Efficient Direct-Connect Topologies for Collective Communications

Liangyu Zhao[1] Siddharth Pal[2] Tapan Chugh[1] Weiyang Wang[3] Jason Fantl[2]
Prithwish Basu[2] Joud Khoury[2] Arvind Krishnamurthy[1]

[1]University of Washington *[2]RTX BBN Technologies* *[3]MIT CSAIL*

ACE Theme 3
To be presented at NSDI '25

Table of Contents

Collective Communication

- **Collective Communication**: a set of communication operations among a group of nodes in a parallel computing system, serving as building blocks for distributed computing.
  - e.g. broadcast, reduce, allgather, reduce-scatter, allreduce, all-to-all, etc.
- Originally a topic in HPC, it is now extensively used for gradient, parameter, and activation synchronization in distributed ML training and inferencing.

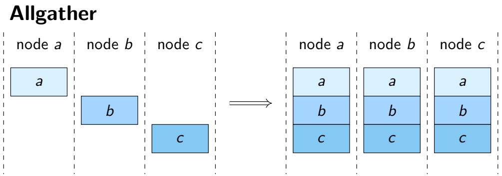


Figure: Allgather Operation

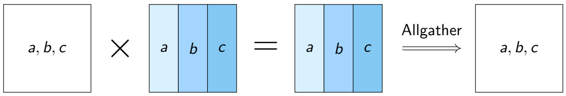
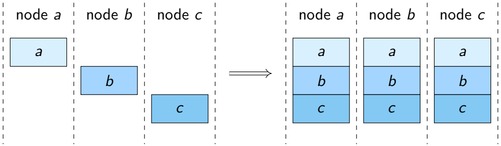


Figure: Distributed Matrix Multiplication

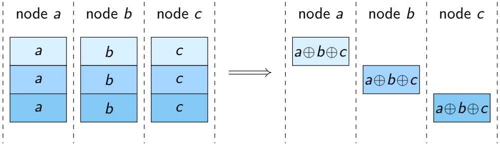
Collective Communication

- We focus on accelerating **allgather**, **reduce-scatter**, and **allreduce**, three widely used collective operations in distributed ML.
- **Focus on Allgather:** allgather can be transformed into reduce-scatter and allreduce.
  - reduce-scatter = *reversed* allgather
  - allreduce = reduce-scatter + allgather

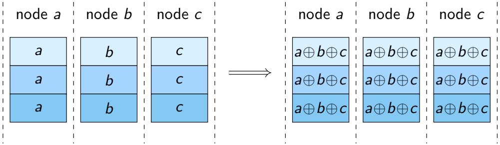
Allgather



Reduce-Scatter

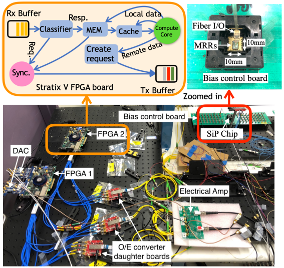


Allreduce

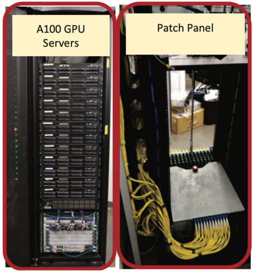


An emerging approach is to use **optical circuit network**:

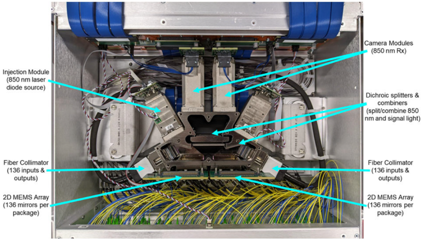
- Higher bandwidth at lower capital expenditure and energy cost.
- The network can be configured into any **direct-connect** topology.
- Exhibit **high reconfiguration latency**, requiring relatively fixed topologies.



(a) SiP-ML (SIGCOMM '21)



(b) TopoOpt (NSDI '23)

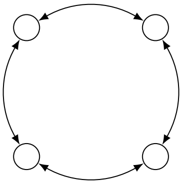


(c) TPU (Google)

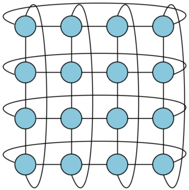
Direct-Connect Network

The topology of an optical circuit network can be modeled as a **direct-connect network**:

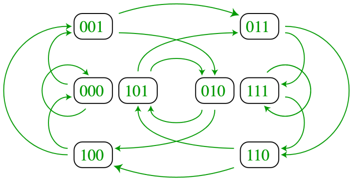
- Nodes are **directly connected** without the use of packet switches. Pairs of unconnected nodes cannot communicate directly.
- The network can be **unidirectional** (directed graph) or **bidirectional** (undirected graph).
- The topology is typically d-**regular** and **homogeneous**.
- α-β **cost model:** the time cost of sending a size-M message over a link is $\alpha + M/b$.



(a) Ring



(b) Torus



(c) de Bruijn

Research Problem

Problem: For a given workload (e.g., ML or HPC), what is the most efficient topology?

- Allreduce-Type Collectives (e.g., allgather, reduce-scatter, allreduce)

latency sensitive **low-diameter topology**	small ⟸ Data Size ⟹ large	throughput sensitive **load-balanced transmission**

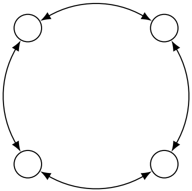
- All-to-All Communication
  - Because of bandwidth tax, point-to-point flows should be as short as possible.
  - All-to-all throughput also requires **low-diameter topology**.
- Workloads may require both low diameter and load-balanced allreduce.
  - e.g., expert-parallel training involving both allreduce and all-to-all.

Ideal Topology: low-diameter topology with load-balanced collective communication.

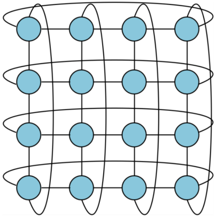
Traditional HPC Topologies

Problem: For a given workload (e.g., ML or HPC), what is the most efficient topology?

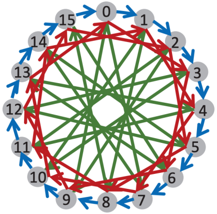
- Traditional HPC topologies are limited to a few ring-based graphs.
  - e.g., ring, torus, multi-ring.
- **Pros:** load-balanced collective, high-throughput allreduce-type collective operations.
- **Cons:** high diameter, detrimental for all-to-all throughput and latency-sensitive small-data allreduce.



(a) Ring



(b) Torus

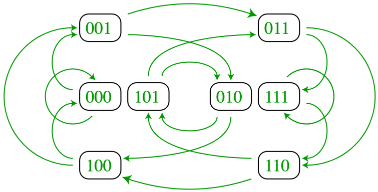


(c) TopoOpt Multi-Ring

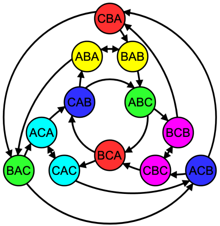
Low-Diameter Expander Graphs

Problem: For a given workload (e.g., ML or HPC), what is the most efficient topology?

- Low-diameter expander graphs from graph theory.
  - e.g., de Bruijn graph, Kautz graph.
- **Pros:** low diameter, ideal for all-to-all throughput and small-data allreduce.
- **Cons:** complex structure, lack of load-balanced allreduce-type schedules.



(a) de Bruijn Graph



(b) Kautz Graph

Topology Dilemma

Problem: For a given workload (e.g., ML or HPC), what is the most efficient topology?

Topology Type	**Small-Data Allreduce** Latensy-Sensitive	**Large-Data Allreduce** Throughput-Sensitive	**All-to-All** Throughput
Traditional HPC Topologies	–	✓	–
Low-Diameter Expander Graphs	✓	–	✓

Topology Dilemma

Problem: For a given workload (e.g., ML or HPC), what is the most efficient topology?

Topology Type	**Small-Data Allreduce** Latensy-Sensitive	**Large-Data Allreduce** Throughput-Sensitive	**All-to-All** Throughput
Traditional HPC Topologies	×	✓	×
Low-Diameter Expander Graphs	✓	–	✓

- Latency and all-to-all throughput are bounded by topology diameter.

Topology Dilemma

Problem: For a given workload (e.g., ML or HPC), what is the most efficient topology?

Topology Type	**Small-Data Allreduce** Latensy-Sensitive	**Large-Data Allreduce** Throughput-Sensitive	**All-to-All** Throughput
Traditional HPC Topologies	×	✓	×
Low-Diameter Expander Graphs	✓	**???**	✓

- Latency and all-to-all throughput are bounded by topology diameter.
- **Question:** Can we have *load-balanced* allreduce schedules on *low-diameter* topologies?

Challenge

Challenge: Optimizing communication schedule can be **computationally intractable**.

- **Data Dependency:** unlike point-to-point traffic, flow conservation is not sufficient to maintain data dependency in collective communication due to multicast/aggregation.
- Earlier works track data dependency in chunks, leading to NP-hard discrete optimization.
  - SCCL [PPoPP '21] uses *satisfiability modulo theories* (SMT).
  - TACCL [NSDI '23], TE-CCL [SIGCOMM '24] use *mixed integer linear program* (MILP).

# of nodes	4	8	16	32	64
SCCL	0.59s	0.86s	21.4s	$>10^4$s	$>10^4$s
TACCL	0.50s	7.39s	1801s	1802s	n/a

Table: Generation Time on Hypercube

# of nodes	4	9	16	25	36
SCCL	0.61s	1.00s	60s	3286s	$>10^4$s
TACCL	0.45s	67.8s	1801s	1802s	n/a

Table: Generation Time on 2D Torus ($n \times n$)

Table of Contents

Solution Overview

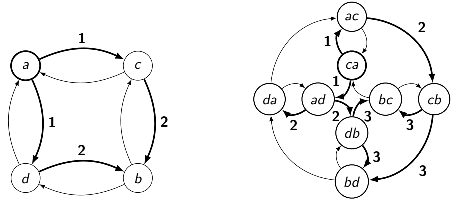
- **Expansion Techniques:** expand small-scale optimized topologies and schedules into large-scale ones.
  - Avoid intractable direct construction of large-scale topologies and schedules.
- **Schedule Generation:** generate optimal Breadth-First-Broadcast (BFB) schedule on large topologies.
  - Optimizing BFB schedule can be done with polynomial-time linear program.

Expansion Techniques

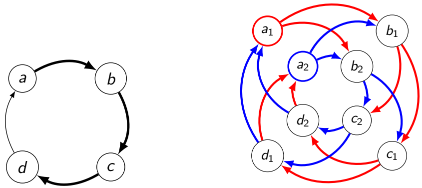
Given a base topology and its associated communication schedule,

- We have graph transformations to expand the **base topology** into larger ones.
- The **base schedule** is also expanded to match the expanded topology.
- The expansion involves **simple mapping** of nodes, edges, and data send/recv.
- The sacrifice in **overall performance** is mathematically bounded during the process.

Line Graph Expansion:

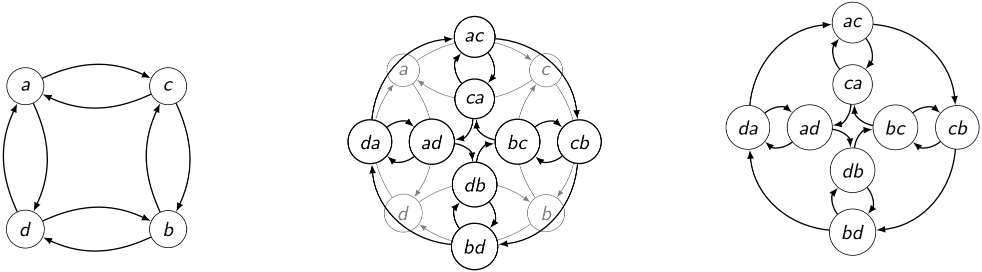


Degree Expansion:



We borrow the concept of **line graph** from graph theory:

- Edge (u, v) in base graph $G \iff$ Node uv in the line graph $L(G)$.
- For every uv, vw node pair in the line graph, there is an edge (uv, vw).
- $N_{L(G)} = N_G \cdot \deg G; \quad \deg L(G) = \deg G$.



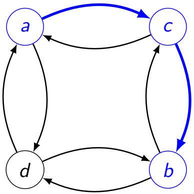
(a) $K_{2,2}$ $(N = 4, d = 2)$

(b) $L(K_{2,2})$ $(N = 8, d = 2)$

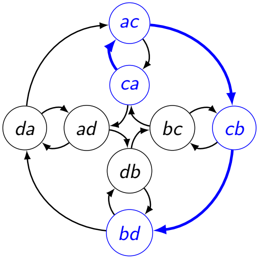
Line Graph Expansion

The **schedule** can be mapped from base topology to the expanded topology:

- Any (shortest) path $w_0 \to w_1 \to \cdots \to w_n$ in $K_{2,2}$ can be mapped to a (shortest) path $w_{-1}w_0 \to w_0w_1 \to \cdots \to w_{n-1}w_n \to w_nw_{n+1}$ in $L(K_{2,2})$, for any $w_{-1}w_0 \neq w_nw_{n+1}$.
- Data going from ca to bd in $L(K_{2,2})$ can follow the corresponding path of a to b in $K_{2,2}$.



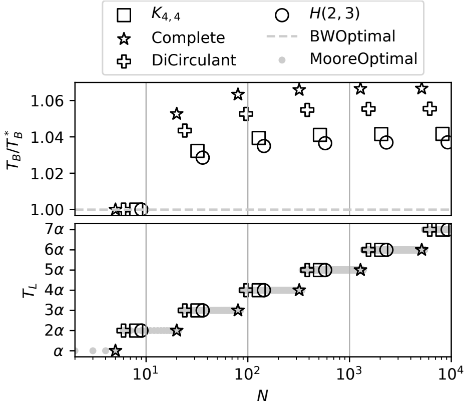
(a) $K_{2,2}$ ($N = 4, d = 2$)



(b) $L(K_{2,2})$ ($N = 8, d = 2$)

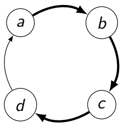
Line Graph Expansion

- Line graph expansion can be **applied repeatedly** to scale topology and schedule indefinitely.
    - Node degree is preserved, friendly to hardware constraints.
- The performance sacrifice is limited.
    - If the base is throughput-optimal with N nodes, then the expanded schedule is **at most** $\frac{1}{(d-1)N}$ away from throughput optimality.
    - Expansion preserves low-diameter property. N increases d-fold while diameter increases by 1.

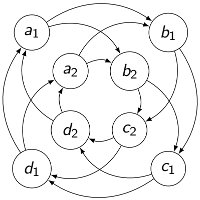


Degree Expansion

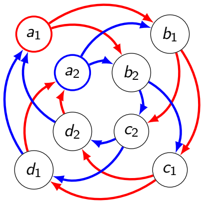
- $G * n$ makes n copies of G, and connect (a_i, b_j) for any (a, b) in G.
  - $\deg(G * n) = n \cdot \deg G$; $N_{G*n} = n \cdot N_G$.
- Degree expansion preserves throughput optimality.
  - Broadcast path in figure (a) is mapped to non-overlapping red and blue paths in (c).



(a) $G(N=4, d=1)$



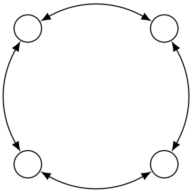
(b) $G*2$ $(N=8, d=2)$



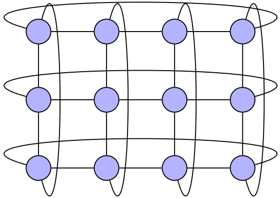
(c) Broadcasts w.r.t. a_1, a_2

Cartesian Product Expansion

- From graph theory, given graphs $G_1, G_2, \ldots, G_n$, we can construct a Cartesian product graph $G_1 \square G_2 \square \ldots \square G_n$.
    - $N_{G_1 \square G_2 \square \ldots \square G_n} = \prod_i N_{G_i}$; $\deg(G_1 \square G_2 \square \ldots \square G_n) = \sum_i \deg(G_i)$.
- $G_1 \square G_2 \square \ldots \square G_n$ is throughput-optimal if each G_i is throughput-optimal.
    - e.g., torus with **arbitrary dimensions** $d_1 \times d_2 \times \cdots \times d_n$. Previously, only torus with **equal dimensions** ($d_1 = d_2 = \cdots = d_n$) has efficient schedules.
    - Use BFB schedule generation (to be introduced later).
    - Cartesian product greatly expands the set of throughput-optimal topologies we construct.



(a) 4-Node Ring $R_4 (N = 4, d = 2)$



(b) 3x4 Torus $R_3 \square R_4 (N = 12, d = 4)$

Topology Finder

- Given a target topology size, the topology finder explores all known base topologies and potential combinations of expansion techniques.
- The resulting candidate topologies and schedules form a **Pareto-frontier**. The best one is then decided by hardware/workload specifications.
  - Pareto-frontier: **low-diameter** vs. **load-balanced allreduce**.
  - All-to-all performance is strongly related to graph diameter $D(G)$.

Expansion Techniques	# of Nodes	Deg	Moore	BW
Line Graph Exp $L^n(G)$	$d^n N$	d	✓	×
Degree Exp $G * n$	nN	nd	×	✓
Cartesian Power $G^{\square n}$	N^n	nd	×	✓
Cartesian Prod $G_1 \square \ldots \square G_n$	$\prod_i N_i$	$\sum_i d_i$	×	✓

Table: Summary of Expansion Techniques

Topology	T_L	T_B	$2(T_L+T_B)$	$D(G)$	All-to-All
$\Pi_{4,1024}$	5α	$1.332 M/B$	323.5us	5	409.1us
$L^3(C(16,\{3,4\}))$	6α	$1.020 M/B$	**291.0us**	6	**403.5us**
$L^2(\text{Diamond}^{\square 2})$	8α	$1.004 M/B$	328.4us	8	446.6us
$L(\text{DBJMod}(2,4)^{\square 2})$	11α	$1.000 M/B$	387.8us	9	529.9us
$(\text{UniRing}(1,4) \square \text{UniRing}(1,8))^{\square 2}$	20α	$0.999 M/B$	567.6us	20	1174.4us
Baseline: 32x32 Torus	62α	$0.999 M/B$	1407.6us	32	1342.2us
Theoretical Bound	$\mathbf{5\alpha}$	$\mathbf{0.999} M/B$	**267.6us**	**5**	**382.3us**

Table: Pareto-frontier for $N=1024$, $d=4$ with $\alpha=10\mu s$ and $M/B=1\text{MB}/100\text{Gbps}$.

Motivation

Observations:

- Expansion techniques have huge gaps in the coverage of topology sizes.
  - Given a base topology with $N = 4, d = 2$, line graph expansion can only generate topologies of $8, 16, 32, \ldots$ $(d^n N)$ number of nodes.
- There exist off-the-shelf low-diameter expander graphs from graph theory.

Question

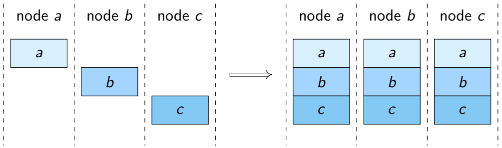
Given a topology from graph theory, can we *efficiently* construct an *efficient* schedule for it?

Breadth-First-Broadcast (BFB)

Allgather: each node broadcasts a shard of data simultaneously.

- We perform a **Breadth-First-Broadcast** (BFB) from each node.
  - At time step t, from each source node, nodes at distance $t-1$ collectively broadcast the data shard to nodes at distance t.
- A **linear program** is used to balance workloads on links at each time step.
- Latency: data always follows the shortest paths, optimal for the given topology.
- Throughput: provably throughput-optimal for many types of graphs.

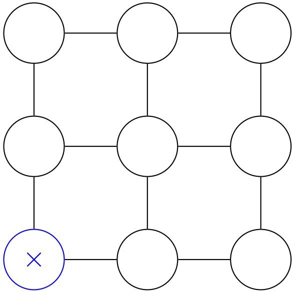
Allgather



$$
\begin{aligned}
\text{minimize} \quad & U_{u,t} \\
\text{subject to} \quad & \sum_{v} x_{v,(w,u),t} \le U_{u,t}, && \forall w \in N^{-}(u) \\
& \sum_{w} x_{v,(w,u),t} = 1, && \forall v \in N_t^{-}(u) \\
& 0 \le x_{v,(w,u),t} \le 1. && \forall w, v
\end{aligned}
$$

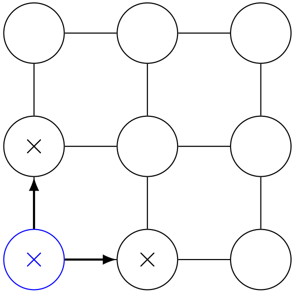
BFB Example

Breadth-First-Broadcast:



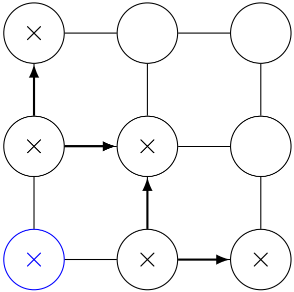
BFB Example

Breadth-First-Broadcast:



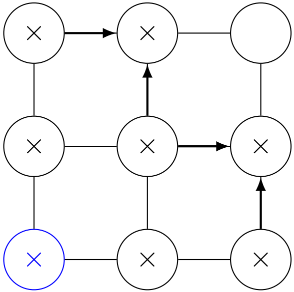
BFB Example

Breadth-First-Broadcast:



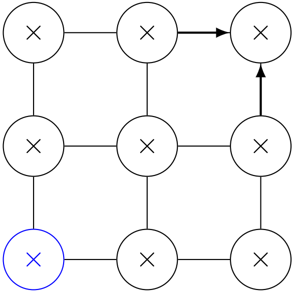
BFB Example

Breadth-First-Broadcast:

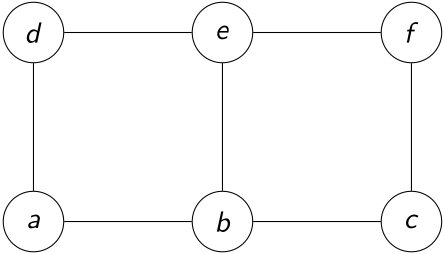


BFB Example

Breadth-First-Broadcast:

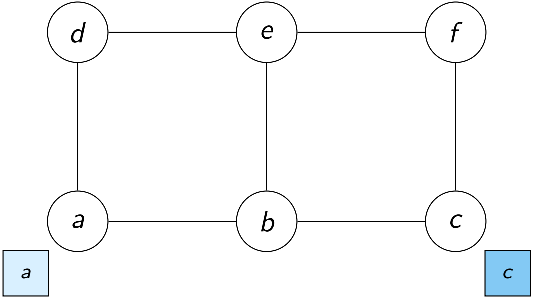


BFB Linear Program



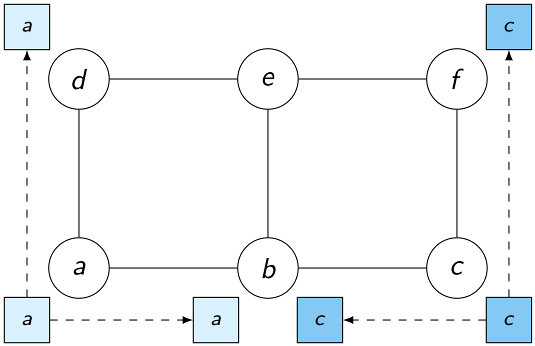
BFB Linear Program

Nodes a and c each have a data shard to broadcast.



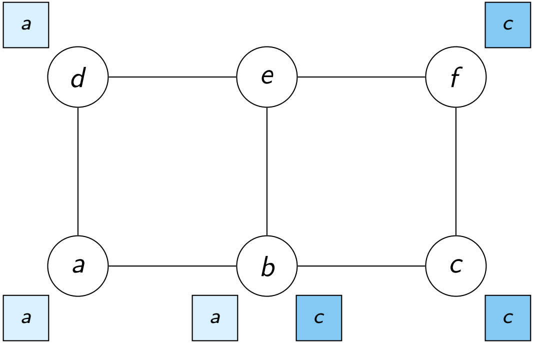
BFB Linear Program

Broadcast data to neighbors.



BFB Linear Program

Broadcast data to neighbors.

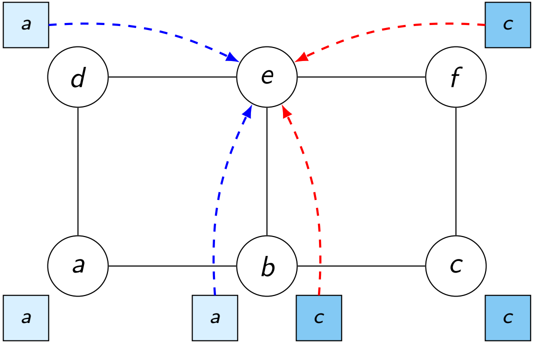


BFB Linear Program

Broadcast data to node e.

- Both b, d can provide shard a.
- Both b, f can provide shard c.

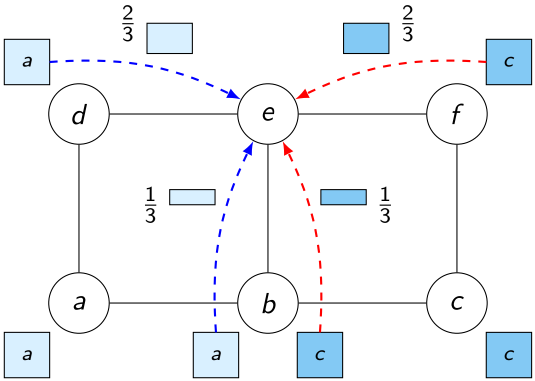
Question: How can data be sent while balancing the workload across links?



BFB Linear Program

Perfect balance is achieved if

- d sends $\frac{2}{3}$ of shard a.
- f sends $\frac{2}{3}$ of shard c.
- b sends $\frac{1}{3}$ of shard a and $\frac{1}{3}$ of shard c.

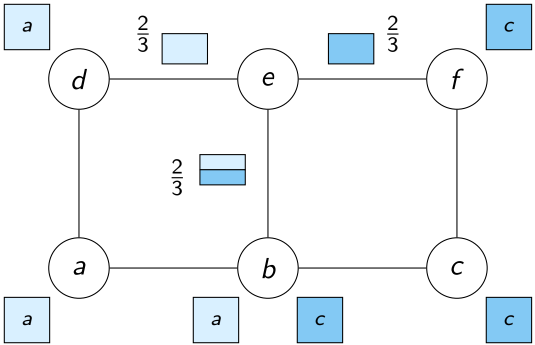


BFB Linear Program

Perfect balance is achieved if

- d sends $\frac{2}{3}$ of shard a.
- f sends $\frac{2}{3}$ of shard c.
- b sends $\frac{1}{3}$ of shard a and $\frac{1}{3}$ of shard c.

Each link sends $\frac{2}{3}$ of a shard in total.



BFB Linear Program

For each node $u \in V$ and time step $t \in \{1, 2, \ldots, D(G)\}$,

- Data shards from source nodes v at distance t **to** u should reach u at step t.
- $x_{v,(w,u),t}$ is the proportion of v's shard sent through link (w, u) at step t.
    - Only nodes w on the shortest paths from v to u can provide v's data shard; otherwise, $x_{v,(w,u),t}$ is undefined.

minimize	$U_{u,t}$		Minimize the max workload across links
subject to	$\sum_v x_{v,(w,u),t} \leq U_{u,t},$	$\forall w \in N^-(u)$	Ensure $U_{u,t}$ is the max workload
	$\sum_w x_{v,(w,u),t} = 1,$	$\forall v \in N_t^-(u)$	Ensure u receives all the data
	$0 \leq x_{v,(w,u),t} \leq 1.$	$\forall w, v$	

- Solve the linear program for each u and t.

BFB Linear Program

For each node $u \in V$ and time step $t \in \{1, 2, \ldots, D(G)\}$,

- Data shards from source nodes v at distance t **to** u should reach u at step t.
- $x_{v,(w,u),t}$ is the proportion of v's shard sent through link (w, u) at step t.
  - Only nodes w on the shortest paths from v to u can provide v's data shard; otherwise, $x_{v,(w,u),t}$ is undefined.

minimize	$U_{u,t}$		Minimize the max workload across links
subject to	$\sum_v x_{v,(w,u),t} \leq U_{u,t},$	$\forall w \in N^-(u)$	Ensure $U_{u,t}$ is the max workload
	$\sum_w x_{v,(w,u),t} = 1,$	$\forall v \in N_t^-(u)$	Ensure u receives all the data
	$0 \leq x_{v,(w,u),t} \leq 1.$	$\forall w, v$	

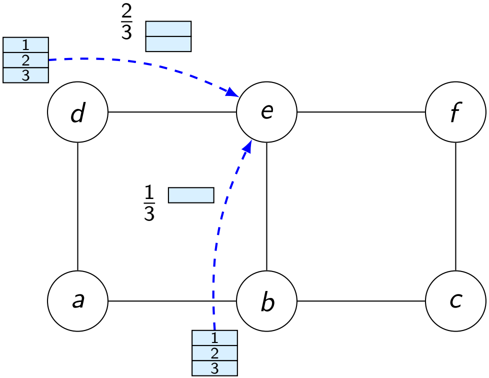
- Solve the linear program for each u and t.

Question: Why is BFB able to use a polynomial-time linear program rather than NP-hard discrete optimizations?

BFB Linear Program

Answer: BFB eliminates the need to track data dependencies using discrete data chunks.

What specific data are the $\frac{2}{3}$ shard sent by d and $\frac{1}{3}$ shard sent by b?

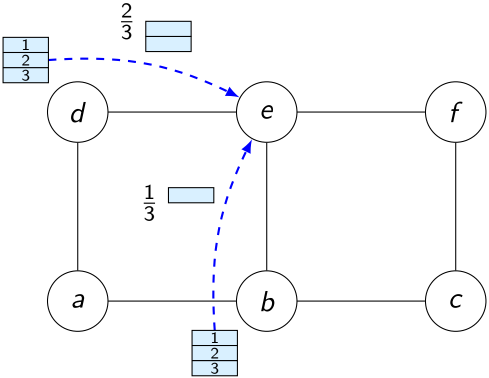


BFB Linear Program

Answer: BFB eliminates the need to track data dependencies using discrete data chunks.

What specific data are the $\frac{2}{3}$ shard sent by d and $\frac{1}{3}$ shard sent by b?

- In BFB, they can be **any parts** of shard a, as long as the union is the whole shard.

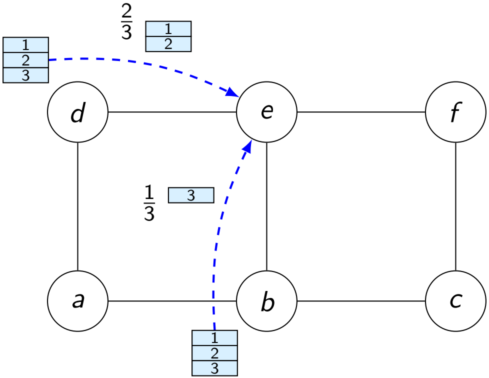


BFB Linear Program

Answer: BFB eliminates the need to track data dependencies using discrete data chunks.

What specific data are the $\frac{2}{3}$ shard sent by d and $\frac{1}{3}$ shard sent by b?

- In BFB, they can be **any parts** of shard a, as long as the union is the whole shard.
  - e.g., $\{1, 2\}$ and $\{3\}$

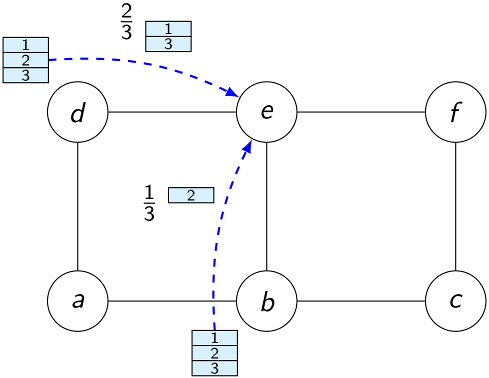


BFB Linear Program

Answer: BFB eliminates the need to track data dependencies using discrete data chunks.

What specific data are the $\frac{2}{3}$ shard sent by d and $\frac{1}{3}$ shard sent by b?

- In BFB, they can be **any parts** of shard a, as long as the union is the whole shard.
  - e.g., $\{1, 2\}$ and $\{3\}$, or $\{1, 3\}$ and $\{2\}$.

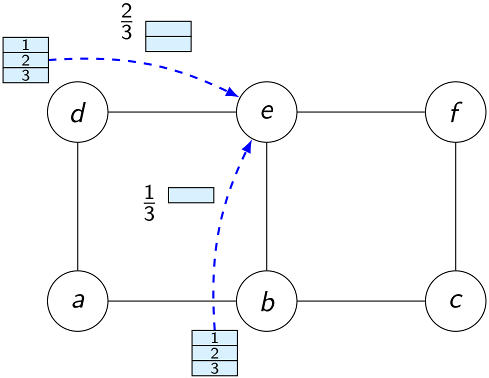


BFB Linear Program

Answer: BFB eliminates the need to track data dependencies using discrete data chunks.

What specific data are the $\frac{2}{3}$ shard sent by d and $\frac{1}{3}$ shard sent by b?

- In BFB, they can be **any parts** of shard a, as long as the union is the whole shard.
  - e.g., $\{1, 2\}$ and $\{3\}$, or $\{1, 3\}$ and $\{2\}$.
- Only the amount of data to be sent needs to be decided, not the specific data chunks, enabling a **continuous optimization**.

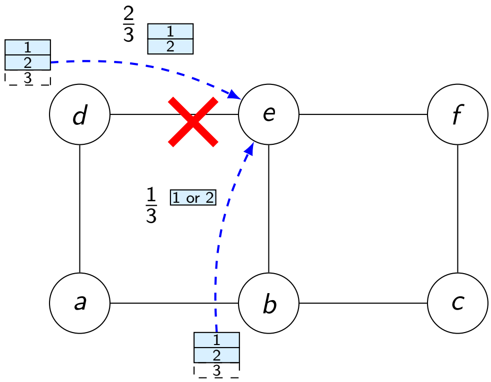


BFB Linear Program

Answer: BFB eliminates the need to track data dependencies using discrete data chunks.

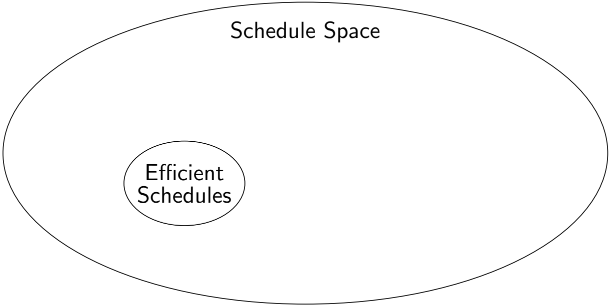
What specific data are the $\frac{2}{3}$ shard sent by d and $\frac{1}{3}$ shard sent by b?

- In BFB, they can be **any parts** of shard a, as long as the union is the whole shard.
  - e.g., $\{1, 2\}$ and $\{3\}$, or $\{1, 3\}$ and $\{2\}$.
- Only the amount of data to be sent needs to be decided, not the specific data chunks, enabling a **continuous optimization**.
- BFB ensures that b, d receive the entire shard before forwarding it to e, a guarantee not provided by all scheduling methods.



BFB Search Space

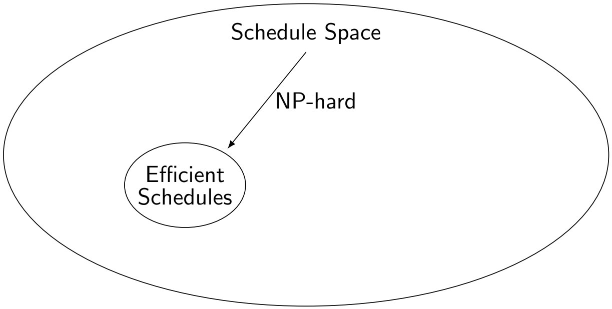
Question: How does BFB achieve polynomial-time schedule generation?



BFB Search Space

Question: How does BFB achieve polynomial-time schedule generation?

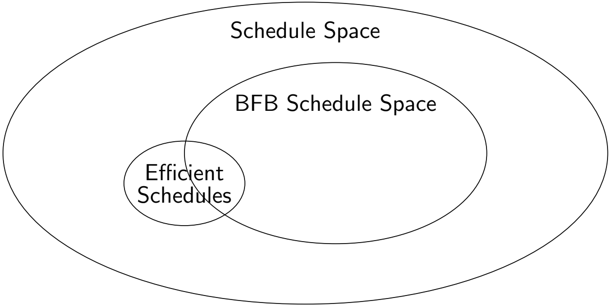
- Finding efficient schedules starting from the whole schedule space is NP-hard.



BFB Search Space

Question: How does BFB achieve polynomial-time schedule generation?

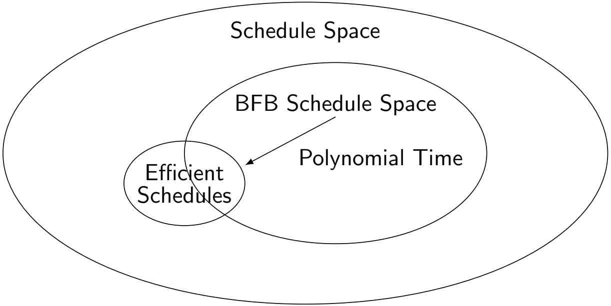
- Finding efficient schedules starting from the whole schedule space is NP-hard.
- BFB schedules are a subset of the schedule space.



BFB Search Space

Question: How does BFB achieve polynomial-time schedule generation?

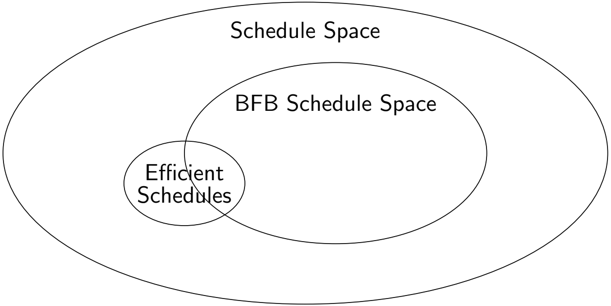
- Finding efficient schedules starting from the whole schedule space is NP-hard.
- BFB schedules are a subset of the schedule space.
- Finding efficient schedules within BFB schedule space is polynomial-time.



BFB Optimality

BFB linear program gives the optimal BFB schedule.

Question: the optimal BFB schedule = the globally optimal schedule?

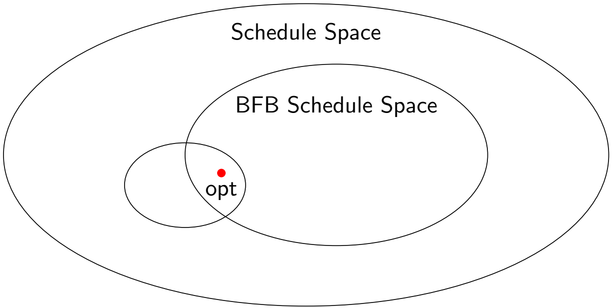


BFB Optimality

BFB linear program gives the optimal BFB schedule.

Question: the optimal BFB schedule = the globally optimal schedule?

- **Case 1:** the optimal BFB schedule is the globally optimal schedule.
  - **Topologies with certain symmetry properties,** e.g., torus with arbitrary dimensions, twisted torus used by TPU v4, circulant graph, distance-regular graph.
  - Cartesian product of graphs with globally optimal BFB schedules.

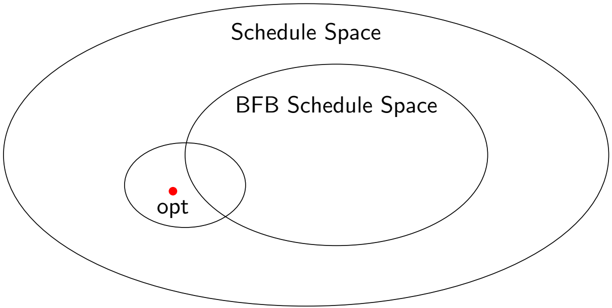


BFB Optimality

BFB linear program gives the optimal BFB schedule.

Question: the optimal BFB schedule = the globally optimal schedule?

- **Case 2:** the optimal BFB schedule is efficient but not the globally optimal schedule.
  - The optimal BFB schedule is **close to** throughput optimality, e.g., generalized Kautz Graphs.

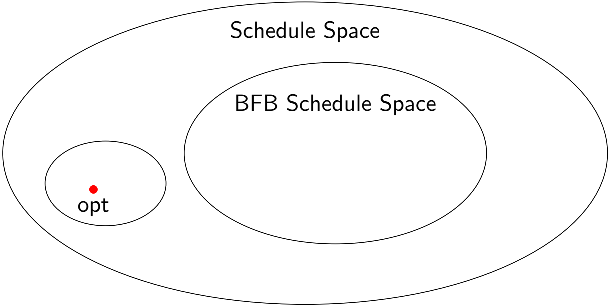


BFB Optimality

BFB linear program gives the optimal BFB schedule.

Question: the optimal BFB schedule = the globally optimal schedule?

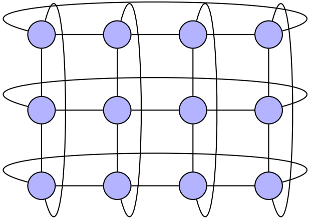
- **Case 3:** the optimal BFB schedule is not efficient at all.
  - Random topologies without any symmetry properties.
  - Throughput optimality in all cases—see follow-up work **ForestColl** (arXiv:2402.06787).



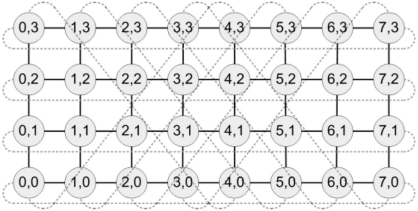
BFB Efficient Topologies

Throughput-optimal topologies with BFB:

- **Torus with arbitrary dimensions**
  - Cartesian product of rings, which have globally optimal BFB schedules.
  - Previous schedules are only efficient on torus with **equal dimensions** (e.g., $n \times n$, $n \times n \times n$)
- **Twisted Torus used by Google TPU v4**
  - Computationally verified for at least $N \leq 10^4$.



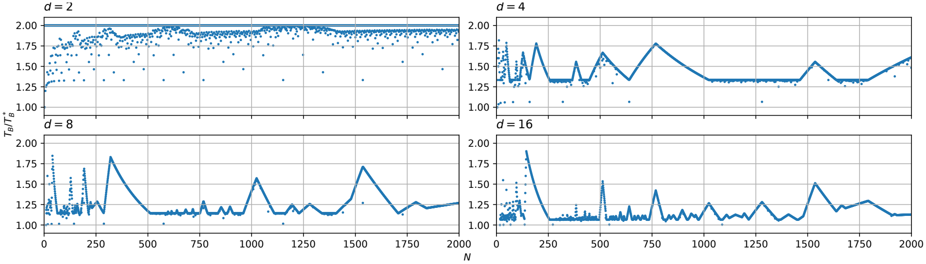
(a) 3x4 2D Torus



(b) 4x2 Twisted Torus

BFB Efficient Topologies

- **Circulant Graph:** throughput-optimal with BFB.
  - Can be constructed for any N and even-value d.
  - Significant improvement over ring in latency if throughput optimality is required.
    - $d = 4$: total-hop latency $\approx \frac{\sqrt{2N}}{2}$ instead of $N - 1$.
- **Generalized Kautz Graph:** diameter is at most one hop away from Moore Bound.
  - Can be constructed for any N and d.
  - Close to throughput optimality:



BFB vs Existing Schedule Generations

- BFB schedule generation is orders of magnitude faster than previous methods.
- BFB schedule is always theoretically optimal on hypercube and 2D torus.

# of nodes	4	8	16	32	64	**1024**
SCCL	0.59s	0.86s	21.4s	$>10^4$s	$>10^4$s	$>10^4$s
TACCL	0.50s	7.39s	1801s	1802s	n/a	n/a
BFB	**<0.01s**	**<0.01s**	**<0.01s**	**0.03s**	**0.17s**	**52.7s**

Table: Generation Time on Hypercube

# of nodes	4	9	16	25	36	**2500**
SCCL	0.61s	1.00s	60s	3286s	$>10^4$s	$>10^4$s
TACCL	0.45s	67.8s	1801s	1802s	n/a	n/a
BFB	**<0.01s**	**<0.01s**	**<0.01s**	**0.01s**	**0.03s**	**61.1s**

Table: Generation Time on 2D Torus ($n \times n$)

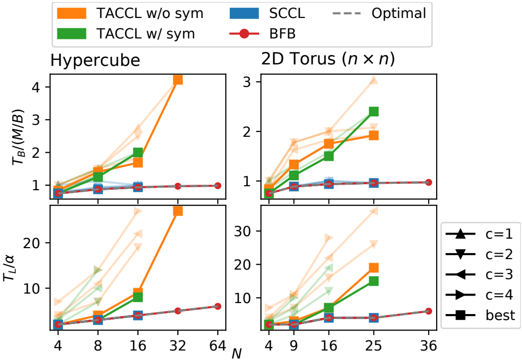


Figure: Theoretical Performance of Schedules

BFB vs Existing Schedule Generations

- Unlike previous methods, BFB does not require parameter sweeps.
- Previous methods require specifying # of chunks for data dependency tracking and heuristic parameters to speedup.

N	SCCL				TACCL w/o Symmetry				TACCL w/ Symmetry				BFB
	$c=1$	$c=2$	$c=3$	$c=4$	$c=1$	$c=2$	$c=3$	$c=4$	$c=1$	$c=2$	$c=3$	$c=4$	
Hypercube													
4	0.59	0.64	0.68	0.72	0.89	0.50	0.83	0.75	0.62	0.51	0.71	0.60	<0.01
8	0.86	1.22	1.86	2.48	96.9	807	63.2	1800	7.97	645	7.39	1801	<0.01
16	21.4	48.4	130	573	1801	1801	1801	1802	1801	n/a	n/a	n/a	<0.01
32	$>10^4$	$>10^4$	$>10^4$	$>10^4$	1802	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.03
64	$>10^4$	$>10^4$	$>10^4$	$>10^4$	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.17
1024	$>10^4$	$>10^4$	$>10^4$	$>10^4$	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	52.7
2D Torus ($n \times n$)													
4	0.61	0.63	0.67	0.76	0.68	0.50	0.82	0.72	0.45	0.51	0.76	0.64	<0.01
9	1.00	1.51	2.22	3.44	1801	189	67.8	262	88.6	71.1	67.8	105	<0.01
16	17.5	60	131	603	1801	1801	1801	1802	1801	1801	1801	n/a	<0.01
25	3286	5641	$>10^4$	$>10^4$	1802	1802	1803	n/a	1802	n/a	n/a	n/a	0.01
36	$>10^4$	$>10^4$	$>10^4$	$>10^4$	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.03
2500	$>10^4$	$>10^4$	$>10^4$	$>10^4$	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	61.1

Table of Contents

Direct-Connect Optical Testbed

(a) A100 Servers

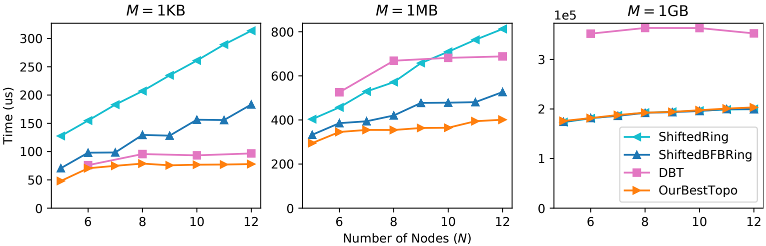
(b) Optical Patch Panel

- 12 servers, each with an NVIDIA A100 GPU.
- 100 Gbps HP NIC, configured as 4x25Gbps breakout interfaces.
- Topology is reconfigurable via a *Telescent* optical patch panel.

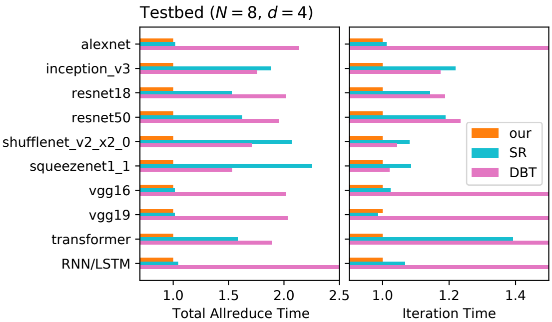
Allreduce Evaluation

- Generate our best bidirectional topologies for $N = 5$ to 12.
- Compare allreduce performance with shifted rings and double binary trees at data sizes 1KB, 1MB, and 1GB.
- **Result:** our topologies consistently outperform baselines across all topology sizes N and allreduce data sizes M.

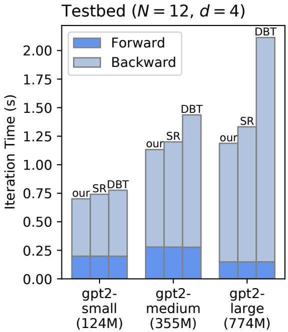
N	**Topology**	T_L
5	Complete Graph: K_5	2α
6	Degree Expansion of Complete graph: $K_3 * 2$	4α
7	Circulant Graph: $C(7, \{2, 3\})$	4α
8	Complete Bipartite Graph: $K_{4,4}$	4α
9	Hamming Graph: $H(2, 3)$	4α
10	Degree Expansion of BFB augmented Bidirectional Ring: $\text{BiRing}(2, 5) * 2$	4α
11	Circulant Graph: $C(11, \{2, 3\})$	4α
12	Circulant Graph: $C(12, \{2, 3\})$	4α



Data-Parallel DNN Training Evaluation



(a) 8-node Small Model Training.

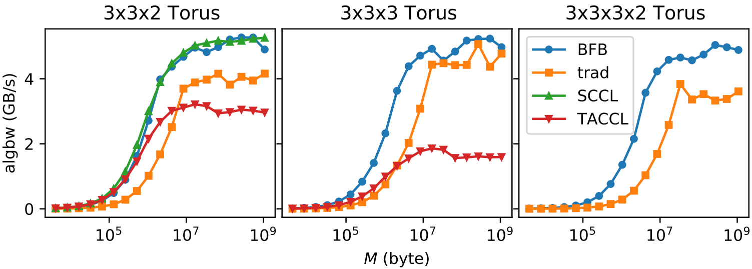


(b) 12-node GPT-2 Training.

Supercomputing Evaluation

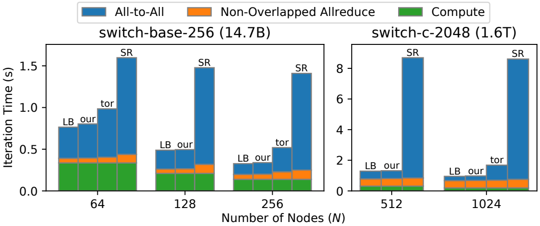
Frontera Supercomputer at the Texas Advanced Computing Center (TACC)

- Intel Xeon CPU nodes in a torus topology with 25 Gbps per link.
- **Result:** BFB torus schedules outperform all other schedules and remain efficient for tori with unequal dimensions.

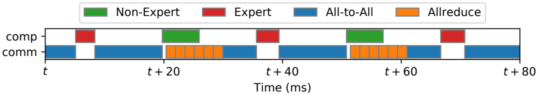


Simulated Expert-Parallel Training

- Expert-parallel training involves both **allreduce** and **all-to-all** communications.
  - While allreduce can be overlapped, all-to-all remains on the critical path.
- At 1024-node training of 1.6T MoE model, our topology outperforms torus by 40%+.
  - Torus spends 58% of the time on all-to-all, while our topology only spends 30%.
- Our topologies remain within 5% of the theoretical lower bound all the time.



(a) Simulated Training of Switch Transformers.



(b) Training Timeline.

Conclusion

- In this work, we introduce
  - **Expansion techniques** to expand small-scale optimized topologies and schedules into large-scale ones.
  - **Breadth-First-Broadcast** method to generate efficient communication schedules for large-scale topologies in polynomial time.

  Together, we enable efficient collective communications with direct-connect topologies.
- In evaluation, we demonstrate significant improvements over existing direct-connect topologies in collective communications and ML training performance.

Efficient Direct-Connect Topologies for Collective Communications
arXiv: https://arxiv.org/abs/2202.03356
To be presented at NSDI '25