ParaGAN: A Cloud Training Framework for Generative Adversarial Networks

Ziji Shi[1,2], Fuzhao Xue[1], Jialin Li[1], Yang You[1]
[1]National University of Singapore, [2]Alibaba Group

Challenge: Large-Scale GAN Training

a. Numerical instability

- Generative Adversaria Network has two subnetworks:
  - Generator **(G):** synthesize images
  - Discriminator **(D):** distinguish fake images from the real ones
- Convergence is often not guaranteed
  - Two subnetworks are optimized for contradicting goals, and the convergence is defined as a Nash Equilibrium
  - **G** may be overly optimized for **D**, producing poor image results

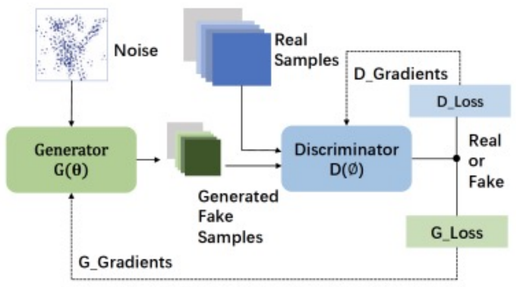


Fig. 1: GAN training process.

b. System challenges for large-scale training

- Successful GAN models take days to weeks to train
- Issues with datacenter-level distributed training:
  - Network congestion prolongs unpredictable latency
  - Each HW accelerator has different characteristics
- Existing works emphasize little on training speed

Table 1: Training Time and Parameters Number for GANs trained on ImageNet 2012 dataset.

GANs	Hardware	Training Time	Parameters
SNGAN [40]	8 V100 GPUs	3d 13.6h	81.44M
SAGAN [54]	8 V100 GPUs	10d 18.7h	81.47M
BigGAN [4]	8 V100 GPUs	30d 6.6h	158.42M
ContraGAN [24]	8 V100 GPUs	5d 3.5h	160.78M

Approach: Numerical and System Optimization Co-Design

a. Numerical optimizations for training stability

- Asynchronous Training
- Asymmetric Optimization

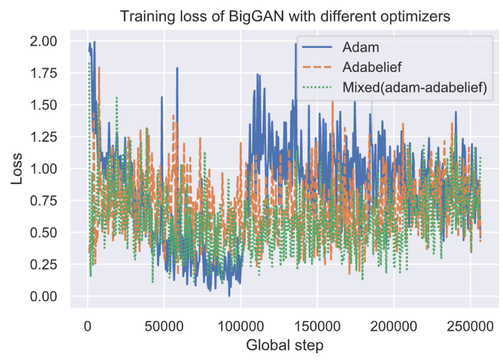


Fig. 2: Asymmetric optimization.

a. System optimizations for scalability

- Model size is rather small => Data parallelism suffices
- Congestion aware data pipeline
  - Dynamically adjust the prefetch buffer size
  - Moderate the impact of network jittering
- Hardware-aware layout transformation
  - Improve accelerator utilization
    - minimize zero padding
    - convert tensor to HW-friendly data layouts
  - Support CPU/GPU/TPU

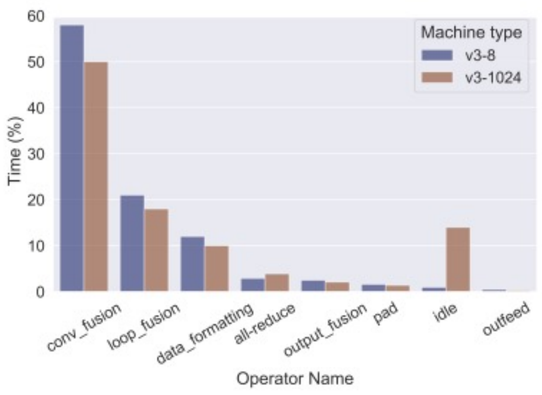


Fig. 3: Profiling results of individual operators on TPU. Model shown is BigGAN.

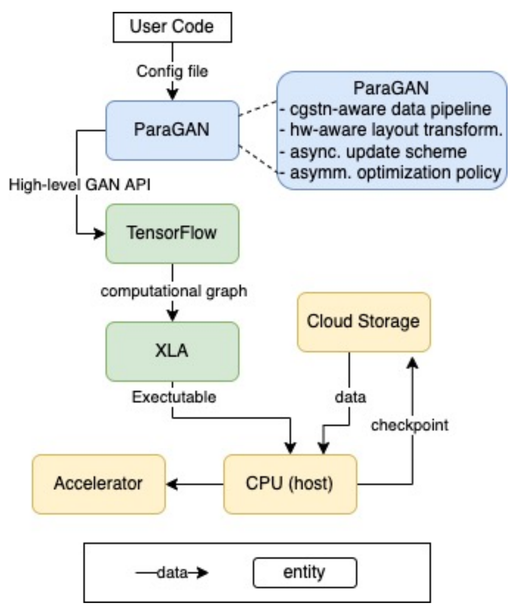


Fig. 4: Overview of ParaGAN.

Example and Implementation

```
import paragan as pg

class Generator:
  def model_fn(latent_var, y):
        # generator model
        return output

class Discriminator:
  def model_fn(x, y):
        # discriminator model
        return output, out_logit

scale_mgr = pg.ScalingManager(config=cfg,
                  bs=2048, num_workers=128)

g = Generator()
d = Discriminator()
gan = pg.Estimator(g,d)

# train
for step in cfg.max_steps:
  scale_mgr.train(gan)

# evaluate
scale_mgr.eval(metric='fid')
```

Fig. 5: ParaGAN example.

a. Scaling manager

- In charge of hyper-parameter tuning
- Scaling strategy is set based on the profile collected from the single-worker

b. Network backbone

- Implements several popular GAN backbones
- Easily extensible to other architectures

c. Evaluation metrics

- Standardized evaluation metrics
  - Frechet Inceptions Distance(FID),
  - Inception Score (IS)
  - …

Asynchronous Training

a. Motivation

- Previous works optimize both generator and discriminator in the same manner
- Can we decouple it by optimizing **G** and **D** separately?

b. Asynchronous training

- Let **G** and **D** run on different nodes within the same TPU Chip (each chip has two nodes)
- Sync gradient globally every other step, and cache the generated fake image
- Why it works?
  - Add perturbation to the gradient (no long update each other immediately)
  - Force **G** not to mimic a limited set of modes
- Also save memory and communication => enables the training of larger models

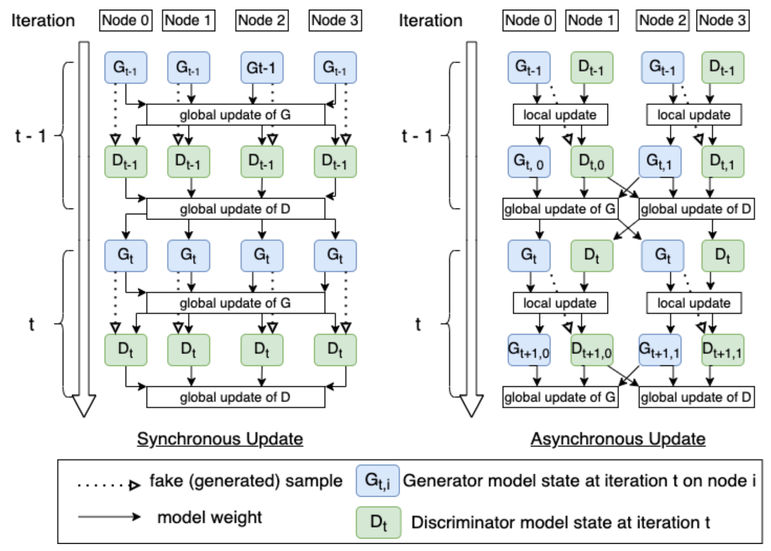


Fig. 6: Async update scheme.

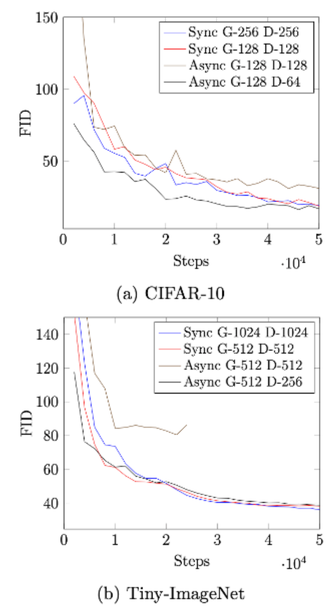


Fig. 7: FID of both update schemes. Lower is better.

Evaluation

1. BigGAN end-to-end training: reduced from 15 days to 14 hours.
2. Scales to 1024 TPU nodes at 91% efficiency.
3. Enables direct generation of 1024x1024 images.

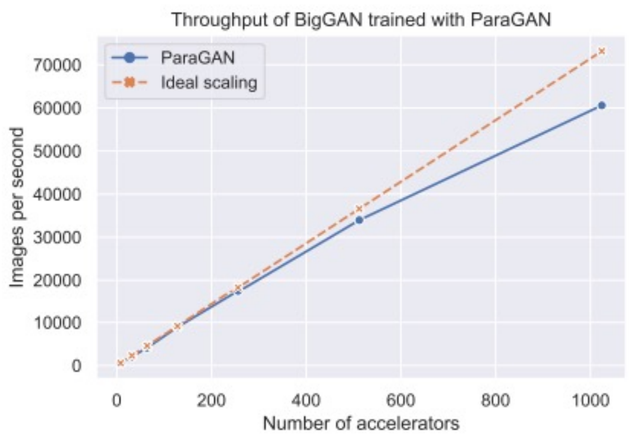


Fig. 8: Scaling on 1024 TPU nodes.

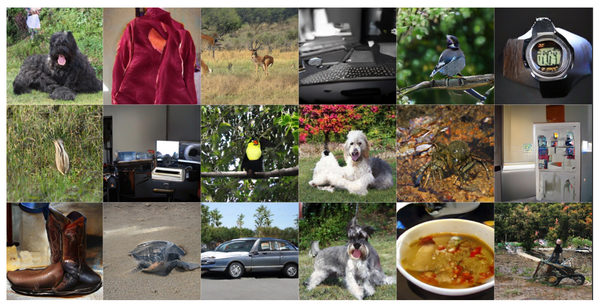
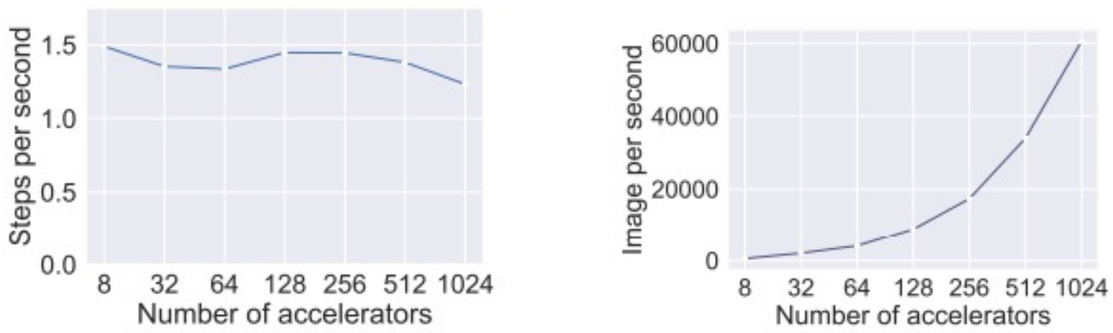


Fig. 9: Image generated at 1024x1024 resolution.



a) Steps per second.

b) Images per second.

Fig. 10: Weak scaling on BigGAN (128x128)

Limitation & Future Work

1. More GAN architectures shall be evaluated.
2. Scale on larger clusters made of GPU
3. Adapt to stable diffusion models