SiP-ML: High-Bandwidth Optical Network Interconnects for Machine Learning Training

Mehrdad Khani[1], Manya Ghobadi[1], Mohammad Alizadeh[1], Ziyi Zhu[2], Madeleine Glick[2], Keren Bergman[2], Amin Vahdat[3], Benjamin Klenk[4], Eiman Ebrahimi[4]
[1]Massachusetts Institute of Technology [2]Columbia University [3]Google [4]NVIDIA

ABSTRACT

This paper proposes optical network interconnects as a key enabler for building high-bandwidth ML training clusters with strong scaling properties. Our design, called SiP-ML, accelerates the training time of popular DNN models using silicon photonics links capable of providing multiple terabits-per-second of bandwidth per GPU. SiP-ML partitions the training job across GPUs with hybrid data and model parallelism while ensuring the communication pattern can be supported efficiently on the network interconnect. We develop task partitioning and device placement methods that take the degree and reconfiguration latency of optical interconnects into account. Simulations using real DNN models show that, compared to the state-of-the-art electrical networks, our approach improves training time by 1.3–9.1×.

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network design and planning algorithms**;

KEYWORDS

Optical networks, Distributed Machine Learning, Silicon photonics, Reconfigurable networks

ACM Reference Format:
Mehrdad Khani, Manya Ghobadi, Mohammad Alizadeh, Ziyi Zhu, Madeleine Glick, Keren Bergman, Amin Vahdat, Benjamin Klenk, Eiman Ebrahimi. 2021. SiP-ML: High-Bandwidth Optical Network Interconnects for Machine Learning Training. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3452296.3472900

1 INTRODUCTION

The ever-growing demand for more accurate machine learning (ML) models has resulted in a steady increase in the dataset and model sizes of deep neural networks (DNNs). Since 2012, the amount of compute used in the largest AI training jobs has been increasing exponentially with a 3.4-month doubling time [1], 50× faster than the pace of Moore's Law.

The computation requirements of large ML models has been partly met by the rapid development of ML hardware accelerators and specialized software stacks. Although hardware accelerators have provided a significant amount of speed-up, today's training tasks can still take days and even weeks [2–4]. Solutions such as NVIDIA DGX [5] enable distributed training on a small number of GPUs (e.g., 8–16) connected with a high-speed electrical switch with Tbps bandwidth, but large-scale ML clusters must resort to connecting GPU servers over much slower infiniband fabrics [6, 7]. We argue that future distributed ML training workloads are likely to require several Tbps of bandwidth per device at large scales, creating a pressing need for entirely new ways to build interconnects for distributed ML systems.

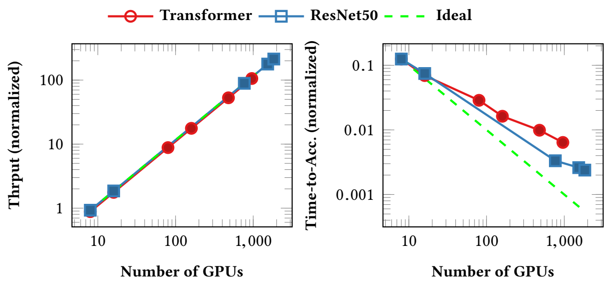
With Silicon Photonic (SiP) technology [8–18], it is now possible to build I/O interfaces integrated with an electronic chip with Tbps bandwidth [8, 19]. These optical I/O chiplets can be directly integrated into a CPU/GPU/FPGA/ASIC package [20], providing significantly higher bandwidth density than today's technologies.

This paper proposes an end-to-end optical solution, called SiP-ML, for strong scaling of ML workloads by leveraging SiP chiplets. SiP-ML exploits the predictability of ML training traffic patterns to find a parallelization strategy that meets the limitations of the optical topology at hand. Specifically, we explore two all-optical architectures: (*i*) SiP-OCS, an Optical Circuit Switch (OCS) design based on commercially available switches; and (*ii*) SiP-Ring, a switchless ring design enabled by reconfigurable Micro-ring resonators (MRRs) [21] embedded in SiP interfaces [22, 23]. Each of these architectures inherits one of the constraints of optical circuit-switched interconnects to an extreme. Optical Circuit Switches are too slow to reconfigure (e.g., 10 ms [24–26]) for ML models with a few milliseconds of iteration time, while the ring topology can only support communication between nearby GPUs. We show that SiP-ML's parallelization algorithm can produce traffic patterns suited to both these constraints by taking the *degree limitation* of all-optical circuit-switched interconnects as an input parameter.

To evaluate SiP-ML, we develop a detailed simulator for distributed neural network training. Our simulation results show the following: (1) for representative Natural Language Processing and Computer Vision DNN models, SiP-ML speeds up the total training time by a factor of 1.3–9.1× compared to today's electrical network fabrics; (2) although SiP-Ring's switchless design constrains the physical topology to a ring, it performs similarly to SiP-OCS because of the fast reconfigurability offered by the MRRs; (3) a SiP-ML interconnect with per-GPU bandwidth B performs as well as or better than an ideal, full-bisection electrical switch with per-GPU bandwidth $B/2$; (4) when per-GPU bandwidth is high (e.g., order of Terabits-per-second), hybrid parallelism strategies outperform data parallelism by up to 2× in terms of time-to-accuracy.

This work does not raise any ethical issues.

SIGCOMM '21, August 23–27, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8383-7/21/08.
https://doi.org/10.1145/3452296.3472900



(a) Throughput (b) Time-to-Accuracy

Figure 1: Weak scaling in today's training systems.

2 BACKGROUND AND MOTIVATION

This section describes the key concepts of designing scalable ML training interconnects. First, we discuss various parallelization strategies for distributed training (§2.1). Then, we describe weak and strong scaling and identify their network bandwidth requirements (§2.2). Finally, we introduce Silicon Photonics as a promising technology to build high-bandwidth ML training interconnects (§2.3).

2.1 Parallelization Strategies

Data Parallelism (DP). A popular parallelization strategy is data parallelism where a *batch* of training data is distributed across multiple workers. Each worker has *an identical copy of the DNN model* but trains on a subset of the training batch, called a *local batch*, in parallel. In DP training, workers need to communicate their model weight updates after each iteration. This step can be performed using various techniques such as broadcasting [27], parameter servers [28], ring-allreduce [29–31], and tree-reduce [32].
Model Parallelism (MP). In this approach, the *DNN model* is partitioned across different workers [33, 34]. The batch is copied to all MP workers, and different parts of the DNN model are computed on different workers, resulting in faster iteration times. Model parallelism is an active area of research, with various proposals for model partitioning [35–38]. Recent work has shown significant gains can be obtained with model parallelism; however, the degree of model parallelism has been limited to a few tens of workers [39–42].
Hybrid Parallelism. We consider a hybrid of the above parallelization strategies. Our proposed interconnects and task partitioning algorithms are designed specifically to support a hybrid of DP and MP.Further, we do not make any assumptions about a specific communication pattern, such as ring-allreduce or all-to-all. Our goal is to support a variety of communication patterns using smart task partitioning and GPU placement algorithms (details in §3).

2.2 Weak and Strong Scaling of ML Jobs

To identify the bandwidth requirements of ML systems, we first describe two fundamental scaling paradigms.
Approach 1: Weak Scaling. The first approach is to scale the throughput of data processing (number of processed data samples/sec) as the number of workers increases. The principal technique for throughput scaling is to keep the local batch size per worker fixed and grow the global batch size as more workers are added to the training job [43]. As a result, the entire system is able to process a larger global batch while *keeping the iteration time of each worker the same.* It is widely thought that training with large batches reduces the time-to-accuracy because large batches can produce better model updates, allowing the training to converge with *fewer total iterations* [44, 45]. However, increasing the global batch size in DNN training does not always translate to improving the number of iterations for all models [46, 47]. As an example, Fig. 1 compares the throughput and time-to-accuracy of two DNN models: Transformer [48] and ResNet-50 [49]. The numbers are obtained from Nvidia's benchmark results [50]. As shown in Fig. 1a, increasing the number of GPUs increases the batch size and thus improves the throughput (images/sec) of both models. However, the time-to-accuracy does not scale at the same rate and starts to plateau at large scales, as shown in Fig. 1b. As we show in our evaluations, reducing the time-to-accuracy at 1000-GPU scale requires significantly higher bandwidth than today's clusters (§4).
Approach 2: Strong Scaling. Instead of reducing the *number of iterations*, a more effective scaling approach is to reduce the *iteration time* as the number of workers increases. This approach is called strong scaling [43]. In contrast to weak scaling where the system operates on a larger global batch size as the system scales, strong scaling parallelizes the computation for a fixed batch size either by reducing the local batch size per worker or by partitioning the computation task across workers. However, achieving strong scaling is challenging, because reducing the iteration time leads to more frequent model updates and, hence, requires the I/O bandwidth to scale with the number of workers [47]. Furthermore, since each worker must perform small granular computations, strong scaling can be sensitive to network latency and small inefficiencies in the compute/network software stack.
Bandwidth Requirements of Weak and Strong Scaling. Today, the technique most commonly used to scale a distributed training job is weak scaling using the DP strategy. This approach is popular because as more workers are added to the job: (*i*) the computation time of each worker remains constant (since the local batch is constant); and (*ii*) the size of data transfers at each iteration remains constant (because it depends on the DNN model).[1] In contrast, in strong scaling approaches, the bandwidth requirement increases (often super linearly) as the system is scaled, since (*i*) strong scaling leads to reduced computation time per worker and shorter training iterations, and (*ii*) the amount of data exchanged at each iteration stays the same or even grows with scale.[2] In today's systems, the degree of MP is limited to 8 or 16 workers within one DGX box [51] with Tbps communication bandwidth per GPU [42, 52–54].

2.3 Silicon Photonics for ML Training

A straightforward approach to meet the high-bandwidth requirement of large-scale training workloads is to augment the bandwidth of existing electrical switches. However, recent trends in SERDES/-packet switching technology suggest that we will hit a wall in

[1] The amount of data transferred in DP in each iteration depends on the all-reduce algorithm. With a ring-reduce implementation, *each worker* exchanges 2×M, where M is the DNN model size. Note that as the number of workers increase, the bandwidth *per worker* remains constant but the total required bandwidth grows.
[2]The amount of data transferred in MP in each iteration depends on the model partitioning strategy but often increases significantly with scale, particularly when a kernel is split on anything other than the batch dimension.

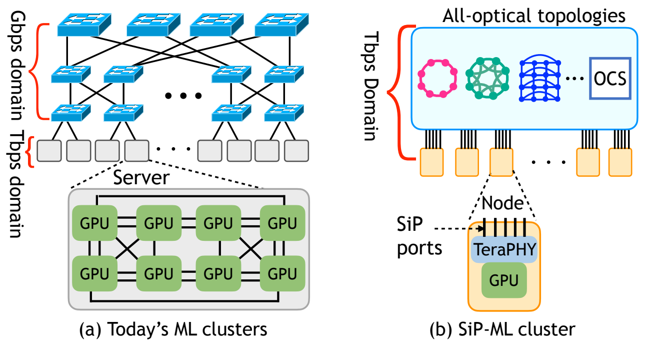


Figure 2: Comparing today's ML cluster with SiP-ML.

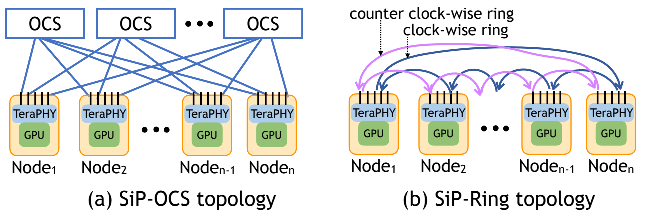


Figure 3: Two topologies we consider for SiP-ML.

capacity with standard electrical packet switching [55–58]. For instance, realizing an electrical packet switch with 100 ports each with 10 Tbps is extremely challenging. This is because the traffic manager ASIC in the switch needs to process packets at 1000 Tbps speed, but today's ASICs can only process packets at 12.8 Tbps speed. To get to 1000 Tbps switching, we need to build a "Clos" of switching ASICs inside each electrical switch [59]. This is a challenging undertaking.

At the same time, substantial progress is being made with Silicon Photonics chiplets to bring optical interconnects very close (essentially on die) to the ASICs. Recent advances in SiP fabrication processes have created an opportunity to build chiplets with optical I/O ports that can transmit data at far higher rates than electrical conductors [9–17, 60–62]. With SiP interfaces, however, it is possible to build I/O interfaces integrated with electronics at 10 Tbps/mm bandwidth (BW) density [14, 19, 20, 60, 63, 64]. Such integration enables building next-generation computer architectures that are fundamentally impossible with today's technologies.

In this paper, we propose all-optical interconnects as an attractive solution to build the next generation of ML systems. We argue that ML workloads present a unique opportunity to build specialized circuit-based interconnects. While conventional datacenter workloads have unpredictable behavior, with short flows dominating the traffic, ML workloads are predictable, periodic, and consist of mostly large transfers. Importantly, the parallelization algorithm determines the circuit schedules, and the entire training repeats the same communication pattern at every iteration. This unique characteristic simplifies the control-plane logic with which datacenter optical designs have grappled for years.

3 SIP-ML DESIGN

In this section, we introduce *degree* and *reconfiguration latency* as fundamental factors affecting all optical circuit-based interconnects (§3.1). We then discuss our parallelization algorithm, explaining how it takes these factors into account to produce a suitable parallelization strategy for a given topology (§3.2). Finally, we discuss SiP-ML's control plane and wavelength allocation (§3.3).

3.1 Degree and Reconfiguration Latency

Fig. 2 illustrates the differences between today's ML training clusters and SiP-ML. The state-of-the-art clusters have two bandwidth domains: (*i*) a Gbps bandwidth domain that interconnects thousands of servers using conventional network fabrics and off-the-shelf NICs; (*ii*) an all-to-all Tbps bandwidth domain that tightly connects a handful of GPUs inside a server or a DGX. In contrast, a SiP-ML cluster consists of disaggregated GPUs, each equipped with Tbps SiP interfaces, interconnected by an all-optical network. An example of a SiP interface is the TeraPHY optical I/O technology developed by Ayar Labs [64], capable of carrying 2 Tbps bandwidth (80 wavelengths each carrying 25 Gbps [65]). A GPU can be equipped with several of these interfaces. To put the choice of topology into perspective, we first introduce two fundamental factors affecting all optical circuit-switched interconnects.

Degree. Unlike packet-switched networks, optical interconnects are circuit-based. Hence, at any point in time, each node has a limited number of active circuits, thereby limiting the number of nodes it can communicate with directly. We refer to this as the node degree. A topology with degree $\mathcal{D}$ means each node can simultaneously maintain, at most, $\mathcal{D}$ circuits. Depending on the traffic pattern, these circuits can be established with one to $\mathcal{D}$ other nodes. Topologies with higher degree are suited for traffic patterns with high fan-out, but they also tend to have a larger cabling footprint.

Reconfiguration Latency. The reconfiguration latency puts a lower bound on how long the circuits should be kept to achieve a high duty cycle [66]. For a topology with reconfiguration latency r, the circuit hold time should be longer than, for instance, $10\times r$ to achieve a 90% duty cycle.

There are various optical topologies that realize SiP-ML's vision. At one end of the spectrum are switch-based interconnects, such as MEMS-based Optical Circuit Switch interconnects [24, 25, 55, 67, 68] and Rotor-based interconnects [66, 69]. On the other end lie switch-free topologies such as ring [26, 70, 71], circulant graphs [72], torus [73, 74], hypercube [75] and dragonfly interconnects [76–78].

In this paper, we consider two topologies at opposite ends of the spectrum, as shown in Fig. 3. SiP-OCS is the first natural topology choice because OCSs are commercially available today [79]. However, their reconfiguration latency is ≈10 ms, making them suitable for circuits that last through the entire training. Fig. 3a illustrates our SiP-OCS topology. SiP-OCS consists of Q optical switches, each with N ports (the same as the number of GPUs), where each GPU is connected to every OCS in a flat topology. Hence, in SiP-OCS, the degree $\mathcal{D}$ is equal to the number of switches (Q).

As an alternate, extreme design point, we also investigate the possibility of removing the switching elements entirely and evaluate the performance of a *minimalistic, switch-free topology* called SiP-Ring. In contrast to SiP-OCS, SiP-Ring reconfigures *wavelengths* *within each port* to achieve logically rich topologies. Reconfiguration is done using Micro-ring resonators (MRRs) [21] embedded in SiP ports [22, 23]. MRRs act as spectral filters to select and forward wavelengths, and they enable the reuse of wavelengths across non-overlapping segments of the ring (Fig. 13a in the appendix illustrates an example). Our experiments show MRRs can switch between different wavelengths within 25μs (§4.4). We discuss the SiP-Ring design in more detail in Appendix A.1.

3.2 Degree-Aware Parallelization Strategy

A DNN can be viewed as a directed acyclic graph (DAG) of operations (ops). To parallelize a DNN training job, we need to decide which GPU is responsible for running each op (or a part of each op). As a simple example, to train a model with global batch size b using DP on N GPUs, we break each op into N parallel *sub-ops*, each operating on a local batch of size b/N (this is referred to as splitting on the sample dimension [38]), and we map one sub-op to each GPU. In general, MP follows similar steps: first partition each op into parallel ops, then place the sub-ops. However, the partitioning and placement decisions are not as straightforward as in DP.

Our parallelization algorithm takes the following as input: (i) a DNN computation graph, $G_{in} = (V, E)$, where V is the set of operations (nodes) and E is the set of data dependencies (edges) between the operations; (ii) the global batch size denoted by b; (iii) a parameter k denoting the number of GPUs to partition the model using MP; (iv) a parameter l denoting the number of GPUs to partition the data using DP; and (v) the physical degree constraint of the optical network topology, denoted by $\mathcal{D}$. Our algorithm finds a hybrid MP-DP strategy with k-way model parallelism and l-way data parallelism for $N = k \times l$ GPUs, such that the training iteration time is minimized while satisfying the degree constraint (i.e., each GPU communicates with no more than $\mathcal{D}$ other GPUs). We assume all GPUs are identical.

The core of the algorithm determines an MP placement of the DNN computation on k GPUs. Specifically, we begin by splitting the GPUs into l groups, with k GPUs per group, and we divide the global batch equally between the groups (i.e., each group is responsible for a local batch of training data of size b/l). Then, we compute an MP placement across k devices. We replicate the same placement in each group to produce the final hybrid MP-DP strategy. Fig. 4 illustrates the key steps in our parallelization algorithm across 8 GPUs, with k = 4-way MP, l = 2-way DP, and degree constraint $\mathcal{D}$=3. We use this as a running example in the remainder of this section.

(i) Partitioning. DNN training involves sequential stages of computation, as dictated by the data dependencies in the computation graph. For example, the graph in Fig. 4(a) has 4 sequential ops, shown as rectangles of different colors. The size of each rectangle represents the computation time of the op. The key to minimizing training time is to balance the computation load across devices at *every* stage of computation to maximize parallelism. Note that balancing per-stage computation is not the same as balancing the total load on each device. Sequentially-dependent ops cannot run in parallel, hence placing them on the same device has no impact on run-time compared to placing them on different devices, even though it increases the total load on the device.

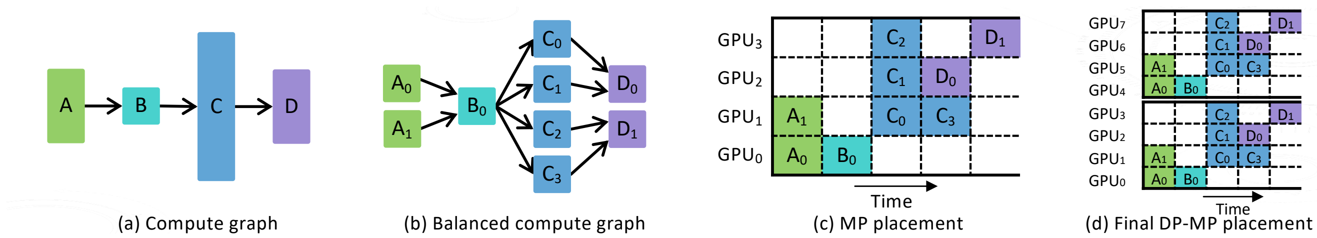
To minimize per-op run-time, it is desirable to split ops into smaller pieces of computation. There are many ways to split an op; for example, a 2D convolution can be split across height, width, and channel dimensions [38]. However, in splitting ops, we must take care not to compromise GPU utilization. GPUs (and other ML accelerators) internally distribute an op over a massive number of cores. If we split an op too finely, it will not have enough *compute intensity* to utilize the cores effectively, and, therefore, we will achieve no reduction in run-time from splitting. As a result, we choose a minimum quantum of computation time, τ, and split ops to sub-ops of a size near τ. We also cap the maximum number of partitions for each op at k (the MP degree), as there is no point in splitting beyond the maximum number of available parallel workers. The result is a balanced computation graph whose vertices are the sub-ops, as shown in Fig. 4(b) for our running example.

The right choice of the split dimension depends on the type of the op and can impact the communication pattern between the sub-ops. For example, in the case of a 2D convolution on an image with multiple output channels, if we divide the op across the height and width dimensions of the input, none of the sub-ops needs to know the entire input image. However, if we split the op across the output channel dimension, every sub-op needs a copy of the input image, leading to a broadcast communication pattern with high overhead. We select the most efficient dimension for each op. Since we always split ops uniformly, sub-ops tend to communicate the same amount of data with their descendants (the edges between the sub-ops at each stage in Fig. 4(b) carry roughly the same amount of traffic).

(ii) Placement. Next, we assign a GPU device to each op in the balanced graph. Our placement aims to minimize the total run-time while respecting the communication degree constraint $\mathcal{D}$ required by the optical interconnect. Each GPU has two types of communications: (i) it must communicate with some of the GPUs in its MP group (depending on the op placement); (ii) given the hybrid DP-MP strategy, there are l MP groups that need to synchronize their parameters through DP. Hence, each GPU must communicate with its counterparts in the other l MP groups to perform an all-reduce operation to synchronize the model parameters across the DP partitions. We use the ring-allreduce [29, 30] algorithm for this step. This requires a ring communication pattern between corresponding GPUs in the MP groups, which requires each GPU to send data to one GPU in another group. Therefore a GPU can communicate with, at most, $\Delta = \mathcal{D} - 1$ other GPUs within its own MP group to meet the overall degree constraint.

We now present a heuristic algorithm for placing ops within an MP group to minimize run-time with a constraint Δ on the degree of communication. While this problem can be written as an Integer Linear Problem (ILP), it is prohibitive to solve this ILP given the scale of the balanced computation graph (e.g., over 20K sub-ops for the Transformer DNN model). Algorithm 1 provides the pseudocode.

The key strategy in our algorithm is to map GPU devices into a metric space and transform the degree constraint into a distance constraint in that space. We select an arbitrary ordering of GPU devices and place ops to maintain a maximum communication distance of Δ; i.e., devices i and j are allowed to communicate only if $(i-j) \mod k \leq \Delta$. This constraint leads to a sparse diagonal traffic



(a) Compute graph (b) Balanced compute graph (c) MP placement (d) Final DP-MP placement

Figure 4: An example of SiP-ML's parallelization strategy with $k = 4$, $l = 2$, $\mathcal{D}$=3, and $\Delta = 2$.

Algorithm 1
Task Placement with a Communication Degree Constraint

```
Input: Balanced compute graph g_in, computation quantum $\tau$, degree
  constraint $\Delta$, local_batchsize $b/l$, mp_degree $k$
Output: A task graph g_out with placed ops
for op in g_in.topological_sort( ) do
    for sub_op in par_ops_map[op] do
        far_id←farthest sub_op's predecessor device id
        near_id←nearest sub_op's predecessor device id
        range_lo←near_id
        range_hi←far_id + $\Delta$
        sub_op.device←get_earliest_avail(avail_times,
  range_lo, range_hi, sub_op.mem_size)
        cand_start←latest end time of predecessors
        start←max(cand_start, avail_time[sub_op.device])
        end←start + sub_op.duration
        avail_time[sub_op.device]←end
    end for
end for
g_out ← add_network_ops(g_out)
```

matrix with zeros outside a Δ distance from the main diagonal, satisfying the communication degree constraint.

The algorithm begins with a topological sort of the balanced computation graph (shown in Fig. 4(b) for our example), such that each sub-op appears in the sorted list after its dependencies. It places the sub-ops in this sorted order, guaranteeing that when a sub-op is placed, all of its dependencies have already been placed. For each sub-op, the algorithm first computes a set of *placement candidates*. These are the devices where the sub-op can be placed without violating the distance constraint mentioned above. We compute the intersection of these ranges for all parents of x to determine its placement candidates. Then, we select the earliest available device among these candidates to place x, and we schedule the op on that device as soon as its dependencies have completed. If there is a tie at this step, we select the GPU with the smallest index so that we can minimize the distance between communicating GPUs.[3] Notice that since we place the sub-ops in order of their dependencies, keeping track of when each op can be scheduled on each device is straightforward. If the intersection of the feasible ranges for all parents of the sub-op x is empty, i.e., the maximum distance between the parents is longer than $\Delta - 1$, we relocate the parent nodes into a smaller device range so that the placement of x becomes feasible. For this purpose, we place x on the GPU that meets a maximal set of range constraints. We then reallocate the remaining parents that violate the constraint into the nearest device that meets the distance constraint with x. As this may create distance violations between parents and grandparents of x, we continue this backward process until all previously placed ops meet the distance constraint with their parents. We then restart a forward pass from the first located op and verify the distance constraints between the placed ops and their children. If any violations have occurred due to reallocation, we relocate the child op. This forward-backward procedure is repeated until all ops are placed. We leave the convergence proof to future work.

Fig. 4(c) shows the MP placement for our running example, with $\Delta = 2$. Notice two properties of this placement: (i) each GPU communicates with, at most, $\Delta = 2$ other GPUs, as required, and (ii) the sub-ops of each op are balanced well across the 4 GPUs. In fact, the only op that is not perfectly balanced is C, but the 4 sub-ops of this op cannot be placed on all 4 GPUs without violating the communication degree constraint, because whichever GPU op B resides on would then need to communicate with the other 3 GPUs.

Putting it all together. Fig. 4(d) shows the final hybrid MP-DP placement for our example. As mentioned earlier, it is created simply by replicating the MP placement in the $l = 2$ GPU groups. As for the communication pattern, each GPU communicates with, at most, $\Delta = 2$ other GPUs in its MP group and one more GPU for the ring topology required for the DP all-reduce step. For example, in Fig. 4(d), GPU 1 must communicate with GPUs 2 and 3 for MP and GPU 5 for DP. Our parallelization algorithm takes the degree of MP and DP (k and l) as input, but it is trivial to optimize over these parameters to find the combination that minimizes training time for a given number of GPUs, as discussed in Appendix 4.2.

3.3 Circuit Scheduling

Given that our SiP-OCS topology reconfigures its circuits only once at the beginning of the training job, its control plane logic is simple. In this case, the main task is to compute the total traffic matrix resulting from the parallelization algorithm and then assign circuits to each pair of GPUs that must communicate, such that the maximum transfer time is minimized. We determine the circuit assignment with a simple ILP run once for each training job (details in §A.2).

The control plane for the SiP-Ring topology is more challenging, as circuits can be reconfigured during training. Hence, our controller needs to estimate the traffic and reschedule the circuits periodically. Therefore, every GPU's host needs to read its GPU transfer buffer counters through PCIe and communicate them to a

[3]This property helps enable wavelength reuse in the ring topology (§A.1).

central controller. Using NVIDIA's `nvml` API, we poll the NVLink counters on a Tesla V100 GPU at a 300-microsecond granularity. However, this API is designed for management purposes and is not optimized for latency. We believe obtaining the counters at a sub-100-microsecond scale should be feasible with further engineering. Our experiments confirm that the observed traffic matrix over the past 100μs is a good estimate of the communication demands over the next 100 μs. Using the traffic matrix, we can solve an ILP (see §A.1) for optimal wavelength scheduling on the ring topology. However, solving an ILP is too slow for short-timescale circuit scheduling. Therefore, we propose a fast, approximate wavelength scheduling algorithm that solves a minimum-cost flow routing problem to schedule wavelengths. Appendix A.1 describes this algorithm in detail. Note that while we currently propose to measure the traffic matrix for dynamic circuit establishment, exploiting the predictability of training workloads is a natural step which we leave for future work.

Supporting Multiple Jobs. We anticipate a SiP-ML cluster will typically be used to run multiple jobs at the same time. Each job will run on a subset of GPUs, dedicated to that job. Supporting multiple jobs with SiP-OCS requires no changes to our design, except that we allocate a subset of available GPUs when a job arrives and correspondingly set the total number of GPUs in our placement algorithm. When a job completes, we release its GPUs and optical circuits. SiP-Ring follows a similar logic, but we ideally prefer to allocate each job to a contiguous block of neighboring GPUs on the ring. Fragmentation of the ring space, as jobs arrive and depart, could make this difficult to achieve at all times. One solution is to use a standard OCS to assign GPU interfaces to arbitrary locations on the ring.

Scalability Considerations. While our current version of SiP-OCS assumes each OCS has enough ports to connect to every GPU in a flat topology, a more realistic setting is to use hierarchical Clos [80] or flat designs such as BCube [81] to scale SiP-OCS. Our SiP-Ring topology can be scaled using Theia [72] and SlimFly [82] to build hierarchical rings. Another way to scale SiP-Ring is to consider 2D rings, where we have K horizontal rings, with N GPUs on each ring. We then connect every K GPUs from K different horizontal rings on a single vertical ring. Hence, there will be $K + N$ rings in total, connecting NK GPUs. Each GPU has direct access to one vertical and one horizontal ring and must divide its SiP interfaces between the two. Depending on the vertical bandwidth requirement of the interconnect, this ratio can be adjusted.

4 EVALUATION

In this section, we quantify the performance of SiP-ML by comparing it to other network interconnects. Our results show:

(*i*) For three representative DNN models (Transformer, ResNet, and Megatron), SiP-ML speeds up training time by a factor of 1.3–9.1× compared to hierarchical electrical network fabrics representative of today's ML clusters. This is because SiP-ML eliminates bandwidth bottlenecks and enables hybrid DP/MP parallelization strategies that cannot be supported efficiently by today's fabrics.

(*ii*) Although SiP-Ring's switchless design constrains connectivity, it performs similarly to SiP-OCS. SiP-Ring's limited connectivity is compensated by its ability to rapidly reschedule wavelengths using MRRs and our parallelization algorithm's ability to adapt its strategy to the topology (e.g., ensuring most communication occurs between nearby nodes on the ring).

(*iii*) A SiP-ML interconnect with per-GPU bandwidth B performs as well as or better than an ideal, full-bisection electrical switch with per-GPU bandwidth $B/2$. For instance, given 1024 GPUs and B = 8 Tbps, SiP-ML's dynamic topology provides at least 4 Tbps of bandwidth, on average, between each pair of GPUs that need to communicate.

(*iv*) When per-GPU bandwidth is high (e.g., order of terabits-per-second), hybrid parallelism strategies outperform data parallelism by up to 2× in terms of time-to-accuracy.

4.1 Methodology & Setup

To evaluate SiP-ML, we implement a detailed simulator, called Rostam, to model several baseline network architectures connecting up to thousands of GPUs. Our simulator is ≈10K lines of code in C++ and is available online at https://github.com/MLNetwork/rostam.git. We discuss the details of our simulator in §4.2. In our evaluations, we set the quantum of computation for balancing the computation graphs, τ, to 10 μs (§3.2).

Comparisons. We consider the following network architectures:

- **Elect-Flat:** an ideal electrical switch that scales to any number of GPUs, N, for any per-GPU bandwidth of B; i.e., each GPU can simultaneously communicate with $N - 1$ other GPUs with a total bandwidth of B in both send and receive directions. This baseline has zero reconfiguration delay. For any pair of (B, N), no network can communicate faster than this baseline. In practice, it can be approximated with full-bisection bandwidth topologies such as fat-tree for relatively small values of B (e.g., 100–400 Gbps), or with a small N (e.g., tens of nodes) with large B. Note that no electrical network would be able to perform better than this flat electrical baseline, as it provides full-bisection bandwidth.
- **Elect-Cluster:** a hierarchical electric network fabric representative of today's ML clusters interconnecting GPUs. Each server hosts eight GPUs, connected with an internal high-speed electrical switch providing per-GPU bandwidth of B, typically in the order of terabits-per-second. The servers are connected with a slower electrical fabric providing 400 Gbps bandwidth per server (unless otherwise stated). In practice, servers can be thought of as DGX [5] boxes with an internal NVSwitch [83] interconnect, communicating over a standard datacenter network fabric (e.g., fat-tree).
- **SiP-Ring:** a ring-based interconnect for SiP-ML, as described in §3.1. Each GPU has W distinct wavelengths that it can dynamically allocate to communicate with its 16 closest neighbors on the ring (in both directions). We assume each wavelength carries 25 Gbps of bandwidth, providing a maximum bandwidth of $B = W \times 25$Gbps for each GPU. Unlike SiP-OCS, this topology is rapidly reconfigurable, with a reconfiguration latency of 25 μs (§4.4). We estimate the traffic every 100 μs as described in §3.3 unless stated otherwise.
- **SiP-OCS:** an optical circuit switch interconnect for SiP-ML, as described in §3.1 with Q OCS switches, each with N ports (the same as the number of the GPUs). Each GPU has Q optical links (each with a bandwidth of B/Q), one to each OCS. Each GPU can communicate with, at most, $\mathcal{D}$=Q other GPUs at the same time. To study the impact of $\mathcal{D}$, we vary the number of OCS switches in the

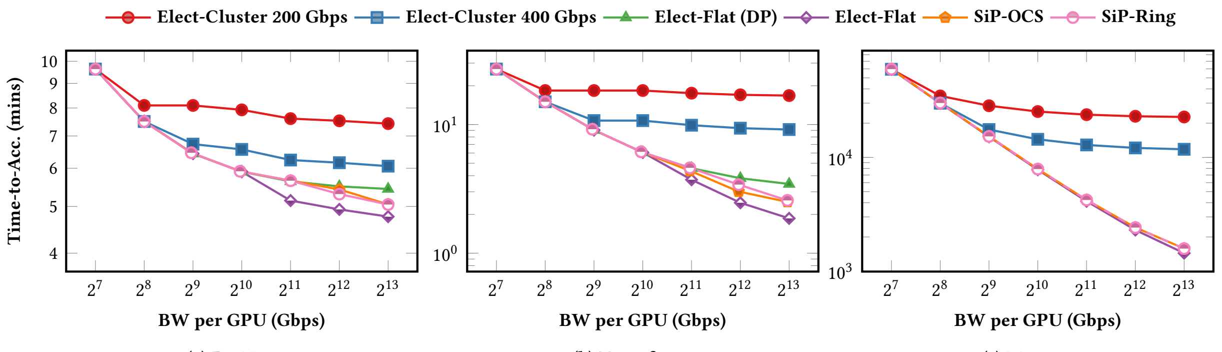


Figure 5: Impact of bandwidth B on the total training time (Time-to-Accuracy) for N=1024 GPUs. DP is not feasible for Megatron because of its huge memory footprint.

interconnect, using a default value of 16. Since OCS reconfiguration delay is too long compared to the typical training iteration time of our DNN models (< 20ms), we compute the best *one-shot* circuit schedule for each workload, as described in §3.3. To evaluate the potential benefits of optical switches with fast reconfiguration [55, 71], we also evaluate the impact of lowering the reconfiguration latency and allowing multiple reconfigurations within each training iteration.[4]

Training workloads. We consider ResNet, Transformer, and Megatron, three representative DNN models widely used in computer vision and natural language processing applications. ResNet [84] is an image classification model with 25 million parameters. Transformer refers to a Universal Transformer with 350 million parameters. Megatron[52] is a variant of the GPT model [85] with 18 billion parameters.

We focus on time-to-accuracy as our primary metric. We determine the time-to-accuracy by multiplying the time for a single training iteration (obtained via our simulator) by the number of training iterations required to reach the target accuracy. We use numbers reported in prior work for the required training iterations for these models at a given batch size. For ResNet and Transformer, Shallue et al. [86] report the number of training iterations across a range of batch sizes. Hence for these models, we optimize over batch size to find the lowest possible time-to-accuracy in each network configuration. For Megatron, we use batch size 1024 and 240,000 training iterations, following [50, 87]. Note that we report the total *pre-training* time for Megatron, which requires significantly more training iterations than a typical *fine-tuning* task. But the relative improvements we report would hold for fine-tuning the model since we are directly decreasing the iteration time.

ResNet and Transformer fit in a typical GPU's memory. Hence the main reason to parallelize them is to speed up training. Megatron, cannot fit on one GPU and therefore cannot be trained with only DP; MP is required to split it across multiple GPU memories.

4.2 Simulator

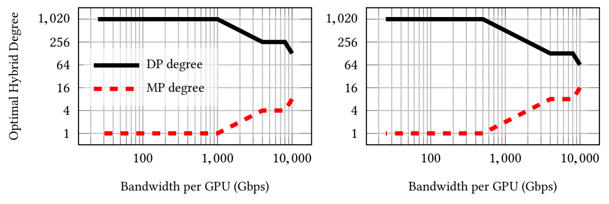
The overall flow of an end-to-end simulation in Rostam is as follows.

[4]In the extreme, eliminating reconfiguration latency entirely would make SiP-OCS equivalent to the ideal Elect-Flat architecture.

Profiling. We first need to profile the average GPU and CPU compute time, peak memory size, and input/output data sizes of each operation in the model in addition to its data dependencies. Each compute operation typically has one or more input/output arrays of data, "tensors". Profiling the operations over different input/output tensor shapes helps predict the speed ups of partitioning each operation in different input/output tensor dimensions. We start profiling over a fair range of batch sizes, typically starting with 1 sample/iteration and continuing until we run out of GPU memory. The profiling step is independent from the simulator and can use any convenient profiling tool. Moreover, profiling along other than the samples dimension (e.g., height and width in the 2D convolution) helps improve the simulation's accuracy. In absence of the profiling data in any dimension, we assume a linear dependency between the total number of splits and each split's compute time in that dimension. Depending on the dimension of the split, Rostam adds the required new data dependencies in the placement stage. In addition to the operations profile, we need to know the required number of iterations to achieve a certain level of model accuracy as a function of the global batch. This profile depends on the DNN model and the training dataset [46]. Rostam can combine the latter two profiles in the placement stage to come up with the best hybrid parallelization strategy. In this paper, we profile all models on an NVIDIA Tesla V100 GPU with 32 GB of memory.

Placement. Our approach to explore the space of hybrid parallelism techniques takes as input: (1) the number of GPUs, (2) the bandwidth available per GPU, (3) the graph profile for the DNN model as described above, and (4) the curve providing the required number of training iterations as a function of the (global) batch size. We search through all possible hybrid parallelizations over a range of global batch size configurations and use the placement algorithm (e.g., Algorithm 1 (§3)) for device placement. We then estimate each configuration's run-time based on the graph profile and the bottleneck bandwidth. To estimate the effect of the network, we also compute the latency for each data transfer (edge) in the graph profile according to the bottleneck bandwidth. We finally select the fastest of all these parallelization strategies.

Two points are worth noting about this procedure. First, one of the strategies that our task parallalization considers is the conventional DP. However, as our results show (see §4.3), in many cases, DP is not the best strategy for large-scale training. Second, the time



(a) ResNet50 **(b) Transformer**

Figure 6: Optimal hybrid trade-off between the degree of MP and DP at different per-node bandwidths for 1024 GPUs.

computed for a configuration in this procedure is only an estimate; in our actual simulations, a GPU's bandwidth can vary over time (e.g., due to circuit reconfiguration). Therefore, our simulator requires a runtime stage to track the effect of dynamic decisions on ops scheduling more precisely.

Runtime. Our runtime simulator relies on three main components: GPUs, an interconnect, and an executive session. The session launches the operations onto the GPUs as soon as their dependencies are met in the DNN graph. The interconnect can be electrical or optical. Our current implementation includes SiP-Ring, SiP-OCS, electrical, and full-mesh interconnects.

Rostam models a latency for each op launched onto the GPU and a minimum completion time for ops that run on the GPU. Hence, there is a lower-bound on how quickly we can run a compute graph that depends on its critical path length. We set the launch latency and the minimum completion to 1 microsecond in our experiments. Moreover, Rostam overlaps the communication and computation whenever possible.

4.3 Results

Fig. 5 compares the time-to-accuracy of our three DNN models with 1024 GPUs on different network architectures. We vary the bandwidth per GPU, B, between 128—8192 Gbps, and compare Elect-Flat, Elect-Cluster with two values of inter-server bandwidth (200 Gbps or 400 Gbps), SiP-OCS, and SiP-Ring. For each value of B and each network architecture, we use Algorithm 1 (§3.2) to search for the best parallelization strategy, as described in §4.2. To compare the different architectures on an equal footing, we run Algorithm 1 for electrical networks by removing the degree constraint. We then compare our results to the state-of-the-art results reported in MLPerf [88] and find that they are comparable or better (§A.3). For reference, we also show data parallel (DP) training on Elect-Flat (except for Megatron which cannot use basic DP).

We also experiment with FlexFlow [38] as a state-of-the-art placement algorithm. FlexFlow's network model does not support the degree constraints required by our optical interconnects. For electrical interconnects, we run the FlexFlow code [89] for our workloads, but the strategies it finds are very similar to DP. We believe there are two reasons for this. First, the scales we consider (e.g., 1000 GPUs) are much larger than those in FlexFlow, making the search space for its Metropolis algorithm significantly larger. Second, FlexFlow's implementation only searches for partitioning strategies across the batch dimension (although the approach in [38] is general).

Analysis. We first consider the Elect-Flat architecture. Recall that Elect-Flat has ideal performance. At every value of B, it provides each GPU with its full interface bandwidth regardless of the communication pattern. Thus Elect-Flat's training time serves as a lower bound for any other network. Fig. 5 shows that increasing B on Elect-Flat improves training time for all models, but the improvement is much larger for Transformer and Megatron than ResNet50. ResNet50 is less sensitive to network bandwidth for two reasons. First, it is a smaller model than the others and therefore requires less bandwidth for all-reduce operations. Second, ResNet50 trains effectively with large batch sizes (via weak scaling), further reducing its bandwidth requirements [86, 90–92].

Comparing DP with the best strategy found using Algorithm 1 on Elect-Flat is also instructive. Consider Transformer: when B is less than 1 Tbps, our placement cannot beat DP. But as B increases to 8 Tbps, SiP-ML's hybrid strategy outperforms DP by ≈50%.

Now let us turn to the Elect-Cluster architectures. For all three models, the training time plateaus as we increase B, with Elect-Cluster (400 Gbps) outperforming Elect-Cluster (200 Gbps). Recall that here, B is the local bandwidth between the GPUs within each server. The results show that scaling this local bandwidth can improve training time to an extent (by enabling some model parallelism), but the slow server-to-server network eventually becomes a bottleneck and prevents further speedups.

Compared to Elect-Cluster architectures, SiP-OCS and SiP-Ring achieve 1.3−9.1× faster training time as we scale B. The benefits are smallest for ResNet50 (which does not require very high communication bandwidth) and most significant for Megatron. SiP-ML architectures are less efficient than the ideal Elect-Flat (which cannot be realized in practice for large values of B and N): to achieve the same training time, SiP-ML architectures require up to 2× higher bandwidth per GPU (B) (e.g., Transformer), with a smaller gap in many cases (e.g., Megatron). This difference reflects the constraints imposed by optical circuit switching. Specifically, in our evaluations, we set the degree constraint for both SiP-OCS and SiP-Ring at $\mathcal{D}$=16. SiP-OCS requires a one-shot reconfiguration, while SiP-Ring imposes a traffic locality requirement on the communication pattern. Despite these constraints, SiP-ML performs quite well, as our placement algorithm adapts the parallelization strategy to suit the degree requirement.

SiP-OCS and SiP-Ring perform similarly overall. Each architecture has pluses and minuses. Unlike SiP-OCS, SiP-Ring has fast reconfiguration, but it makes communication between more distant GPUs on the ring less efficient. Our results show that the impacts of these factors on overall performance effectively cancel each other out.

Parallelization strategies. Fig. 6 plots the degrees of DP and MP for each value of B in SiP-OCS. The figure shows that as the per-node bandwidth increases on the x-axis, the optimal strategy uses more model parallelism to decrease the total training time. This is consistent with current practice: when the network is slow, DP is more efficient but on a fast network, combining MP and DP improves training time. For instance, the Transformer model shown in Fig. 6b starts with 1024-way DP and 1-way MP, but at 10 Tbps bandwidth per-GPU, the best training time is achieved with 16-way MP and 64-way DP.

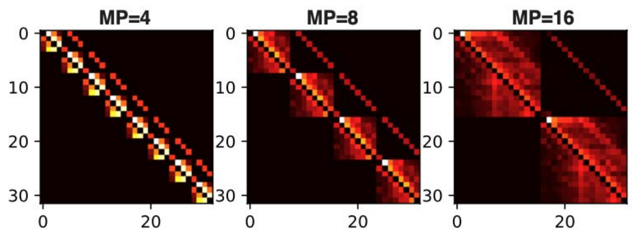


Figure 7: Traffic matrices generated by SiP-ML for the Transformer model on 1024 GPUs (displaying only the first 32 GPUs for brevity).

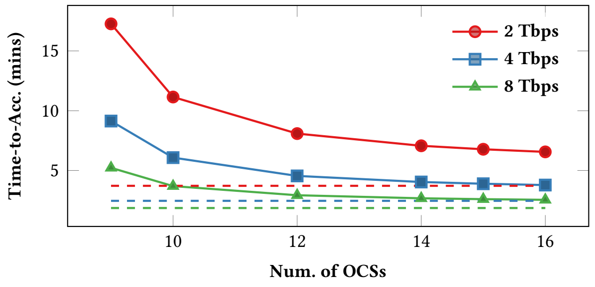


Figure 8: Impact of number of OCSs in SiP-OCS on time-to-accuracy of a hybrid training of Transformer with one-shot configuration. The lines correspond to different per-GPU bandwidth (B). Dashed horizontal lines of the same color show performance achieved by Elect-Flat at the same bandwidth.

Communication patterns. To better understand the communication patterns produced by Algorithm 1, Fig. 7 shows the traffic matrices for the Transformer model with MP degree $k = 4, 8, 16$, corresponding to 2 Tbps, 6 Tbps, and 10 Tbps per-GPU bandwidth, respectively. These traffic matrices have two main components: (i) a set of identical $k \times k$ blocks, corresponding to the traffic between the nodes in each MP group (brighter colors represent larger values); (ii) an off-diagonal component, corresponding to the DP ring-all-reduce traffic used by each GPU to synchronize its parameters with its peers in other MP groups (holding the same part of the model). Within the $k \times k$ blocks, the entries near the diagonal are larger (brighter), indicating the GPUs communicate more with their immediate neighbors. This property helps when mapping the communication to SiP-Ring. The off-diagonal entries (DP traffic) are smaller than the largest entries for the MP traffic, but they are still significant. This is the downside of current hierarchical electrical fabrics, as shown in Fig. 5, the low server-to-server bandwidth becomes a chokepoint.

The traffic matrices also show how SiP-ML meets the degree constraint. For example, in SiP-OCS, each GPU establishes circuits with members of its MP group and is also part of a ring with its peers in other MP groups. The resulting topology is effectively the union of $l = N/k$ identical direct-connect topologies and k rings. The number of circuits to each destination is chosen based on the traffic intensity towards that destination, although finding the optimal circuit allocation is more subtle and requires solving an ILP (§3.3).

Impact of number of OCSs and reconfiguration latency. Increasing the number of OCSs (or the total number of ports on each OCS) in SiP-OCS can improve performance in two ways: (i) we can increase the maximum permissible communication degree; or (ii) for the same communication degree, we can allow a more fine-grained allocation of circuits (with less bandwidth per circuit). The latter enables SiP-ML to align circuit bandwidth to traffic demands more closely, resulting in less wasted bandwidth. Fig. 8 shows the time-to-accuracy vs. number of OCSs for a one-shot circuit configuration of the Transformer model. Performance improves with more OCSs, but benefits are marginal beyond 12 OCSs. Also, unsurprisingly, a larger bandwidth per GPU (B) reduces sensitivity to the number of OCSs; it has more headroom, thus masking the inefficiencies caused by fewer OCSs.

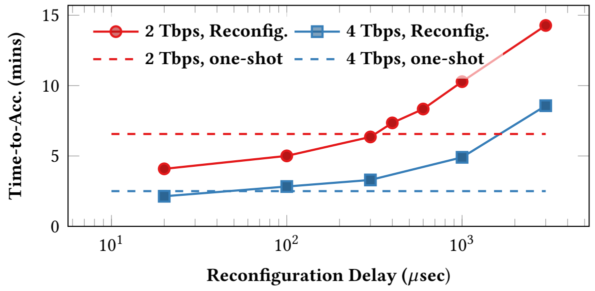


Figure 9: Impact of OCS reconfiguration delay on time-to-accuracy of Transformer in SiP-OCS for two per-GPU bandwidths. The critical reconfiguration delay when choosing between reconfigurable and one-shot allocation depends on the bandwidth.

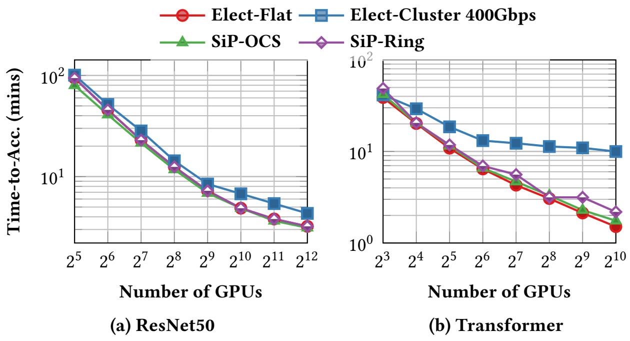


Figure 10: Overall performance of SiP-ML's OCS and Ring topologies at different scales.

Fig. 9 shows how future OCSs with faster reconfiguration time could improve the total training time of a Transformer model. For a reconfiguration delay of d, we use the traffic matrix of the past $5d$ seconds to reconfigure the circuit allocations. We maintain circuits for $5d$ to amortize the reconfiguration delay overhead. As expected, reducing the reconfiguration delay always helps. However, note that for $d > 300\mu s$, a one-shot allocation outperforms a dynamic reconfiguration. Once again, higher bandwidth per GPU masks inefficiencies, and one-shot allocation performs as well as rapid dynamic reconfiguration.

Impact of scale. Fig. 10 compares the training time of Resnet50 and Transformer on different network architectures across different scales, with $B = 8$ Tbps of bandwidth per GPU. As in Fig. 5, we see that SiP-OCS and SiP-Ring are close to the performance of

#GPUs	Fabric Latency				
	1 μsec	3 μsec	10 μsec	30 μsec	100 μsec
32	**1×**	0.99×	0.83×	0.73×	0.64×
128	2.11×	2.10×	1.52×	1.36×	1.29×
512	4.27×	4.04×	3.03×	2.49×	2.03×

Table 1: Impact of interconnect latency on the scaling efficiency. Training speed-ups are normalized by the speed-up at 32 GPUs with 1μsec latency.

the ideal Elect-Flat at all scales, with SiP-Ring occasionally slightly worse. With Elect-Cluster, the training time improves up to a certain scale, and then the benefits taper off as the low server-to-server bandwidth becomes a bottleneck. Once again, ResNet scales quite well with Elect-Cluster, in line with current practice [31]. But larger models and those less amenable to large-batch training, such as Transformer, can benefit significantly from SiP-ML's high per-GPU bandwidth at moderate-to-large system scales.

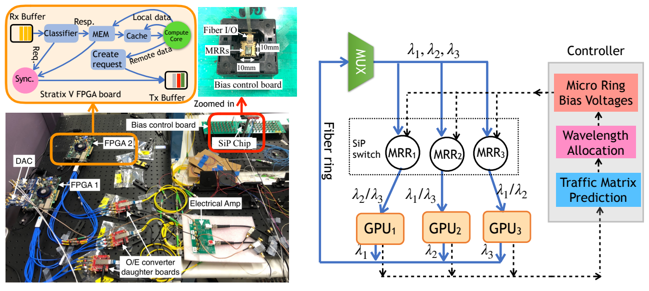
Impact of network latency. Network fabric latency can play an important role in scaling ML workloads at multi Tbps network speeds. Table 1 shows the impact of different minimum interconnect latencies on training performance. The results show the training speedup relative to an Elect-Flat network with 32 GPUs with B = 10 Tbps, and 1 μs fabric latency. Latencies above ∼10μs degrade performance. This suggests another potential advantage of optical networks over electrical switching fabrics, the latter can suffer from variable latency due to the presence of buffers. To compare to the best-case performance of the baselines, our simulations do not model buffering within electrical fabrics, as this depends on factors such as the details of the transport protocols [93, 94].

SiP-Ring reconfiguration delay. While Tbps SiP-enabled chiplets are just about to hit the market [8, 63, 95], their reconfiguration latency has not been evaluated. To evaluate the reconfiguration latency of SiP-ML's ring topology, we build a small-scale testbed (details in §4.4). Our testbed includes a thermo-optic SiP chip which has six micro-ring resonators (MRRs). To hit 10 Tbps bandwidth we must package 400 MRRs (each modulating light at 25 Gbps). As a result, our testbed only supports 10 Gbps bandwidth. Rather than bandwidth, we focus on validating reconfigurability. Our measurements show a reconfiguration delay of 25 μs (Fig. 12b and Fig. 12c in §4.4).

4.4 Testbed

To benchmark the switching time and throughput of a SiP-based architecture, we build a small-scale testbed.

Testbed setup. Fig. 11a shows a photograph of our experimental testbed. We built a three-node prototype of SiP-ML using FPGA development boards (to emulate GPUs), and a thermo-optic SiP chip which has six micro-ring resonators (MRRs). Each MRR is tuned to select one wavelength by receiving the appropriate bias signal from the bias control board. We use Stratix V FPGAs to emulate the GPU training workflow, as no commercial GPU chip supports optical interfaces. Our FPGAs have 50 Mb embedded memory and 1152 MB DDR3 memory. The FPGAs are programmed and configured as individual compute nodes with their own local memory. The controller logic is implemented using one of the FPGAs. A digital-to-analog converter (DAC) provides the necessary bias signals to the SiP chip



(a) Photo (b) Logical schematic

Figure 11: SiP-ML's testbed.

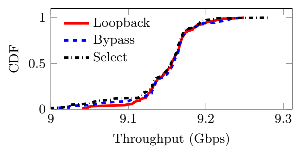
to cause a state change in the MRRs, depending on the scheduling decision. We use commodity SFP+ transceivers connected to the high-speed serial transceiver port on the FPGA board to achieve the conversion between electrical and optical domains. Our three input wavelengths are λ_1 =1546.92 nm, λ_2 =1554.94 nm, and λ_3 = 1556.55 nm. Our SiP optical chip consists of six MRRs (we use three of them as shown in Fig. 11b) to select and forward any of the wavelengths to the target emulated GPUs. To evaluate our prototype, we implement 2D convolutional computation workloads in Verilog to perform data fetching, computing, and storing between emulated GPU nodes. A GPU node can get access to the other GPU node's memory and perform read/write operations, similar to how real GPUs communicate today.

Example: programming the MRRs. We set the first configuration such that GPU_1 is connected to GPU_2; this means MRR_1 is tuned to select and forward λ_2 to GPU_1, while MRR_2 is tuned to select and forward λ_1 to GPU_2. For simplicity of the configuration logic, MRR_3 is always tuned to λ_1 but is effectively in idle mode, as the optical power of λ_1 has been dropped through MRR_2. To change the state to $Configuration_2$ where GPU_1 is connected to GPU_3, MRR_1 should be tuned to select and forward λ_3, while MRR_2 should be detuned from λ_1 for the optical power of λ_1 to pass through MRR_3 to GPU_3. Note that in this configuration, MRR_3, can remain tuned to λ_1.

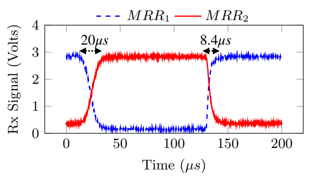
Testbed limitations. Our use of commodity FPGAs and transceivers is driven by pragmatic concerns. It allows us to implement workloads without needing separate modulation logic at the transmitter or demodulation logic at the receiver. Packets are forwarded to the SFP+ transceiver which modulates the light for us. However, this method has limitations as well. Implementing convolutional neural networks in an FPGA, rather than a GPU as would be the case in the actual system, introduces complex Verilog logic with overhead on (de)serializing the remote memory access commands.

To validate the feasibility of our optical design, we answer the following four key questions. (*i*) What is the impact of using MRRs to select/bypass wavelengths on throughput? (*ii*) How fast can we reconfigure the MRRs to dynamically tune to appropriate wavelengths? (*iii*) What is the end-to-end switching time? (*iv*) What is the impact of our scheduling algorithm on throughput?

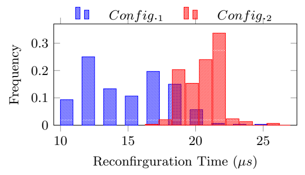
MRRs as select/bypass interfaces. We first examine the select/bypass functions of our MRR-based interfaces. A transceiver channel is instantiated on the FPGA and a SFP+ optical transceiver at



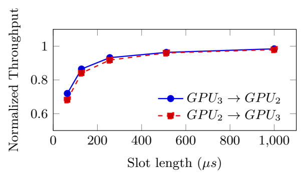
(a) Micro-ring select/bypass throughput



(b) Micro-ring reconfiguration time



(c) End-to-end reconfiguration time



(d) End-to-end throughput

Figure 12: Testbed benchmarks.

1546.92 nm is used to perform the throughput measurements for select, bypass and loopback cases. As shown in Fig. 12a, the throughput measurement of the select mode (the MRR tuned at 1546.92 nm) is the curve in black while the result for bypassing the MRR is in blue. The red curve is the baseline measurement where the optical transmitter is connected directly to the receiver channel without coupling the optical signal in/out the SiP chip. Our measurements show in all three cases, the throughput is 9 to 9.3 Gbps confirming the feasibility of the idea of using MRRs as select/bypass interfaces.

MRR reconfiguration time. To measure the reconfiguration time of our MRRs, we place InGaAs PIN photodetectors after MRR_1 and MRR_2 in Fig. 11b and change the bias voltage from $Config_1$ to $Config_2$, where MRR_1 and MRR_2 are tuned into and out of resonance with λ_1. We switch light between the two photodetectors by applying different bias signals to the SiP chip every 125 μs. The photodetectors convert the received photocurrent into voltage. We use an oscilloscope to measure real time light intensity and can therefore measure the reconfiguration speed. Fig. 12b shows the receive signal at the photodetectors. In one case, the signal reaches stable state in approximately 20 μs, and in another case, it takes only 8.4 μs. This is because tuning the MRR into the chosen wavelength is faster than tuning out of that wavelength due to our use of the thermal tuning effect. We conservatively, consider 25 μs as the switching time in our simulations. This experiment micro-benchmarks the micro-ring reconfiguration time; additional time might be required for transceivers to start decoding bits. This additional time is not fundamental, and next we show how we measured the end-to-end reconfiguration time between FPGAs.

End-to-end reconfiguration time. The end-to-end reconfiguration time includes the MRRs' reconfiguration time, the transceivers' locking time, and the handshaking time between newly connected nodes. The distribution of end-to-end switching time between $Config_1$ and $Config_2$ is shown in Fig. 12c. We perform 300 measurements to obtain the distribution, showing that the average switching time to $Config_1$ is 13 μs and $Config_2$ is 15 μs. Indeed, it is reasonable that the fastest end-to-end reconfiguration time may be less than the micro-ring reconfiguration time, as the receiver at the FPGA receives enough optical power to start the synchronization process before stabilization of the light output power. As described above, the micro-ring reconfiguration times for tuning and detuning are not equal, leading to two distinct distributions. The additional variations in the distribution of the reconfiguration time are a consequence of the time required for the transceiver to lock onto the new signal and carry out the handshaking protocol.

Putting it all together. We also measure the achieved throughput while changing the scheduling slot length between the two configurations. We conduct five different case studies with slot lengths of 64, 128, 256, 512 and 1000 μs and measure the ideal throughput. The curve in blue in Fig. 12d indicates the switching state from GPU_3 to GPU_2 lasting the duration set by the experiment; the curve in red indicates the switching from GPU_2 to GPU_3. As the plot shows, the link can achieve above 90% of the ideal throughput, when the scheduling slot length is 220 μs. This is because the end-to-end reconfiguration takes only about 20 μs; hence, having a scheduling slot 10 times larger will result in near optimal throughput.

5 DISCUSSION

Power budget and scalability. Optical power loss is a key measure for any optical system. To estimate the $\mathcal{D}$ of our SiP-Ring topology, we measure the loss of light in our testbed. Our experiments indicate that the loss per MRR is negligible (0.125–0.025 dB per MRR). However, coupling the light in and out of each node creates 0.5 dB loss because each I/O interface has an input and output coupler with loss. Overall, the total loss incurred by passing through each node on SiP-Ring is 0.625–0.525 dB. Hence, assuming a 10 dB power budget based on transmit power and receiver sensitivity [96], SiP-Ring can send light to 16 back-to-back neighbors without requiring amplification. At first blush, it appears infeasible to scale SiP-Ring, as building a cluster with more than 16 nodes needs amplifiers which add non-linear noise to the system. However, SiP-Ring can capture path length limitations in its placement algorithm. For instance, the path length in our evaluations is limited to 16 nodes (Appendix A.1). This is because the placement algorithm is able to place GPUs locally close to each other such that every GPU only interacts with, at most, a GPU that is 15 nodes away (i.e., the node degree is 16). As a result, SiP-Ring's design can take path length into account to scale to large numbers of nodes.

Cost of SiP-ML. The entire field of silicon photonics is based on the concept that the fundamental way to reduce the cost of photonic devices is to leverage the high volume manufacturing capabilities of the silicon electronics industry. As a result, it is impossible to provide an accurate cost estimation for SiP-ML. Prior work has built TeraPHY SiP interfaces with size 8.86 mm × 5.5 mm [20, Slide 41]. This area contains optical transmit, receive, and MRRs. The cost of manufacturing this SiP interface is \$44,082 for a volume of 20 chips (\$4,408/chip) based on 2020 Europractice pricelist [97].[5]

[5]Europractice is an EC initiative that provides the industry and academia with a platform to develop smart integrated systems, ranging from advanced prototype design to volume production. The cost is listed as €80,000 on page 10 under imec Si-Photonics iSiPP50G; the volume is listed as 20 samples on page 6 under iSiPP50G.

Hence, assuming the cost will drop by a factor of 10 at mass production, our current cost estimation for each SiP interface in SiP-ML is ≈\$440. We further estimate the cost of on-chip electrical circuitry (drivers, MRR's tuning control logic, and CMOS transimpedance amplification) to be ≈\$300. This estimate is based on Europractice pricelist for a 10 mm^2 chip area [14, 19, 98, 99].[6] Another approach to observe the potential cost effectiveness of SiP solutions is to look at it from the standpoint of pluggable transceivers and active copper cables. Today's SiP-based pluggable optics at 100 Gbps cost roughly \$1/Gbps (SiP PSM4 and CWDM4). In comparison, a non SiP-based SR-4 pluggable transceiver is around \$3/Gbps (multimode and VCSEL based). Similarly, a 400 Gbps SR8 is \$3/Gbps, while a SiP based 400 Gbps DR4 and FR4 is projected to be \$1/Gbps. We note that there is a large distinction between the cost of commodity DWDM transponders used in wide-area networks and SiP-ML's SiP interfaces. In particular, DWDM transponders are designed to operate at long distances; this imposes strict challenges on the laser, manufacturing, forward-error correction, photodiode sensitivity, modulation scheme, and light coupling. In contrast, SiP interfaces are designed for short distances and do not require coherent detection; hence, they can take advantage of the development and commercialization of photonics components for short distance datacenters.

6 RELATED WORK

Our work builds on two lines of related work.

Software/hardware systems for distributed ML. Many software platforms and techniques have focused on enabling large-scale distributed machine learning in recent years [100–105]. In particular several papers focus on enabling large-scale data parallel training [45, 100–104, 106]. Relevant to this paper, several aim to reduce communication overhead using techniques such as compression [107–110], asynchronous updates [28, 111–114], partially-exchanged gradients [115], and smart parameter propagation [2, 45, 116–119]. In addition, a variety of algorithmic approaches have been developed to accelerate communication among devices customized for the underlying network [120], or to improve model parallel training using smart task device placement [121, 122], and more efficient pipelining strategies [4, 123]. There is also a significant body of work on new electrical hardware designs to accelerate machine learning computations [118, 124–129]. The work proposed here is orthogonal to the above mentioned techniques, as they can still be applied to further improve both data and model parallel training. Our work differs in that we investigate the system requirements of using SiP as a new underlying technology to interconnect hundreds of GPUs in an all-optical architecture.

Datacenter Interconnects. The broad vision for this paper is to use all-optical interconnects for future distributed ML systems. Optical interconnects have a long and rich history in the datacenter research community [24–26, 55, 66, 70, 71, 130–135]. Prior work shows the benefits of reconfigurable topologies in datacenter networks by adding optical links to the electrical topology [24, 66, 71, 133, 136] or by creating all-optical datacenter interconnects [26, 55, 70, 131, 132]. The unpredictability of legacy datacenter workloads and the complexity of managing hybrid topologies are two main reasons for the lack of adoption of all-optical datacenters so far. In contrast, this paper builds an all-optical interconnect with a simple and practical task placement algorithm primarily used to accelerate ML workloads. Our ring topology (SiP-Ring) is inspired by Quartz [70], Mordia [71], and Megaswitch [26]. They all use a fiber ring to interconnect the datacenter topology, but they do not leverage MRRs. Moreover, Mordia realizes a microsecond switching circuit switch, but it does not reuse wavelengths, and this significantly reduces its bandwidth efficiency compared to SiP-Ring. As a result, Mordia's number of ports is limited by the number of wavelengths. Jellyfish [137], Rotornet [66], and Opera [69] take advantage of the unpredictability of datacenter workloads and use expander-based topologies to improve the completion time of short and long flows. Random permutations are not ideal for ML workloads, as a training workload is a periodic repetition of thousands of iterations. Shoal [135], Larry [138], XFabric [139], and Sirius [55] have proposed reconfigurable datacenter interconnects with nanosecond switching fabric. We believe these proposals have the potential to change the game in datacenter environments, but they are not commercially available yet and they do not support Tbps bandwidth between communicating nodes. Moreover, our results show μs reconfiguration latency is close to optimal for ML; a control plane with nanosecond response time might be needed for a general purpose datacenter traffic, but it is an overkill for distributed ML training. Finally, there is rich body of research on silicon photonics [17, 140–142], embedding silicon photonics switches in High Performance Computing clusters [143] and energy-efficient datacenters [144]. By focusing on ML, our work takes an application-level perspective to build an interconnect with SiP components.

7 CONCLUSION

In this paper, we propose optical network interconnects for distributed ML training clusters capable of providing multiple terabits-per-second of bandwidth per GPU. Our results show that the predictability of ML workloads makes them a great fit for optical interconnects. We develop a new task partitioning and placement algorithm that exploits the degree requirement of optical networks to find a parallelization strategy suitable for a given network topology. We show this approach can mitigate and in fact largely overcome concerns such as limited communication degree and reconfigurability of optical circuit-switched networks. Simulations using three real DNN models show that, compared to today's electrical network fabrics with limited server-to-server bandwidth, SiP-ML improves the training time by 1.3–9.1× at scale.

8 ACKNOWLEDGMENTS

We would like to thank our shepherd Hitesh Ballani and anonymous reviewers for their feedback. This work was partly supported by AEPA-E ENLITENED PINE, DARPA FastNICs, DARPA PIPES, a Cisco Research Center Award, NSF ASCENT-2023468, NSF CNS-2008624, NSF CNS-1751009, NSF CNS-2006827, NSF CNS-1563826 as well as by a SystemsThatLearn@CSAIL Ignite Grant and a MachineLearningApplications@CSAIL Award.

[6]Page 6 under GLOBALFOUNDRIES 22 nm FDSOI lists €14,000/mm^2 for 50 samples.

REFERENCES

[1] AI and Compute. https://openai.com/blog/ai-and-compute/.

[2] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. In *SysML Conference*, 2019.

[3] Siddharth Das. CNN Architectures, 2017.

[4] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP âĂŽ19, page 1âĂŞ15, New York, NY, USA, 2019. Association for Computing Machinery.

[5] NVIDIA DGX A100. https://www.nvidia.com/en-us/data-center/dgx-a100/.

[6] NVIDIA Selene Cluster. https://blogs.nvidia.com/blog/2020/12/18/nvidia-selene-busy/.

[7] S S Vazhkudai, B R de Supinski, A S Bland, A Geist, J Sexton, J Kahle, C J Zimmer, S Atchley, S H Oral, D E Maxwell, V G Vergara Larrea, A Bertsch, R Goldstone, W Joubert, C Chambreau, D Appelhans, R Blackmore, B Casses, G Chochia, G Davison, M A Ezell, E Gonsiorowski, L Grinberg, B Hanson, B Hartner, I Karlin, M L Leininger, D Leverman, C Marroquin, A Moody, M Ohmacht, R Pankajakshan, F Pizzano, J H Rogers, B Rosenburg, D Schmidt, M Shankar, F Wang, P Watson, B Walkup, L D Weems, and J Yin. The design, deployment, and evaluation of the coral pre-exascale systems. 7 2018.

[8] Valerie Coffey. DARPA PIPES Program demonstrates 2 Tbit/s optical interconnects at the chip level, July 2020. https://www.laserfocusworld.com/fiber-optics/article/14176186/darpa-pipes-program-demonstrates-2-tbits-optical-interconnects-at-the-chip-level.

[9] Mark Wade. Optical i/o chiplets eliminate bottlenecks to unleash innovation, 2020. https://ayarlabs.com/ayar-labs-solving-critical-computing-challenges-through-optical-i-o/.

[10] Yutaka Urino, Takahiro Nakamura, and Yasuhiko Arakawa. *Silicon Optical Interposers for High-Density Optical Interconnects*, pages 1–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[11] D. Kim, K. Y. Au, H. Y. L. X. Luo, Y. L. Ye, S. Bhattacharya, and G. Q. Lo. 2.5d silicon optical interposer for 400 gbps electronic-photonic integrated circuit platform packaging. In *2017 IEEE 19th Electronics Packaging Technology Conference (EPTC)*, pages 1–4, Dec 2017.

[12] E. R. H. Fuchs, R. E. Kirchain, and S. Liu. The future of silicon photonics: Not so fast? insights from 100g ethernet lan transceivers. *Journal of Lightwave Technology*, 29(15):2319–2326, Aug 2011.

[13] David Thomson, Aaron Zilkie, John E Bowers, Tin Komljenovic, Graham T Reed, Laurent Vivien, Delphine Marris-Morini, Eric Cassan, Leopold Virot, Jean-Marc Fedeli, Jean-Michel Hartmann, Jens H Schmid, Dan-Xia Xu, Frederic Boeuf, Peter O'Brien, Goran Z Mashanovich, and M Nedeljkovic. Roadmap on silicon photonics. *Journal of Optics*, 18(7):073003, 2016.

[14] M. Wade, M. Davenport, M. De Cea Falco, P. Bhargava, J. Fini, D. Van Orden, R. Meade, E. Yeung, R. Ram, M. Popovic, V. Stojanovic, and C. Sun. A bandwidth-dense, low power electronic-photonic platform and architecture for multi-tbps optical i/o. pages 1–3, Sep. 2018.

[15] N. Ophir, C. Mineo, D. Mountain, and K. Bergman. Silicon photonic microring links for high-bandwidth-density, low-power chip i/o. *IEEE Micro*, 33(1):54–67, Jan 2013.

[16] G.T. Reed and A.P. Knights. *Silicon Photonics: An Introduction.* Wiley, 2004.

[17] Qixiang Cheng, Meisam Bahadori, Madeleine Glick, Sebastien Rumley, and Keren Bergman. Recent advances in optical technologies for data centers: a review. *Optica*, 5(11):1354–1370, Nov 2018.

[18] Madeleine Glick, Lionel C. Kimmerling, and Robert C. Pfahl. A roadmap for integrated photonics. *Opt. Photon. News*, 29(3):36–41, Mar 2018.

[19] Amir H. Atabaki, Sajjad Moazeni, Fabio Pavanello, Hayk Gevorgyan, Jelena Notaros, Luca Alloatti, Mark T. Wade, Chen Sun, Seth A. Kruger, Huaiyu Meng, Kenaish Al Qubaisi, Imbert Wang, Bohan Zhang, Anatol Khilo, Christopher V. Baiocco, Milovs A. Popovic, Vladimir M. Stojanovic, and Rajeev J. Ram. Integrating photonics with silicon nanoelectronics for the next generation of systems on a chip. *Nature*, 556(7701):349–354, 2018.

[20] Mark Wade, Erik Anderson, Shahab Ardalan, Pavan Bhargava, Sidney Buchbinder, Michael Davenport, John Fini, Anatoly Khilo, Chandru Ramamurthy Roy Meade, Michael Rust, Vladimir Stojanovic Forrest Sedgwick, Derek Van Orden, Chong Zhang Edward Wang, Chen Sun, Sergey Shumarayev, Conor O'Keeffe, Tim T. Hoang, David Kehlet, Ravi V. Mahajan, Allen Chan, and Tina Tran. TeraPHY: A Chiplet Technology for Low-Power, High-Bandwidth Optical I/O. *HotChips*, pages i–xlviii, August 2019. https://www.hotchips.org/hc31/HC31$_{2}$.9$_{A}$yarLabs$_{2}$0190820$_{H}$C$_{F}$INAL.pdf.

[21] Valentina Donzella, Ahmed Sherwali, Jonas Flueckiger, Samantha M. Grist, Sahba Talebi Fard, and Lukas Chrostowski. Design and fabrication of soi microring resonators based on sub-wavelength grating waveguides. *Opt. Express*, 23(4):4791–4803, Feb 2015.

[22] W. Bogaerts, P. De Heyn, T. Van Vaerenbergh, K. De Vos, S. Kumar Selvaraja, T. Claes, P. Dumon, P. Bienstman, D. Van Thourhout, and R. Baets. Silicon microring resonators. *Laser & Photonics Reviews*, 6(1):47–73, 2012. https://onlinelibrary.wiley.com/doi/abs/10.1002/lpor.201100017.

[23] Q. Cheng, M. Bahadori, Y. Hung, Y. Huang, N. Abrams, and K. Bergman. Scalable microring-based silicon clos switch fabric with switch-and-select stages. *IEEE Journal of Selected Topics in Quantum Electronics*, 25(5):1–11, Sep. 2019.

[24] Nathan Farrington, George Porter, Sivasankar Radhakrishnan, Hamid Hajabdolali Bazzaz, Vikram Subramanya, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Helios: A hybrid electrical/optical switch architecture for modular data centers. *SIGCOMM'10*, pages 339–350.

[25] Guohui Wang, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, T.S. Eugene Ng, Michael Kozuch, and Michael Ryan. c-Through: Part-time optics in data centers. *SIGCOMM'10*, pages 327–338.

[26] Li Chen, Kai Chen, Zhonghua Zhu, Minlan Yu, George Porter, Chunming Qiao, and Shan Zhong. Enabling wide-spread communications on optical fabric with megaswitch. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 577–593, Boston, MA, 2017. USENIX Association.

[27] Pengtao Xie, Jin Kyu Kim, Yi Zhou, Qirong Ho, Abhimanu Kumar, Yaoliang Yu, and Eric Xing. Lighter-communication distributed machine learning via sufficient factor broadcasting. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 795–804, Arlington, Virginia, USA, 2016. AUAI Press.

[28] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. OSDI'14, pages 583–598. USENIX Association, 2014.

[29] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *Int. J. High Perform. Comput. Appl.*, 19(1):49–66, February 2005.

[30] Baidu, 2017. https://github.com/baidu-research/baidu-allreduce.

[31] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018.

[32] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[33] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. On model parallelization and scheduling strategies for distributed machine learning. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2834–2842. Curran Associates, Inc., 2014.

[34] Zhihao Jia, Sina Lin, Charles R. Qi, and Alex Aiken. Exploring hidden dimensions in accelerating convolutional neural networks. volume 80 of *Proceedings of Machine Learning Research*, pages 2274–2283, StockholmsmÃďssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[35] Tal BenNun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *CoRR*, abs/1802.09941, 2018.

[36] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen. Accpar: Tensor partitioning for heterogeneous deep learning accelerators. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 342–355, 2020.

[37] Nikoli Dryden, Naoya Maruyama, Tim Moon, Tom Benson, Marc Snir, and Brian Van Essen. Channel and filter parallelism for large-scale cnn training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'19, New York, NY, USA, 2019. Association for Computing Machinery.

[38] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *SysML*, 2019.

[39] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large scale distributed deep networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

[40] Amir Gholami, Ariful Azad, Kurt Keutzer, and Aydin Buluç. Integrated model and data parallelism in training neural networks. *CoRR*, abs/1712.04432, 2017.

[41] Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. Learning generalizable device placement algorithms for distributed machine learning. In *Advances in Neural Information Processing Systems 32*, pages 3983–3993. Curran Associates, Inc., 2019.

[42] Shar Narasimhan. NVIDIA Clocks World's Fastest BERT Training Time and Largest Transformer Based Model, Paving Path For Advanced Conversational AI, Aug. 2019. https://devblogs.nvidia.com/training-bert-with-gpus/.

[43] Nikoli Dryden, Naoya Maruyama, Tom Benson, Tim Moon, Marc Snir, and Brian Van Essen. Improving strong-scaling of cnn training by exploiting finer-grained parallelism, 2019.

[44] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In

Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pages 1729–1739, Red Hook, NY, USA, 2017. Curran Associates Inc.

[45] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[46] Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019.

[47] Yosuke Oyama, Naoya Maruyama, Nikoli Dryden, Erin McCarthy, Peter Harrington, Jan Balewski, Satoshi Matsuoka, Peter Nugent, and Brian Van Essen. The case for strong scaling in deep learning: Training large 3d cnns with hybrid parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2020.

[48] MLPerf v0.6: NVIDIA Implementation of Attention Mechanisms for Translation, Aug. 2019. https://github.com/mlperf/training$_r$esults$_v$0.6/tree/master/NVIDIA/benchmarks/transformer/implementations/pytorch.

[49] ResNet v1.5 for TensorFlow, 2020.

[50] NVIDIA Data Center Deep Learning Product Performance. https://developer.nvidia.com/deep-learning-performance-training-inference.

[51] Nvidia DGX-2. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/dgx-2/dgx-2-print-datasheet-738070-nvidia-a4-web-uk.pdf.

[52] MegatronLM: Training Billion+ Parameter Language Models Using GPU Model Parallelism, Jul. 2019. https://nv-adlr.github.io/MegatronLM.

[53] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models, 2019. https://www.deepspeed.ai/.

[54] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, Matthew Denton, and Tushar Krishna. Efficient communication acceleration for next-gen scale-up deep learning training platforms, 2020.

[55] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. Sirius: A Flat Datacenter Network with Nanosecond Optical Switching. *SIGCOMM'20*, Aug. 2020.

[56] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 365–376, June 2011.

[57] R. Colwell. The chip design game at the end of moore's law. In *2013 IEEE Hot Chips 25 Symposium (HCS)*, pages 1–16, Aug 2013.

[58] H. J. S. Dorren, E. H. M. Wittebol, R. de Kluijver, G. Guelbenzu de Villota, P. Duan, and O. Raz. Challenges for optically enabled high-radix switches for data center networks. *Journal of Lightwave Technology*, 33(5):1117–1125, March 2015.

[59] Alexis BjÃűrlin and Manish Mehta. Broadcom discusses its co-packaged optics plans. http://www.gazettabyte.com/home/2021/4/27/broadcom-discusses-its-co-packaged-optics-plans.html, 2021. [Online; last accessed 25-June-2021].

[60] Steven Leibson. Ayar labs and Intel demo FPGA with optical transceivers in DARPA PIPES project: 2 Tbps now, >100 Tbps is the goal, Mar. 2020. https://blogs.intel.com/psg/ayar-labs-and-intel-demo-fpga-with-optical-transceivers-in-darpa-pipes-project-2-tbps-now-100-tbps-is-the-goal/.

[61] Pipes researchers demonstrate optical interconnects to improve performance of digital microelectronics, Mar. 2020. https://www.darpa.mil/news-events/2020-03-25.

[62] Tiffany Trader. Ayar Labs to Demo Photonics Chiplet in FPGA Package at Hot Chips, Aug. 2019. https://www.hpcwire.com/2019/08/19/ayar-labs-to-demo-photonics-chiplet-in-fpga-package-at-hot-chips/.

[63] F. Douglis, S. Robertson, E. Van den Berg, J. Micallef, M. Pucci, A. Aiken, M. Hattink, M. Seok, and K. Bergman. Fleet—fast lanes for expedited execution at 10 terabits: Program overview. *IEEE Internet Computing*, (01):1–1, apr 5555.

[64] Ayar Labs TeraPHY Silicon Chip. https://ayarlabs.com/products/.

[65] Demonstration of Ayar Labs' Optical I/O Multi-Chip Package and Single-Die Package solutions, Aug. 2020. https://vimeo.com/449164007.

[66] William M. Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C. Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. *SIGCOMM '17*, pages 267–280, 2017.

[67] Tae Joon Seok, Niels Quack, Sangyoon Han, Richard S. Muller, and Ming C. Wu. Large-scale broadband digital silicon photonic switches with vertical adiabatic couplers. *Optica*, 3(1):64–70, Jan 2016.

[68] Kyungmok Kwon, Tae Joon Seok, Johannes Henriksson, Jianheng Luo, Lane Ochikubo, John Jacobs, Richard S Muller, and Ming C Wu. 128× 128 silicon photonic mems switch with scalable row/column addressing. In *CLEO: Science and Innovations*, pages SF1A–4. Optical Society of America, 2018.

[69] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. *NSDI'20*, 2020.

[70] Yunpeng James Liu, Peter Xiang Gao, Bernard Wong, and Srinivasan Keshav. Quartz: A new design element for low-latency dcns. *SIGCOMM'14*, pages 283–294.

[71] George Porter, Richard Strong, Nathan Farrington, Alex Forencich, Pang Chen-Sun, Tajana Rosing, Yeshaiahu Fainman, George Papen, and Amin Vahdat. Integrating microsecond circuit switching into the data center. *SIGCOMM'13*, pages 447–458.

[72] meg walraed sullivan, Jitu Padhye, and Dave Maltz. Theia: Simple and cheap networking for ultra-dense data centers. In *HotNets-XIII Proceedings of the 13th ACM Workshop on Hot Topics in Networks*. ACM, October 2014.

[73] Paolo Costa, Austin Donnelly, Greg O'Shea, and Antony Rowstron. Camcubeos: A key-based network stack for 3d torus cluster topologies. In *Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '13, pages 73–84, New York, NY, USA, 2013. Association for Computing Machinery.

[74] Hussam Abu-Libdeh, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. Symbiotic routing in future data centers. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, page 51?62, New York, NY, USA, 2010. Association for Computing Machinery.

[75] J. M. Kumar and L. M. Patnaik. Extended hypercube: a hierarchical interconnection network of hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):45–57, 1992.

[76] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Comput. Archit. News*, 36(3):77âĂŞ88, June 2008.

[77] Min Yee Teh, Jeremiah J. Wilke, Keren Bergman, and Sébastien Rumley. Design space exploration of the dragonfly topology. In Julian M. Kunkel, Rio Yokota, Michela Taufer, and John Shalf, editors, *High Performance Computing*, pages 57–74, Cham, 2017. Springer International Publishing.

[78] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77–88, 2008.

[79] Calient Optical Circuit Switch. https://www.calient.net/products/edge640-optical-circuit-switch/.

[80] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, August 2008.

[81] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: A high performance, server-centric network architecture for modular data centers. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 63?74, New York, NY, USA, 2009. Association for Computing Machinery.

[82] M. Besta and T. Hoefler. 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*, pages 348–359, Nov 2014.

[83] Alexander Ishii, Denis Foley, Eric Anderson, Bill Dally, Glenn Dearth Larry Dennison, Mark Hummel, and John Schafer. NVIDIA's NVLink-Switching Chip and Scale-Up GPU-Compute Server. *HotChips*, 2018. https://www.hotchips.org/hc30/2conf/2.01$_N$vidia$_N$Vswitch$_H$otChips2018$_D$GX2NVS$_F$inal.pdf.

[84] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[85] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training.

[86] Christopher J. Shallue, Jaehoon Lee, Joseph M. Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *CoRR*, abs/1811.03600, 2018.

[87] Raul Puri. Megatron: a large, powerful transformer, Aug. 2019. https://github.com/NVIDIA/Megatron-LM.

[88] MLPerf: A broad ML benchmark suite. https://mlperf.org/.

[89] FlexFlow Github. https://github.com/flexflow/FlexFlow.git.

[90] Takuya Akiba, Shuji Suzuki, and Keisuke Fukuda. Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. *arXiv preprint arXiv:1711.04325*, 2017.

[91] Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel, and Kurt Keutzer. Imagenet training in minutes. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[92] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[93] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[94] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. 2019.

[95] Roy Meade, Shahab Ardalan, Michael Davenport, John Fini, Chen Sun, Mark Wade, Alexandra Wright-Gladstein, and Chong Zhang. Teraphy: A high-density electronic-photonic chiplet for optical i/o from a multi-chip module. In *Optical Fiber Communication Conference (OFC) 2019*, page M4D.7. Optical Society of America, 2019.

[96] Alvaro moscoso martir, Juliana MÃijller, Johannes Hauck, Nicolas Chimot, Rony Setter, Avner Badihi, Daniel Rasmussen, Alexandre Garreau, Mads Nielsen, Elmira Islamova, Sebastian Romero-GarcÃŋa, Bin Shen, Anna Sandomirsky, Sylvie Rockman, Chao Li, Saeed Sharif Azadeh, Guo-Qiang Lo, Elad Mentovich, Florian Merget, and Jeremy Witzens. Silicon photonics wdm transceiver with soa and semiconductor mode-locked laser. *Scientific Reports*, 7, 05 2016.

[97] 2020 General Europractice Pricelist, Jan. 2020. https://europractice-ic.com/wp-content/uploads/2020/01/General-MPW-EUROPRACTICE-200123-v3.pdf.

[98] D. Kim, K. Y. Au, H. Y. L. X. Luo, Y. L. Ye, S. Bhattacharya, and G. Q. Lo. 2.5d silicon optical interposer for 400 gbps electronic-photonic integrated circuit platform packaging. In *2017 IEEE 19th Electronics Packaging Technology Conference (EPTC)*, pages 1–4, Dec 2017.

[99] Chen Sun, Mark T. Wade, Yunsup Lee, Jason S. Orcutt, Luca Alloatti, Michael S. Georgas, Andrew S. Waterman, Jeffrey M. Shainline, Rimas R. Avizienis, Sen Lin, Benjamin R. Moss, Rajesh Kumar, Fabio Pavanello, Amir H. Atabaki, Henry M. Cook, Albert J. Ou, Jonathan C. Leu, Yu-Hsin Chen, Krste Asanović, Rajeev J. Ram, MilošA. Popović, and Vladimir M. Stojanović. Single-chip microprocessor that communicates directly using light. *Nature*, 528(7583):534–538, 2015.

[100] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[101] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *International conference on machine learning*, pages 1337–1345, 2013.

[102] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI'14*, pages 571–582, 2014.

[103] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In *NSDI'19*, pages 485–500, 2019.

[104] Peng Sun, Wansen Feng, Ruobing Han, Shengen Yan, and Yonggang Wen. Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes. *arXiv preprint arXiv:1902.06855*, 2019.

[105] Adam Lerer, Ledell Wu, Jiajun Shen, Timothée Lacroix, Luca Wehrstedt, Abhijit Bose, and Alexander Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. *CoRR*, abs/1903.12287, 2019.

[106] Luo Mai, Chuntao Hong, and Paolo Costa. Optimizing network performance in distributed machine learning. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, 2015. USENIX Association.

[107] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[108] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. QSGD: Communication-efficient SGD via randomized quantization and encoding. volume 3, pages 1710 – 1721, 2018.

[109] Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. *arXiv preprint arXiv:1802.07389*, 2018.

[110] Peng Jiang and Gagan Agrawal. A linear speedup analysis of distributed deep learning with sparse and quantized communication. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2525–2536. Curran Associates, Inc., 2018.

[111] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[112] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *Advances in Neural Information Processing Systems*, pages 685–693, 2015.

[113] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[114] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 693–701, 2011.

[115] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. Ako: Decentralised deep learning with partial gradient exchange. *SoCC '16*, 2016.

[116] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy H. Campbell. Communication scheduling as a first-class citizen in distributed machine learning systems. *CoRR*, abs/1803.03288, 2018.

[117] Forrest N. Iandola, Khalid Ashraf, Matthew W. Moskewicz, and Kurt Keutzer. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. *CoRR*, abs/1511.00175, 2015.

[118] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengil, Ming Liu, Daniel Lo, Shlomi Alkalay, and Michael Haselman. Accelerating persistent neural networks at datacenter scale. In *Hot Chips*, volume 29, 2017.

[119] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed DNN training. *CoRR*, abs/1905.03960, 2019.

[120] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

[121] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2430–2439. JMLR. org, 2017.

[122] Linnan Wang, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuaiwen Leon Song, Zenglin Xu, and Tim Kraska. Superneurons: dynamic gpu memory management for training deep neural networks. In *ACM SIGPLAN Notices*, volume 53, pages 41–53. ACM, 2018.

[123] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *NeurIPS*, 2019.

[124] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[125] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, 11(7):441, 2017.

[126] Mahdi Nazm Bojnordi and Engin Ipek. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–13. IEEE, 2016.

[127] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2017.

[128] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017.

[129] Stephen W Keckler, William J Dally, Brucek Khailany, Michael Garland, and David Glasco. Gpus and the future of parallel computing. *IEEE Micro*, 31(5):7–17, 2011.

[130] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[131] Navid Hamedazimi, Zafar Qazi, Himanshu Gupta, Vyas Sekar, Samir R. Das, Jon P. Longtin, Himanshu Shah, and Ashish Tanwer. Firefly: A reconfigurable wireless data center fabric using free-space optics. *SIGCOMM'14*, pages 319–330.

[132] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P. Blanche, H. Rastegarfar, M. Glick, and D. Kilper. Projector: Agile reconfigurable data center interconnect. *SIGCOMM '16*, pages 216–229, 2016.

[133] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael Kaminsky, George Porter, and Alex C. Snoeren. Scheduling techniques for hybrid circuit/packet networks. In *CoNEXT*, pages 41:1–41:13. ACM, 2015.

[134] Ankit Singla, Atul Singh, and Yan Chen. OSA: An optical switching architecture for data center networks with unprecedented flexibility. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 239–252, San Jose, CA, 2012. USENIX.

[135] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. Shoal: A network architecture for disaggregated racks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*. USENIX, February 2019.

[136] He Liu, Feng Lu, Alex Forencich, Rishi Kapoor, Malveeka Tewari, Geoffrey M. Voelker, George Papen, Alex C. Snoeren, and George Porter. Circuit switching under the radar with REACToR. *NSDI'14*, pages 1–15.

[137] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. Jellyfish: Networking data centers randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 17–17, Berkeley, CA, USA, 2012. USENIX Association.

[138] Andromachi Chatzieleftheriou, Sergey Legtchenko, Hugh Williams, and Antony Rowstron. Larry: Practical network reconfigurability in the data center. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 141–156, Renton, WA, April 2018. USENIX Association.

[139] Sergey Legtchenko, Nicholas Chen, Daniel Cletheroe, Antony Rowstron, Hugh Williams, and Xiaohan Zhao. Xfabric: A reconfigurable in-rack network for rack-scale computers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 15–29, Santa Clara, CA, March 2016. USENIX Association.

[140] Sebastien Rumley, Meisam Bahadori, Robert Polster, Simon D. Hammond, David M. Calhoun, Ke Wen, Arun Rodrigues, and Keren Bergman. Optical interconnects for extreme scale computing systems. *Parallel Computing*, 64:65 – 80, 2017. High-End Computing for Next-Generation Scientific Discovery.

[141] Nicolas Sherwood-Droz, Howard Wang, Long Chen, Benjamin G. Lee, Aleksandr Biberman, Keren Bergman, and Michal Lipson. Optical 4×4 hitless silicon router for optical networks-on-chip (noc). *Opt. Express*, 16(20):15915–15922, Sep 2008.

[142] Qixiang Cheng, Sebastien Rumley, Meisam Bahadori, and Keren Bergman. Photonic switching in high performance datacenters. *Opt. Express*, 26(12):16022–16043, Jun 2018.

[143] G. Michelogiannakis, Y. Shen, X. Meng M. Y. Teh, B. Aivazi, T. Groves, J. Shalf, M. Glick, M. Ghobadi, L. Dennison, and K. Bergman. Bandwidth steering for hpc using silicon nanophotonics. *ACM/IEEE Supercomputing Conference (SC)*, 10 2019.

[144] Keren Bergman, John Shalf, George Michelogiannakis, Sebastien Rumley, Larry Dennison, and Monia Ghobadi. Pine: An energy efficient flexibly interconnected photonic data center architecture for extreme scalability. In *2018 IEEE Optical Interconnects Conference (OI)*, OI '18, 2018.

[145] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[146] Robert E Tarjan. Dynamic trees as search trees via euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78(2):169–177, 1997.

[147] James B Orlin, Serge A Plotkin, and Éva Tardos. Polynomial dual network simplex algorithms. *Mathematical programming*, 60(1-3):255–276, 1993.

[148] Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. Technical Report UCB/CSD-85-242, EECS Department, University of California, Berkeley, May 1985.

A APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A.1 SiP-Ring

One of the core properties of SiP-ML-Ring is the ability to dynamically place bandwidth around the static topology to maximize the throughput between communicating nodes for model parallel jobs. Note that for ring-allreduce data parallel jobs, there is no need to reschedule the bandwidth once a physical ring is established by the patch panel. However, we find that model parallel jobs benefit from bandwidth rescheduling. The optimal bandwidth allocation maximizes the throughput while ensuring no two paths sharing the same fiber are assigned the same wavelength. More formally, the bandwidth allocation problem corresponds to the following optimization problem. Let TM_{ij} denote the predicted GPU-to-GPU traffic matrix, and W denote the total number of wavelengths (a.k.a available bandwidth). We can represent a wavelength allocation as a 3-dimensional binary matrix, Λ, where Λ_{ijk} is 1 if GPU i sends data to GPU j using λ_k and is zero otherwise. There are several possible objectives. A natural one is to minimize the maximum completion time for any GPU-to-GPU transfer, where the completion time is $\frac{TM_{ij}}{\sum_k \Lambda_{ijk}}$. This can be expressed as an Integer Linear Program (ILP) by maximizing the minimum *inverse of the completion time*, as follows:

$$\begin{array}{llll}
 & \text{maximize}_{\Lambda \in \{0,1\}^{N\times N\times W}} \min\limits_{ij:TM_{ij}>0} & \sum_k \Lambda_{ijk}/TM_{ij} & \\
\text{s.t.} & & & \\
(1) & \sum_{(j\le i\le N+l)\cap(l<j)} \Lambda_{(i \bmod N)jk} & \le 1 & \forall l,k \\
(2) & \sum_i \Lambda_{ijk} & \le 1 & \forall j,k \\
(3) & \sum_j \Lambda_{ijk} & \le 1 & \forall i,k
\end{array} \qquad (1)$$

The constraints are (1) ensure fiber segments do not contain overlapping wavelengths (ring constraint), and (2) ensure each GPU can use each wavelength for communication with, at most, one other GPU (node constraint).

Note that the size of the ILP solution space, $\Lambda \in \{0,1\}^{N\times N\times W}$, grows with the number of nodes in the network, rendering it intractable at larger scale. Therefore, instead of solving the ILP, we present a more practical algorithm that turns this discrete optimization problem into a min-cost flow routing problem which can be solved efficiently.

Step 1: Communication graph construction. We construct a directed communication graph, $G = (V, E)$, where V is the set of nodes and for every $TM_{uv} > 0$, there is a directed edge $e = (u, v)$. After including edges for the entire TM in G, we check whether every adjacent node-pair on the topology is connected in G. If not, we add a "dummy" edge between them to E. The direction of all edges in G is the same as that of wave propagation on the fiber.We then add dummy sink and source nodes by cutting the edges in G along an arbitrary topology segment. For simplicity, let us assume for now that this process cuts only one edge of the graph.We add two terminal nodes on the two ends of the cut edge to be the source and sink. The source node injects a unit-sized flow into the ring and the sink node receives it.

Step 2: Compute min-cost flow. Having constructed the graph G, we solve the following flow routing problem:

$$\text{maximize} \sum_{e\in E:TM_e>0} \frac{f_e}{TM_e}, \qquad (2)$$

where for an edge $e = (u, v)$, f_e is the flow on the edge, and $TM_e = TM_{uv}$ is the traffic demand on that edge. The constraints (not shown for brevity) are the standard flow conservation constraints. The intuition for the above objective is that we wish to maximize throughput but preferentially allocate a larger flow (more wavelengths) to GPU-to-GPU paths with smaller demand. The reason for favoring smaller demands is to complete them quickly, reducing the number of nodes with which each node must communicate. This keeps the unsatisfied traffic pattern sparse over time, allowing the remaining traffic to be handled efficiently in future wavelength reconfiguration events.

The objective in Eq. (2) can be equivalently be written as a min-cost flow routing problem [145] by defining the *weight* of edge e as $w_e = -1/TM_e$ if $TM_e > 0$, and $w_e = 0$ if e is a dummy edge. The problem is then to minimize $\sum_e w_e f_e$. Min-cost flow routing can be solved using the *network simplex* algorithm [145–147]. The procedure for constructing the graph and defining the flow routing problem is slightly more complicated when the cut chosen for adding the source and sink nodes includes more than one edge. In this case, we need additional constraints to ensure consistency of flows between the cut edges.

In the more general case of cuts with higher degrees, suppose we would like to inject the flows at the segment between Node$_3$ and Node$_4$. The problem remains basically the same, but we need to add the following three constraints in addition to the flow conservation constraints: (1) $X = X'$, (2) $Y = Y'$, and (3) $X + Y = 1$. We can simply add these constraints to our simplex problem as well.

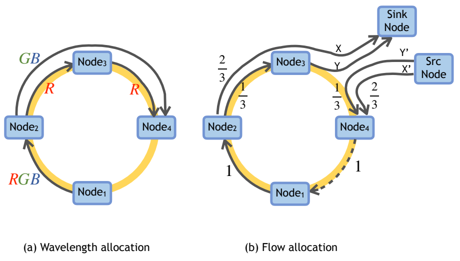


Figure 13: Wavelength allocation and its equivalent flow routing translation for multiedge cut.

Step 3: Remove and repeat. The solution obtained by solving the above min-cost flow problem may result in some GPU-to-GPU demands completing very quickly. However, since reconfiguration incurs delay (e.g., 25 μs in our prototype), we cannot reconfigure wavelengths too quickly without hurting efficiency (more on this below). Therefore we should plan the wavelength allocation based on a time horizon rather than looking only at the instantaneous traffic demands. To this end, we iteratively solve the min-cost flow problem in Equation (2), serving the TM with step-size of Δ based on the flows obtained after each iteration, and repeating this procedure until there is no unserved demand left in the TM. We compute the mean of the flow allocations over all iterations as the final flow allocation.

Step 4: Mapping flows to bandwidth. Finally, we scale the flows from the previous step by W and map them to integer numbers using a technique called randomized rounding [148]. This produces the final compute and bandwidth allocation. An important consideration in SiP-ML's design is how frequently to reschedule the bandwidth allocations. By rescheduling frequently, we can better tailor the bandwidth allocation to meet the traffic demands. But rescheduling too quickly is undesirable, because each reconfiguration incurs a delay during which no traffic can be sent. In our experiments, we found setting the rescheduling period to 100 μs (4× the reconfiguration delay) provides the best performance.

A.2 SiP-OCS ILP

Similar to §A.1, we assume TM_{ij} denotes the estimated traffic matrix between GPUs i and j. We have N GPUs and Q OCSs each with N ports. There is B/Q bandwidth available between each GPU and OCS. Let $P \in \{0,1\}^{N\times N\times Q}$ stand for the permutation configuration of all OCSs with $P_{ijk} = 1$ if there is a circuit between GPUs i and j on OCS k. Therefore, the total available bandwidth between GPUs i and j would be: $(B/Q)\left(\sum_{k=1}^{Q} P_{ijk}\right)$. Our circuit scheduling goal can be expressed as an Integer Linear Program (ILP) by maximizing the minimum inverse of the completion time, as follows:

$$\begin{array}{llll} \text{maximize}_{P\in\{0,1\}^{N\times N\times Q}} & \min\limits_{ij:TM_{ij}>0} & \sum_k P_{ijk}/TM_{ij} & \\ & \text{s.t.} & & \\ & (1) \quad \sum_i P_{ijk} & \leq \quad 1 & \forall j,k \\ & (2) \quad \sum_j P_{ijk} & \leq \quad 1 & \forall i,k \end{array} \tag{3}$$

where constraints (1) and (2) would enforce the OCS configurations to be in the form of a permutation for each OCS; i.e., each GPU can establish a circuit with only one other GPU on each OCS. For commercial OCSs that have orders of magnitude higher reconfiguration delay than MRRs, we only use one-shot configuration. For such configurations, our experiments show ILPs can be solved reasonably fast enough for thousands of nodes. Note that with one-time scheduling, this optimization happens only once at the beginning of training each new workload.

A.3 Scaling Efficiency of the Placement

In Fig. 14, we compare the scaling efficiency of SiP-ML's placement algorithm on 1024 GPUs to the efficiency achieved in the most recent version of the MLPerf training benchmark [88]. We highlight the following takeaways: 1) workloads like ResNet50 are too small to be efficiently scaled to 1000s of GPUs; 2) our placement generalizes to electrical topologies without degree constraint; 3) placement with optical degree constraints respects the compute efficiency in addition to interconnect constraints; 4) overall, SiP-ML achieves up to 4.3× better scaling efficiency than today's expert-designed parallelization strategies for clusters in MLPerf benchmark.

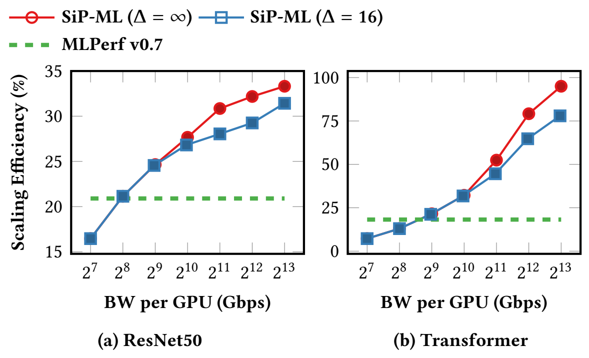


Figure 14: Comparing the scaling efficiency of our placement algorithm at different bandwidths to state-of-the-art expert designed placements in MLPerf benchmark for 1024 GPUs.

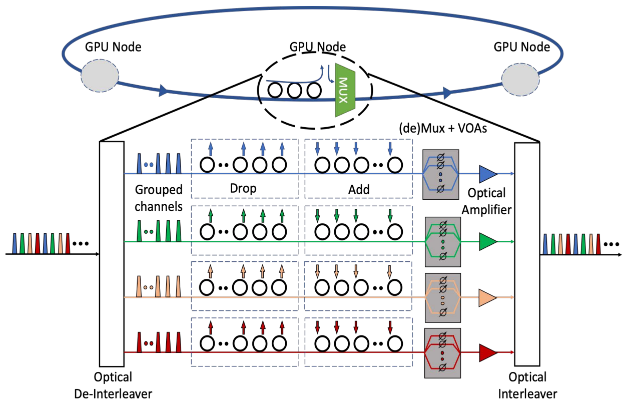


Figure 15: System level diagram of GPU nodes with scalable SiP select/bypass interface. The incoming 64 wavelengths are separated into four groups with 16 wavelengths each for select/bypass.

A.4 Optical Simulations

Fig. 15 demonstrates our approach to achieve SiP interfaced GPU nodes at large scale. Every WDM input of 64 wavelengths from the previous GPU node is first de-interleaved into 4 groups with 16 wavelengths each. We use cascaded SiP micro-ring filters to perform wavelength selective add/drop or to pass wavelength(s) through the node based on the requirement of global scheduler. To overcome the spectral power variability caused by the multi-staged optical components, we add optical amplifiers, optical (de)multiplexers and variable optical attenuators (VOAs) to equalize the optical power for each wavelength at the output of the GPU node. An interleaver then combines all 4 groups and forwards the new WDM signal to the next GPU node. We simulate our SiP add/drop interface using the American Institute for Manufacturing Integrated Photonics (AIM

Photonics) process design kit (PDK) in OptSim software. The add/-drop filters are from the AIM PDK and the (de)interleavers are built with cascaded 2-stage MZI. The optical multiplexer/demultiplexers are designed using ideal OptSim models with a bandwidth of 0.5nm. The multiplexer/demultiplexer function can also be implemented with multimode interference (MMI) couplers. In the simulation, we achieve an equalized optical spectrum at the output of a GPU node for two cases: 1) 64 bypass wavelengths; 2) 64 wavelengths with 32 wavelengths being dropped and added, while the other 32 wavelengths bypassing the node.