Highly Available Data Parallel ML training on Mesh Networks

Sameer Kumar, Norm Jouppi
Google Inc
`{sameerkm, jouppi}@google.com`

Abstract

Data parallel ML models can take several days or weeks to train on several accelerators. The long duration of training relies on the cluster of resources to be available for the job to keep running for the entire duration. On a mesh network this is challenging because failures will create holes in the mesh. Packets must be routed around the failed chips for full connectivity. In this paper, we present techniques to route gradient summation allreduce traffic around failed chips on 2-D meshes. We evaluate performance of our fault tolerant allreduce techniques via the MLPerf-v0.7 ResNet-50 and BERT benchmarks. Performance results show minimal impact to training throughput on 512 and 1024 TPU-v3 chips.

1 Introduction

Deep learning has found a wide range of applications from image classification [12], object detection [13], language modeling [21, 10], content recommendation [20], speech recognition [9], reinforcement models for gaming and self driving cars. To enable high quality, the models are trained on large datasets typically for several days on tens to hundreds of accelerators such as NVIDIA GPUs. The popular algorithm for distributed training is mini-batch data parallel training [18]. Here each worker executes ML training forward and backward passes on a mini-batch. The computed gradients from the loss function are then summed globally via an allreduce operation.

Large scale ML data parallel training relies on a scalable global allreduce library optimized for the training platform. As several training steps are typically executed with the same batch size over several days, the same number of accelerators must be dedicated to the training job. Typically, in a datacenter cluster, when a failure happens the job will restart from a recent checkpoint and the failed server is swapped with a spare server on the data center network. This approach works well on a fully connected data center network.

On a 2-D mesh network, when there is a failure, any of the following approaches could be used:

- Fire Fighter approach: here data center specialists or even robots can quickly go and repair the failed host and make all servers in the mesh available for the job.
- Sub-mesh jobs: the ML training job is executed on a mesh smaller than the original mesh. If the failure is in the middle of the mesh, the training job may only execute on half the mesh resulting in significant loss to training throughput in that job, while the unavailable servers are repaired.
- Rebuild mesh with hot spares [7]: in this case, when there is a failure, the mesh data network is rebuilt via the use of spares. Note, there is additional cost to having spares in rows and columns of the mesh.
- Fault tolerant technique: here there are no additional spares and network packets are routed around the failed nodes in the mesh network. The main challenge here is to execute the gradient global summation efficiently on the entire mesh even with failed chips.

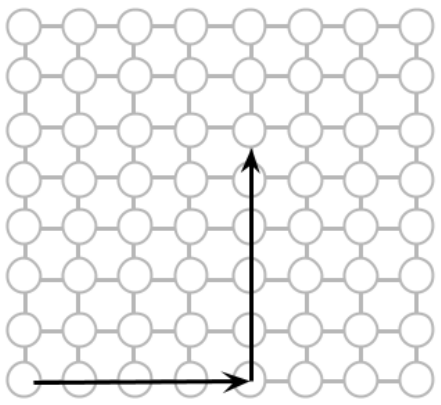
Figure 1: Dimension order routing on 2-D meshes.

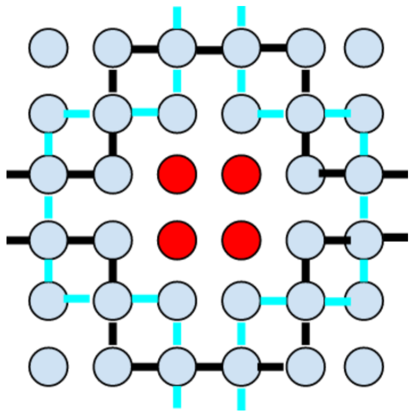
Figure 2: Non-minimal routing on a 2-D mesh with a 2x2 failed region of chips.

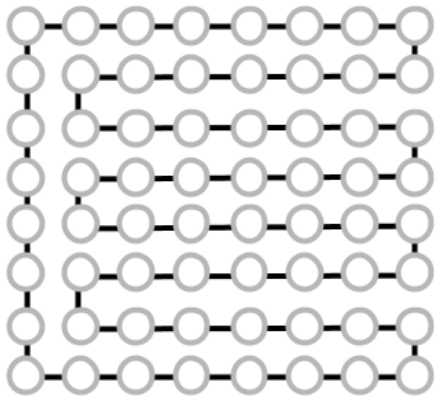
Figure 3: 1-D algorithm for building a near-neighbor Hamiltonian ring on a 2-D mesh.

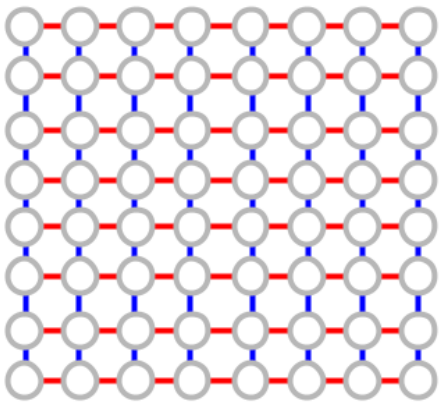
Figure 4: 2-D algorithm for allreduce on 2-D meshes. Here, there are two concurrent reductions colored red and blue

We present techniques to execute fast global summation operations on 2-D meshes with failed chips. We present performance results on the Google TPU-v3 machine [15, 1, 17]. The next section describes optimized allreduce algorithms on 2-D meshes with failures.

2 Mesh Algorithms

Routing on the TPU-v3 mesh communication network uses standard dimension order routing (Figure 1). To increase availability of TPU meshes, we explore a mode where we enable high throughput allreduce with failed regions that are 2x2 or 4x2 blocks of chips. These correspond to a single board or 2 boards on a single host of the TPU-v3 machine. In the presence of a failed contiguous region of chips the network routing would use non-minimal paths around the failed region as shown in Figure 2. As long as the non-minimal paths don't form cycles, significant additional Virtual channel resources are not required on a 2D mesh [16, 11].

2.1 Allreduce on 2-D mesh networks

The ring algorithm [5] is widely used when implementing the allreduce collective operation. For example, the NVIDIA NCCL [3] uses a ring algorithm for reductions between GPUs over the NVLink network [4]. Ring reductions schemes for allreduce on a 2-D mesh can use either the 1-D algorithm (Figure 3) or the 2-D algorithm [14] as shown in Figure 4. In the 1-D scheme, a Hamiltonian circuit is built such that nodes only communicate with a downstream and upstream near neighbor on the

2-D mesh. This scheme can have a high latency of $O(N^2)$ store-forward transfers on an $N \times N$ mesh. This may be significant for short and medium sized transfers. In the 2-D algorithm (Figures 4 and 5), nodes execute allreduce rings along the X dimension first (shown in red) and then along the y dimension (shown in blue). After the first phase along the X dimension of an $N \times N$ mesh, each node has a reduced shard of size $1/N$ the total allreduce payload. In the second phase, the small summed shard is summed along the Y dimension to produce a shard of size $1/N^2$ the allreduce payload. The result shard is broadcast over two gather phases on the Y and then the X dimensions. Note, the 2-D algorithm has a lower latency of O(N) on an $N \times N$ mesh. For full throughput, we can execute two concurrent flips over half the payload along X and Y dimensions and then execute the second phase along Y and X dimensions, respectively. This results in twice the throughput in the 2-D algorithm.

A possible downside of the 2-D scheme is that links are shared by traffic in two directions on a 2-D mesh resulting in network contention. An alternative scheme that does not require multiple colors (concurrent flips) is presented in Figures 6 and 7. Note, in this scheme we build rings of size $2 \times N$ nodes. As each link is only used by one all-reduce ring, this scheme can achieve high link throughput in the first phase. However, note in the second phase (Figure 7) nodes must communicate with ring neighbors that skip rows and that may result in some network congestion. However, on large meshes the communication volume is significantly reduced in the second phase and this phase will not significantly impact the throughput of the allreduce operation.

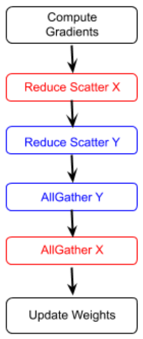

Figure 5: Steps in the data parallel gradient summation allreduce operation on a 2-D mesh.

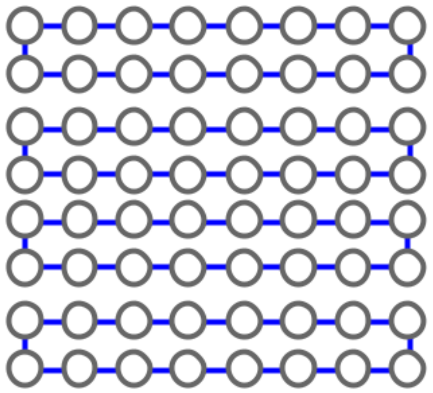

Figure 6: First phase in the alternate 2-D allreduce scheme, where pairs of two rows form a ring and execute a ring allreduce.

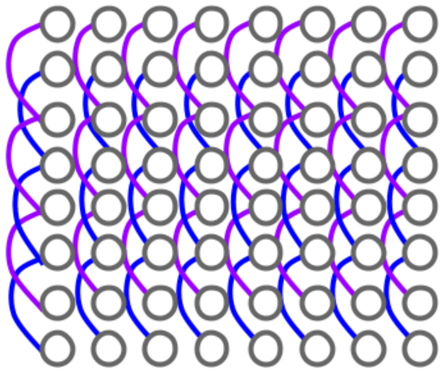

Figure 7: Second phase of alternate 2-D allreduce scheme, where nodes in alternate rows form a ring.

2.2 Fault tolerant allreduce schemes

We next present algorithms for allreduce when there are failures. Figure 8 shows the 1-D scheme on a 2-D mesh with a contiguous failed region of size 2x2. Note, the 1-D Hamiltonian circuit can be built when the failed chips are form a contiguous region that is of even size and starts on even rows and columns.

When the shape of the failed region is 2kx2 or 2x2k we can build optimal 2-D allreduce rings as shown in Figure 9. Here we build rings on nodes along two consecutive rows along the X dimension similar to the allreduce scheme in Figures 6 and 7. Neighbors of the failed chips (shown in yellow) form smaller rings in similar 2x2 blocks as shown in Figure 9. After a ring reduction round, the partial sums on yellow nodes are forwarded to neighbors on full blue rings as shown in Figures 9 and 10. In the first phase of the allreduce, the blue rings do not share network links and that results in

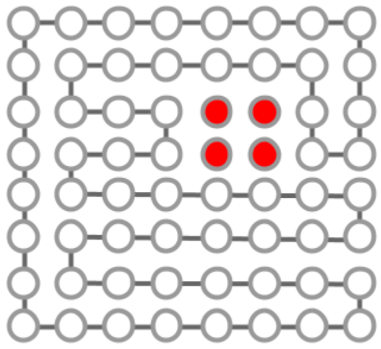

Figure 8: 1-D scheme to build an allreduce ring on a 2-D mesh with a 2x2 failed region. Failed chips marked in red

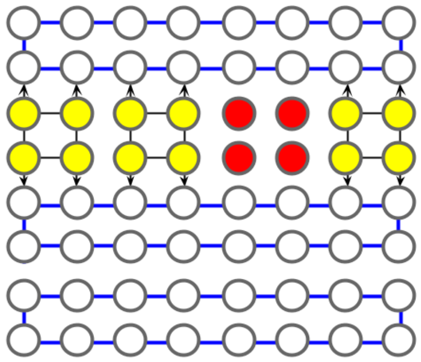

Figure 9: Fault tolerant allreduce rings built using a 2-D algorithm. The failed chips are marked in red and the peers of failed chips marked in yellow forward data to the full blue rings.

high throughput. A similar forwarding scheme can be used in the second phase, where nodes that are Y neighbors of the failed chips forward their contributions to Y neighbors on columns that don't have failed chips. However, for simplicity, we just use the route around scheme shown in Figure 2 to execute ring reductions in the second phase. We find the route around scheme works quite well as the second phase transfers 1/2N less payload than the first phase.

3 Experiments

We compare the performance of fault tolerant 2-D allreduce with standard 2-D allreduce on the Google TPU-v3 machine. We use the MLPerf-v0.7 [2] ResNet-50 and BERT benchmarks to compare both schemes. These benchmarks are developed in the TensorFlow [8] programming framework. ResNet-50 [12, 23] is an image classification model that is one of the most widely-used models for ML benchmarking. The MLPerf [2] ResNet-50 benchmark trains the model on the ImageNet-1K [19] dataset. The BERT [10] model is a pre-training task for language understanding with a bi-directional transformer architecture that trains on the Wikipedia dataset. For a fine grained analysis of allreduce we disable the weight update sharding [22] technique in the XLA compilers for TPUs [6]. Note the weight update sharding technique distributes the optimizer weight updates by executing them on the partially summed shards produced during the ring-allreduce algorithm.

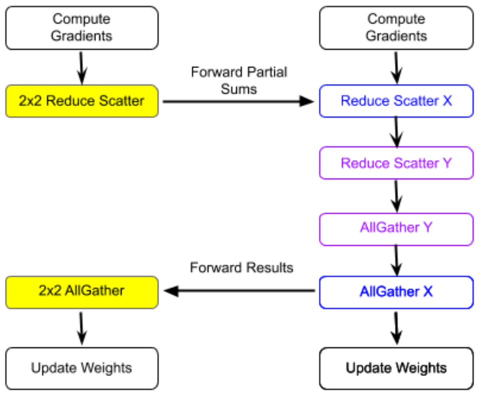

Figure 10: Steps in the forwarding scheme with a failed 2x2 region on a 2-D mesh.

Table 1 shows the end to end time with the two MLPerf benchmarks on 512 and 1024 TPU chips that had 16x32 and 32x32 mesh topologies. The failed region here has a shape of 4x2 with 8 total failed chips. Note, the run-to-run variance here is under 2%. The table also shows the relative efficiency of fault tolerant vs full meshes. The relative efficiency also compensates for the reduction in the number of chips in addition to overheads from the fault tolerant allreduce scheme. From the table we can conclude the maximum overhead from fault tolerant allreduce is 5.4%. On 512 chips, the fault tolerant job is more efficient than the full mesh job. This may be because the fault tolerant training job results in better regularization than the full mesh job. Table 2 compares the communication overheads from standard allreduce vs fault tolerant allreduce.

Benchmark	Full Mesh		Fault Tolerant Mesh		Relative Efficiency
	TPU Chips	Benchmark Time	TPU Chips	Benchmark Time	
ResNet-50	512	1.80 minutes	504	1.84 minutes	0.99
ResNet-50	1024	1.08 minutes	1016	1.15 minutes	0.946
BERT	512	1.90 minutes	504	1.92 minutes	1.02
BERT	1024	1.16 minutes	1016	1.19 minutes	0.986

Table 1: End to end times from the MLPerf-v0.7 benchmarks are shown on both the standard 2-D mesh and the fault tolerant 2-D mesh with a failed 4x2 region. The relative efficiency shows the performance degradation from the fault tolerant mesh.

Benchmark	Full Mesh		Fault Tolerant Mesh	
	TPU Chips	Allreduce overhead	TPU Chips	Allreduce Overhead
ResNet-50	512	4.2%	504	6.4%
ResNet-50	1024	8.8%	1016	11%
BERT	512	3.7%	504	4.7%
BERT	1024	6.0%	1016	7.8%

Table 2: Here we show the communication overhead percent in the device execution step time.

4 Summary

We presented a fault tolerant allreduce algorithm on 2-D mesh networks. The fault tolerant allreduce algorithm enabled higher availability as ML training jobs could execute with a failed region of up to 8 chips (in a contiguous 4x2 topology). This resulted in ML training running on 504 of the 512 chips and 1016 out of 1024 chips, respectively. In the MLPerf BERT and ResNet-50 benchmarks, the training time was minimally affected with under 6% overheads in the worst case. This scheme has been available in production on Google data centers for training of Google ML models. Techniques presented are general and the fault tolerant schemes could be extended to other architectures that have 2-D meshes.

In future, we plan to implement the weight update sharding optimization [22] on meshes with failures. As the fault tolerant allreduce algorithm builds reduce-scatter and all-gather rings on complete dimensions, the optimizer weight updates can be computed at the end of the reduce-scatter phase and the updated weights can be forwarded to the nodes that are neighbors of the failed chips and do not participate in those allreduce rings.

The schemes presented in this paper can be extended to n-D mesh and n-D torus networks. On such networks multi-dimensional rings will need to be built to reach peak performance. The main challenge would be to ensure efficient forwarding of partial sums from neighbors of failed chips to the full allreduce rings.

5 Acknowledgements

We would like to thank Bjarke Roune, Jeremy Lau, Steve Lacy, Andy Swing, Peter Gavin, Blake Hechtman, David Manemer, Robert Hundt, Naveen Kumar, Cliff Young, George Kurian, Peter Brandt and Pankaj Kanwar for their support with TPU compilers, systems and performance analysis.

References

[1] Empowering businesses with Google Cloud AI. `https://cloud.google.com/tpu`.

[2] Mlperf: Fair and useful benchmarks for measuring training and inference performance of ml hardware, software, and services. `http://mlperf.org`.

[3] NVIDIA Collective Communications Library (NCCL). `https://developer.nvidia.com/nccl`.

[4] nvlink and nvswitch: The building blocks of advanced multi-gpu communication.

[5] bandwidth optimal all-reduce algorithms for clusters of workstations.

[6] Xla: Optimizing compiler for tensorflow. `https://www.tensorflow.org/xla`.

[7] Wafer scale deep learning. *Cerebras tutorial at Hotchips'19*, 2019. URL `https://www.hotchips.org/hc31/HC31_1.13_Cerebras.SeanLie.v02.pdf`.

[8] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[9] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin, 2015.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, and H. Tenhunen. Lear – a low-weight and highly adaptive routing method for distributing congestions in on-chip networks. In *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 520–524, 2012.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL `http://arxiv.org/abs/1703.06870`.

[14] Nikhil Jain and Yogish Sabharwal. Optimal bucket algorithms for large MPI collectives on torus interconnects. *In proceedings of the International Conference on SUpercomputing*, 2010.

[15] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, Richard C. Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of ISCA'17*, 2017. URL `http://arxiv.org/abs/1704.04760`.

[16] M. Kumar, V. Laxmi, M. S. Gaur, M. Daneshtalab, M. Ebrahimi, and M. Zwolinski. Fault tolerant and highly adaptive routing for 2d nocs. In *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 104–109, 2014.

[17] Sameer Kumar, Victor Bitorff, Dehao Chen, Chiachen Chou, Blake Hechtman, HyoukJoong Lee, Naveen Kumar, Peter Mattson, Shibo Wang, Tao Wang, et al. Scale MLPerf-0.6 models on Google TPU-v3 pods. *arXiv preprint arXiv:1909.09756*, 2019.

[18] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020.

[19] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[20] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf`.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

[22] Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Hongjun Choi, Blake Hechtman, and Shibo Wang. Automatic Cross-Replica Sharding of Weight Update in Data-Parallel Training. *arXiv preprint arXiv:2004.13336*, 2020.

[23] Chris Ying, Sameer Kumar, Dehao Chen, Tao Wang, and Youlong Cheng. Image classification at supercomputer scale. *CoRR*, abs/1811.06992, 2018. URL `http://arxiv.org/abs/1811.06992`.