Challenges in large scale training of Giant Transformers on Google TPU machines

Sameer Kumar
Google
sameerkm@google.com

HotChips ML Tutorial, August 22nd, 2021

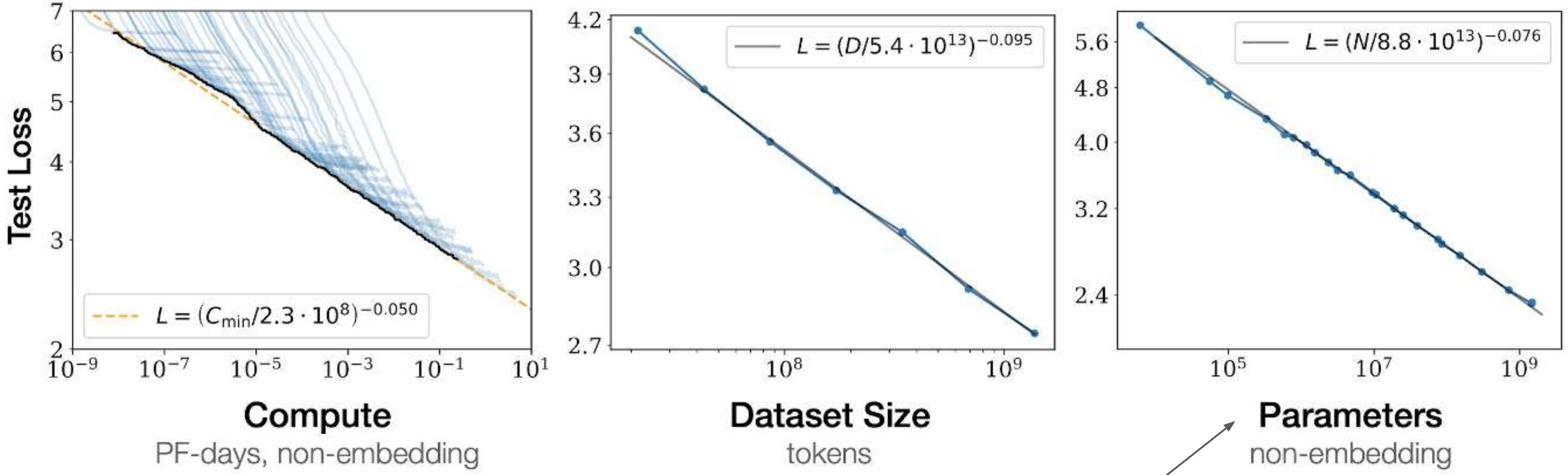
Acknowledgements

- James Bradbury
- Dehao Chen
- Ryan Doherty
- Liam Fedus
- Samer Hasan
- Blake Hechtman
- Yanping Huang
- Naveen Kumar
- Marcello Maggioni
- Karthik Murthy
- Noam Shazeer
- Amit Sabne
- Shibo Wang
- Tao Wang
- Jinliang Wei
- Yuanzhong Xu
- Cliff Young
- Zongwei Zhou

Neural Scaling Laws

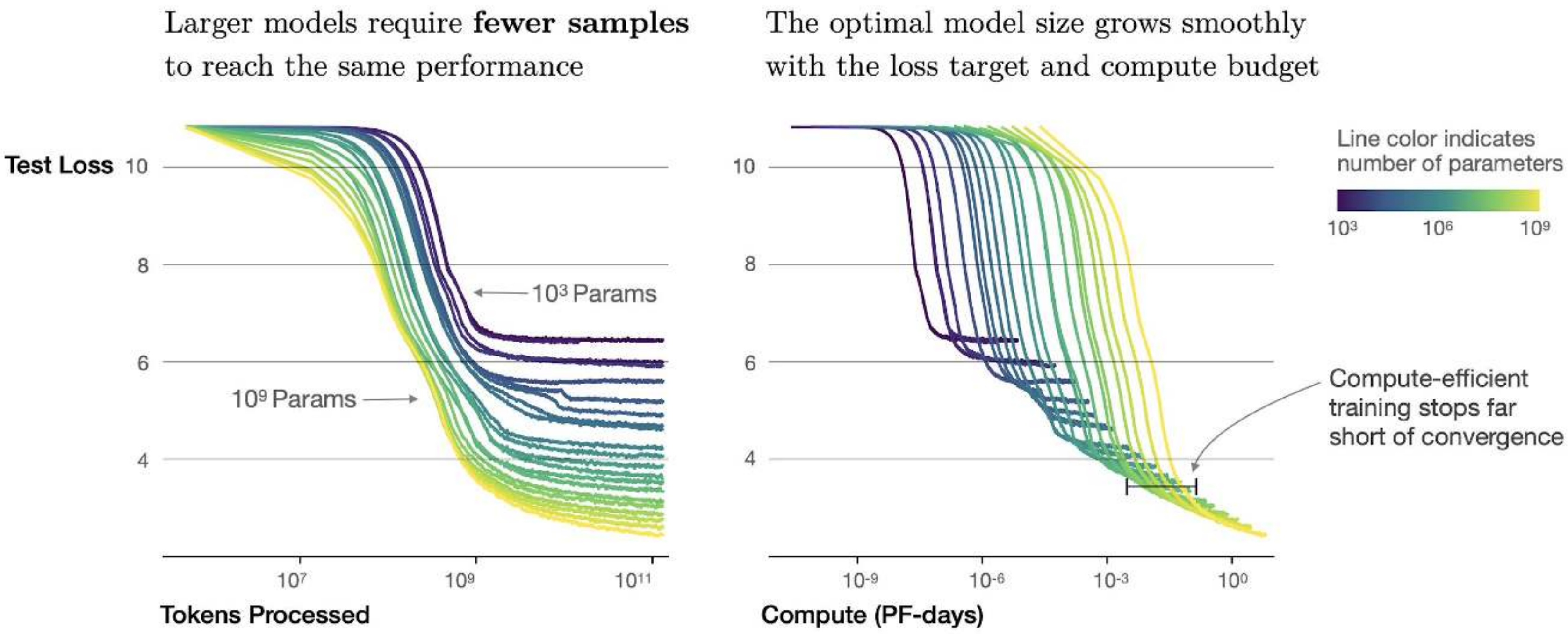
Language modeling performance, when not bounded by model-capacity, data size or compute, demonstrates power-law scaling over many orders of magnitude

Scaling Laws for Neural Language Models, Kaplan et. al. 2020 (https://arxiv.org/abs/2001.08361)



(With Increasing FLOPS per example)

Neural Scaling Laws



Google

Giant Language Models

OpenAI GPT3 - 200+ billion parameter model

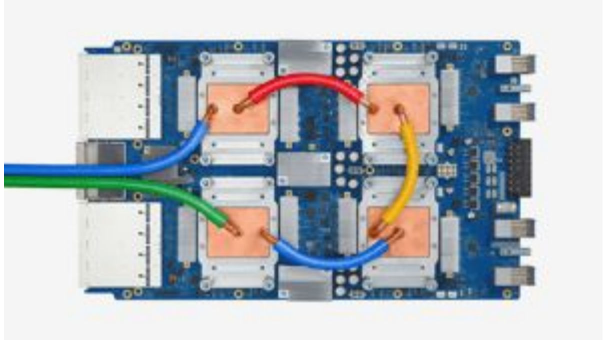
Mixture of experts: Gshard (1 trillion parameters)

Switch transformers (1.6 trillion parameters)

Google LaMDA model

Google T5 text-to-text transformer framework

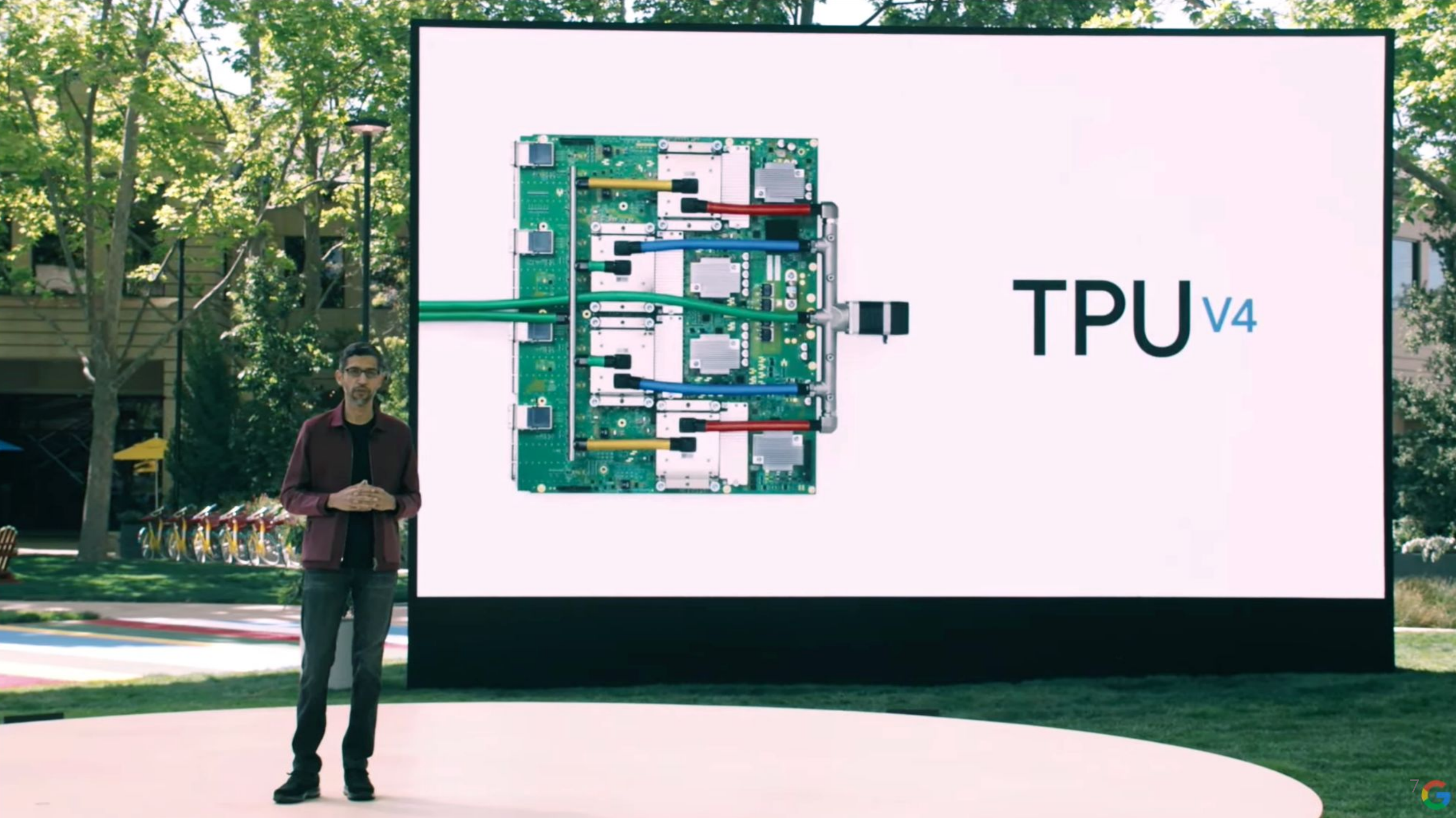
Google Tensor Processing Units (TPUv3)

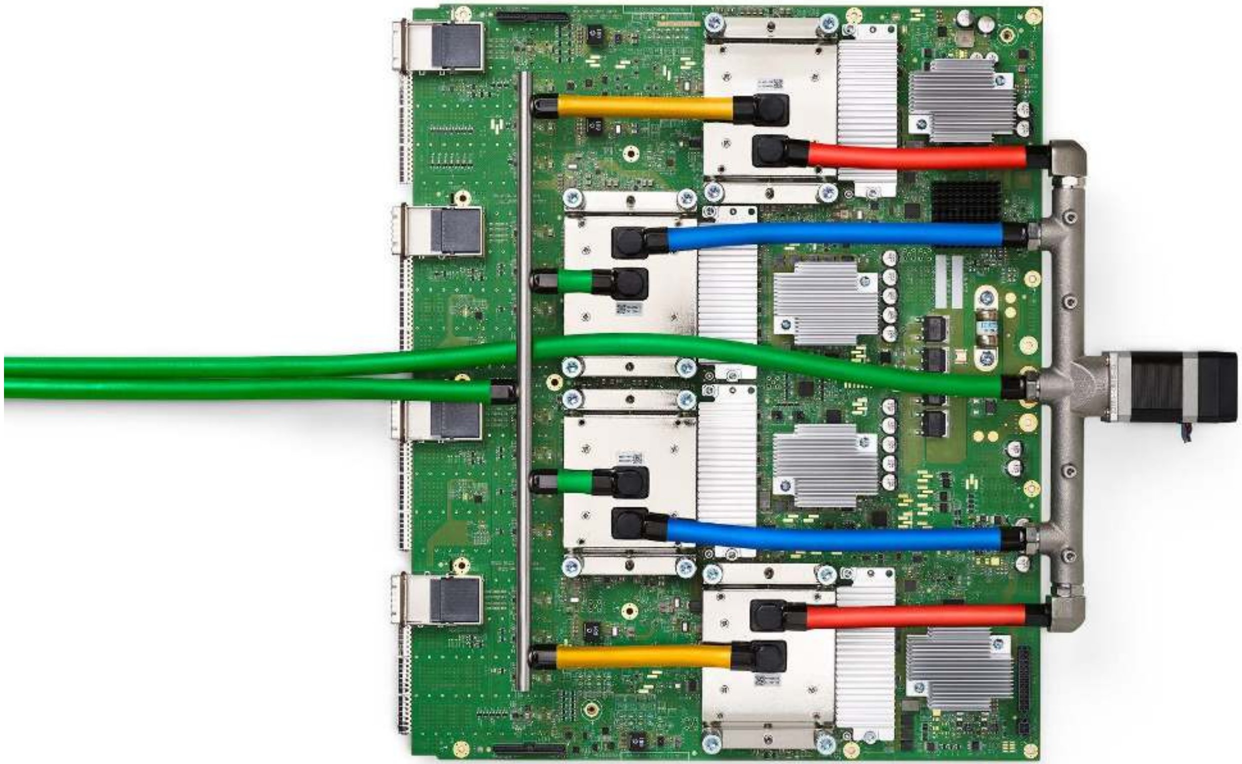


420 TFLOPS, 128 GB HBM

TPU Pod: 100+ PFLOPS, 32 TB HBM, 2-D Toroidal Mesh Network

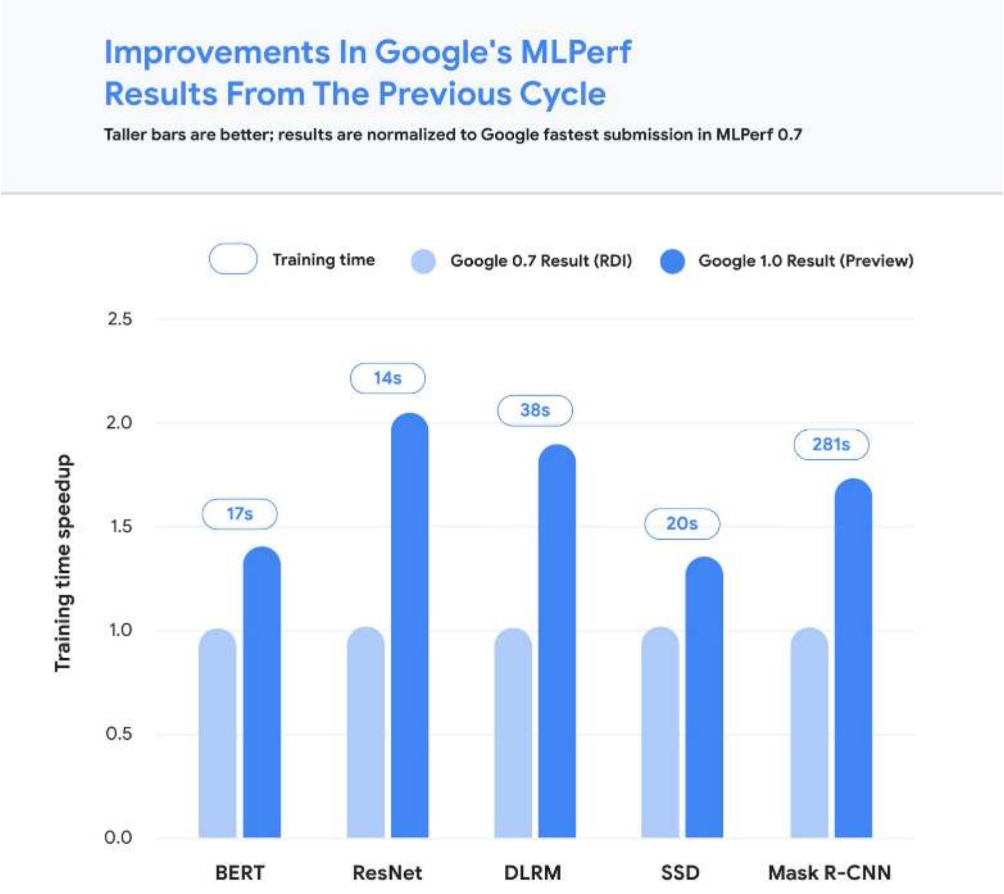
Image Source: https://cloud.google.com/tpu/


TPU V4

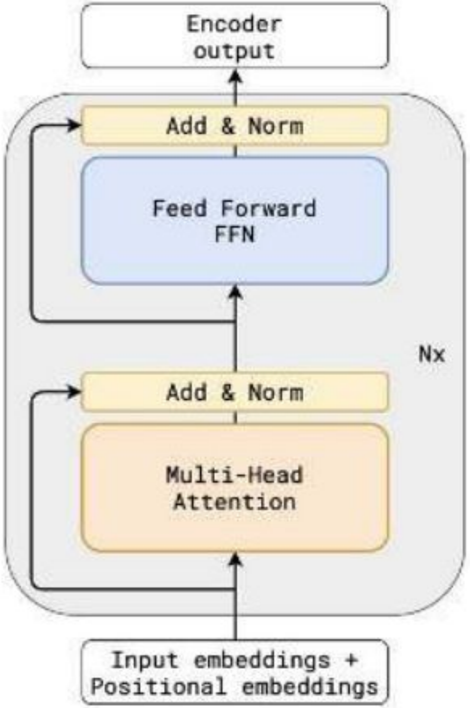


MLPERF on Google TPU-v4

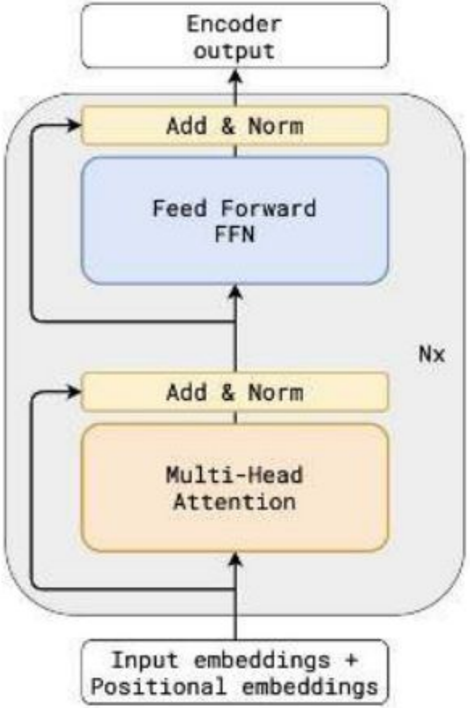
- Pods with 4096 chips for 1.1 exa flops of mixed precision
- Torus network for data transfer



ML Transformers



Transformer Computation



(B x d_model) X (d_model x d_h)

(B x d_h) X (d_h x d_model)

- B : batch
- d_model : embedding table size
- d_h : hidden dimension per worker
- Input/output shape : B x d_model
- Weight shapes: d_model x d_h or d_h x d_model

Communication to compute ratio = 1 / (2 x d_h) bytes/flop

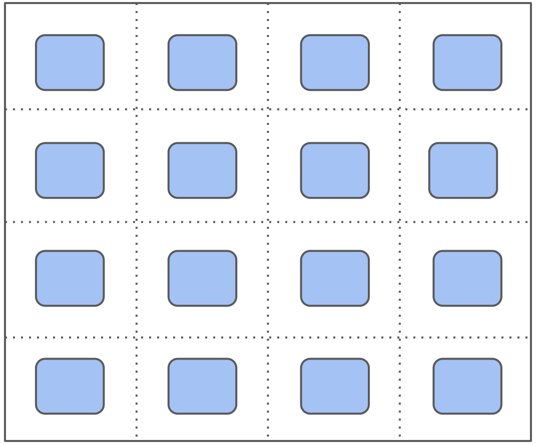
- For example, if hidden dimension size is 16k and we have 16 workers ratio is 4.88e-4
- For example, if hidden dimension size is 16k and we have 64 workers ratio is 1.95e-3

Giant Model Parallelism Techniques

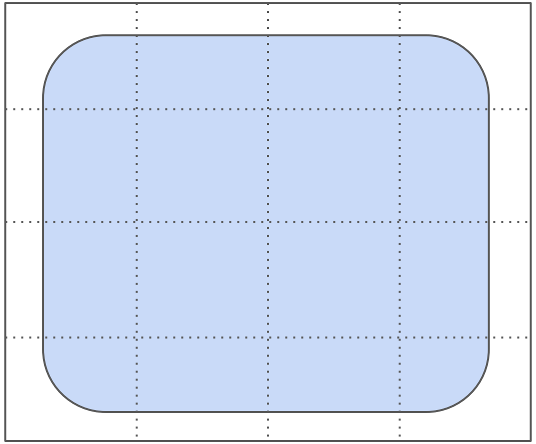
- Pipelining
    - Assign layers to different accelerators
- MeshTF : execute training on a virtual 2-D mesh
    - Model dimension
        - Shard the weights and activations along the feature dimension
        - Execute allreduce along the model dimension to sum partial results
    - Batch dimension: execute gradient summation along this dimension
    - https://arxiv.org/abs/1811.02084
- GShard: train a large number of mixture of experts
    - Extend model parallelism to all (or most) of the workers in the train job
    - All-to-all communication used to exchange activations between workers
    - https://arxiv.org/abs/2006.16668

Data vs Model Parallelism

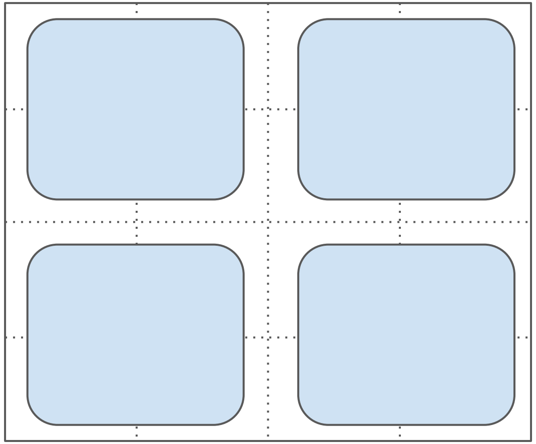
Sharding of Parameters between workers



Data Parallelism



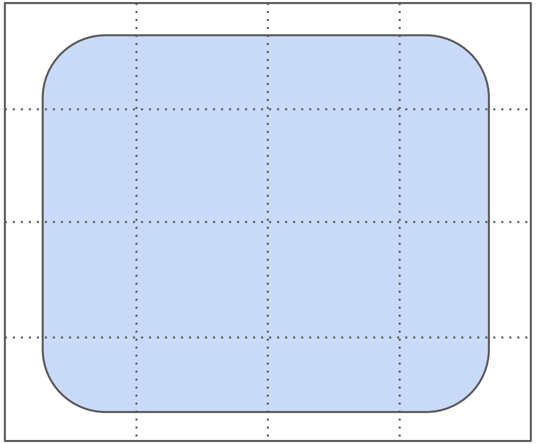
Model Parallelism



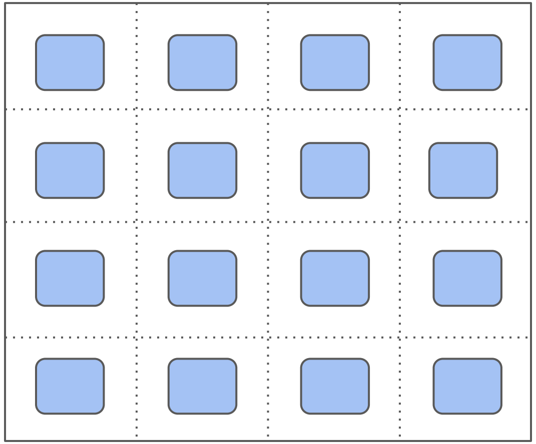
Mixed mode: Model + Data Parallelism

Data vs Model Parallelism

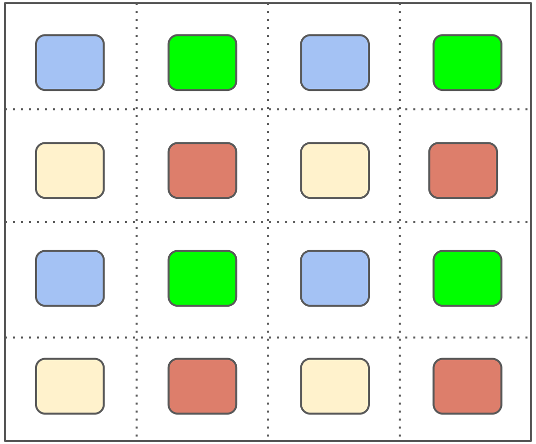
Sharding of Inputs between workers



Data Parallelism



Model Parallelism



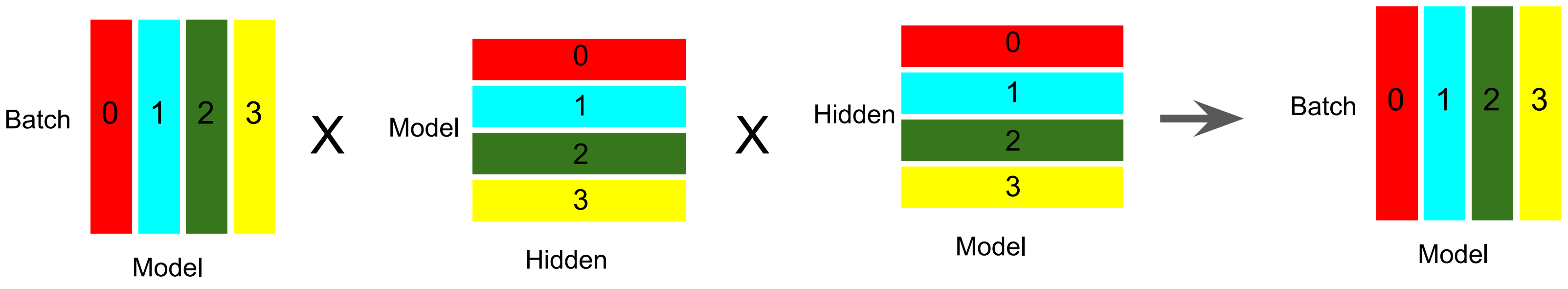
Mixed mode: Model + Data Parallelism

Google

Balance communication overheads vs space usage

- Remat vs communication overheads
- Pipelining
  - Point to point communication between neighboring layers
  - Entire layer must fit on a single accelerator
  - Scale limited to number of layers
- Weight sharding
  - Needs all-reduce (or a reduce-scatter) to sum partial results from workers
- Activation sharding
  - Needs an all-gather to concat activation contributions from all workers

Einsum Dot Sharding



AllGather collective to gather activation inputs

MatMul Compute

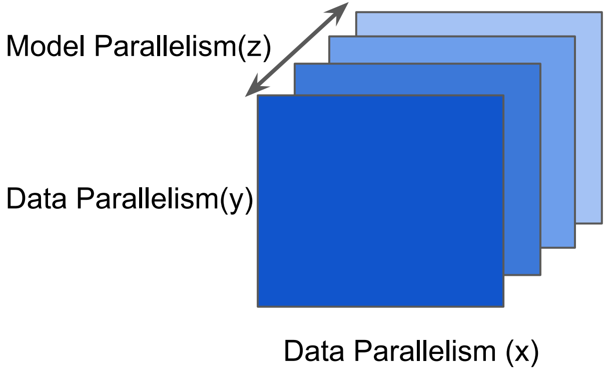
MatMul Compute

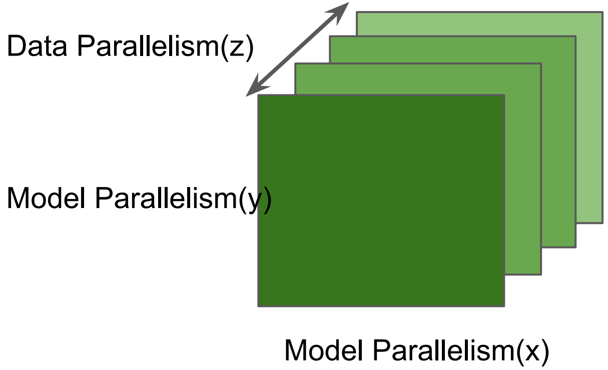
Reduce Scatter collective to sum partial contributions

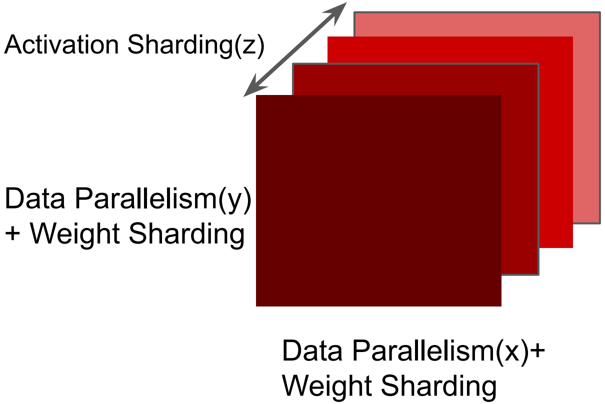
Related work: 1-D, 2-D and 2.5-D Canon's algorithms

[Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms Edgar Solomonik and James Demmel, EuroPar 2011]

Mapping Giant Models to N-D Meshes



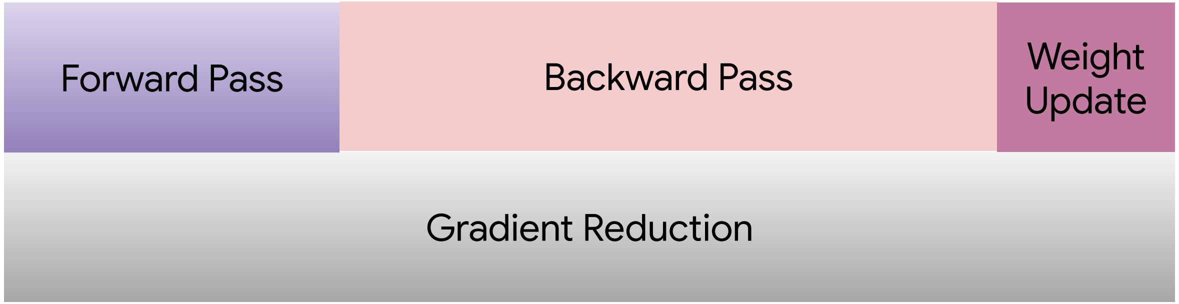




Scalability challenges

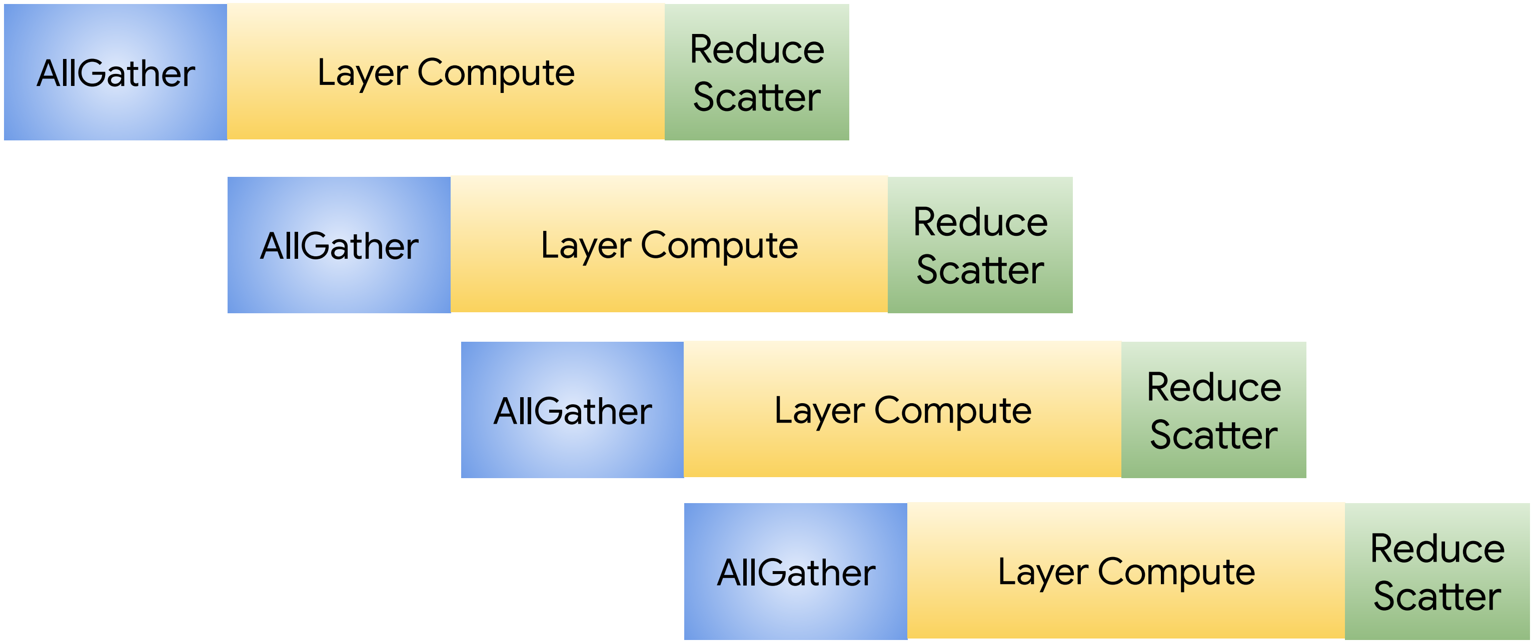
- Debugging
  - If model only trains on a very large scale debugging can be challenging
  - Use of alternative sharding techniques to run models at smaller scale can help debugging the model
- Fine tuning
  - Enable transfer learning
  - Ability to run models at smaller scale crucial here as well as datasets and activation sizes can be smaller here
- Overlapping computation with communication
- Strided collectives when inner dimensions of the tensors are split
  - Network transfers must access inputs and outputs with strides
  - DMA subsystem must achieve high throughput with strided access

Overlap communication with computation



Overlap gradient reduction with backward pass, weight update and forward pass in the next step

Over-decomposition to increase overlap



Split model on batch dimension to create the overlap opportunity
Over-decomposition proposed and used in the Charm++ programming model

Communication overlap mechanisms

- Dynamic co-processor mode
  - Selectively use one of the TPU cores as a communication co-processor
  - Suitable when communication overheads dominate or matmul is memory bounded
- Decomposed collectives
  - Schedule collective DMAs from outer loops of convolutions
  - Use double buffering to overlap computation and communication
  - Works well when transformer layer is dominated by matmul computation

Summary

- Giant models are challenging to optimize
- Communication overheads significant in Giant models
- Network throughput must scale with compute and number of accelerators
- Scaling giant models needs a cocktail of optimizations
- Can benefit from HPC literature on distributed matrix multiplication