
Mehrdad Khani1  Manya Ghobadi1  Mohammad Alizadeh1  Ziyi Zhu2  Madeleine Glick2  Keren Bergman2  Amin Vahdat3  Benjamin Klenk4  Eiman Ebrahimi4
1 MIT  2 Columbia University  3 Google  4 NVIDIA

Abstract

We explore a novel, silicon photonics-based approach to build a high bandwidth rack designed for machine learning training. Our goal is to scale state-of-the-art ML training platforms, such as NVIDIA’s DGX and Intel’s Gaudi, from a handful of GPUs in one platform to 256 GPUs in a rack while maintaining Tbpps communication bandwidth. Our design, called TeraRack, leverages the emergence of silicon photonics technology to achieve Tbpps bandwidth in/out of the GPU chip. TeraRack enables accelerating the training time of popular ML models using (i) a scheduling algorithm that finds the best wavelength allocation to maximize the throughput between communicating nodes; and (ii) a device placement algorithm that partitions ML models across nodes to ensure a sparse and local communication pattern that can be supported efficiently on the interconnect. We build a small prototype with FPGA boards and a 10 mm × 10 mm silicon photonics chip. Simulation results show that TeraRack’s performance on realistic ML training workloads is equivalent to a full-bisection 256×1.2Tbps electrical fabric at 6× lower cost, enabling faster model/data parallel training.

1 INTRODUCTION

Today’s approach to assembling custom platforms for machine learning (ML) training—electrically interconnecting a few GPUs within one box—has a fundamental scaling challenge. The number of compute nodes in the box is limited by the wiring complexity of the interconnect between the nodes. For instance, Nvidia’s DGX-2 [1], one of today’s fastest ML training platforms, offers 1.2 Tbps communication bandwidth between GPU pairs at all times, but it is limited to interconnecting 16 GPUs using a custom-designed network fabric (NVSwitch [2]). Similarly, Intel’s Gaudi training platform [3] interconnects 8 Tensor Processing Cores using an on-chip wiring mesh that provides 1 Tbps bandwidth for every pair [4].

Many researchers have recognized this cliff in scaling from a handful of accelerators to sizes needed by large ML training workloads. There are a number of proposals to scale across HPC-like clusters, such as IBM Summit [5, 6], with thousands of commodity or even customized servers [7–12]. However, these clusters have limited server-to-server bandwidth, making the network a bottleneck for large DNN training [13–17]. The reason is that as the number of parallel workers increases, the amount of computation at each worker decreases but the amount of communication remains relatively constant (since it depends primarily on model size [11, 12, 18, 19]). Consequently, the total training time becomes dominated by the communication time, resulting in diminishing returns with additional workers. Data-parallel training methods aim to mitigate this problem by processing more data in each training iteration as the system scales, thereby keeping computation and communication time balanced. However, this approach also hits scaling limits, since processing more data in each training iteration does not always translate to faster training [20]. Further, recent work has shown that other parallelization strategies with higher bandwidth requirements, such as model parallelism or hybrid model and data parallelism, can significantly improve performance [21]. We posit that the community will soon require 100s of accelerators for single training workloads and such deployments will require Tbps bandwidth per accelerator to maintain a scalable, cost-balanced system.

Figure 1 compares the bandwidth and scale of existing ML platforms. Custom platforms [1, 4, 22–24] have high per-node bandwidth but are limited in scale, whereas HPC-like clusters [5, 27–30] can scale to thousands of nodes, but the training speed suffers from limited per-node bandwidth.
TeraPHY chip [32–36], present a unique opportunity to design interconnects with Tb/s bandwidth.

Silicon photonics have been proposed as a potential game changer for ML-based systems [37–39], with a projected cost of $1–$5/Gbps, thus matching the cost of today’s datacenter interconnects [40–42]. Micro-ring resonators (MRRs) [43] play an important role in the success of silicon photonics [44, 45]. MRRs act as spectral filters to select and forward wavelengths, and they can switch between different wavelengths within a few microseconds. Prior work shows the feasibility of MRRs in optical switches and transceivers [34, 46–59]. In this work, we explore the use of MRRs to build rack-scale interconnects.

An immediate challenge of optical interconnects is that they tend to have either a millisecond reconfiguration time [60–63], or a limited port count [55, 64–66]. For instance, the Calient switch has 320 ports, but it takes roughly 30 ms to reconfigure the switch [67]. Recent proposals, such as Mordia [64], introduce microsecond switching time using wavelength-selective switches, but they are inherently limited to 4–8 ports. Low port count limits the scalability, and slow reconfiguration time hurts the performance of training (§2). We overcome this challenge by eliminating the need for an optical switch to interconnect the nodes in TeraRack altogether. Instead, we connect the MRRs on each node with fiber cables to form several rings-of-micro-rings and reconfigure the wavelengths on each ring on demand.

Although a ring topology eliminates the need for an optical switch, it introduces a well-known challenge: wavelength assignment [68]. The wavelength assignment problem is to select a suitable wavelengths, among the many possible choices for each communicating node, such that no two paths sharing the same fiber are assigned the same wavelength. We develop a wavelength scheduling algorithm that exploits a connection with min-cost flow routing. Our algorithm provides a near-optimal solution, achieving training times that are within less than 5% of an integer linear programming (ILP) wavelength allocation solution.

To evaluate TeraRack, we develop an accurate simulator for parallel neural network training. Our simulations compare the cost and training iteration time of TeraRack with those of the state-of-the-art electrical and optical architectures for data/model parallelism workloads. Our results demonstrate two key takeaways. First, for three representative DNN models (ResNet152 [69], VGG16 [70], Transformer [71]), TeraRack speeds up training time by factors of 2.12–4.09× and 1.65–3× compared to cost-equivalent electrical-switch and OCS-based interconnects respectively, at 256-GPU scale. Second, TeraRack achieves up to 55× speedup in total training time from fast (rescheduling allocation once every 100µs) wavelength reuse at 256 GPUs. Finally, based on component costs, we estimate that a TeraRack interconnect will be an order of magnitude cheaper than a full-bisection bandwidth electrical interconnect.

We also build a small, three-node prototype using FPGA boards to emulate GPUs and three MRRs to show the feasibility of our architecture. We use Stratix V FPGAs and commodity 10 Gbps SFP+ transceivers to emulate the GPU training workflow, as no commercial GPU chip currently supports high bandwidth optical interfaces.2 We use the MRRs to select and forward any of the wavelengths to the target emulated GPUs. Using our prototype, we benchmark the throughput, reconfiguration time, and optical power budget of TeraRack’s architecture. Our prototype demonstrates that (i) using MRRs to select/bypass traffic does not impact TCP’s throughput; (ii) MRR’s reconfiguration latency is within 25 µs; and (iii) the optical power loss of each MRR is 0.025–0.05 dB.

2 BACKGROUND AND MOTIVATION

A common approach to distributed training is data parallelism (DP) where the training data is distributed across multiple workers (e.g., GPU, TPU, CPU). Each worker has an identical copy of the model but trains on an independent subset of the training data, called a mini-batch, in parallel. In DP training, workers need to communicate their model parameters after each training iteration. This can be done in a variety of ways, including parameter servers [73], ring-allreduce [74, 75], tree-reduce [76], and hierarchical all-reduce [8].

In addition to DP training, there is a rapid increase in model parallelism (MP) training, motivated by the rapid increase in the computation and memory requirements of neural network training. The size of deep learning models has been doubling about every 3.5 months [9, 14, 77]. Many models, such as Google’s Neural Machine Translation [78, 79] and Nvidia’s Megatron [80], no longer fit on a single device [81–83] and need to be distributed across multiple GPUs [84]. To train such models, model parallelism (and hybrid data-model parallelism) approaches partition the model and data across different workers [85]. Model parallelism is an active area of research, with various model partitioning techniques [13, 14, 81–86]. For example, pipeline parallel approaches, such as PipeDream [13] and GPipe [14], have emerged as a sub-area of model parallelism. Recent work [21] explores optimizing over a large space of fine-grained parallelization strategies (e.g., parallelizing each operator in a DNN computation graph separately), demonstrating an increase in training throughput of up to 3.8×. We posit that all of these approaches will benefit from a platform with high network bandwidth.

2.1 Need for Tb/s Per-node Bandwidth

We profile traffic traces from deep neural networks distributed across a DGX-1V system [24] to motivate the need for a rack with 256 nodes and 2.4 Tb/s per-node bandwidth, or 615.4 Tb/s of total interconnect bandwidth. To put this into context, the interconnect bandwidth of today’s commodity electrical packet switches is 12.8 Tb/s (128 ports, each 100 Gbps); 48 times slower than our proposal.

Our DGX system contains eight GPUs connected by NVLinks with 1.2 Tb/s bandwidth per GPU; each GPU has six NVLinks

1Note that this is far cheaper than wide-area networks’ DWDM technology (details in Section 4).

2The first chips are expected to hit the market later this year [72].
The intuition behind the periodicity is that each training iteration repeats the same pattern of computation dependencies. To quantify this observation, we calculate the autocorrelation of time-series representing the bandwidth demand between GPU pairs. These measurements illuminate the nature of ML workloads and provide a general framework to evaluate TeraRack. For reproducibility, our traffic trace data and GPU instrumentation code will be publicly available.

Fig. 2 shows the GPU-to-GPU bandwidth utilization for three training networks (Transformer [71], VGG [70], and GPT2 [91]) as examples of data, model, and hybrid data-model parallel workloads, respectively. We run the Transformer model in data parallel mode on our DGX-1 box across all eight GPUs, the VGG model on four GPUs in a 4-way model parallel mode, and the GPT2 model on eight GPUs, with a hybrid 4-way model and 2-way data parallelism [84]. The figure shows that the throughput can hit the peak capacity of ≈1.2 Tbps. Moreover, we find that the throughput is periodic. The intuition behind the periodicity is that each training iteration repeats the same pattern of computation dependencies. To quantify this observation, we calculate the autocorrelation of each workload and find the autocorrelation peaks correspond to training iterations (figure shown in Appendix A). Prior work has reported a periodic behavior for GPU-to-memory transfers [92], our measurements confirm their findings but for GPU-to-GPU traffic.

To understand the growing need for Tbps bandwidth for each node, we now provide the intuition on the bandwidth requirements of distributed training tasks from first principles. In a balanced parallel training system, the communication time and computation time should be roughly equal, with near-perfect overlap. The advent of new hardware architectures, such as Cerebras [93] and Google’s TPU [94], along with improvements in software stacks, are leading to a continuous decrease in compute time. Hence, bandwidth has to increase to avoid making the network a bottleneck. In data-parallel training, this effect can be partially compensated by increasing the amount of data that each GPU trains (i.e., local batch size). However, there is a limit on decreasing the total training time with increasing the batch size [20]. Importantly, model-parallel training cannot leverage this technique since the size of the communication data is proportional to the batch size.

As an example, we calculate the required bandwidth for data parallelism with the Transformer model [90]. Today, the batch size used for Transformer at the largest scale is 1280. We measure the backward pass as 23.5 ms on a V100 GPU, during which 420 MB need to be all-reduced. To achieve maximal overlap, the system needs to provide an effective all-reduce bandwidth of about $\frac{420 \text{MB}}{23.5 \text{ms}} \approx 150 \text{ Gbps per node}$. Hence, as we increase the system scale or improve the processor’s performance through hardware or software, the required bandwidth grows rapidly. For example, a 5x processor improvement and a 2x larger system requires an effective bandwidth of about 1.36 Tbps per GPU for Transformer model.

### 2.2 The Solution: An Optical Interconnect with μs Reconfigurability

The previous section establishes the need for Tbps per node bandwidth. This section discusses the desired properties of an interconnect to achieve such high bandwidth at moderate scale (e.g., 256 nodes).

**The strawperson design.** An interconnect with electrical packet switches is the first design option to consider. For instance, the DGX-2 fabric uses 12 NVSwitch chips [2] to interconnect 16 GPUs. An NVSwitch offers the highest on-chip bandwidth, with 18 ports of 200 Gbps capacity (3.6 Tbps total bandwidth). However, scaling this fabric to 256 GPUs in one densely connected box would require a large number of these switching chips arranged in a multi-layer topology. For instance, the standard fat-tree topology [95] providing 2.4 Tbps of bandwidth to 256 GPUs would require 960 NVSwitches and 9216 NVLinks. The cost of such an interconnect is 10x higher than TeraRack (see Section 4 for detailed cost comparison). However, using commodity electrical packet switches to interconnect 16 DGX-2 boxes (each with 16 GPUs) does not give the desired per-GPU bandwidth: each DGX box can provide up to 800 Gbps bandwidth shared across 16 GPUs (50 Gbps per GPU). Section 5 demonstrates that such a design cannot support the high bandwidth requirement of hybrid data-model parallel jobs.
Optical interconnects. Given the limited bandwidth provided by electrical fabrics, another attractive solution is a circuit-based optical interconnect. At first glance, it appears that standard optical interconnects fit the bill, as ML training consists of thousands of iterations with the same communication pattern, making it suitable for long-lasting optical circuits. An important consideration is that with a 256 port-count, MEMS-based optical switches have a reconfiguration latency of 30 ms. As a result, they are suitable only for circuits that can last for several hundreds of milliseconds, such as large data center flows. Although ML jobs last many hours, and the communication pattern of ML jobs does not change between training iterations, the optical circuits may have to be reconfigured within iterations. Consequently, the iteration time plays an important role in determining potential circuit reconfiguration times. For instance, a circuit-based optical interconnect with a 30 ms reconfiguration time is not ideal if the iteration time is a few milliseconds: reconfiguring the circuits during the iteration will bloat the total training time significantly. In fact, our evaluations (§5) show that all of model parallel training iterations require circuit reconfiguration within the iteration.

Estimating the total iteration time. The computation time plays an important role in determining the training iteration time. Section 2.1 shows that scaling the number of nodes or improving the compute capabilities of each node will result in a decrease in the computation time per node. To prevent the network from becoming a bottleneck, the communication time (which depends on the size of messages transferred between nodes) should overlap entirely with the computation time, making it a suitable proxy to calculate the lower bound on training iteration time for each node. Fig. 3 shows the cumulative distribution function of transfer sizes during iterations for different workloads, including ResNet50, ResNet152, VGG16, InceptionV3, GPT2, and Transformer. We train the Transformer model in 8-way data-parallel mode and ResNet, VGG16, InceptionV3 are 4-way model parallel. GPT2 is trained with a mix of 2-way data and 4-way model parallelism. In most cases, the transfer sizes vary from 1 MB to 1 GB. For instance, the median transfer size of Transformer is about 200 MB, or 1.6 ms on a 1 Tbps link. This means reconfiguring circuits for this workload requires an optical interconnect with a circuit reconfiguration time shorter than 160 μs to ensure the reconfiguration latency is a small fraction (<10%) of the circuit-hold time.

2.3 Prior Proposals Are Insufficient

In this section, we briefly summarize why prior proposals fall short in supporting the above requirements of Tbps bandwidth and μs reconfiguration time simultaneously. Sections 4 and 5 quantitatively compare the cost and performance of TeraRack with relevant prior work.

Static topologies. Fat-tree [95], Dragonfly [96, 97], SlimFly [98], Diamond [99], WaveCube [100], and Quartz [101] are all examples of static topologies proposed for datacenters. However, increasing the bandwidth of static topologies to support Tbps bandwidth is prohibitively expensive and requires non-trivial cabling and topology changes. For example, in Section 4, we show the cost of a static topology to support the scale and bandwidth of TeraRack is 10× larger than the projected cost of TeraRack.

Traffic oblivious topologies. Jellyfish [102], Rotornet [103], and Opera [104] take advantage of the unpredictability of datacenter workloads and use expander-based topologies to improve the flow completion time of short and long flows. Random permutations are not ideal for ML workloads, however, as a training workload is a periodic repetition of thousands of iterations.

3D MEMS-based technologies. Helios [61], c-Through [62], and OSA [63] use commercialized high port count optical switches with \( \approx 30 \) ms reconfiguration latency. These proposals might be suitable for data parallel workloads using the ring-reduce algorithm, but they are inefficient to support tree-reduce, hierarchical all-reduce [8], and hybrid data-model parallel training jobs. Section 5 evaluates the impact of reconfiguration delay on ML training time.

2D MEMS-based technologies. The lower port count 2D MEMS is often a component of a Wavelength Selective Switch (WSS) which, in combination with a wavelength selective element such as a grating, has been used as a microsecond reconfigurable multiplexer in Mordia [64] and REACToR [65]. However, such designs have limited port count. Megaswitch [60] reveals that as WSS port count increases, the reconfiguration time changes from 11.5 μs to 3 ms, making such proposals unsuitable for our needs. Moreover, each Megaswitch node broadcasts its wavelengths on a dedicated fiber, thus limiting the number of nodes in the topology to 30. In contrast, TeraRack can support up to 256 nodes.

Free-space interconnects. ProjecToR [105] and FireFly [106] have high port count and microsecond reconfiguration latency, but they use free-space optical links which have alignment challenges and are sensitive to environmental vibrations.

Nanosecond switching fabrics. Sheol [107], Larry [108] and XFabric [109] have proposed reconfigurable datacenter interconnects with nanosecond switching fabric. Similarly, Sirius [25] and WS-TDMA [26, 110–112] have demonstrated the feasibility of sub-nanosecond wavelength switching using fast tunable lasers and arrayed waveguide grating routers (AWGRs). We believe these proposals have the potential to become widely deployed in datacenter environments, but these fabrics do not support Tbps bandwidth between communicating nodes.
Fig. 4 illustrates the components in a TeraRack node. The capacity of each interface is 600 Gbps, achieving a total full-duplex bandwidth of 2.4 Tbps per node. On the send side, a second array of MRRs selects the wavelengths to connect the four interfaces of each node (two clock-wise rings and two counter clock-wise). The figure shows one ring in each direction for clarity.

**Wavelength scheduling.** One of the core properties of TeraRack is the ability to dynamically place wavelengths around the fiber ring, to maximize the throughput between communicating nodes. The optimal wavelength allocation maximizes the throughput while ensuring that no two paths sharing the same fiber are assigned the same wavelength. More formally, the wavelength allocation problem corresponds to the following optimization problem. Let $T_{Mij}$ denote the predicted GPU-to-GPU traffic matrix, and $\lambda^t$ denote the total number of wavelengths. We can represent a wavelength allocation as a 3-dimensional binary matrix, $\Lambda$, where $\Lambda_{ijk}$ is 1 if GPU $i$ sends data to GPU $j$ using $\lambda_k$ 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 $\sum_{\lambda_k} \Lambda_{ijk} T_{Mij}$. This can be expressed as an Integer Linear Program (ILP) by maximizing the minimum inverse of the completion time, as follows:

$$\text{maximize}_{\Lambda \in \{0,1\}^{N \times N \times \lambda^t}} \min_{ij : T_{Mij} > 0} \sum_{\lambda_k} \Lambda_{ijk} / T_{Mij}$$  \hspace{1cm} (1)

The full formulation is presented in Appendix D. The constraints are (1) ensure fiber segments do not contain overlapping wavelengths (ring constraint) and (2) ensure that each GPU can use each wavelength for communication with at most one other GPU (node constraint).

Moreover, building a control plane with nanosecond response time is overkill for our use-case.

**3 TERARACK DESIGN**

TeraRack interconnects 256 compute nodes and is tailored for the communication patterns and bandwidth requirements of ML workloads.

**TeraRack node.** Fig. 4a illustrates the components in a TeraRack node designed based on TeraPHY silicon photonics technology [32, pages 25–28]. Each node in TeraRack has four optical interfaces [32, 34–36]. Note that the optical interfaces do not need the expensive and bulky DWDM transceivers used in wide-area networks. Instead, each interface can be connected to other nodes via an MTP fiber connector [113]. The capacity of each interface is 600 Gbps, achieving a total full-duplex bandwidth of 2.4 Tbps per node. On the send side, an off-chip comb laser generates light that is steered into the node via a fiber coupler, toward an array of MRRs modulating the GPU’s transmitting data at 25 Gbps per wavelength. On the receive side, a second array of MRRs selects the wavelengths targeted to the GPU and passes through the remaining wavelengths. Each interface has 24 micro-ring resonators to select and forward any sub-set of 24 wavelengths. Section 6 measures the reconfiguration latency of MRRs in our testbed at 25 μs. Our laser technology is based on a comb laser source, called SuperNova Light Supply [114], that produces a spectrum consisting of multiple equidistant frequencies. At its simplest, an optical comb can be used to replace an array of independent lasers, like those of a dense wavelength division multiplexed (DWDM) communications system. The SuperNova laser source is capable of supplying light for 256 wavelengths.

**TeraRack topology.** TeraRack’s design does not require an optical switch to interconnect its nodes. Instead, we use fiber rings to interconnect the micro-rings on each node. Fig. 4b shows the high-level view of TeraRack’s topology. Data plane traffic is distributed across four single-mode fiber rings that connect the four interfaces of each node (two clock-wise rings and two counter clock-wise). The figure shows one ring in each direction for clarity.

**Figure 4: TeraRack’s design.**

![TeraRack node](image1)

![Topology with 256 nodes](image2)

![Figure 4: TeraRack’s design.](image3)

![Wavelength allocation and its equivalent flow routing translation.](image4)

![Wavelength allocation and its equivalent flow routing translation.](image5)
The FlowRing algorithm. Our approach is inspired by prior work on formulating wavelength assignment in optical networks as flow routing problems [68, 115–119]. To develop intuition, consider the example in Fig. 5a with four TeraRack nodes placed around a fiber ring. The arrows represent the traffic demand between nodes and the colors on each arrow shows one possible wavelength allocation. Notice the reuse of wavelengths on non-overlapping ring segments. For example, wavelength $R$ is used to satisfy three demands on three different ring segments, and wavelengths $G$ and $B$ are used to satisfy two different demands. Also notice that, on each ring segment, a wavelength is only assigned to a single demand.

We can construct a flow routing problem corresponding to this example as shown in Fig. 5b. We cut the ring at Node$_1$, and then divide Node$_1$ into two nodes: a source and a sink. The source node injects a continuous flow of size 1 units into the ring and the sink node absorbs a flow of 1 unit. As the flow circles the graph between the source and the sink, it gets split across different outgoing edges at each node. These outgoing edges correspond to traffic demands out of that node, and the fraction of the flow on each edge corresponds to the fraction of wavelengths assigned to carry that demand. For instance, in Fig. 5b, the flow is split at Node$_2$, with 1/3 going to the Node$_2$→Node$_3$ edge, and 2/3 going to the Node$_2$→Node$_4$ edge. With $W = 3$ total wavelengths, this corresponds to the allocation in Fig. 5a.

A standard property of a valid flow routing is “flow conservation,” i.e., the fact that the total flow into any node must be the same as the total flow out of that node (except for the source and sink nodes). In our setting, the implication of flow conservation for wavelengths is that when a wavelength is dropped at a GPU, it must be added back to the fiber ring on the next segment (provided there is any demand between that GPU and its subsequent GPUs). Flow conservation enforces the invariant that the total flow across a cut at any segment of the ring has size 1. This invariant captures the idea that, in any throughput-maximizing allocation, each ring segment must carry all wavelengths if there is traffic demand between GPUs that come before and after that segment.

Our FlowRing algorithm involves the following steps:

**Step 1: Graph construction.** We construct a directed graph, $G = (V, E)$, where $V$ is the set of nodes on the ring 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 ring is connected in $G$. If not, we add a "dummy" edge between them to $E$. For instance, in Fig. 5b, the dotted line between Node$_4$ and Node$_1$ is a dummy edge, since it does not correspond to a demand in the traffic matrix. The direction of all edges in $G$ is the same as that of wave propagation on the fiber. We then add the sink and source nodes by cutting an 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 [121, 122] and there exist a variety of combinatorial network simplex algorithms [120]. 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 (details in Appendix B).

**Step 3: Remove and repeat.** The solution obtained by solving the above min-cost flow routing 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 $\Delta$ 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 wavelengths.** Finally, we scale the flows from the previous step by $W$ and then map them to integer numbers using a technique called randomized rounding [123]. This produces the final wavelength allocation.

**Scheduling frequency.** An important consideration in TeraRack’s design is how frequently to reschedule the wavelength allocations. By rescheduling frequently, we can tailor the wavelength allocation better to the traffic demands. But rescheduling too quickly is also undesirable because each reconfiguration would

---

3We solve separate instances of the flow routing problem for the clockwise and counter-clockwise rings, including each demand on the ring for which it needs the fewest hops.
incurs a delay during which no traffic can be sent. In our experiments, we found that setting the rescheduling period to 100 µs (4× the reconfiguration delay) provides the best performance.

**Device placement.** Splitting a DNN’s computation graph across multiple devices is non-trivial in the case of model/pipeline parallelism due to the need to balance computation and memory usage while maintaining low communication between devices [13, 14, 21, 124, 125], TeraRack can use any existing model partitioning algorithm, but we present a simple heuristic in order to split a computation graph. The goal of this heuristic is to try to keep the communication between GPUs both sparse (i.e., a GPU communicates with few other GPUs) and local (i.e., a GPU communicates with nearby GPUs along the ring). Given the run-time of each operation, we first split the large operations across the batch dimension into equal-sized parallel operations following a similar approach to [126]. We then topologically sort the resulting computation graph. We traverse the operations in this sort order, placing each operation on the next GPU in round-robin manner. The motivation for this algorithm is that DNN computation graphs often have long chains of operations (with each operation sending data only to the next operation on the chain). Now consider two operations $i$ and $i+1$ along such a chain, and suppose that operation $i$ is assigned to GPU $j$. Then the above algorithm is likely to assign operation $i+1$ to GPU $j+1$. Hence most communication will be between adjacent GPU pairs along the ring (or more generally, nearby GPU pairs). This reduces the interference between GPU-GPU communications, enabling more wavelength reuse for TeraRack.

## 4 COST OF TERARACK

We estimate the cost of a TeraRack interconnect with 256 nodes and 2.4 Tbps bandwidth per node and compare it with the cost of a scaled DGX fabric (ElectNet), an OCS-based interconnect (OcsNet), and Megaswitch [60], an optical ring interconnect. The cost of Mordia [64], Quartz [101], and OSA [63] fabrics are higher than that of Megaswitch, hence we omit them in our cost comparison. Table 1 compares the cost of these interconnects.

### Cost of ElectNet fabric.

We posit that a fully electrical fabric comparable with TeraRack is an on-chip scaled DGX platform with 256 GPUs and 2.4 Tbps bandwidth per GPU. We reverse engineer the cost of today’s DGX-2 and use the components’ cost to estimate the cost of ElectNet fabric. The retail price of a commercial DGX-2 box with 16 V100 GPUs and 12 NVSwitches (each with 18 NVLink ports) is $400K [127]. We assume that the actual cost of a DGX-2 fabric is $100k (a profit margin of $300K). We estimate the cost of a 200 Gbps NVLink is $100; 10× lower than the price of a commodity electrical transceivers [128]) and the cost of a V100 GPU is $1000 (8× lower than the price tag on Amazon [129]). As a result, we estimate the cost of an NVSwitch is $5,200. To scale the current fabric to 256 GPUs, we scale the DGX into an on-chip fat-tree interconnect with $k=16$ using NVSwitches and NVLinks. A fat-tree with $k=16$ can interconnect 1024 nodes, however, given that the bandwidth of NVLinks is only 200 Gbps, we bundle every four ports together to achieve 800 Gbps per-GPU bandwidth. To further scale the per-node bandwidth to 2.4 Tbps, we replicate the same fat-tree interconnect three times. The resulting interconnect has 960 NVSwitches interconnected with 9216 NVLinks.

### Cost of OcsNet.

To interconnect 256 GPUs optically, we design an OCS-based fabric following the state-of-the-art optical proposals [61, 62, 103, 104] using 12 OCS switches. We interconnect the GPUs to each OCS using 200 Gbps optical transceivers to achieve 2.4 Tbps per GPU bandwidth. The cost of the OCS and transceivers is obtained from online quotes [130, 131]. We estimate the cost of fiber cables based on $0.44/m [132] and average length of 10 meters.

### Cost of Megaswitch.

The required components for Megaswitch are obtained from the paper [60, Table 1]. We note that the current design of Megaswitch does not scale beyond 32 nodes however, we conservatively assume the fabric is scalable without requiring new components. The prices are based on quotes we obtained from vendors. The most dominating component in the cost of Megaswitch is the price of DWDM transceivers. We note that there is an important distinction between the cost of commodity DWDM transponders and TeraRack’s SiP-based design. In particular, DWDM transponders are designed to operate at long distances which will impose strict challenges on the laser, manufacturing, forward-error correction, photodiode sensitivity, modulation scheme, as well as light coupling. In contrast, SiP interfaces are designed for short distances, and do not require coherent detection, hence they can take advantage of considerable development and commercialization of photonic components for short distance datacenters. However, even if we replace the WDMD interfaces in Megaswitch with TeraRack’s SiP interface, the price will come down to $30,384M, that is still a factor of 50 higher than that of TeraRack. Intuitively, this is because Megaswitch’s design consumes too

<table>
<thead>
<tr>
<th>Component</th>
<th>Unit Cost($)</th>
<th>Count</th>
<th>Total ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ElectNet</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVLink</td>
<td>100</td>
<td>256</td>
<td>9216</td>
</tr>
<tr>
<td>NVSwitch</td>
<td>5200</td>
<td>1</td>
<td>4992</td>
</tr>
<tr>
<td>Packaging/box</td>
<td>1000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>All together</td>
<td></td>
<td></td>
<td>9,914</td>
</tr>
<tr>
<td><strong>OcsNet</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCS switch</td>
<td>100000</td>
<td>12</td>
<td>1200</td>
</tr>
<tr>
<td>Transceiver</td>
<td>100</td>
<td>3072</td>
<td>3072.2</td>
</tr>
<tr>
<td>Fiber cable</td>
<td>4.4</td>
<td>3072</td>
<td>13.5</td>
</tr>
<tr>
<td>All together</td>
<td></td>
<td></td>
<td>1,520.7</td>
</tr>
<tr>
<td><strong>Megaswitch</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSS</td>
<td>60000</td>
<td>256</td>
<td>15360</td>
</tr>
<tr>
<td>Amplifier</td>
<td>100</td>
<td>256</td>
<td>25.6</td>
</tr>
<tr>
<td>DWDM transceiver</td>
<td>100000</td>
<td>24832</td>
<td>248320</td>
</tr>
<tr>
<td>All together</td>
<td></td>
<td></td>
<td>263,705</td>
</tr>
<tr>
<td><strong>TeraRack</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SiP interface</td>
<td>441</td>
<td>1024</td>
<td>451.6</td>
</tr>
<tr>
<td>Electrical circuits</td>
<td>308</td>
<td>256</td>
<td>78.8</td>
</tr>
<tr>
<td>SiP amplifier</td>
<td>100</td>
<td>256</td>
<td>25.6</td>
</tr>
<tr>
<td>Comb laser</td>
<td>100</td>
<td>256</td>
<td>25.6</td>
</tr>
<tr>
<td>Packaging</td>
<td>30</td>
<td>256</td>
<td>7.7</td>
</tr>
<tr>
<td>Single-mode fiber</td>
<td>3</td>
<td>1024</td>
<td>3.1</td>
</tr>
<tr>
<td>PCI-E slot</td>
<td>45</td>
<td>256</td>
<td>11.5</td>
</tr>
<tr>
<td>All together</td>
<td></td>
<td></td>
<td>604</td>
</tr>
</tbody>
</table>

Table 1: Component and interconnect costs of TeraRack and its electrical and optical counterparts. The table excludes the cost of GPUs, assuming it is roughly the same for all fabrics.
many interfaces and does not take advantage of wavelength reuse, a key point in TeraRack’s design (§3).

Cost of TeraRack. A TeraRack node consists of four TeraPHY SiP interfaces each with 600 Gbps capacity [32]. The manufacturing cost of a silicon photonics chip depends on the area of the chip. Prior work has built TeraPHY SiP interfaces with size 8.86 mm × 5.5 mm [32, 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 [133]. Hence, assuming the cost will drop by a factor of 10 at mass production, our cost estimation for each SiP interface in TeraRack is $441. Similarly, we use Europractice pricelist to provide the estimated cost for on-chip electrical circuitry based on a 10 mm² chip area [34–36, 134]. The electrical circuitry includes drivers, MRR’s tuning control logic, and CMOS transimpedance amplification (TIA). Our estimates on the cost of on-chip SiP amplifier [135], off-chip comb laser [114], and packaging are based on quotes from manufacturers such as Tyndall [136] and PLCC [137], but their price lists are, unfortunately, confidential. The cost of single-mode fiber to interconnect TeraRack nodes into rings is obtained assuming $3/meter [106]. Finally, to put the rack together, we estimate the cost of PCI-E slots using online sellers [138].

5 EVALUATION

In this section, we quantify TeraRack’s performance and cost with respect to other electrical and optical network interconnects (§5.2), analyze its performance for different parallelization strategies (§5.3), and evaluate its design decisions (§5.4).

Our simulation results show:

- For three representative DNN models, TeraRack speeds up training time by factors of 2.12–4.09× and 1.65–3× compared to cost-equivalent electrical-switch and OCS-based interconnects respectively, at 256-GPU scale. TeraRack’s performance is equivalent to electrical-switch and OCS-based interconnects that cost at least 6× and 4× more respectively.

- For data-parallel training, TeraRack effectively acts as a 256×2.4 Tbps electrical switch. Further, TeraRack’s high bandwidth enables hybrid data and model parallel training strategies that outperform data-parallel-only training by up to 2× at 256-GPU scale.

- TeraRack’s performance strongly depends on its ability to reuse wavelengths: it outperforms a baseline without wavelength reuse by 55× at 256-GPU scale. TeraRack also relies on fast wavelength rescheduling (once every 100 µs) to deliver good performance.

5.1 Methodology and Setup

Simulator. Our simulator models GPUs and the network interconnect. We simulate a TensorFlow application by defining ML computation graphs that are trained on multiple GPUs with data or model parallelism, or a hybrid of the two. We simulate a system with a high degree of communication-computation overlap. Specifically, in order to achieve high utilization for model-parallel workloads, we simulate microbatches as in GPipe [14]. Each device involved in model-parallel training sends the results of its microbatch as soon as it is ready, hence reducing the idle time of dependent workers. The interconnect component simulates our optical ring and SiP interfaces, as well as electrical, full-mesh, and optical circuit switching (OCS) topologies.

To simulate GPU behavior, we profile each kernel in each of our DNN models. We use the Tensorflow graph profiler tool while running each workload on a V100 GPU near its full utilization. We then construct a graph profile for each DNN model that we provide as input to our simulator. Each node in this graph is an operation kernel, labeled with the node’s GPU time, CPU time, peak memory size, output bytes, and the number of floating-point operations required for running the operation. We profile three DNN models: ResNet152, VGG16, and Transformer. We use the ImageNet [139] training specifications for ResNet152 and VGG16, and LM1B [140] for Transformer.

Parallelization strategy search. We develop a simple approach to explore the space of hybrid data and model parallelism techniques and pick the best one for a given DNN model and network interconnect. Our algorithm 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) a curve providing the number of training iterations required to reach desired accuracy as a function of the (global) batch size. The training iterations vs. batch size curves were measured for the three models we consider in [20]. These curves play an important role in determining a good parallelization strategy. In particular, for DP training, we use these curves to determine the impact of increasing batch size (to reduce communication bandwidth requirement, see §2.1) on training time. Given this information, we loop through all possible hybrid DP-MP configurations and use our device placement algorithm (§3) to allocate the operations to devices. 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 simply 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 it considers as part of this process is basic DP (no MP). However, as we show, in many cases DP is not the best strategy for large-scale training (§5.3). Second, the runtime 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 wavelength reconfiguration). However, the full simulator is too slow to use as part of the
search procedure, and we have found that the configurations found by the simple method align well with simulations.

**Compared schemes.** We consider five classes of architectures:

1. **Optimal**: a fully-connected interconnect with 256 GPUs connected in a mesh topology. The bandwidth of each node is 255×2.4 Tbps and we assume zero propagation latency.
2. **ElectNet**: an electrical switch fabric with 256 nodes. We vary the per-port bandwidth between 200 Gbps to 2.4 Tbps to compare the cost and efficiency of different design points. For each bandwidth value, we design a topology with commodity 64×200 Gbps switches as described in §4.
3. **OcsNet**: an optical ring interconnect as described in §4.
4. **Megaswitch**: an optical circuit switch interconnect with 256-port switches as described in §4. We connect each GPU to all the optical switches, similar to state-of-the-art optical interconnect proposals [61, 62, 103, 104]. We vary the number of OCS switches in the interconnect between 4–32, resulting in bandwidth per GPU between 800 Gbps–6.4 Tbps. Since OCS reconfiguration delay of 30 ms is too long compared to the typical training iteration time of our DNN models (<20 ms), we compute the best single-shot circuit schedule for each workload to achieve the best possible performance with the OCS-based interconnects. We find the optimal schedule by solving an Integer Linear Program, assuming complete knowledge of the traffic matrix.
5. **TeraRack**: our design with 256 GPU nodes and 2.4 Tbps bandwidth per-GPU. We assume reconfiguration delay of 25 µs (as measured in §6). We use FlowRing as our scheduling algorithm with rescheduling frequency of once every 100 µs, unless otherwise stated.

### 5.2 Overall System Performance

**Comparison between cost equivalent architectures.** We first compare the total training time of TeraRack with cost-equivalent optical and electrical architectures for different DNN training workloads. Using components costs in Table 1, we construct each interconnect such that the total cost of the interconnect matches that of TeraRack. For ElectNet interconnect, this translates into an electrical fat-tree architecture with 256 nodes and total bandwidth of 51.2 Tbps. For OcsNet interconnect, this translates into an OCS-based interconnect with total bandwidth of 204.8 Tbps. Fig. 6 shows that scaling the number of GPUs reduces the total training time. TeraRack is able to speed up the training time by factors of 2.12–4.09× and 1.65–3× compared to ElectNet and OcsNet, respectively at 256 GPUs scale. In particular, the gains with the Transformer model are more pronounced where TeraRack is within 2× of the Optimal architecture at all scales, while ElectNet and OcsNet are up to 8.7× and 3.5× worse than Optimal. This is because the DNN architecture of the Transformer model is highly amenable to model parallelism and requires very high per-node bandwidth. We will discuss this in more detail later in this section.

**How much more would it cost to match TeraRack’s performance?** We next compare the total training time of the Transformer model with electrical and optical architectures at various price points to find the extra investment needed to match the performance of TeraRack. We use the Transformer model because Fig. 6 shows it is a more challenging DNN for all the architectures and it has the largest model size (2.1 GB). The figure besides each approach indicates cost ratio compared to TeraRack. For instance, ElectNet $3 is an ElectNet architecture that costs 3 times more than TeraRack. We observe that TeraRack’s performance is between ElectNet architecture with 6–12 times higher cost. In other words, to match the performance of TeraRack, operators need to invest in electrical fabrics that are 6–12 times more expensive. Similarly, Fig. 7b shows that the performance of TeraRack matches that of an OcsNet interconnect that is 4 times more expensive.

![Figure 6: TeraRack’s performance at different scales compared to its equivalent cost counterparts.](image)

![Figure 7: Comparison with electrical and optical architectures at various cost points. The model is Transformer.](image)
The training time with and without this feature as the number of GPUs are scaled for the Transformer model. The figure shows that wavelength reuse dramatically improves the performance of TeraRack (a $55\times$ speed up when the number of GPUs is 256). Indeed, without wavelength reuse, performance degrades at larger scale because the total bandwidth across the ring is fixed (96 wavelengths) but the amount of communication increases with scale.

**Traffic locality.** The main enabler of wavelength reuse is the locality in GPU-to-GPU communication achieved by TeraRack’s device placement algorithm. In Fig. 11, we plot the Probability Density Function (PDF) of the number of segments that a GPU-to-GPU flow traverses on the ring. We see that more than 70% of the flows are between GPU pairs that are less than four nodes apart. The maximum distance traversed by a flow in all of our workloads was 16 hops. Another implication of Fig. 11 is that the majority of wavelengths in TeraRack pass less than 3-4 amplifiers on the ring, therefore limiting the impact of amplifiers on signal-to-noise ratio.

**Impact of wavelength rescheduling frequency.** Fig. 10b shows the impact of the wavelength rescheduling frequency on TeraRack’s performance. We observe that holding scheduled circuits for more than 100 $\mu$s can degrade the performance, and training time gets substantially worse as the rescheduling interval reaches a millisecond.

**FlowRing algorithm.** TeraRack’s performance with the FlowRing algorithm is within 5% of its performance using the ILP solution for all three models at 256-GPU scale.
allows us to benchmark the switching time and throughput of SiP-based architecture.

6.2 Prototype Architecture

Fig. 12 shows a photograph of our experimental testbed. We built a three-node prototype of TeraRack 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 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 \(\lambda_1 = 1546.92\) nm, \(\lambda_2 = 1554.94\) nm, and \(\lambda_3 = 1556.55\) nm. Our SiP optical chip consists of six MRRs (we use three of them as shown in Fig. 13) 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.

Fig. 13 illustrates the logical details of our testbed. The physical connections between FPGAs are managed using the control logic of the FPGA to configure the optical SiP chip. Our controller provides the necessary bias signals on the SiP chip to cause a state change in the MRRs, depending on the scheduling decision. Each GPU transmits at a uniquely assigned wavelength, and the wavelengths are multiplexed together, forming the input to the bus waveguide of the SiP chip. Each MRR can be tuned to select and forward any of the input wavelengths. Depending on which wavelengths are selected at each MRR, the SiP device will reconfigure the connectivity between chips.

**Example: programming the MRRs.** Assume in the first configuration, GPU 1 is connected to GPU 2; this means MRR 1 is tuned to select and forward \(\lambda_2\) to GPU 1, while MRR 2 is tuned to select and forward \(\lambda_1\) to GPU 2. For simplicity of the configuration logic, MRR 3 is always tuned to \(\lambda_1\) but is effectively in idle mode, as the optical power of \(\lambda_1\) 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 \(\lambda_1\), while MRR 2 should be detuned from \(\lambda_1\) for the optical power of \(\lambda_1\) to pass through MRR 2 to GPU 3. Note that in this configuration, MRR 3, can remain tuned to \(\lambda_1\).

**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 1546.92 nm is used to perform the throughput measurements for select, bypass and loopback cases. As shown in Fig. 14a, 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 MRR1 and MRR2 in Fig. 13 and change the bias voltage from Config1 to Config2, where MRR1 and MRR2 are tuned into and out of resonance with $\lambda_1$. We switch light between the two photodetectors by applying different bias signals to the SiP chip every 125 $\mu$s. 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. 14b shows the receive signal at the photodetectors. In one case, the signal reaches stable state in approximately 20 $\mu$s, and in another case, it takes only 8.4 $\mu$s. 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 $\mu$s 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 Config1 and Config2 is shown in Fig. 14c. We perform 300 measurements to obtain the distribution, showing that the average switching time to Config1 is 13 $\mu$s and Config2 is 15 $\mu$s. 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 $\mu$s and measure the ideal throughput. The curve in blue in Fig. 14d indicates the switching state from GPU3 to GPU2 lasting the duration set by the experiment; the curve in red indicates the switching from GPU1 to GPU3. As the plot shows, the link can achieve above 90% of the ideal throughput, when the scheduling slot length is 220 $\mu$s. This is because the end-to-end reconfiguration takes only about 20 $\mu$s; hence, having a scheduling slot 10 times larger will result in near optimal throughput.

Power budget. Optical power loss is a key measure for any optical system. Table 2 shows the power loss measured at each component of a SiP interface used in TeraRack nodes. The loss per MRR is negligible (0.025–0.025 per MRR), however coupling the light in and out of each node creates 4 dB loss. This is because each SiP interface has an input and output coupler with 2 dB loss. In addition to coupling loss, waveguide loss on the chip can attenuate the light by 1–2 dB, depending on the length and bends. The fiber connector fixtures outside each node adds another 1 dB from node to node. Overall, the total loss incurred by passing through each SiP interface is 6.6–8.2 dB. Hence, assuming a 20 dB power budget based on transmit power and receiver sensitivity [141], TeraRack requires amplification every other node. At first blush, it appears infeasible to scale TeraRack to 256 nodes on a ring, since amplifiers add non-linear noise and hence a light signal cannot traverse more than 30 amplifiers, limiting the scale of TeraRack to 60 nodes. However, recall in Fig. 11 that the path length in TeraRack is limited to 16 nodes, indicating that although the size of the ring is 256 nodes, our scheduling 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. As a result, TeraRack’s design can scale to 256 nodes since its device placement algorithm (§3) partitions ML models across nodes to ensure a sparse and local communication pattern with path lengths shorter than 15 hops.

<table>
<thead>
<tr>
<th>Component</th>
<th>count</th>
<th>loss per component (dB)</th>
<th>total loss (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRR</td>
<td>24</td>
<td>0.025–0.05</td>
<td>0.6–1.2</td>
</tr>
<tr>
<td>Coupler</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Waveguide on chip</td>
<td>1</td>
<td>1–2</td>
<td>1–2</td>
</tr>
<tr>
<td>Fiber connector</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>End-to-end</td>
<td></td>
<td></td>
<td>6.6–8.2</td>
</tr>
</tbody>
</table>

Table 2: Optical loss measured at each SiP interface.
This paper investigated a silicon photonics-based approach for building a 256×2.4Tbps interconnect for a rack of 256 GPUs tailored for parallel machine learning training. We use a small prototype to evaluate the design choices of TeraRack. Using large scale simulations, we showed how our design leverages the microsecond reconfiguration time of MRRs in addition to effective dynamic reuse of wavelengths in the interconnect to achieve high performance on three typical DNN training workloads. TeraRack’s performance is equivalent to a 256×2.1×7 Tbps electrical fabric with at least 6× lower cost. Compared to optical interconnects, TeraRack has 4× lower cost.

This work does not raise any ethical issues.

REFERENCES


Wavecube: A scalable, fault-tolerant, high-performance optical network design and implementation (NSDI 18)

Alex C. Snoeren, and George Porter. Expanding across time to deliver high performance computing (INFOCOM)


NVIDIA unveils 350lb ‘DGX-2’ supercomputer with a $399,000 price tag, 2018.


0.5m (2ft) generic compatible 200g qsfp56 passive. http://www.tarluz.com/products.

A technical/mtp-fiber-connector-guide.


0.5m (2ft) generic compatible 200g qsfp56 passive. http://www.tarluz.com/products.

A technical/mtp-fiber-connector-guide.


Figure 15: Throughput autocorrelation: Peer-to-peer communications between each two GPUs is predictable and repetitive at training time-scale. Peak-to-Peak horizontal distance is equivalent of the training time per iteration.

Figure 16: Wavelength allocation and its equivalent flow routing translation for multiedge cut.

A ML WORKLOADS ARE PERIODIC

To measure the periodicity of the communication, we calculate the autocorrelation of each workload. For example, Fig. 15 plots the autocorrelation of a VGG workload in Fig. 2b, with peaks corresponding to training iterations. The autocorrelation is periodic with a slow decay, indicating future communication requirements between GPUs can be easily predicted from recent measurements. This observation helps in making better scheduling decisions as the future state of demands are correlated with their current states.

B MULTI-EDGE SEGMENT CUT

In section 3, we explain how we can express a wavelength allocation problem as a flow problem if we cut at a segment which only one traffic demand passes through. In Fig. 16, we show a more general case of cuts with higher degrees. Suppose that we want to inject the flows at the segment between Node3 and Node4. The problem basically remains the same while 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.

C OPTICAL SIMULATIONS

Figure 17 demonstrates our approach to achieve SiP interfaced GPU node 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. Cascaded SiP micro-ring filters are used 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 perform the simulation of 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 couplers (MMI). 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 bypass the node.

Figure 17: 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.
D INTEGER LINEAR PROGRAM FORMULATION

We can formulate the wavelength allocation ILP in section 3 as follows:

\[
\text{maximize}_{\Lambda \in \{0,1\}^{N \times N \times W}} \min_{i:j, T_{M_{ij}} > 0} \sum_k \Lambda_{ijk} / T_{M_{ij}}
\]

s.t.

(1) \[\sum_{(j \leq i \leq N+1) \cap (l < j)} \Lambda_{(i \mod N)jk} \leq 1 \quad \forall l,k\]  
(2) \[\sum_i \Lambda_{ijk} \leq 1 \quad \forall j,k\]
(3) \[\sum_j \Lambda_{ijk} \leq 1 \quad \forall i,k\]