Investigating Reconfigurable Topologies for Optical Circuits

Nithya Gottipati
Harvard University

Sahil Kuchlous
Harvard University

Dalila Oliva
Harvard University

***Abstract*—In the rapidly evolving field of large models in machine learning, it has become increasingly important to maximize the throughput of model training. Thus, optical circuit-based supercomputers like TPU v4 offer a unique opportunity to reconfigure network topologies based on the training task being run. An important family of topologies for such applications are *twisted tori*, which have been shown to significantly improve throughput for all-to-all communication patterns. Over 25% of workloads run on TPU v4 utilize twisted tori topologies. However, prior works only evaluate a subset of twisted tori topologies, and do not consider communication patterns seen in modern machine learning training workflows. In this paper, we use a network simulator to systematically evaluate potential twisting patterns for various communication patterns and parallelism schemes, and provide conceptual and mathematical justification for why certain patterns in communication time are observed. Our results demonstrate that twisted tori topologies offer improvements of up to 40% in average communication delay for model-tensor parallelism communication patterns, and identify the optimal twists for a number of common network dimensions.**

I. Introduction

In recent years, there has been an explosion of progress in machine learning, from deep neural networks to large language models, with applications across a large range of fields. As such models have become more powerful, model size and training data have grown exponentially. Thus, it has become more and more important to maximize the runtime of machine learning training tasks.

In 2023, Google released TPU v4, an optically reconfigurable supercomputer designed primarily for ML training workloads. [JKL+23] The use of optical circuit switches allowed for the dynamic reconfigurability of the topology based on the training task being run, improving "scale, availability, utilization, modularity, deployment, security, power, and performance." In the paper, the authors include statistics on the popularity of different TPU v4 slices, revealing that over 25% of all workloads use 'twisted tori' topologies, which are known to provide better throughput for all-to-all communication on asymmetric slices like $4 \times 4 \times 8$ and $4 \times 8 \times 8$. However, the paper does not make clear whether these results are based on the singly or doubly-twisted tori defined by Cámara et. al. [CMV+10] There are also a number of additional hyperparameters when designing twisted tori topologies that neither paper explores.

In this work, we begin by verifying the results of Jouppi et. al. and Cámara et. al., demonstrating that twisted tori topologies outperform regular tori topologies for all-to-all communication patterns. We also verify that twisted tori topologies still offer an improvement in communication times when evaluated on communication patterns that resemble those seen in data-tensor parallelism for model training. Next, we mathematically formalize the idea of twisted tori topologies, creating a complete search space for possible twists. We use this to simulate common training communication patterns (all-to-all and data-tensor parallelism) on different combinations of topologies and twisting patterns to empirically evaluate how such decisions interact. Finally, we provide mathematical justifications for some of our observations, found in the appendix.

Our results demonstrate that twisted tori topologies offer a significant improvement for data-tensor parallelism communication, reducing average delay by up to 40% as compared to regular tori topologies. However, we find that the dimension that is being twisted can have a significant impact on the communication delays. Thus, we perform a complete search to identify the optimal twisting patterns for a number of common topology dimensions, and dive deeper into the results on $2 \times 2 \times 4$ and $4 \times 4 \times 8$ topologies. Finally, we demonstrate that communication delay is strongly correlated with average distance between vertices in the network graph, providing a theoretical foundation for our work.

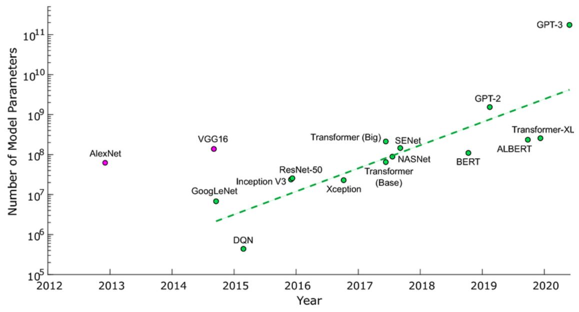

Fig. 1. The number of parameters of state-of-the-art models has grown exponentially. [BSH+21]

II. Design

Our design can be broken down into 3 parts: the construction of the topology, which involves customizing twisting patterns for different topology dimensions, the design of communication patterns, including all-to-all communication and data-tensor parallelism communication, and the actual simulation.

A. *Topology Construction*

When constructing our topologies, it was important to identify a formal way to enumerate through all possible twisted tori, going beyond what was discussed in past work. To do so, we had to allow for the twisting of every dimension in every other dimension. We chose to notate this using the following structure: $([x|y, x|z], [y|x, y|z], [z|x, z|y])$. Here, the first parameter is the dimension being twisted and the second is the axis in which it does so (so $x|y$ represents twisting the x-axis links in the y-axis). Each value is binary, indicating whether the twist is performed or not. In total, this creates $2^6 = 64$ twisting patterns.

As an example, let us consider the topologies analyzed by Cámara et. al. [CMV+10] The topologies listed in the paper can be translated to our notation as follows:

1) 3D Regular Tori: $([0, 0], [0, 0], [0, 0])$
2) 3D Prismatic Twisted Tori: $([0, 0], [1, 0], [0, 0])$
3) 3D Prismatic Doubly Twisted Tori: $([0, 0], [1, 0], [1, 0])$

Let us consider some further examples to illustrate the differences between twisting patterns $([0, 0], [0, 0], [0, 0])$, $([1, 0], [0, 0], [0, 0])$ and $([0, 1], [0, 0], [0, 0])$. Fig. 2 depicts a $4 \times 4 \times 4$ network with twisting pattern $([0, 0], [0, 0], [0, 0])$, also known as a regular tori. This is a regular tori because each vertex connects directly to its x, y and z neighbour, wrapping around in each dimension.

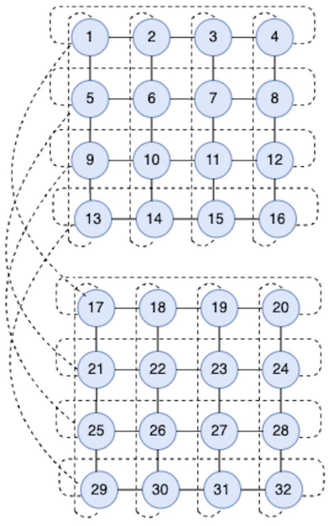

Fig. 2. $4 \times 4 \times 2$ Regular Tori (twists: $([0, 0], [0, 0], [0, 0])$). Some z edges are excluded for clarity.

If we want to add a single twist to the x-axis, we can choose to twist the edges on the y-axis or the z-axis. Twisting the x-axis edges on the y-axis would result in the topology in Fig. 3, corresponding to twisting pattern $([1, 0], [0, 0], [0, 0])$. The twist shifts the wrap-around edges in the x-axis halfway across the y-axis.

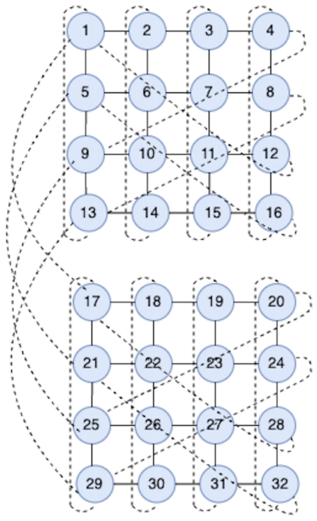

Fig. 3. $4 \times 4 \times 2$ Singly-Twisted Tori (twists: $([1, 0], [0, 0], [0, 0])$). Some z edges are excluded for clarity.

On the other hand, twisting the x-axis edges on the z-axis would result in the topology in Fig. 4, corresponding to twisting pattern $([0, 1], [0, 0], [0, 0])$. The twist shifts the wrap-around edges in the x-axis halfway across the z-axis instead. Note that this is slightly awkward in the figure, since the size of the z dimension is 2.

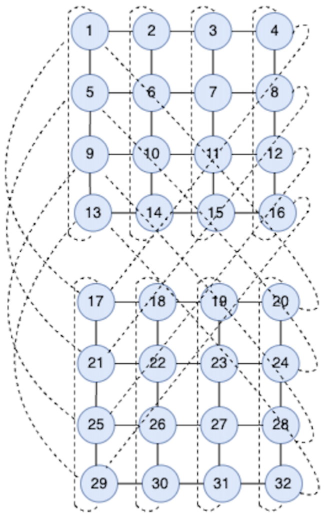

Fig. 4. $4 \times 4 \times 2$ Singly-Twisted Tori (twists: $([0, 1], [0, 0], [0, 0])$). Some z edges are excluded for clarity.

B. *Communication Patterns for Parallelism Schemes*

Communication patterns play a fundamental part in our experiments, as they can significantly change the communication delay that different topologies might have on different jobs. Thus, in this paper, we evaluate two different communication patterns: all-to-all communication, corresponding to just tensor parallelism, and a more complex communication pattern that combines the all-to-all communication of tensor parallelism with the ring all-reduce algorithm of data parallelism. We will refer to these as all-to-all communication and data-tensor parallelism respectively.

1) Tensor Parallelism (All-to-All): The communication pattern used in tensor parallelism is all-to-all, where packets are communicated between every pair of nodes in the network.

2) Data-Tensor Parallelism (Ring All-Reduce + All-to-All): To simulate a combination of data and tensor parallelism, we created alternating patterns of all-to-all communication and ring all-reduce communication. [JGLY20] The topology is sliced across the z-dimension, and all-to-all communication is run within the slices to simulate tensor parallelism. Every node also forms a ring with its immediate z neighbors, and communicates with them until the new gradients have been distributed to every layer, simulating data parallelism. Thus, by alternating between these two communication phases, we generate communication patterns that simulate data-tensor parallelism. For simplicity, we assume that both phases run for the same duration, and fix the number of iterations at 3.

For data-tensor parallelism, we restrict the possible twisting patterns to the following 4: $([0,0],[0,0],[0,0])$, $([1,0],[0,0],[0,0])$, $([0,0],[1,0],[0,0])$, and $([1,0][1,0][0,0])$. This is because it is unclear how twisting on the z-axis interacts with the ring all-reduce algorithm, so we only consider twisting on the x and y axes.

C. Simulation

To simulate communication patterns on different twisted tori topologies, we used NS-3, a powerful discrete-event network simulator targeted for research use. [RH10] The customizability of NS-3 gave us a lot of flexibility when it came to specifying communication patterns that resembled specific training tasks.

The data rate of every connection was fixed at 1 Mbps. All data was transmitted using OnOff applications with a data rate of 448 kbps (unless otherwise specified). The simulation was run for 10 seconds, with a total transmission time of 0.1 seconds for the all-to-all communication and 0.6 seconds for the data-tensor parallelism communication.

III. Evaluation

A. Comparison of Regular, Singly and Doubly-Twisted Tori

We start by evaluating regular, singly, and doubly-twisted tori on a $8 \times 4 \times 4$ slice with all-to-all communication. Our results can be seen in Fig. 5. As expected, we see that the singly-twisted tors outperforms the regular torus, and the doubly-twisted torus outperforms the singly-twisted torus. The differences in performance are consistent across data rates, with the doubly-twisted torus improving delay by over 30% as compared to the regular torus. Note that we are using twisting patterns $([0,0],[1,0],[0,0])$ for the singly-twisted torus and $([0,0],[1,0],[1,0])$ for the doubly-twisted torus, in line with the analysis in Cámara et. al. [CMV$^+$10]

Next, we perform a similar evaluation on a $8 \times 4 \times 4$ slice with data-tensor parallelism communication patterns. Our results can be seen in Fig. 6. In this case, we do not evaluate

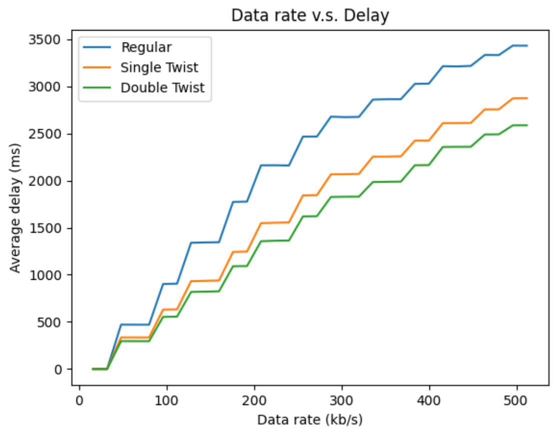

Fig. 5. Comparison of average delay for regular, singly-twisted and doubly-twisted tori topologies on a $8 \times 4 \times 4$ slice with all-to-all communication. The single and double twist match those evaluated by Cámara et. al. [CMV$^+$10] The data rate indicates the rate of all-to-all communication.

doubly-twisted tori, since our method for implementing data parallelism does not allow twisting in the z dimension. We see that performing a single twist in each layer improves the delay by over 40% as compared to the regular torus. Note that we are using twisting pattern $([0,0],[1,0],[0,0])$ for the singly-twisted torus. Thus, we demonstrate that twisted tori topologies offer significant improvement even for common model training tasks that involve combinations of data and tensor parallelism.

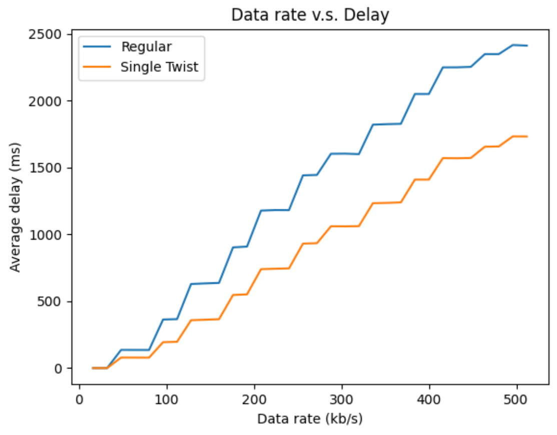

Fig. 6. Comparison of average delay for regular and singly-twisted tori topologies on a $8 \times 4 \times 4$ slice with data-tensor parallelism communication patterns. The single twist matches those evaluated by Cámara et. al. [CMV$^+$10] The data rate indicates the rate of all-to-all communication.

Finally, we compare different single-twist patterns on a $8 \times 4 \times 4$ slice with all-to-all communication. Our results can be seen in Fig. 7. From the data, it is clear that twisting patterns $([0,0],[1,0],[0,0])$ and $([0,0],[0,0],[1,0])$ outperform every other single twist. Based on our dimensions, these correspond to twisting y and z on x respectively. Note that y and z are our two shorter dimensions, and therefore twisting y on x is symmetric to twisting z on x. Also note that the single twists that perform best correspond to those analyzed by Cámara et. al. Thus, we demonstrate that choosing which twist to perform is essential to improving communication delays. A mathematical analysis for this difference can be found in the appendix.

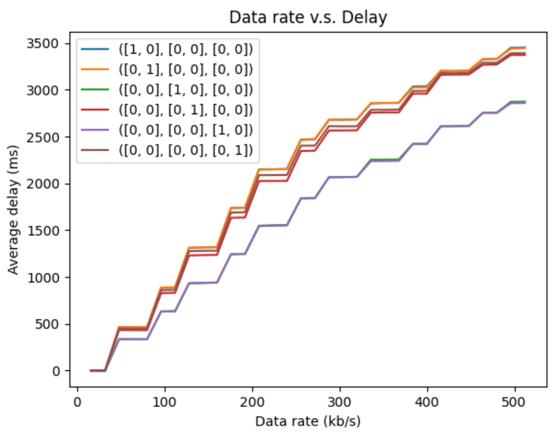

Fig. 7. Comparison of average delay for different single twists on a $8 \times 4 \times 4$ slice with all-to-all communication. The data rate indicates the rate of all-to-all communication.

B. *Identifying Optimal Twists for a Given Dimension*

By exhaustively evaluating all twist combinations on each dimension of interest, we identified the resulting optimal twists (i.e. the twist indices yielding the least average delay) for both all-to-all and data-tensor parallelism communication patterns.

For data-tensor parallelism, the 4 possible twist combinations on each dimension resulted in the following results:

Dimension	Optimal Twist
$2 \times 1 \times 1$	All Same
$2 \times 2 \times 1$	All Same
$2 \times 2 \times 2$	All Same
$4 \times 2 \times 2$	([0, 0], [1, 0], [0, 0])
$4 \times 4 \times 2$	([0, 0], [1, 0], [0, 0])
$4 \times 4 \times 4$	([0, 0], [1, 0], [0, 0])
$8 \times 4 \times 4$	([0, 0], [1, 0], [0, 0])
$12 \times 4 \times 4$	([1, 0], [1, 0], [0, 0])

TABLE I
OPTIMAL TWIST FOR GIVEN DIMENSION WITH DATA-TENSOR PARALLELISM

Notably, every optimal twisting strategy involves twisting the shorter dimension (y) on the longer dimension (x), which agrees with our observations in the previous section.

For all-to-all communication, the 64 possible twist combinations on each dimension resulted in the following:

Dimension	Optimal Twist
$1 \times 1 \times 2$	All Same
$1 \times 2 \times 2$	([1,1], [0,0], [0,0])
$2 \times 2 \times 2$	([1,0], [0,1], [1,0])
$2 \times 2 \times 4$	([1,0], [0,1], [0,1])
$2 \times 4 \times 4$	([1,1], [1,0], [1,0])
$4 \times 4 \times 4$	([1,0], [1,1], [1,0])
$4 \times 4 \times 8$	([0,1], [0,1], [1,1])
$4 \times 4 \times 12$	([0,1], [0,1], [0,1])

TABLE II
OPTIMAL TWIST FOR GIVEN DIMENSION WITH ALL-TO-ALL COMMUNICATION

Notably, the optimal twists of this parallelism strategy often acted on more than two axes, with a tendency to include at least one twist along each axis.

Of particular interest were the dimensions $2 \times 2 \times 4$ and $4 \times 4 \times 8$, which demonstrate optimal twists on asymmetric typologies when the network is scaled by a factor of 2. Despite not exhibiting identical optimal twist indices, implementing the optimal twist for $2 \times 2 \times 4$ in the $4 \times 4 \times 8$ dimension resulted in a remarkably close delay value to its empirical minimum. To determine if the slight gap between the optimal twists for these dimensions could be attributed to noise, we generated heatmaps representing the average delay for every twist combination for both dimensions, respectively.

The arrangement of twist indices in the 8×8 heatmap follows a systematic progression that reflects the different combinations of twisting operations on the x, y, and z axes. Starting from the top-left corner with the index 000000, subsequent indices are generated by adding 1 in binary to the previous cell in a little-endian manner, wrapping around to the next row once the end of the current row is reached. This creates a binary counting pattern that systematically explores all possible twist combinations from 000000 to 111111, which is then converted to our twisting notation.

Consider the $4 \times 4 \times 8$ heatmap, in which a clear trend emerges: twisting the longest axis (z) without an accompanying x or y twist results in the highest delay values, surpassing even the scenario with no twist at all. Generally, elevated delay values are associated with the act of twisting the longest axis (z) on the shorter axes (x or y). Additionally, high delays are observed when twisting a short axis on another short axis, as evident in cases where x is twisted on y or y is twisted on x, respectively. Conversely, the most favorable delay values are found when twisting the x or y axes on the z axis — i.e., in twisting the short axes on the long axis.

Similar results can be extrapolated from the $2 \times 2 \times 4$ heatmap, albeit with some nuanced distinctions. Notably, in the $2 \times 2 \times 4$ scenario, having no twist at all yields the worst delay, even surpassing the delay incurred by twisting the long axis (z) on the shorter axes (x or y), showcasing the influence of dimensionality on optimal twist configurations.

IV. FURTHER WORK

While our current study sheds light on the effectiveness of twisted topologies for specific dimensions and parallelism schemes, there are several avenues for further exploration.

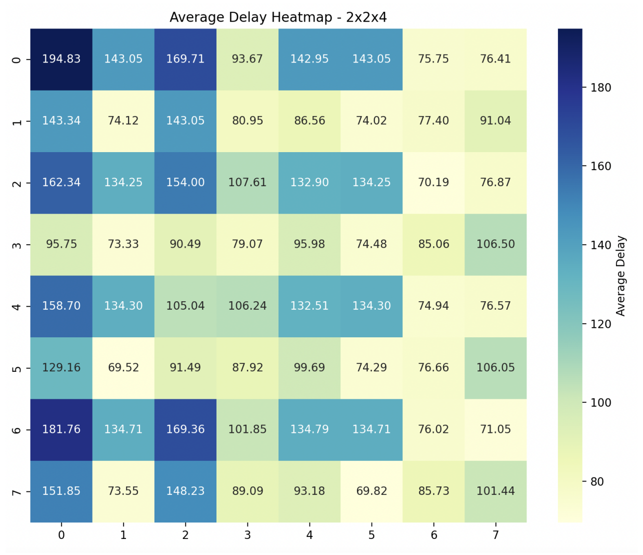

Fig. 8. The average delay for all 64 twist combinations in $2 \times 2 \times 4$.

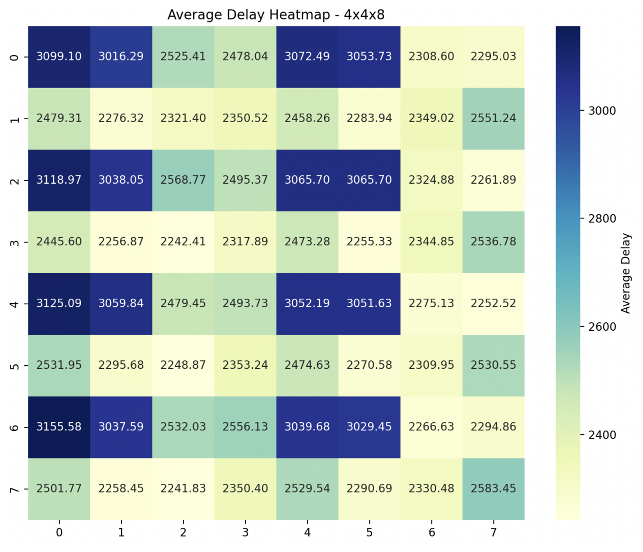

Fig. 9. The average delay for all 64 twist combinations in $4 \times 4 \times 8$.

1) Scaling to Larger Topologies: Building upon the insights gained from our experiments, it would be helpful to extend our study to larger dimensions, completing an investigation of the configurations used on TPU v4. Currently, our results exclude 37% of the slices used in practice. [CMV+10] Investigating the scalability of twisted topologies on larger dimensions can provide valuable insights into the performance of optical circuits at an enterprise scale.
2) Using Doubly-Twisted Tori with Data-Tensor Parallelism: Based on our current implementation, it is not possible to twist in the z-axis when training with data-tensor parallelism, as this causes the Ring All-Reduce algorithm for the data parallelism step to fail. Thus, understanding how to modify All-Reduce algorithms to work with doubly-twisted tori could result in significant improvements in training time.
3) Exploring Additional Forms of Parallelism: Our experiments primarily focus on tensor and data parallelism with specific communication patterns. Future work should delve into alternative forms of parallelism, such as model parallelism and other hybrid approaches, to understand how twisting patterns respond to varying workload characteristics.
4) Investigating Different Assignment Patterns: Extending our research to encompass a broader spectrum of assignment patterns for both tensor and data parallelism is crucial. Currently, our investigations of data-tensor parallelism utilize the framework described by Jouppi et. al., where data parallelism is split across 2-dimensional slices and tensor parallelism is performed within each slice. More interesting assignments of parallelism schemes allow for more flexibility in the division of resources, but also require novel all-reduce algorithms, since Ring All-Reduce is no longer sufficient. One possible approach is outlined in Jiang et. al., which proposes All-Reduce algorithms for 2-dimensional regular torus topologies. Benchmarking these techniques and understanding how to adapt them to higher dimensions and twisted tori could result in more diverse parallelism schemes, improving communication efficiency and compute allocation.

V. Related Work

In the paper that first introduced twisted and doubly-twisted tori, the authors mathematically and empirically evaluate specific single and double twists on $2a \times a \times a$ dimensional topologies, and find that singly twisted tori provide better link utilization than regular tori, and that doubly twisted tori improve link utilization further [CMV+10]. They evaluate these topologies on a number of different workloads, including random uniform traffic, random non-uniform traffic and permutation based traffic. However, these evaluation results do not include traffic patterns observed in real machine learning workloads, which often use combinations of parallelism techniques. This is an important contribution of our paper. We also evaluate a larger set of possible twisting patterns, extending the results.

For optical circuit based supercomputers, there has been some prior work on identifying optimal reconfigurable topologies for a given training workload. In particular, Jouppi et. al. propose a search algorithm to tailor the TPU v4 topology to a DNN Model. [JKL+23] However, a major limitation of this approach is that it lacks transparency, providing little justification for why certain topologies may be better suited for a training task than others. It is important to understand these interactions so that models can be designed with topology in mind and vice versa.

VI. Conclusion

Our investigation into reconfigurable topologies for optical circuits and machine learning training workloads yields critical insights into the interplay between topology configurations and communication efficiency. Through an exhaustive exploration of twisted tori structures, we discern nuanced patterns governing the optimal twist combinations for diverse parallelism schemes and dimensions.

Our empirical findings corroborate the efficacy of twisted tori as established by Jouppi et. al. and Cámara et. al., with twisted topologies consistently outperforming their regular counterparts. [CMV+10] [JKL+23] Moreover, the strategic selection of twist indices, particularly in favoring twists along shorter axes on the longest axis, emerges as a crucial determinant in minimizing communication delays. These results hold true across various dimensions, emphasizing the scalability and generalizability of twisted tori for large-scale machine learning models.

Notably, our results demonstrate that twisted tori topologies yield substantial improvements, slashing average communication delays by up to 40% in the context of model-tensor parallelism communication patterns. However, our study also identifies additional avenues for exploration, including scaling to larger topologies, accommodating doubly-twisted tori with data-tensor parallelism, and investigating alternative forms of parallelism.

By systematically bridging theoretical formulations with empirical evaluations, our work contributes to a deeper understanding of the dynamics shaping communication efficiency in machine learning training workflows. We anticipate that these findings will not only inform the design of future optical circuit-based supercomputers but also prompt advancements in the intersection of topology optimization and machine learning model training.

References

[BSH+21] Liane Bernstein, Alexander Sludds, Ryan Hamerly, Vivienne Sze, Joel Emer, and Dirk Englund. Freely scalable and reconfigurable optical hardware for deep learning. *Scientific Reports*, 11, 02 2021.

[CMV+10] José M Cámara, Miquel Moretó, Enrique Vallejo, Ramon Beivide, Jose Miguel-Alonso, Carmen Martínez, and Javier Navaridas. Twisted torus topologies for enhanced interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 21(12):1765–1778, 2010.

[JGLY20] Youhe Jiang, Huaxi Gu, Yunfeng Lu, and Xiaoshan Yu. 2d-hra: Two-dimensional hierarchical ring-based all-reduce algorithm in large-scale distributed machine learning. *IEEE Access*, 8:183488–183494, 2020.

[JKL+23] Norman P. Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings, 2023.

[RH10] George F. Riley and Thomas R. Henderson. *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

Appendix

We start by explaining why, for single twists on tori $2a \times a \times a$ nodes, twisting the shorter dimensions (y and z) on the longer dimension (x) results in the largest improvement in delay. To do so, we will use the techniques developed by Cámara et. al. in their analysis. [CMV+10] The key idea is that the torus obtained by twisting the shorter dimension on the longer dimension results in a graph equivalent to the Cartesian product of the twisted torus on $2a \times a$ nodes with a ring of a nodes, where the twisted torus on $2a \times a$ has edges that maximize the theoretical link utilization. In contrast, other twists either reduce the link utilization of the twisted torus on $2a \times a$ nodes, or are instead formed by taking the Cartesian product of the twisted torus on $a \times a$ nodes with the ring of $2a$ nodes, which does not improve link utilization since the regular torus on $a \times a$ nodes is already edge-symmetric.

To formalize this further, we compare the average pairwise distance in the following twisting patterns on $8 \times 4 \times 4$ nodes: $([0,0],[1,0],[0,0])$, $([1,0],[0,0],[0,0])$ and $([0,0],[0,1],[0,0])$. Note that these correspond to our 3 unique cases for a single twist: twisting a short dimension on the long dimension, twisting the long dimension on a short dimension, and twisting a short dimension on a short dimension. It is possible to calculate the average pairwise distances between vertices in each of these graphs theoretically, but this is often inelegant. Thus, as done in Cámara et. al., we utilize a program to compute the average pairwise distances on these graphs, running a BFS algorithm from every vertex. [CMV+10] The results can be found in Table III. Note that the first pattern has the shortest average distance, which further supports the argument that twisting the short dimension on the long

Twisting Pattern	Average Distance
$([0,0],[1,0],[0,0])$	3.625
$([1,0],[0,0],[0,0])$	3.9375
$([0,0],[0,1],[0,0])$	3.875

TABLE III
Average Distance for Different Twisting Patterns

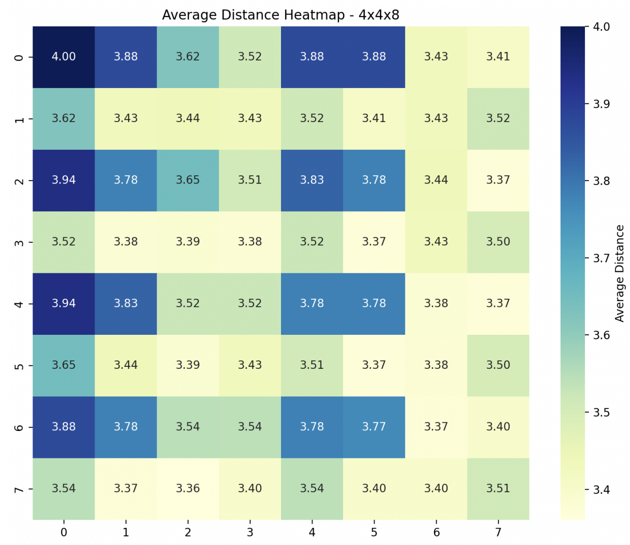

Fig. 10. The average distance between vertices in a $4 \times 4 \times 8$ graph with each twisting pattern.

dimension is optimal for a single twist.

Next, we provide a theoretical justification for the average delays seen in Fig. 9. As earlier, we use a program to compute the average distances for each of the twisting patterns evaluated on a $4 \times 4 \times 8$ vertex graph. The results can be seen in Fig. 10. Note the strong correlation between the average distance and resultant delay. Thus, we conclude that for all-to-all communication, throughput is heavily dependent on the average distance between vertices in a network topology.