -node topology. \square

Graphs achieving all-to-all time bound Generalized Kautz graphs [21] constitute a family of *expander graphs* that comes close to achieving the bound in Theorem 1. A benefit of these graphs is that we can generate an instance for *any* value of N and d . Such coverage in N and d is not possible for most graph families popular in the HPC community, such as mesh, tori, SlimFly [13], SpectralFly [58], etc. Fig. 10(left) shows the simulated all-to-all performance of the GenKautz graph for $d = 4$ with respect to the derived lower bound. We observe that these graphs get very close to the all-to-all lower bound for any d -regular graph (with the ratio between the two approaching 1 for large N) thus making them optimal expanders for all-to-all collectives.

Fig. 10 (right) compares the performance of GenKautz with non-expanders (e.g., 2D-tori) and other well-known expander graphs, e.g., Xpander [54] and random regular

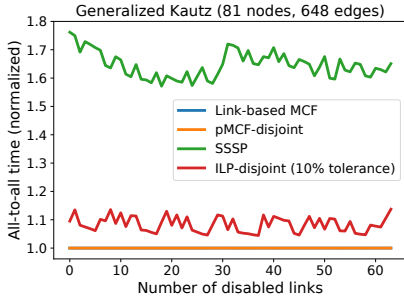


Figure 9: Performance on $N=81$ Gen Kautz w/ links disabled normalized by Link-based MCF

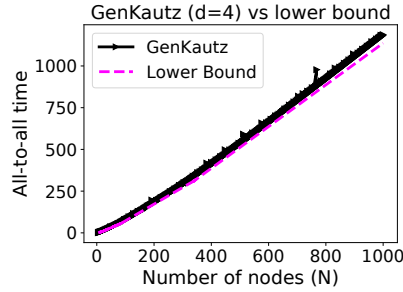
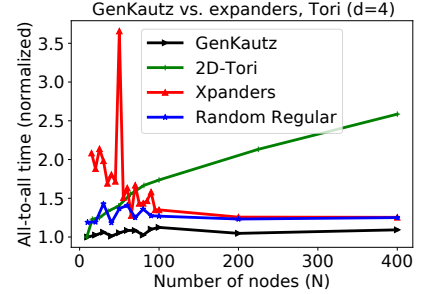


Figure 10: GenKautz topology performance relative to lower bound (left), and expanders and 2D-Tori (right, normalized by lower bound)



graphs /Jellyfish [48]. The last two are also known to exhibit good coverage in N and d . While the expanders significantly outperform non-expanders (for $d = 4$, GenKautz has about $2.4\times$ lower latency than 2D-tori for large N), GenKautz has the best performance among the expanders (e.g., it is 10% better than Xpander and random regular graphs). Similar trends are observed for higher degrees.

5.5 Conclusion and Discussion

We develop efficient schedules and topologies for all-to-all collective communications geared for large-scale direct-connect fabrics. Our results demonstrate that the time-stepped MCF approach and compiler are highly scalable and achieve near-optimal bandwidth performance in practice. This is valuable for many applications that rely on all-to-all when run on GPU (or CPU) clusters, such as deep learning training and inference. For path-based MCF-extP and pMCF, our results similarly demonstrate excellent performance in practice at the smaller 8 and 27-node scales. Path-based MCF is able to exploit additional forwarding bandwidth to speed up the collective. Our experiments with MCF, however, uncovered additional theoretical and practical challenges, which we are addressing as part of our ongoing (and future) work.

Deadlock-free routing: When lowering path-based MCF routes to fabrics with wormhole (flit-based) routing such as Cerio, routes must be deadlock-free [17]. We implemented several variants of common algorithms for breaking deadlocks, such as DF-SSSP [19] and LASH [49], which assign virtual channels to routes after the routes are computed. We found that a variant of LASH [49], which we call LASH-sequential performed best in terms of requiring the least number of layers; specifically, it required no more than 4 layers across all the algorithms (MCF, ILP, EwSP, etc.) and topologies we evaluated. Minimizing the number of VCs to make a given set of routes deadlock free is NP-hard [19]. An open question is *how to generate all-to-all schedules that optimize throughput while ensuring deadlock freedom*.

Injection rate control: On the practical front, a limitation of path-based MCF solutions when lowered to existing fabrics is support for *injection rate control*. As described in §4, we implemented an approximation of injection rate control by

splitting shards into granular equal-sized chunks/flows and steering them on routes. While this approach worked very well at an 8-node scale as a proof of concept, and achieved reasonable performance at a larger 27-node scale, it is in general not scalable. Granular chunking significantly increases the total number of active Queue Pairs (QPs) in the network. And our all-to-all micro-benchmarking experiments clearly showed a reduction in the achievable per-flow bandwidth as the number of QPs increased on the Cerio fabric, likely due to increased contention. We are pursuing two approaches to address this challenge: (1) introduce time steps into the routed MCF schedules and partition the flows across multiple timesteps, and (2) work with the Cerio vendor on options to expose injection rate control in the hardware.

Clustered/Hybrid Configurations: We are extending the general MCF formulation and the implementation to handle hybrid clustered settings with possibly severe imbalance between internal link bandwidth within a server, and external bandwidth (e.g., several Tbps internal bandwidth vs several Gbps external bandwidth in GPU servers from NVIDIA and AMD), and with possibly internal switching.

Future work: Our solution computes a static schedule that assumes dedicated underlying link bandwidth for the duration of the all-to-all collective. This is a reasonable assumption in several direct-connect settings where a cluster manager allocates circuits to jobs on a non-interfering basis. Handling more dynamic environments with multiple jobs contending for bandwidth is left for future work.

6 Acknowledgement

We thank the Rockport Networks team, Matthew Williams, Doug Taylor, Nick Tkotz, and Shaun Hennesey for their help and insights with the Rockport fabric experiments, and with making their slice of the TACC supercomputer available to us. We also thank Amit Ruhela for his help and insights. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) under contract number HR001120C0089. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

- [1] 2021. NVIDIA A100 Tensor Core GPU. <https://www.nvidia.com/en-us/data-center/a100/>.
- [2] 2021. TACC establishes new Center of Excellence with Rockport Networks. <https://rockportnetworks.com/tacc-establishes-new-center-of-excellence-with-rockport-networks/>.
- [3] 2023. Cerio. <https://www.cerio.io/>
- [4] 2023. Parallelization of Particle-mesh Ewald (PME) in GROMACS for Molecular Dynamics. Available at <https://manual.gromacs.org/documentation/current/user-guide/mdrun-performance.html>, accessed on 7.15.2023.
- [5] 2023. Texas Advanced Computing Center. <https://www.tacc.utexas.edu/>.
- [6] 2024. Fastest Fourier Transform in the West. <https://fftw.org/>.
- [7] AMD. 2023. ROCm Communication Collectives Library. Available at <https://github.com/ROCmSoftwarePlatform/rccl>, accessed on 7.15.2023.
- [8] MOSEK ApS. 2023. *MOSEK Solver version 10.1*. <https://www.mosek.com/>
- [9] Behnaz Arzani, Siva Kesava Reddy Kakarla, Miguel Castro, Srikanth Kandula, Saeed Maleki, and Luke Marshall. 2023. Rethinking Machine Learning Collective Communication as a Multi-Commodity Flow Problem. arXiv:2305.13479 [cs.NI]
- [10] Alan Ayala, Stanimire Tomov, Xi Luo, Hejer Shaeik, Azzam Haidar, George Bosilca, and Jack Dongarra. 2019. Impacts of multi-gpu mpi collective communications on large fft computation. In *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*. IEEE, 12–18.
- [11] Prithwish Basu, Feng Yu, Amotz Bar-Noy, and Dror Rawitz. [n. d.]. *To Sample or To Smash? Estimating reachability in large time-varying graphs*. 983–991. <https://doi.org/10.1137/1.9781611973440.112> arXiv:<https://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.112>
- [12] Herman JC Berendsen, David van der Spoel, and Rudi van Drunen. 1995. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer physics communications* 91, 1-3 (1995), 43–56.
- [13] Maciej Besta and Torsten Hoefler. 2014. Slim fly: A cost effective low-diameter network topology. In *SC'14: proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE, 348–359.
- [14] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 62–75.
- [15] Michael B. Cohen, Yin Tat Lee, and Zhao Song. 2019. Solving Linear Programs in the Current Matrix Multiplication Time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (Phoenix, AZ, USA) (STOC 2019)*. Association for Computing Machinery, New York, NY, USA, 938?942. <https://doi.org/10.1145/3313276.3316303>
- [16] Kenneth Czechowski, Casey Battaglini, Chris McClanahan, Kartik Iyer, P-K Yeung, and Richard Vuduc. 2012. On the communication complexity of 3D FFTs and its implications for exascale. In *Proceedings of the 26th ACM international conference on Supercomputing*. 205–214.
- [17] William J. Dally and Charles L. Seitz. 1987. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on computers* 100, 5 (1987), 547–553.
- [18] William James Dally and Brian Patrick Towles. 2004. *Principles and practices of interconnection networks*. Elsevier.
- [19] Jens Domke, Torsten Hoefler, and Wolfgang E. Nagel. 2011. Deadlock-Free Oblivious Routing for Arbitrary Topologies. In *2011 IEEE International Parallel & Distributed Processing Symposium*. 616–627. <https://doi.org/10.1109/IPDPS.2011.65>
- [20] Lisa K Fleischer. 2000. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics* 13, 4 (2000), 505–520.
- [21] Imase and Itoh. 1983. A Design for Directed Graphs with Minimum Diameter. *IEEE Trans. Comput.* C-32, 8 (1983), 782–784. <https://doi.org/10.1109/TC.1983.1676323>
- [22] Intel. 2023. oneAPI Collective Communications Library (oneCCL). Available at <https://github.com/oneapi-src/oneCCL>, accessed on 07.15.2023.
- [23] S Lennart Johnsson and C-T Ho. 1989. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on computers* 38, 9 (1989), 1249–1268.
- [24] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, et al. 2023. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–14.
- [25] Sangeetha Abdu Jyothi, Ankit Singla, P Brighten Godfrey, and Alexandra Kolla. 2016. Measuring and understanding throughput of network topologies. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 761–772.
- [26] George Karakostas. 2008. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms (TALG)* 4, 1 (2008), 1–17.
- [27] Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, and Eiman Ebrahimi. 2021. SiP-ML: High-Bandwidth Optical Network Interconnects for Machine Learning Training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (Virtual Event, USA) (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 657–675. <https://doi.org/10.1145/3452296.3472900>
- [28] John Kim, William J Dally, Steve Scott, and Dennis Abts. 2008. Technology-driven, highly-scalable dragonfly topology. *ACM SIGARCH Computer Architecture News* 36, 3 (2008), 77–88.
- [29] Tom Leighton and Satish Rao. 1989. *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*. Technical Report. Massachusetts Inst Of Tech Cambridge Microsystems Research Center.
- [30] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).
- [31] Hong Liu, Ryohei Urata, Kevin Yasumura, Xiang Zhou, Roy Bannon, Jill Berger, Pedram Dashti, Norm Jouppi, Cedric Lam, Sheng Li, Erji Mao, Daniel Nelson, George Papan, Mukarram Tariq, and Amin Vahdat. 2023. Lightwave Fabrics: At-Scale Optical Circuit Switching for Datacenter and Machine Learning Systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA.
- [32] Yunfeng Lu, Huaxi Gu, Xiaoshan Yu, and Peng Li. 2021. X-NEST: A Scalable, Flexible, and High-Performance Network Architecture for Distributed Machine Learning. *Journal of Lightwave Technology* 39, 13 (2021), 4247–4254. <https://doi.org/10.1109/JLT.2021.3073277>
- [33] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. 2020. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 1–18.
- [34] Microsoft. 2022. Microsoft Collective Communication Library. Available at <https://github.com/microsoft/msccl>.
- [35] Parviz Moin and Krishnan Mahesh. 1998. Direct numerical simulation: a tool in turbulence research. *Annual review of fluid mechanics* 30, 1 (1998), 539–578.

- [36] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 993–1011.
- [37] Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, et al. 2020. Deep learning training in facebook data centers: Design of scale-up and scale-out systems. *arXiv preprint arXiv:2003.09518* (2020).
- [38] NVIDIA. 2023. NVIDIA Collective Communication Library (NCCL). Available at <https://github.com/NVIDIA/nccl>, accessed on 7.15.2023.
- [39] Open MPI. [n. d.]. Open Source High Performance Computing. <https://www.open-mpi.org/>.
- [40] Dmitry Pekurovsky. 2012. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. *SIAM Journal on Scientific Computing* 34, 4 (2012), C192–C209. <https://doi.org/10.1137/11082748X> arXiv:<https://doi.org/10.1137/11082748X>
- [41] Polatis 2023. Polatis Optical Circuit Switch. Available at <https://www.polatis.com/series-7000-384x384-port-software-controlled-optical-circuit-switch-sdn-enabled.asp>.
- [42] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beaugard, Patrick Conner, Steve Gribble, et al. 2022. Jupiter evolving: transforming google’s datacenter network via optical circuit switches and software-defined networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 66–85.
- [43] Sarunya Puma and Abhinav Vishnu. 2021. Semantic-Aware Lossless Data Compression for Deep Learning Recommendation Model (DLRM). In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. IEEE, 1–8.
- [44] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 Conference* (Toronto, Ontario, Canada) (SIGCOMM '11). Association for Computing Machinery, New York, NY, USA, 266?277. <https://doi.org/10.1145/2018436.2018467>
- [45] David S Scott. 1991. Efficient all-to-all communication patterns in hypercube and mesh topologies. In *The Sixth Distributed Memory Computing Conference, 1991. Proceedings*. IEEE Computer Society, 398–399.
- [46] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 593–612. <https://www.usenix.org/conference/nsdi23/presentation/shah>
- [47] Farhad Shahrokhi and David W Matula. 1990. The maximum concurrent flow problem. *Journal of the ACM (JACM)* 37, 2 (1990), 318–334.
- [48] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P Brighten Godfrey. 2012. Jellyfish: Networking data centers randomly. In *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*. 225–238.
- [49] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. 2002. Layered Shortest Path (LASH) Routing in Irregular System Area Networks.. In *ipdps*, Vol. 2. 194.
- [50] Evandro DE SOUZA. 2022. Deadlock-free multipath routing for direct interconnect networks. Available at <https://patents.google.com/patent/WO2022269357A1/en>.
- [51] Young-Joo Suh and S Valamanchili. 1998. All to-all communication with minimum start-up costs in 2d/3d tori and meshes. *IEEE Transactions on Parallel and Distributed Systems* 9, 5 (1998), 442–458.
- [52] Telescent. 2021. G4 Network Topology Manager. <https://www.telescent.com/products>
- [53] Thao-Nguyen TRUONG and Ryousei TAKANO. 2021. Hybrid Electrical/Optical Switch Architectures for Training Distributed Deep Learning in Large-Scale. *IEICE Transactions on Information and Systems* E104.D, 8 (2021), 1332–1339. <https://doi.org/10.1587/transinf.2020EDP7201>
- [54] Asaf Valadarsky, Gal Shahaf, Michael Dinitz, and Michael Schapira. 2016. Xpander: Towards optimal-performance datacenters. In *Proceedings of the 12th International Conference on emerging Networking Experiments and Technologies*. 205–219.
- [55] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. {TopoOpt}: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 739–767.
- [56] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. {TopoOpt}: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 739–767.
- [57] Yuanyuan Yang and Jianchao Wang. 1999. Efficient all-to-all broadcast in all-port mesh and torus networks. In *Proceedings Fifth International Symposium on High-Performance Computer Architecture*. IEEE, 290–299.
- [58] Stephen Young, Sinan Aksoy, Jesun Firoz, Roberto Gioiosa, Tobias Hagge, Mark Kempton, Juan Escobedo, and Mark Raugas. 2022. Spectrally: Ramanujan graphs as flexible and efficient interconnection networks. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1040–1050.
- [59] Liangyu Zhao, Siddharth Pal, Tapan Chugh, Weiyang Wang, Prithwish Basu, Joud Khoury, and Arvind Krishnamurthy. 2022. Optimal Direct-Connect Topologies for Collective Communications. *arXiv preprint arXiv:2202.03356* (2022).
- [60] Liangyu Zhao, Siddharth Pal, Tapan Chugh, Weiyang Wang, Jason Fantl, Prithwish Basu, Joud Khoury, and Arvind Krishnamurthy. 2022. Efficient Direct-Connect Topologies for Collective Communications. *arXiv preprint arXiv:2202.03356* (2022).
- [61] Ziyi Zhu, Min Yee Teh, Zhenguo Wu, Madeleine Strom Glick, Shijia Yan, Maarten Hattink, and Keren Bergman. 2022. Distributed deep learning training using silicon photonic switched architectures. *APL Photonics* 7, 3 (2022), 030901. <https://doi.org/10.1063/5.0070711>