Google

Exploring Limits of ML Training on Google TPUs

Sameer Kumar, Dehao Chen
{sameerkm, dehao}@google.com

Hot Chips 2020 Tutorial on “Machine Learning Scaleout”
08/16/2020

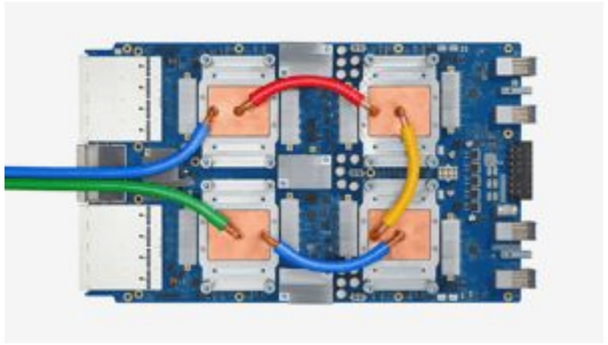
Overview

- Scalable Architecture
    - TPU-v3 **Multipods** with 4096 TPU-v3 chips
    - Fast and scalable collective implementation
- Scalability Techniques
    - Weight update sharding
    - Model parallelism
- Performance results and conclusion

Space-race for the biggest ML machine

- AI supercomputer at Azure with 10K GPUs
- NVIDIA DGX SuperPods with 2k GPUs
- Google TPU MultiPods with 4k chips

Tensor Processing Units (TPUv3)

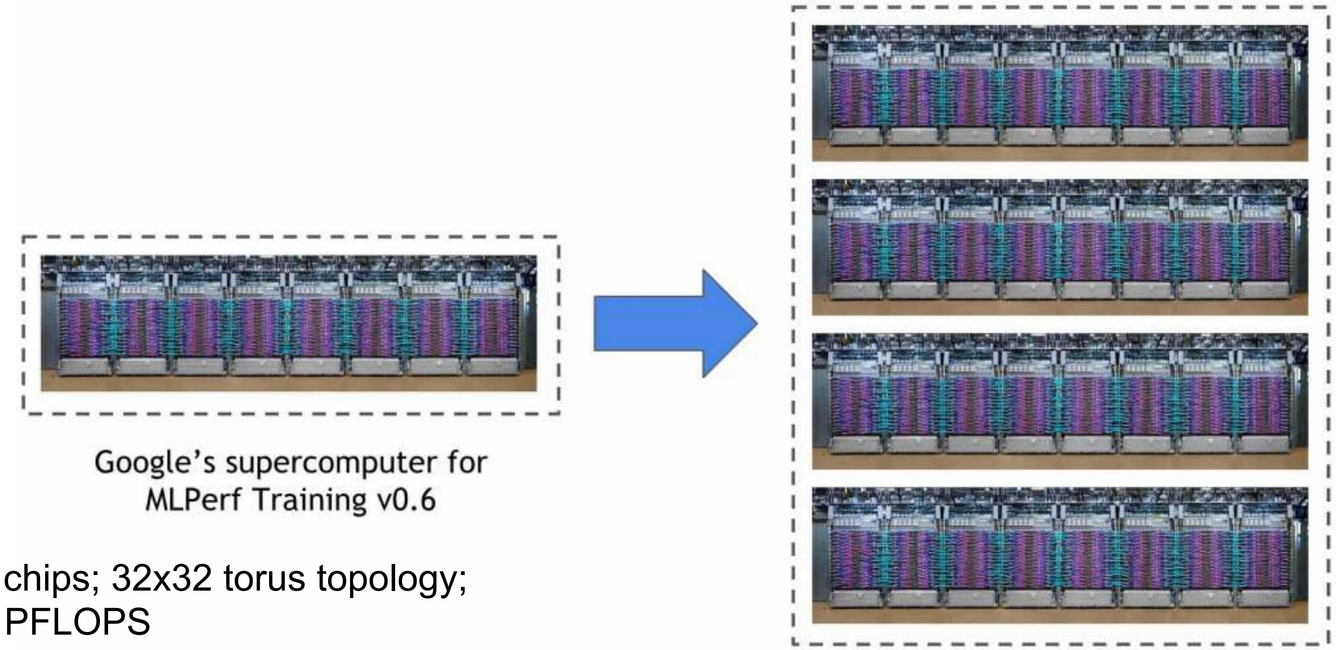


420 TFLOPS, 128 GB HBM

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

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

The Google TPU multipods



Google's supercomputer for MLPerf Training v0.6

Google's supercomputer for MLPerf Training v0.7

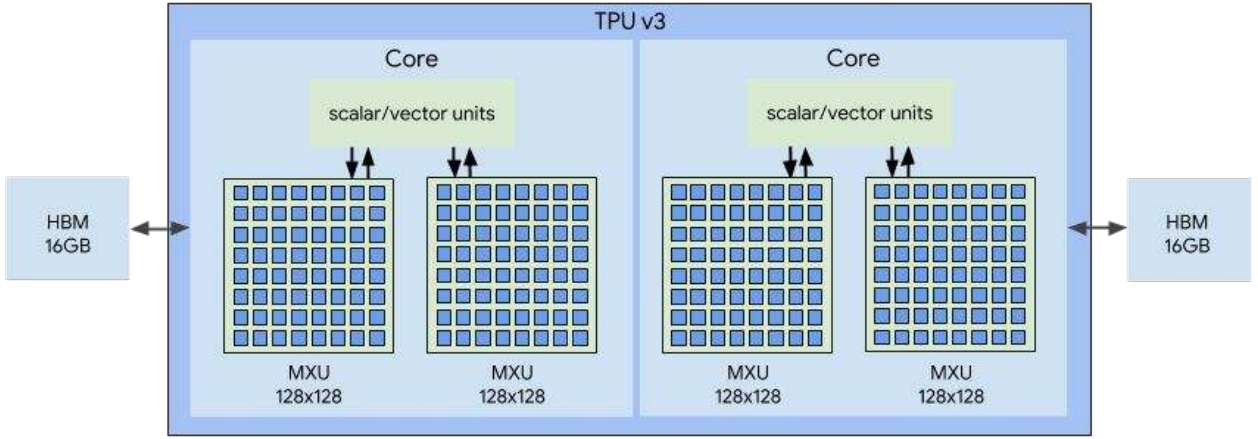
1024 chips; 32x32 torus topology; 100+ PFLOPS

4096 chips; 128x32 mesh topology; 400+ PFFLOPS

TPUv3: System Architecture

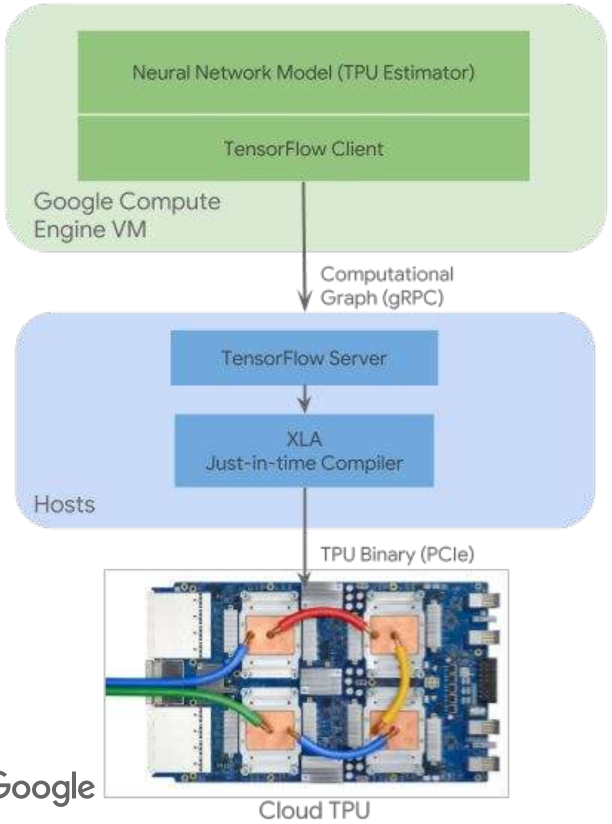
Key Features

- 128x128 Systolic arrays provide the massive horsepower; bfloat16 numerics
- Scalar, vector units to perform data formatting and non-matmul operations
- HBM accessed via on-chip interconnect



Google

TPUv3: Software



Key Features

- Models → TensorFlow → XLA → TPU instructions
- Just-in-time compiled, launched synchronously on a “slice” of a TPU Pod
- Several communication primitives

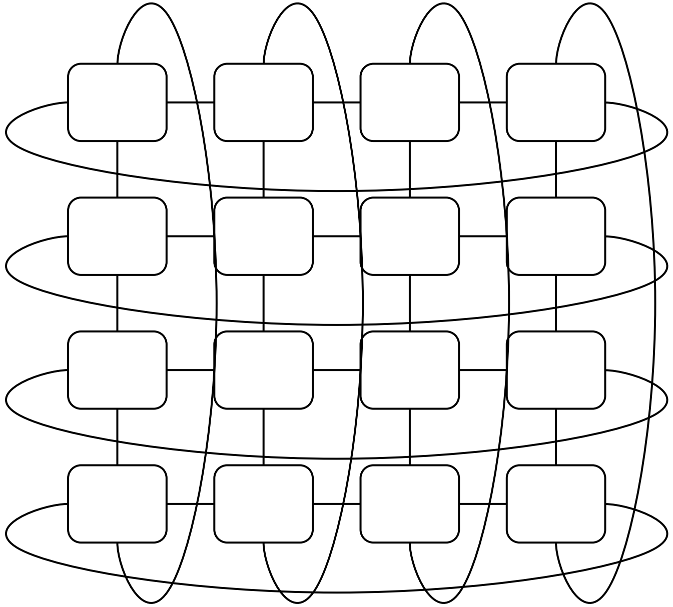
Multiple programming paradigms on TPUs

- TF
  - TF 1.x and TF 2.x enabled on TPUs
- JAX
  - Enable high performance machine learning research through composable transformations of Python and NumPy functions
- Pytorch

Challenges of large scale training

- Accelerate throughput of a large mini-batch SGD training
  - Execute linear algebra at high throughput on an accelerator (XLA Compiler)
  - Execute fast gradient summation
- Increase availability of this large machine : the fire fighter approach
- Scalable launch of ML Ops
- Scalable weight initialization
- Optimized host input pipelines

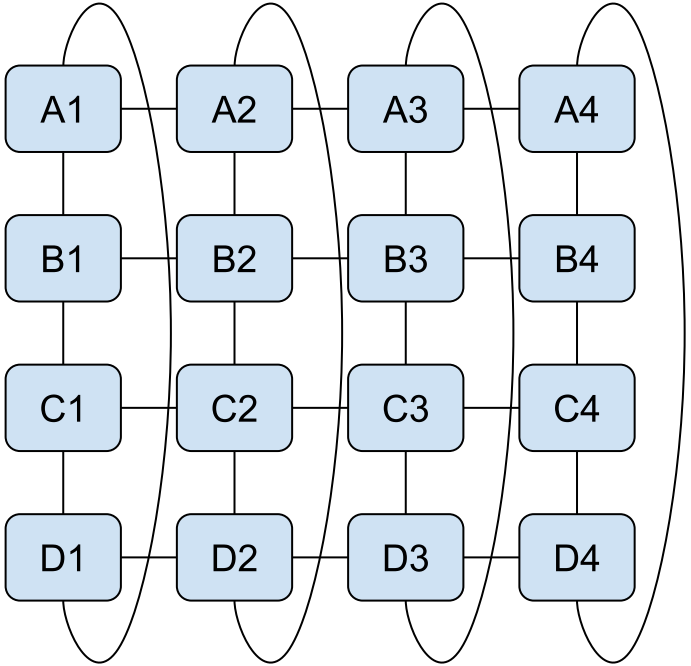
N-D Mesh/Torus Network Overview



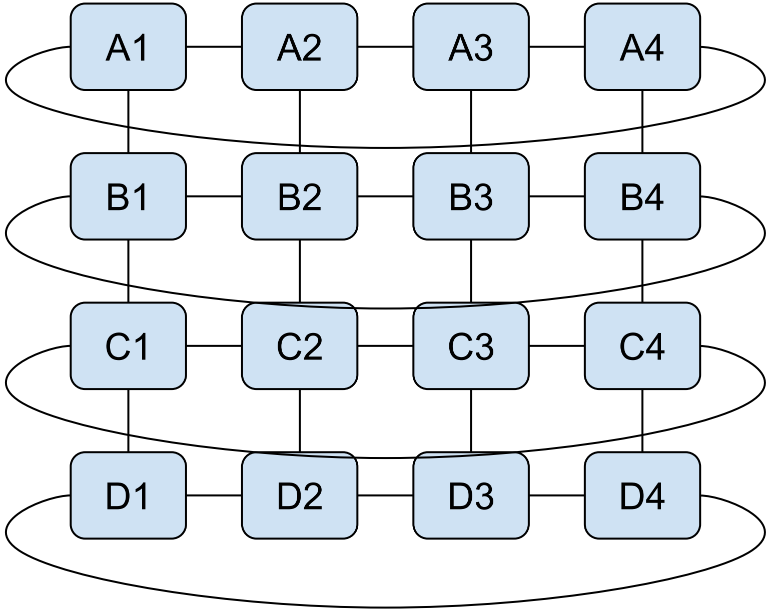
- Nodes connected 2*n neighbors via bidirectional network links
- Optimized for near neighbor communication
- All-to-all traffic can be bisection limited

Allreduce on a 2-D Torus

Phase 0: Execute sums along columns



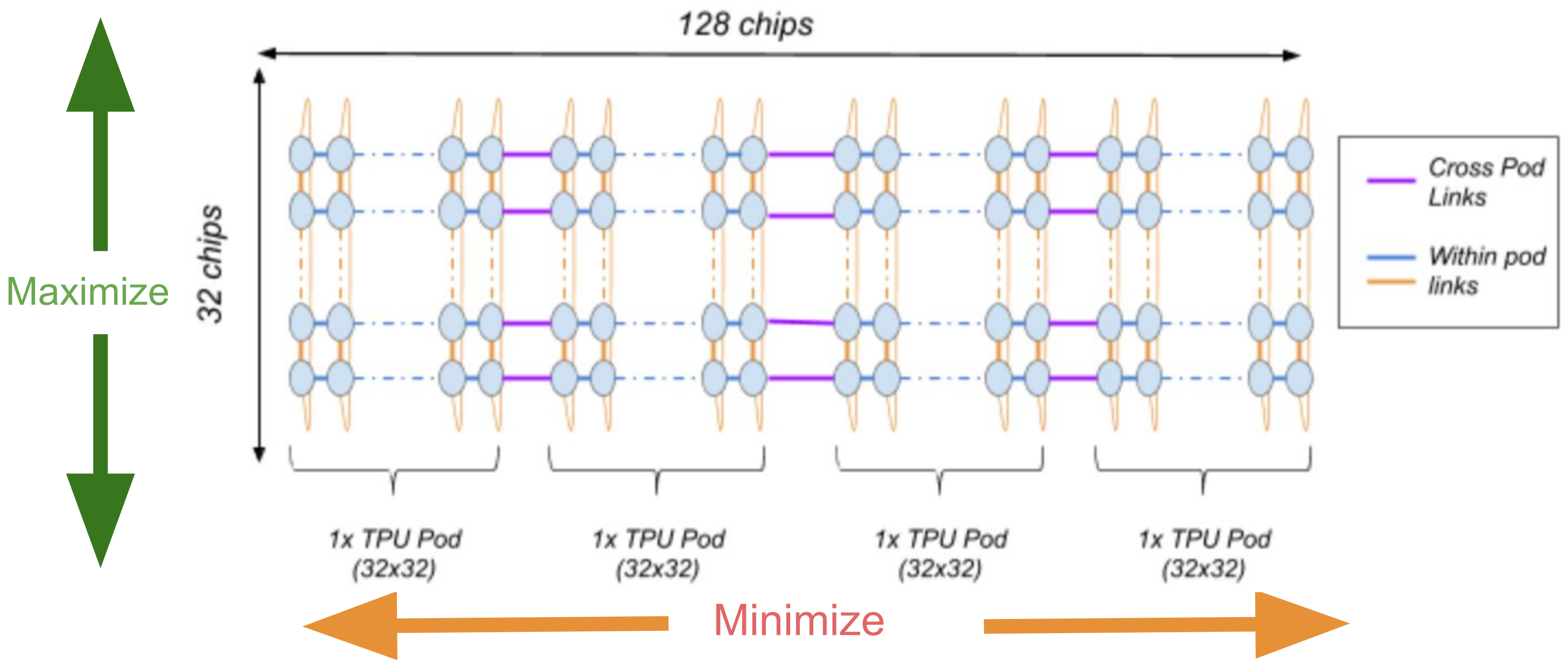
Phase 1: Execute sums along rows. Payload in phase 1 is scaled down by the size of the columns



We enable global summation in float32 and bfloat16 precision

Google

Communication Scaling on Multipods



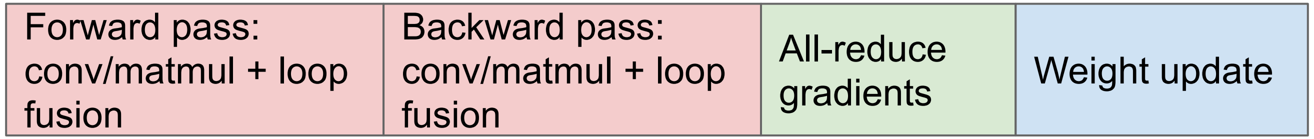
Google

Scalability Techniques

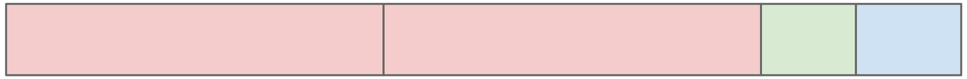
How do we scale training to this large system

- Compiler (XLA) automatically optimizes for scalability
  - Automatic fusion to overlap different computations
  - Automatic layout optimization to minimize data formatting
  - Automatic memory locality optimization to get most out of SRAM
  - Automatic cross replica optimization to maximize parallelism
- Simple API with compiler optimization to partition the model
  - Flexible annotation on a small part of the model
  - Automatic propagation to fully partition the model
  - Model compiled by SPMD to achieve good compile time and run time performance

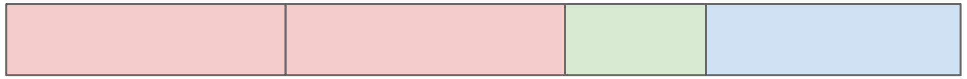
Inside a training step


Forward pass: conv/matmul + loop fusion
Backward pass: conv/matmul + loop fusion
All-reduce gradients
Weight update

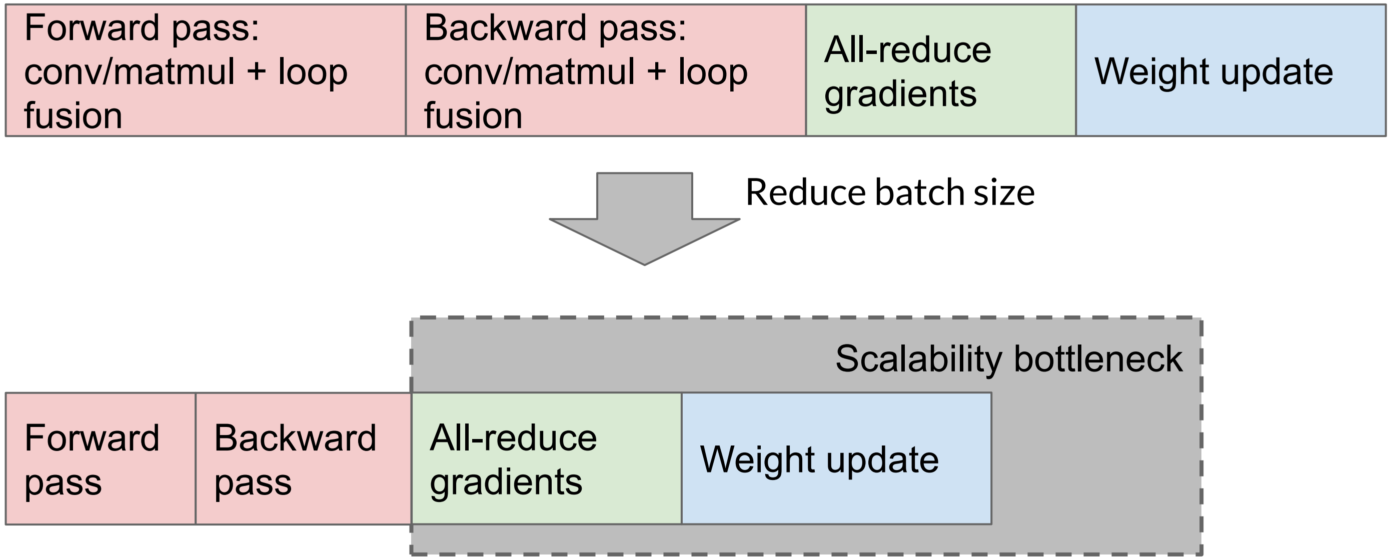
Typical image models: small weights, large input



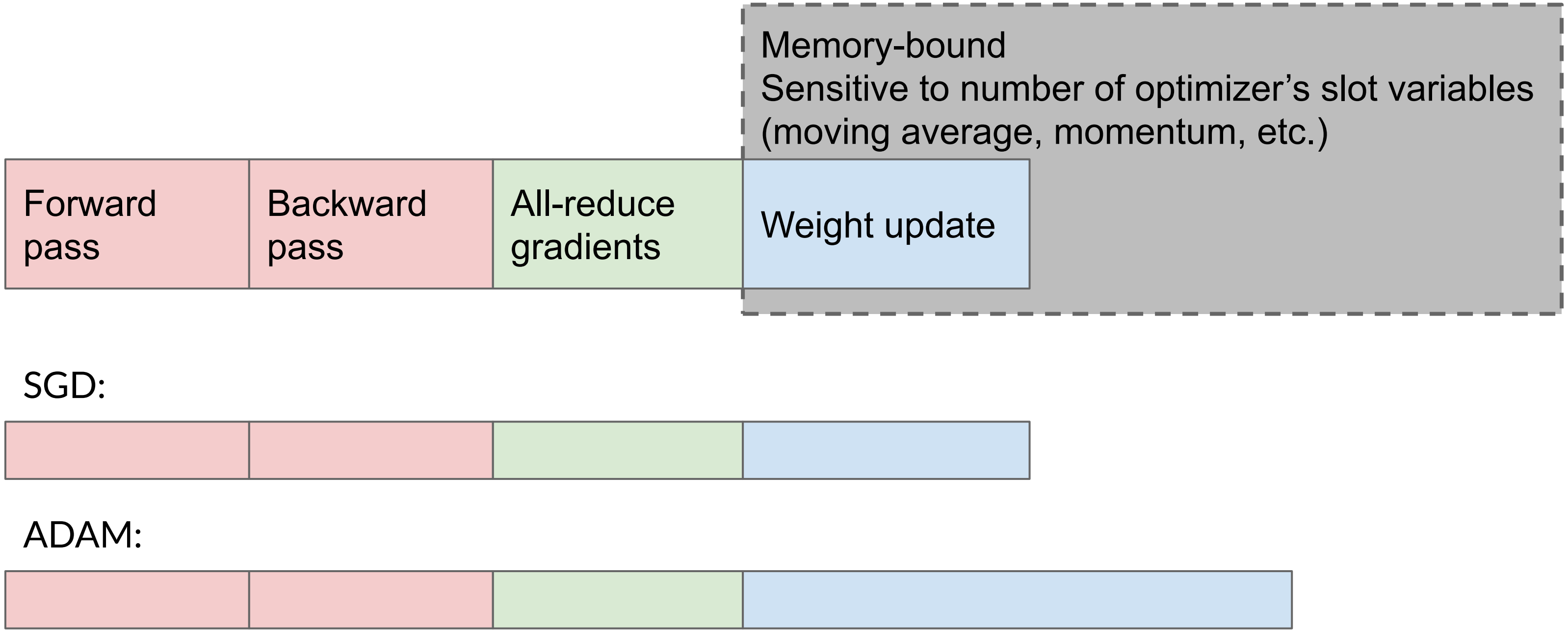
Typical sequence models: large weights, small input



Batch only affects forward/backward passes

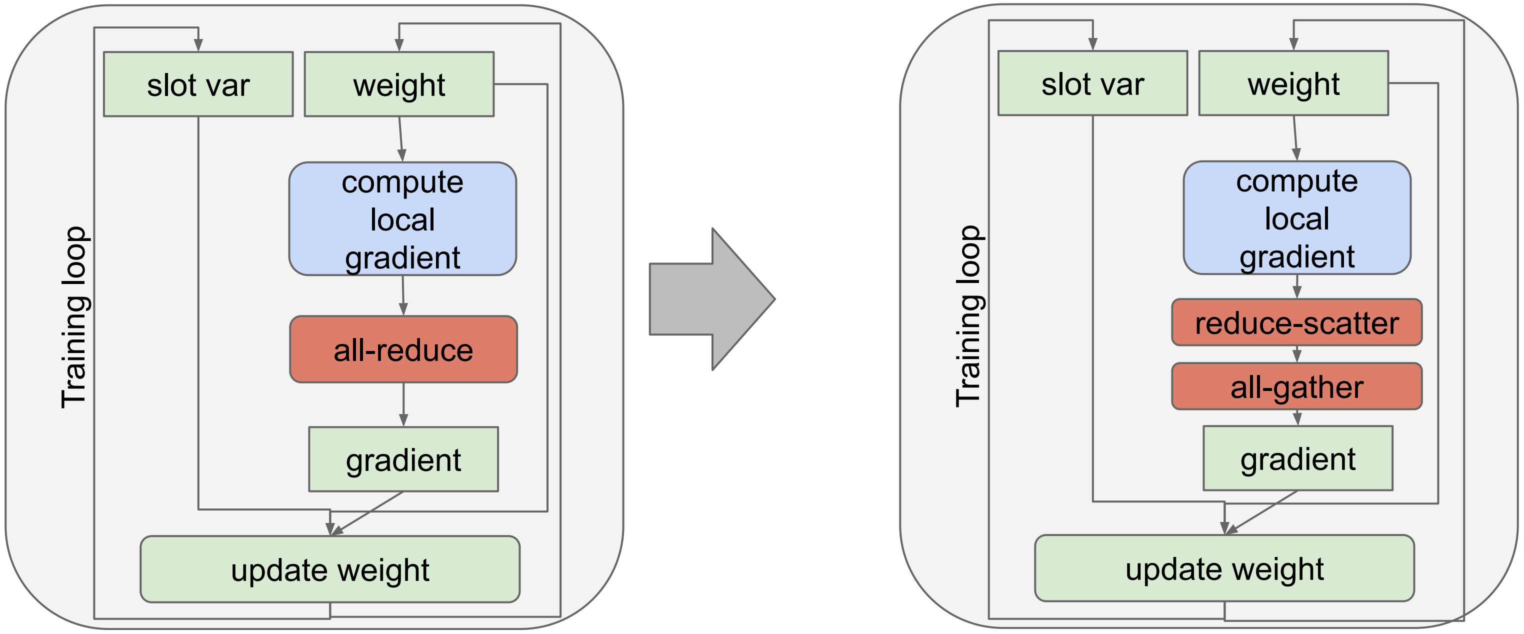


Weight-update time sensitive to optimizers


Memory-bound
Sensitive to number of optimizer’s slot variables (moving average, momentum, etc.)
Forward pass
Backward pass
All-reduce gradients
Weight update
SGD:
ADAM:

Weight-update sharding

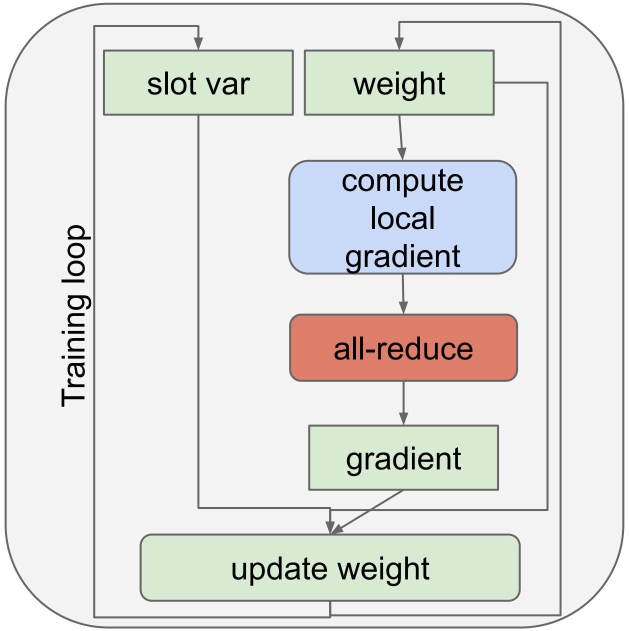
Replicated weight update

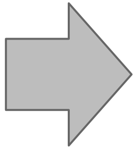


[Yuanzhong Xu et al: Automatic Cross-Replica Sharding of Weight Update in Data-Parallel Training]

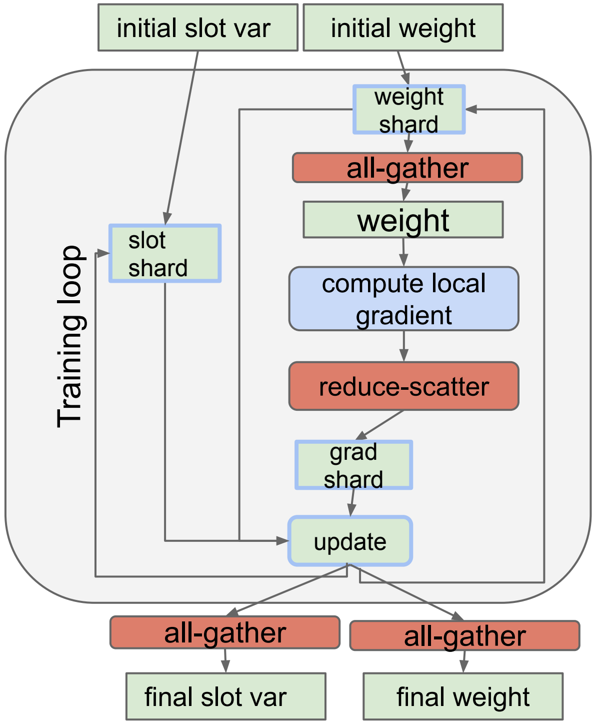
Weight-update sharding

Replicated weight update





Sharded weight update

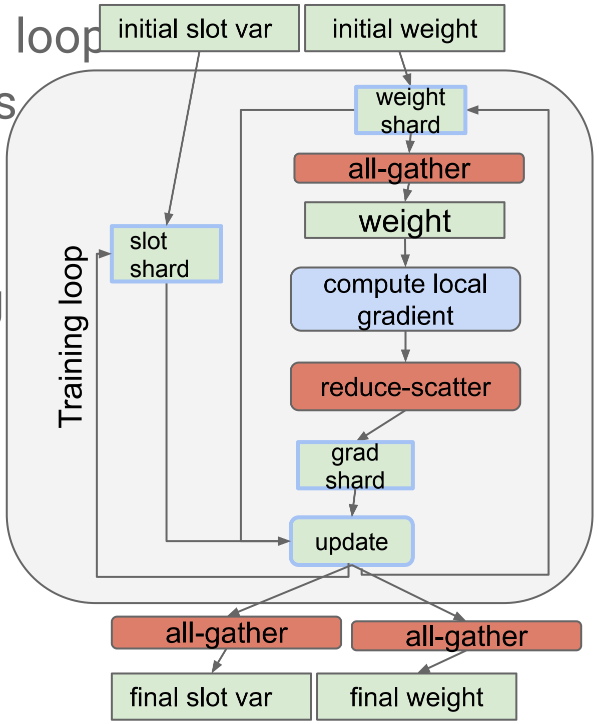


[Yuanzhong Xu et al: Automatic Cross-Replica Sharding of Weight Update in Data-Parallel Training]

Google

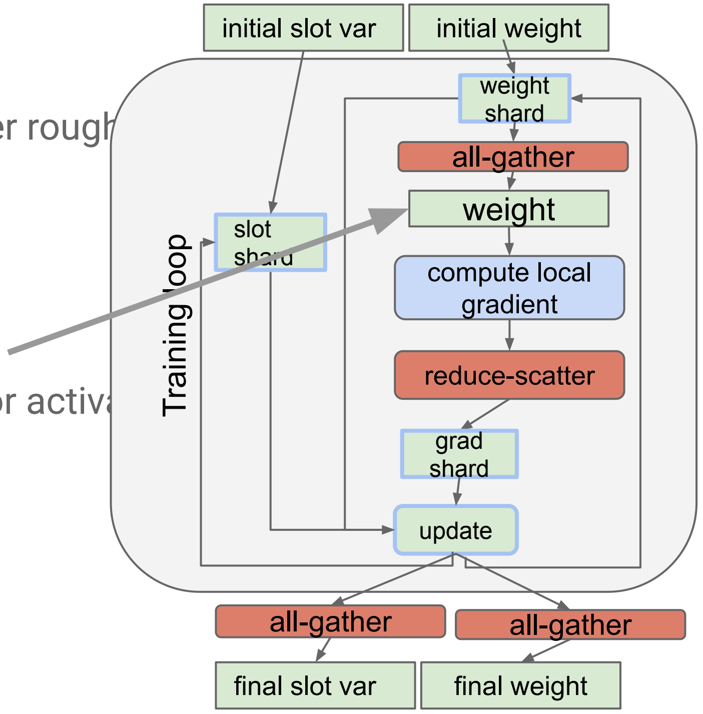
Weight-update sharding

- Weight and slot vars are sharded before training loop
- All-gather right before forward/backward passes
- Reduce-scatter on gradients
- Weight-update on shards
- An automatic optimization with no model chang



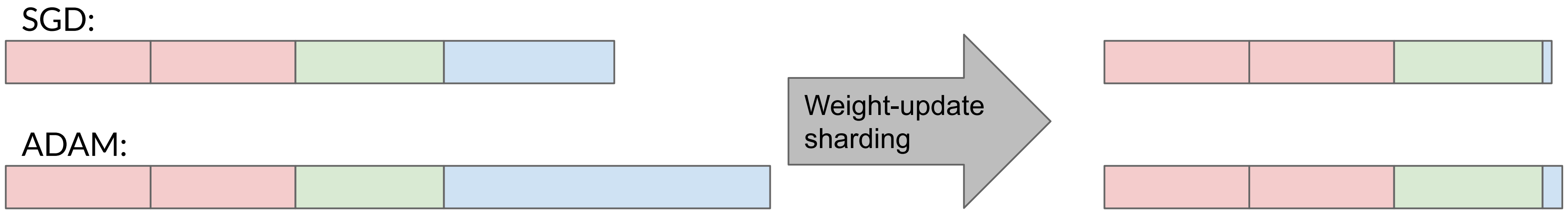
Weight-update sharding

- Performance
    - Weight-update takes much less time
    - In each step: 1 reduce-scatter Inside + 1 all-gather rough
    - Automatically convert weight to BF16
        - Faster all-gather
        - Faster convolutions
- Memory saving
    - Slot vars are sharded, memory could be reused for activa



Weight-update sharding

- Step time will be less sensitive to optimizers: weight-update is already fast



Use Model Parallelism for Small Batch per Core

Graph Partitioning: place subgraphs of operators across different cores (e.g. Inception)

Spatial Partitioning: partition individual operator (e.g. conv) across different cores (e.g., ResNet-like)

GPipe: place layers on different cores and pipeline the execution (e.g., RNN)

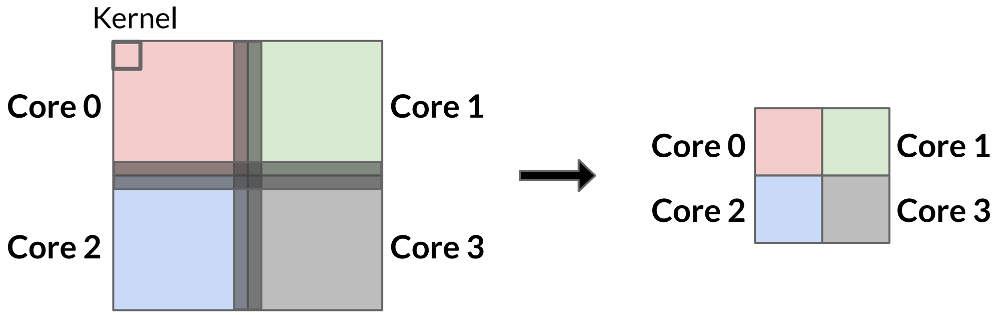
GShard: partition individual operator across different cores using data-parallelism (e.g., Xformer)

Ref: TPU Model Parallelism

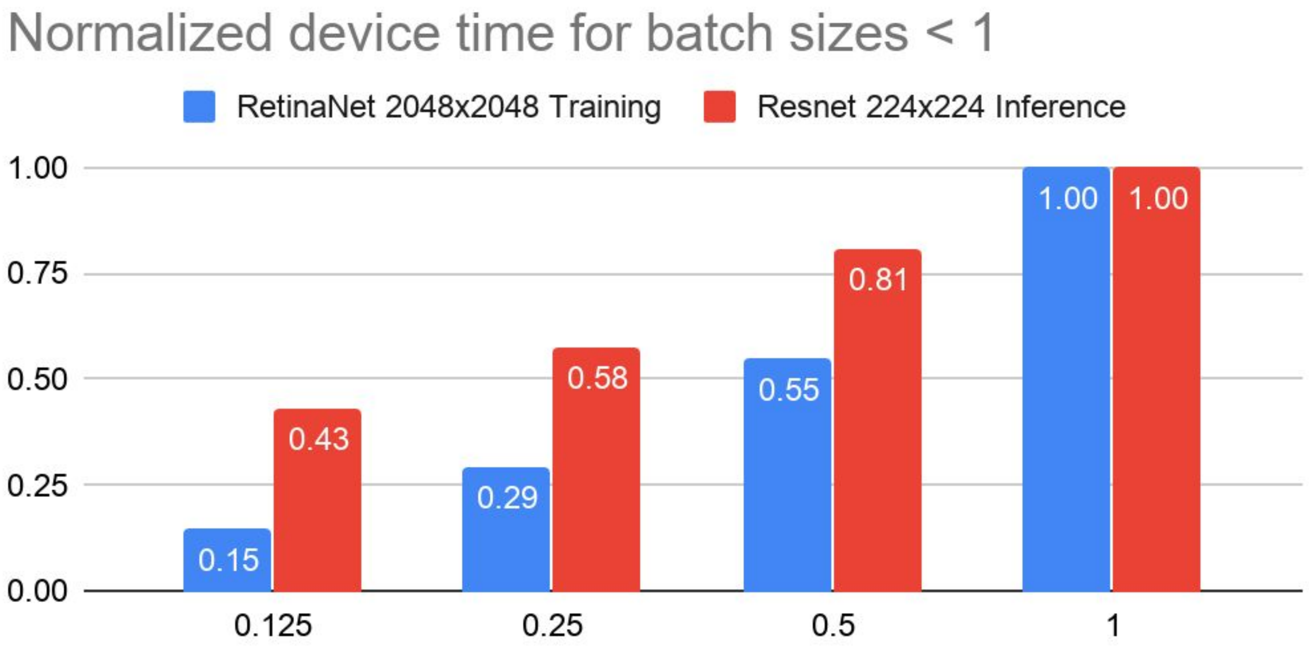
Spatial Partitioning for Image Models

- Halo exchange is required for overlapping windows
- Weights are replicated; only activations are partitioned

```
tpu_config=tpu_config.TPUConfig(
    iterations_per_loop=100,
    num_shards=2,
    num_cores_per_replica=4,
    input_partition_dims=[[1, 2, 2, 1], None]]
)
```

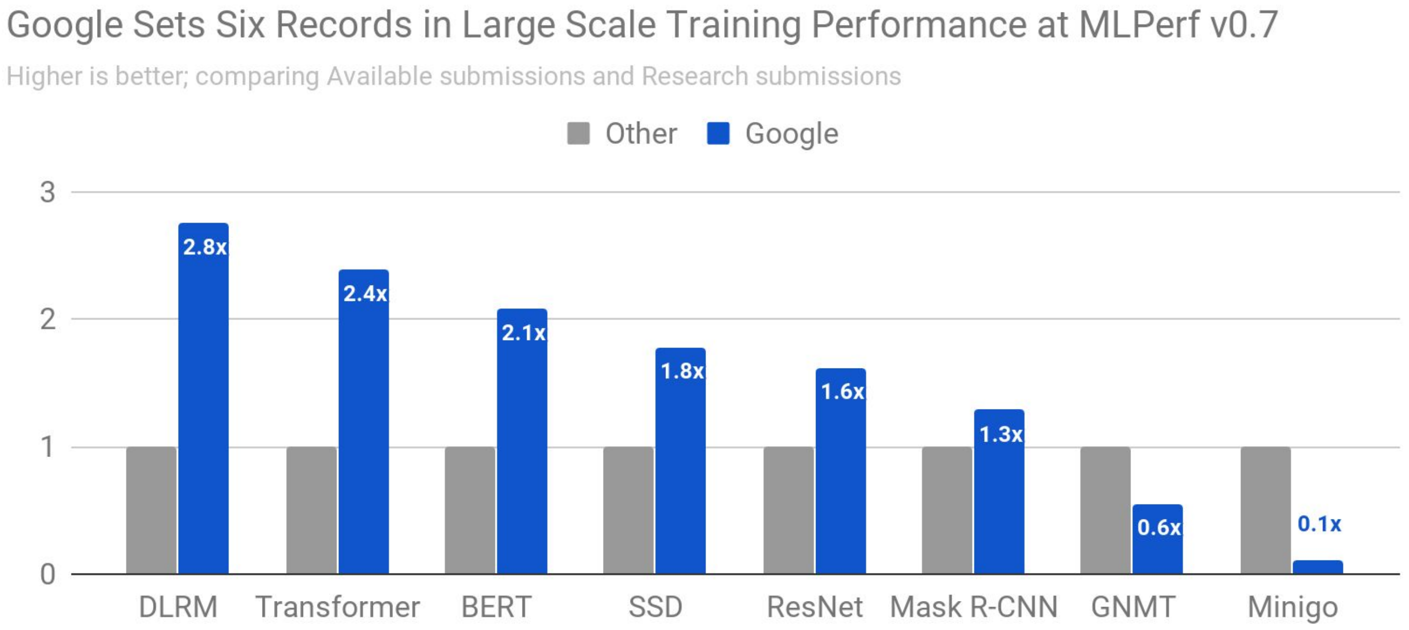


An example of 4-way convolution partitioning


Normalized device time for batch sizes < 1
RetinaNet 2048x2048 Training
Resnet 224x224 Inference
1.00
0.75
0.50
0.25
0.00
0.15
0.43
0.29
0.58
0.55
0.81
1.00
1.00
0.125
0.25
0.5
1

Google

MLPerf 0.7 results



Takeaways

- Scaling workloads to large systems is hard
- Fast and flexible interconnect is essential
- Software innovations can remove bottlenecks in scaling
- Model parallelism helps to further scale up

TPUs are available on Google Cloud