CERN openlab

Google

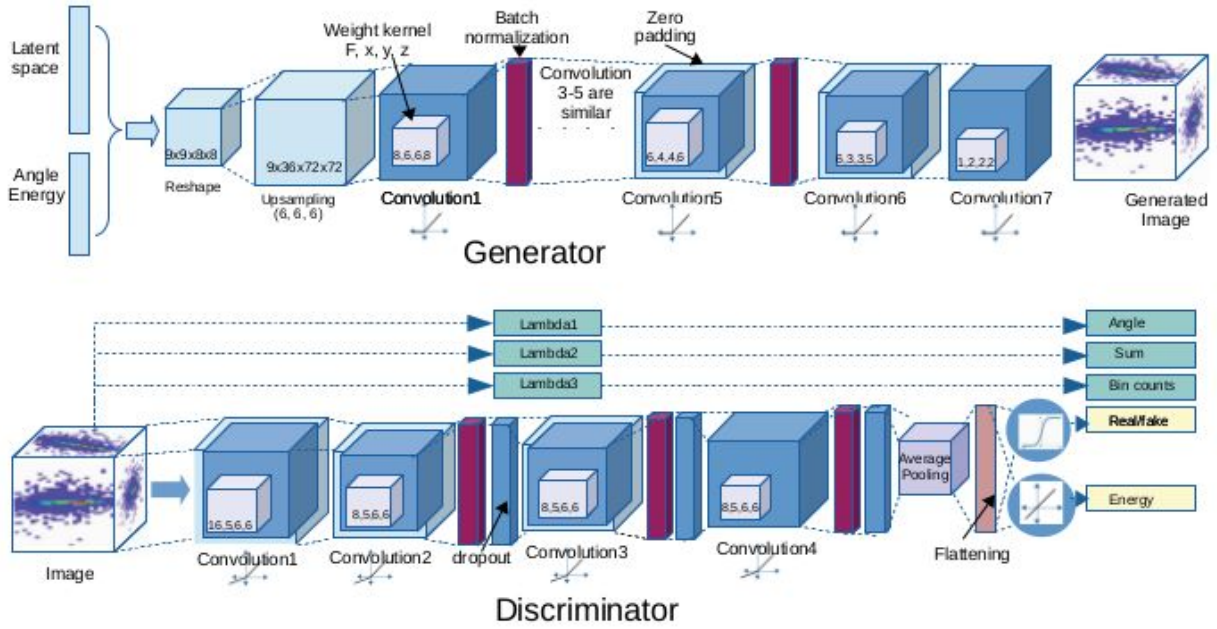
Accelerating GAN training using distributed tensorflow and highly parallel hardware

Renato Cardoso IT-DI-OPL, Sofia Vallecorsa IT-DI-OPL

IML 2020

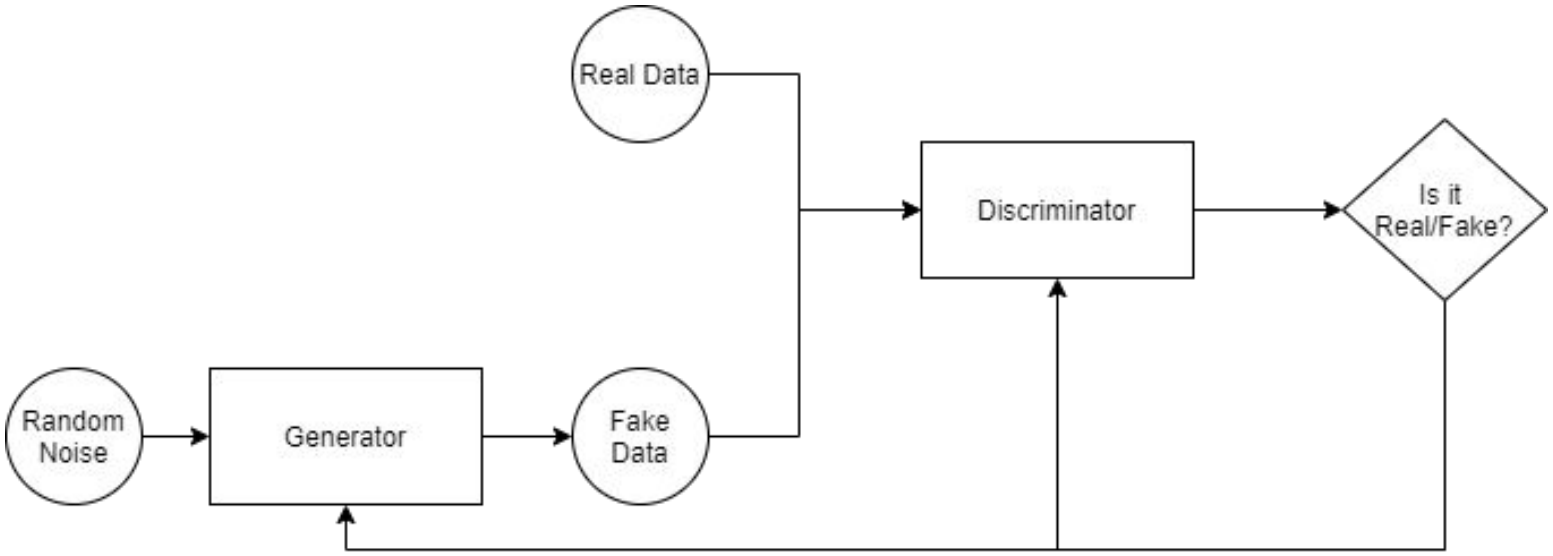
Architecture of the GAN

- 3D convolutional Generative Adversarial Network using physics constraints.
- Generates 51x51x25 pixels images, representing energy depositions in calorimeters, similar to the ones generated by Monte Carlo.



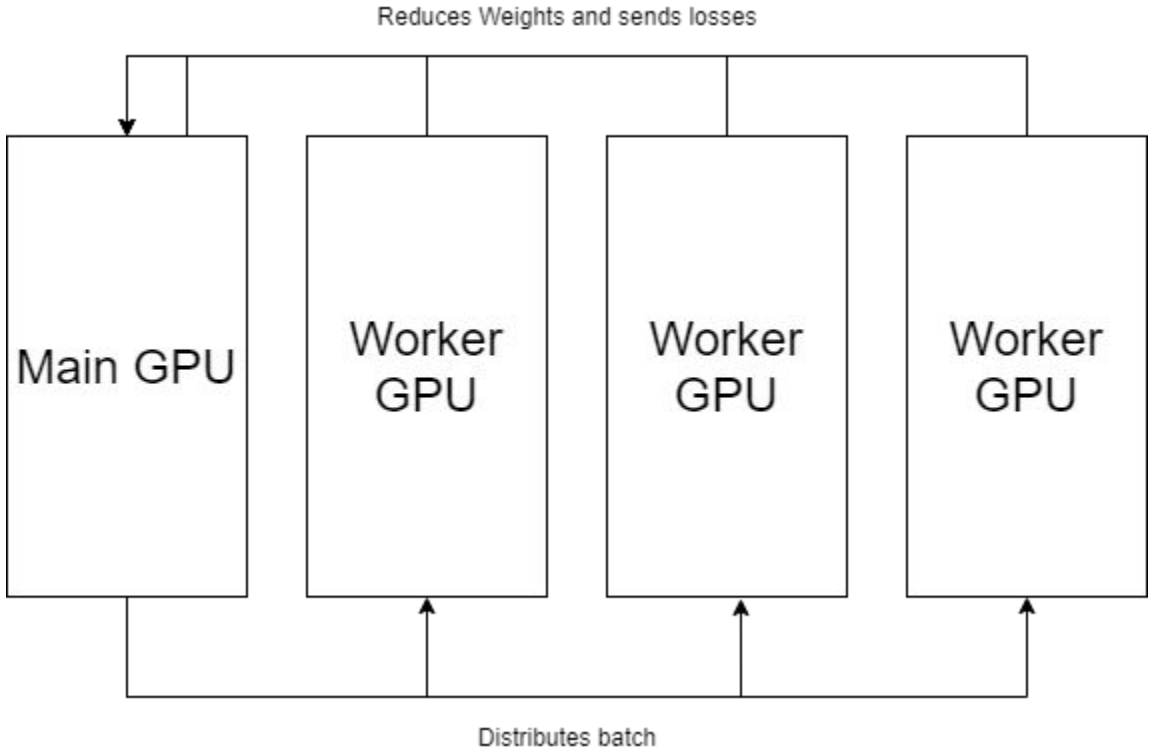
GAN Training - General Idea

- The GAN training is divided into two steps: the Discriminator Training and the Generator Training
- We train the Discriminator and then the Generator for each batch of data.
- This process of training is repeated for a specific amount of epochs.



GPUs - Parallel Approach

- Using Tensorflow Mirrored Strategy
- Mirrored Strategy uses synchronous training with NVIDIA NCCL as the all-reduce implementation
- Model needs to be initialized in strategy scope
- Tensorflow Train_on_batch function takes care of distributed training.



GPU Configuration

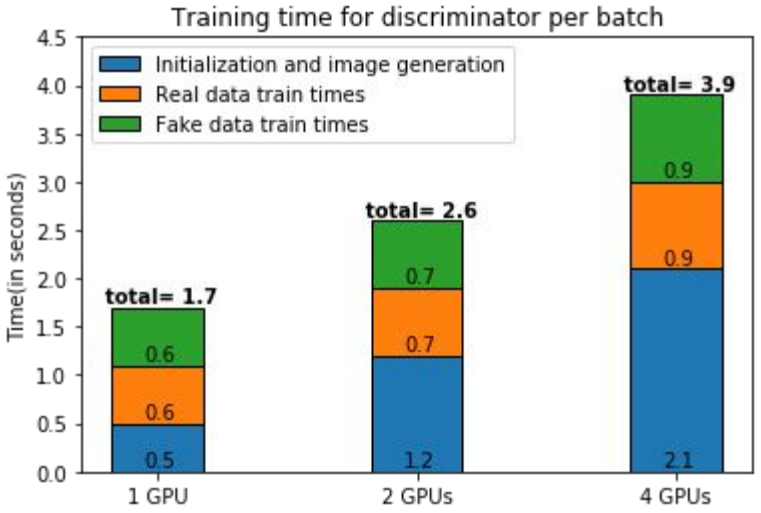
- The models were run using the Mirrored Strategy to implement parallelization across the multiple GPUs
- We used 1, 2 and 4 V100 GPUs present in Google Cloud
- The batch used for each GPU was 128, meaning that for each step in the training (update of the discriminator and the generator) the global batch size is 128 x N_{GPUs}.
- We measured the average time for each individual training for a batch of data, as well as the average training for a single epoch
- All times are in seconds

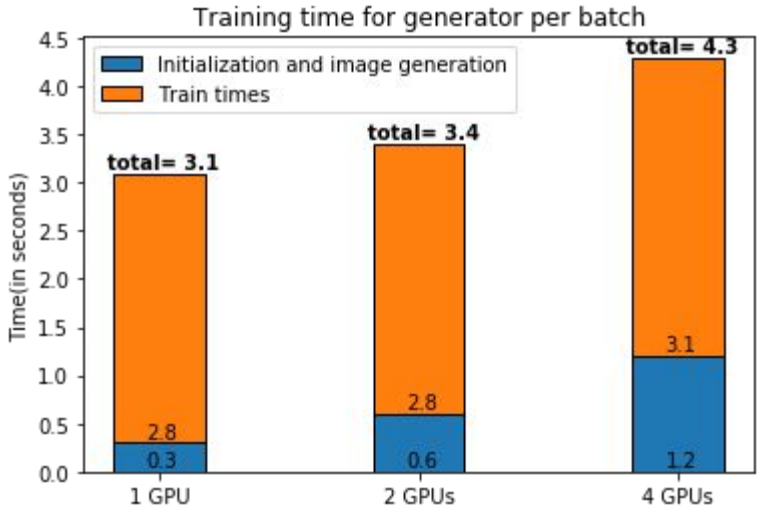
GPU Results

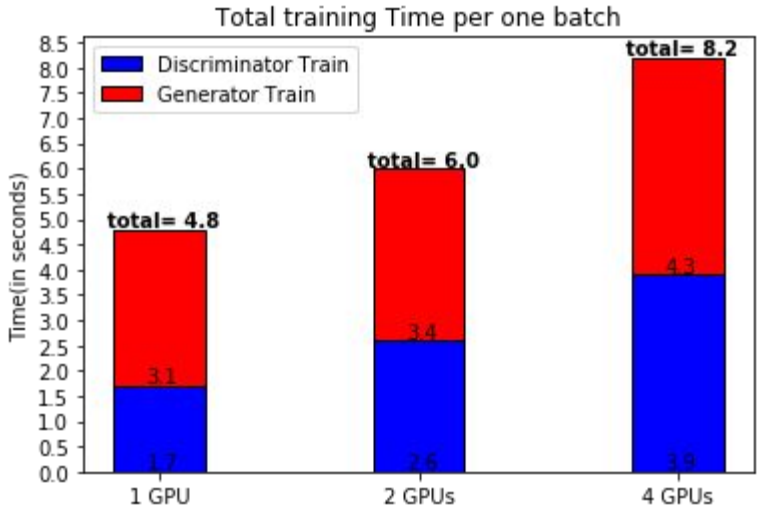
- Batch size:
  - 1 GPU = 1 x 128 = 128
  - 4 GPUs = 4 x 128 = 512.
- 4 GPUs process 4x more data than 1 GPU per batch
- 1 GPU needs 4.8 x 4 = 19.2 seconds to process the same data.

Average Time per epoch:

- 1 GPU: 4999
- 2 GPU: 3481
- 4 GPU: 2565

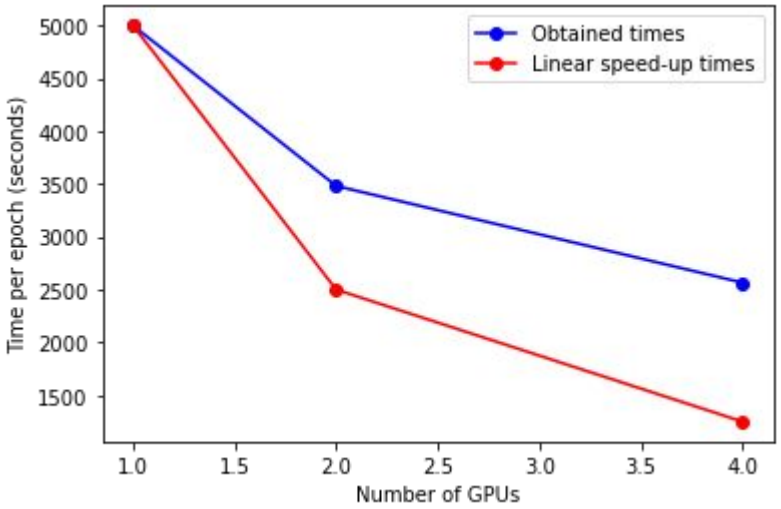






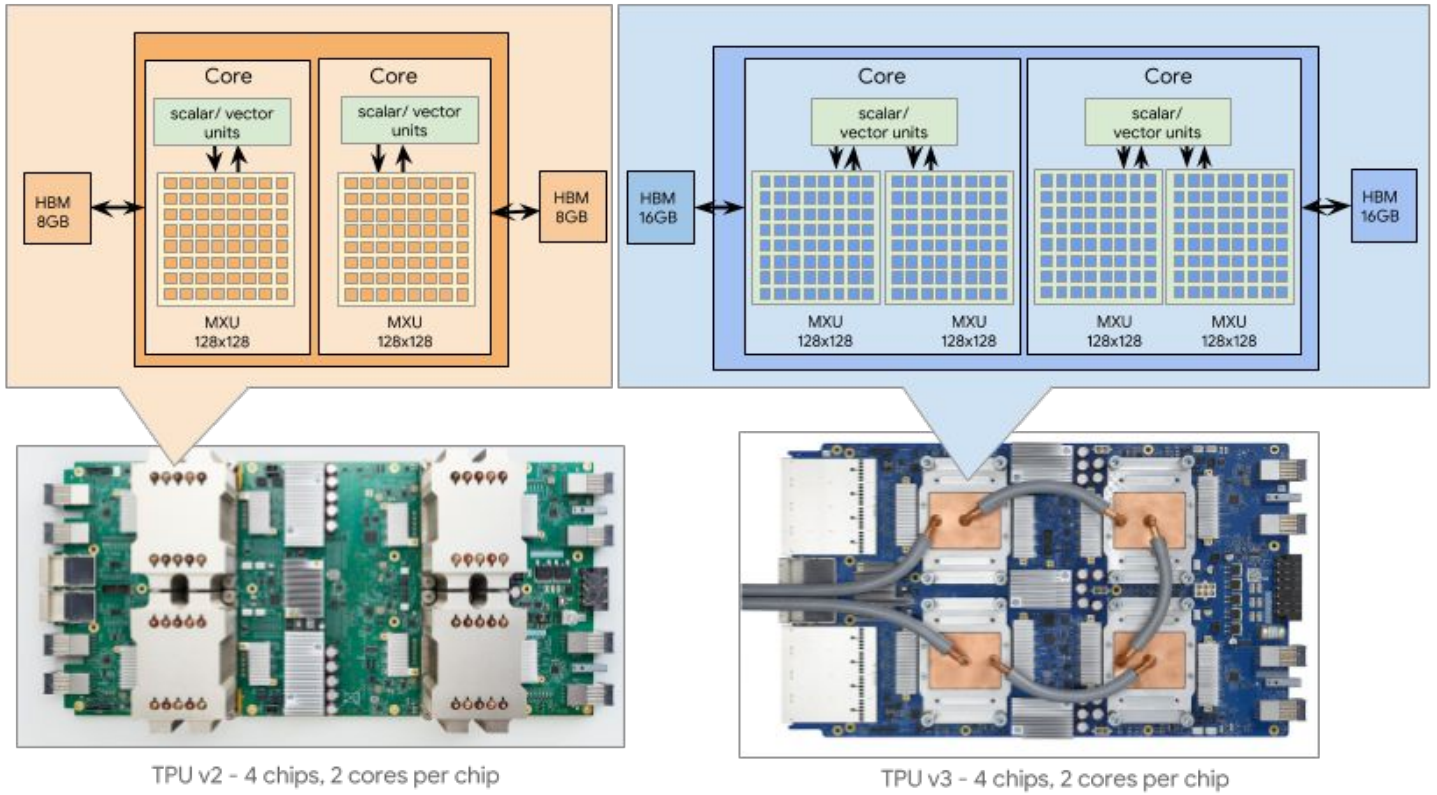
GPU Conclusions

- Overall Decrease in Training time per epoch 4999 -> 3481 -> 2565
- Small variances of training time while increasing number of GPUs 0.6 -> 0.7 -> 0.9
- Bottleneck in initialization of data and image generation
- Times obtained are not near linear speed-up.



TPUs - Tensor Processing Units

Tensor Processing Units (TPUs) are application-specific integrated circuits (ASICs) developed by Google in order to accelerate the machine learning workload.



TPU v2 - 4 chips, 2 cores per chip

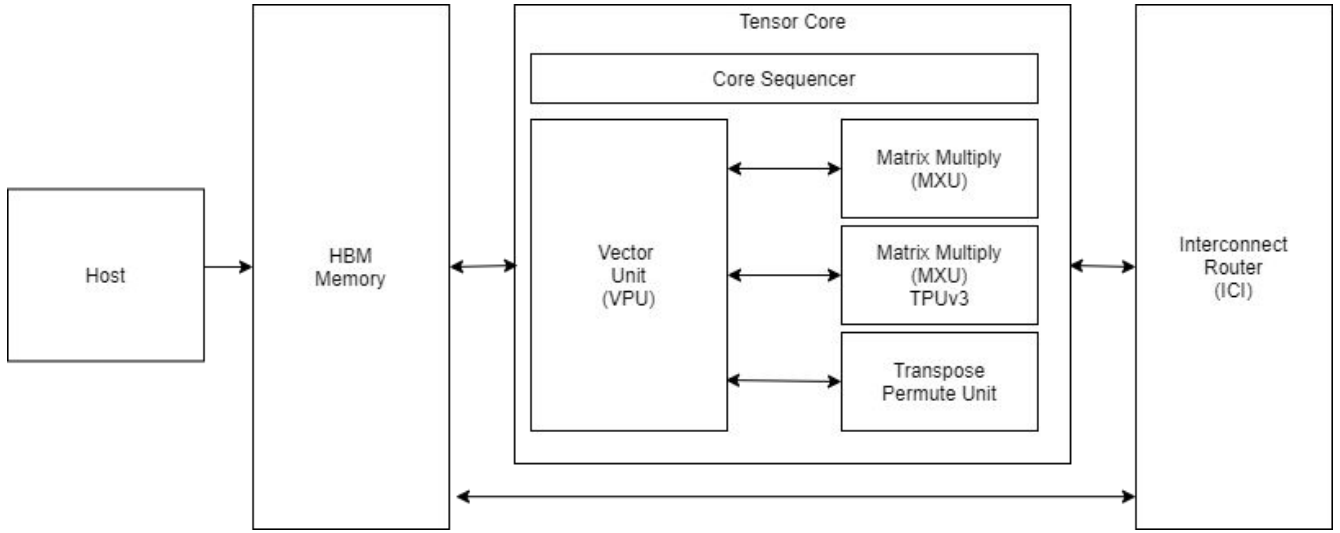
TPU v3 - 4 chips, 2 cores per chip

https://cloud.google.com/tpu/docs/system-architecture

TPUs Hardware

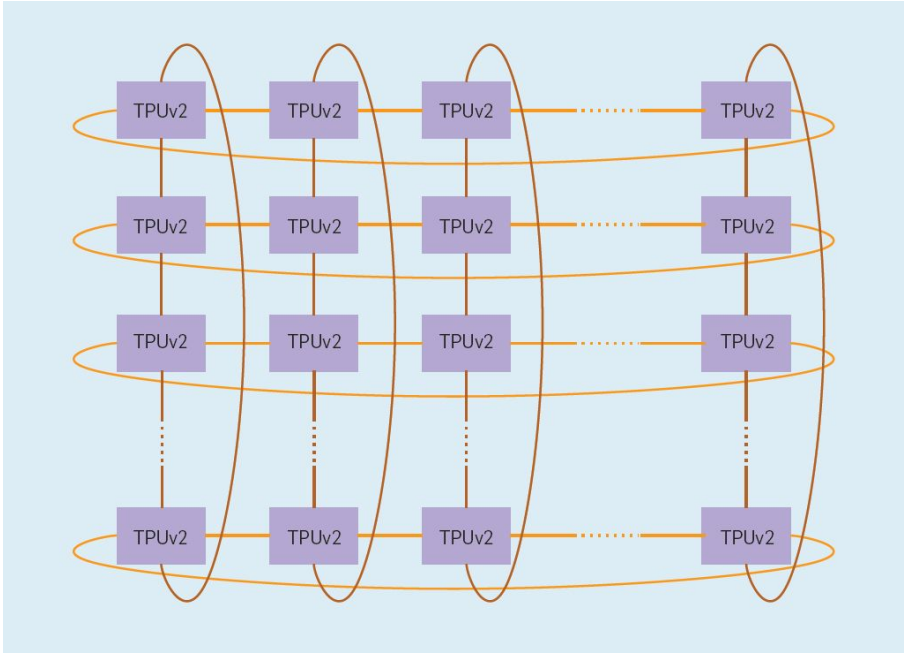
- A large two-dimensional matrix multiply unit (MXU) size of 128x128
- Inter-Core Interconnect (ICI)
- High Bandwidth Memory (HBM)
- Core Sequencer
- Vector Processing Unit
- The Transpose Reduction Permute Unit

- two cores per chip
- 2 MXUs per core for TPUsv3



TPU Parallel Approach

- Using Tensorflow TPU Strategy
- Weight updates using All-reduce and a 2D torus topology
- Model needs to be initialized in strategy scope
- Tensorflow Train_on_batch function takes care of distributed training.
- Cores on a Cloud TPU execute an identical program residing in their own respective HBM in a synchronous manner.
- A reduction operation is performed at the end of each neural network step across all the cores.



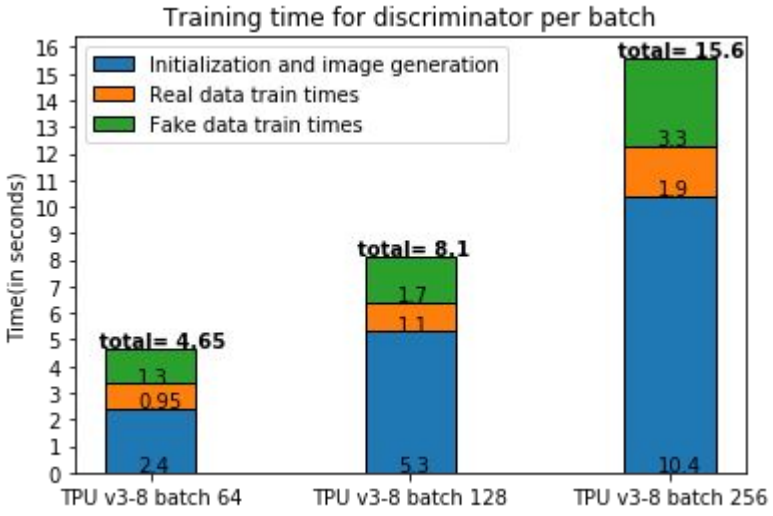
DOI: 10.1145/3360307

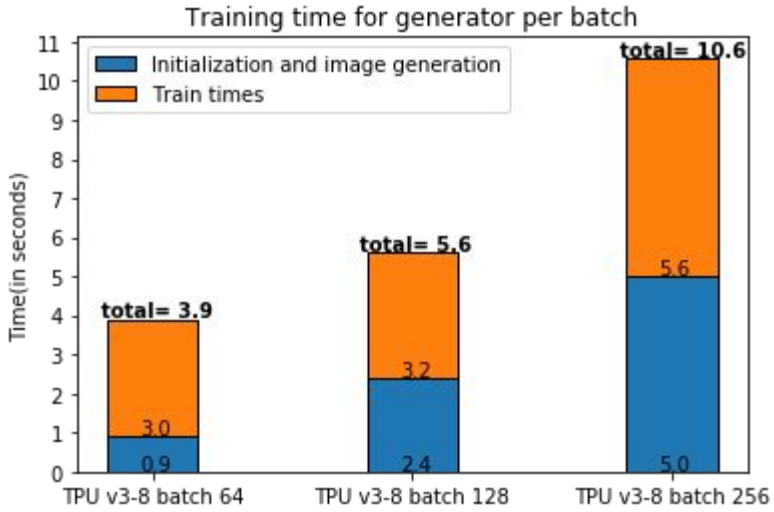
TPU Configuration

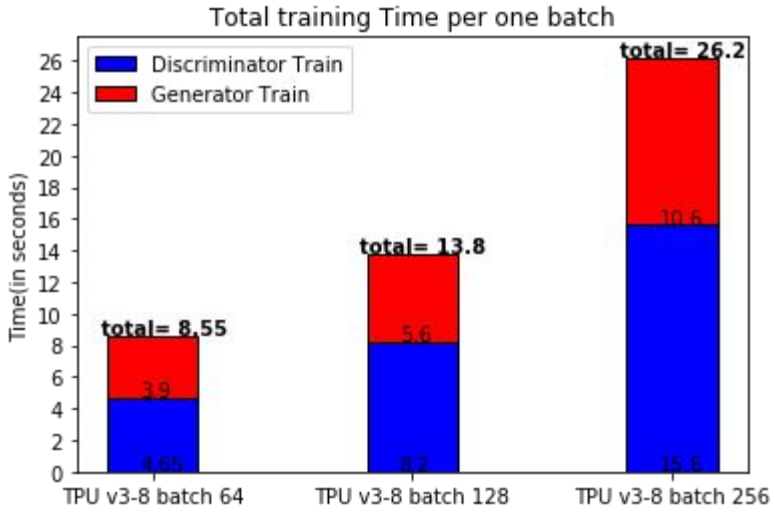
- Test were run on TPU version 3 with 8 cores
- Multiple batch sizes were used to see the difference in performance.
- Batch specification for one TPU core:
  - 8 cores -> Global batch size = 8 x batch size
- We measured the average of time for each individual training.
- In the end we measured the average time for one epoch for the best setup
- All times are in seconds

TPU Results

- Best Result - 8 cores batch size of 128 for each core.
- 64 batch size - similar training time - double the batches
- 256 batch size - double training time
- Average time per epoch of best Result: 2070 seconds

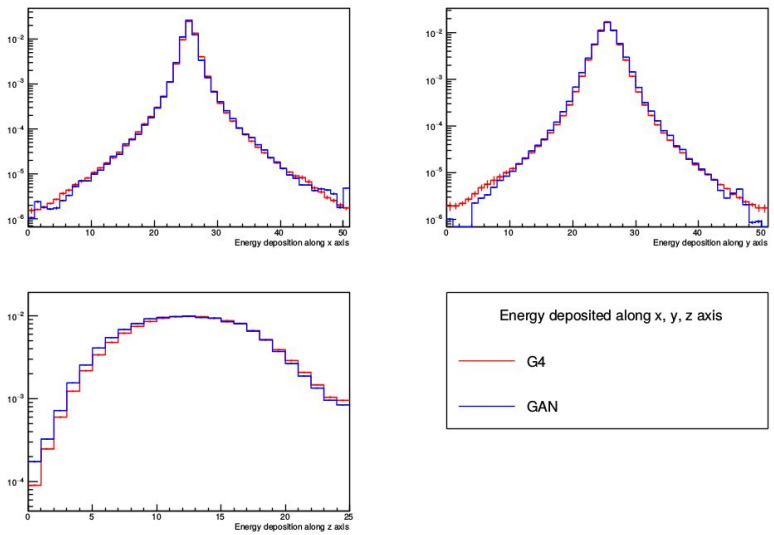




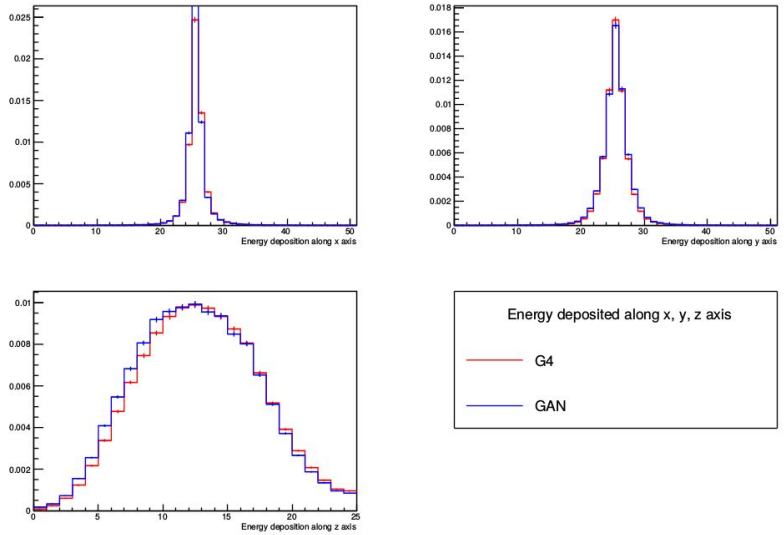


TPU Results Validation

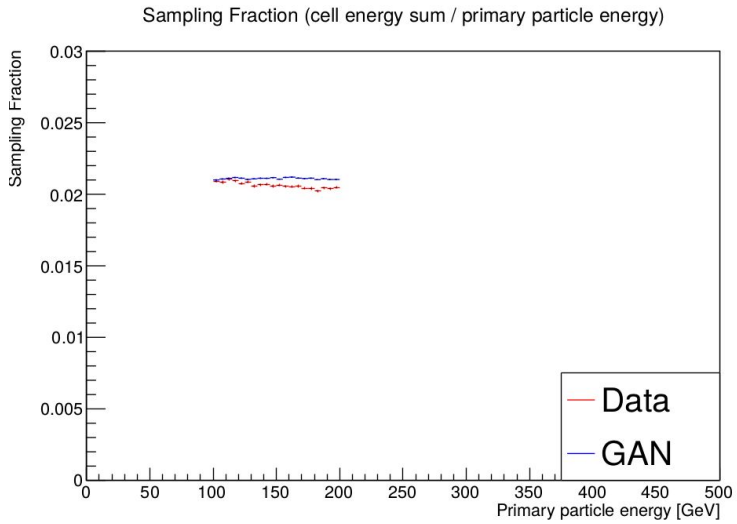
Results obtained from running the on TPUs v3 with 8 cores and a batch per core of 128, during 40 epochs.



Shower Shapes for Geant4 vs. GAN events along x, y and z in log scale



Shower Shapes for Geant4 vs. GAN events along x, y and z



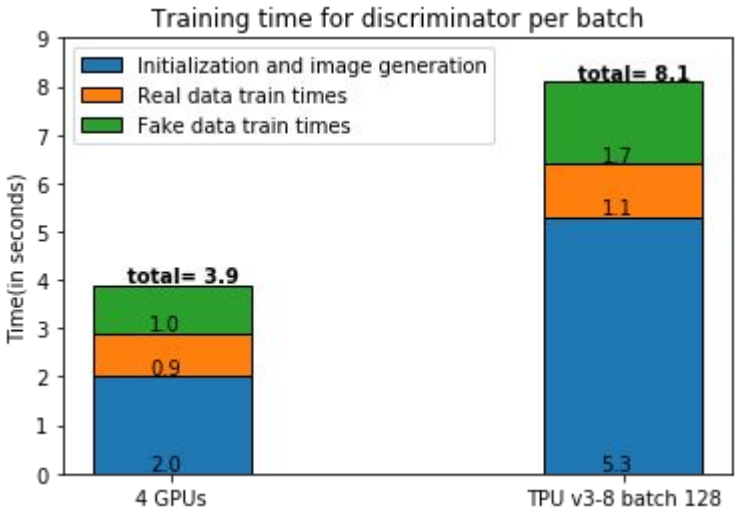
Sampling fraction GAN and Geant4 events for 100 – 200 GeV primary energy

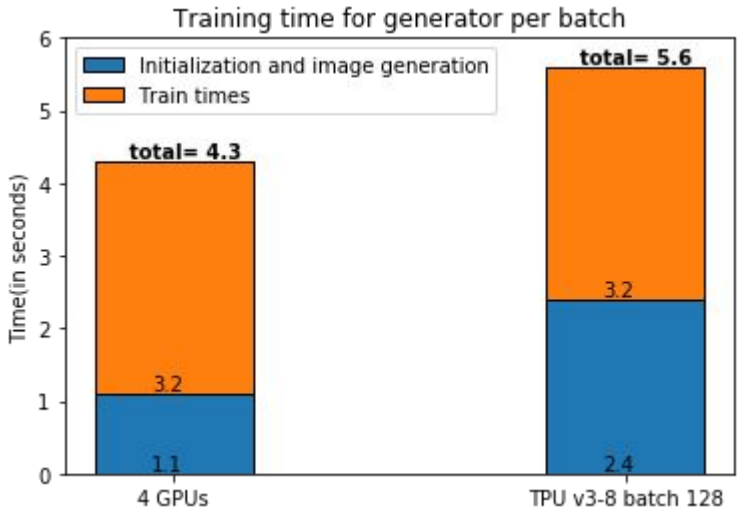
GPU+TPU Comparison

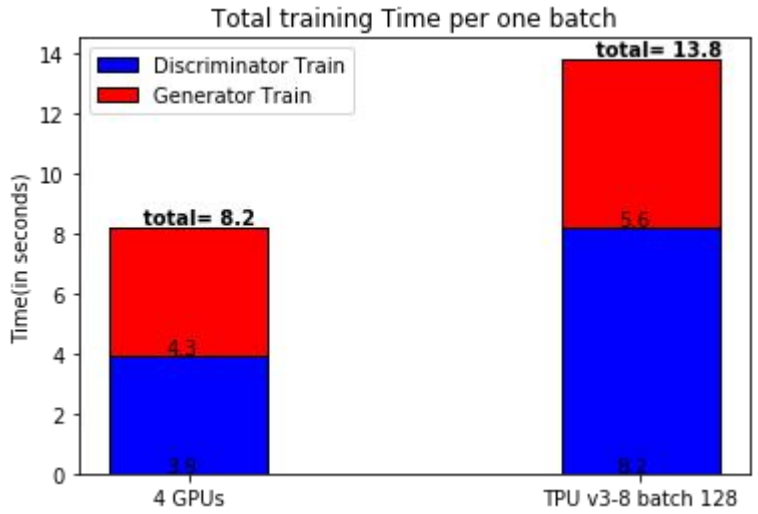
- Batch size:
  - 4 GPU = 4 x 128 = 512
  - 8 cores TPU = 8 x 128 = 1024.
- TPU are better than 4 GPUs
- Price:
  - TPU: 1 USD/hour -> 4,6 per epoch
  - GPU: 1,8 USD/hour -> 5,13 per epoch

Average Time per epoch:

- 4 GPU: 2565
- TPU v3-8: 2070







Summary

- Deployment of 3DGAN training on GPUs and TPUs
- Benchmark of TPUs and GPUs
- TPUs offer better performance than GPUs but still present a bottleneck in training

Future Work

- Increase the amount of code being parallelized
- Better strategies for data input
- Modify models run on TPUs to achieve better performance

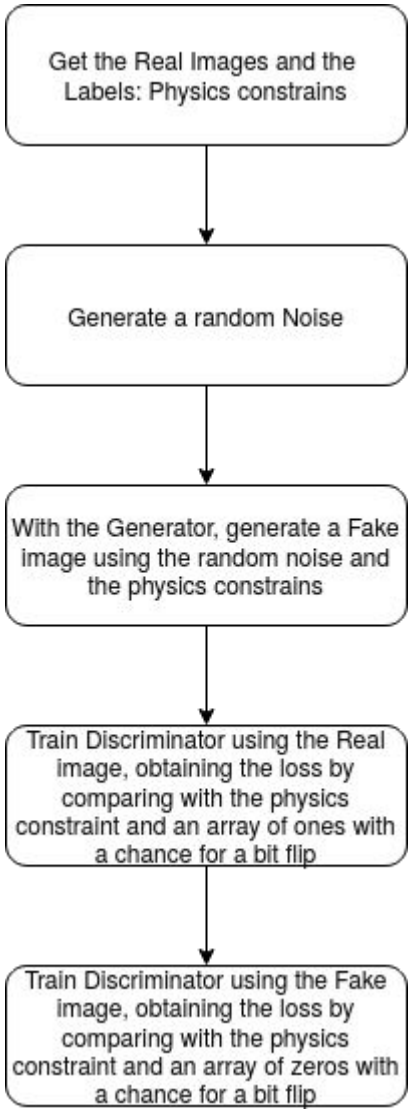
Resources accessed on GCloud thanks to the CERN openlab + Google collaboration

Any questions?

Thank you for your Attention!

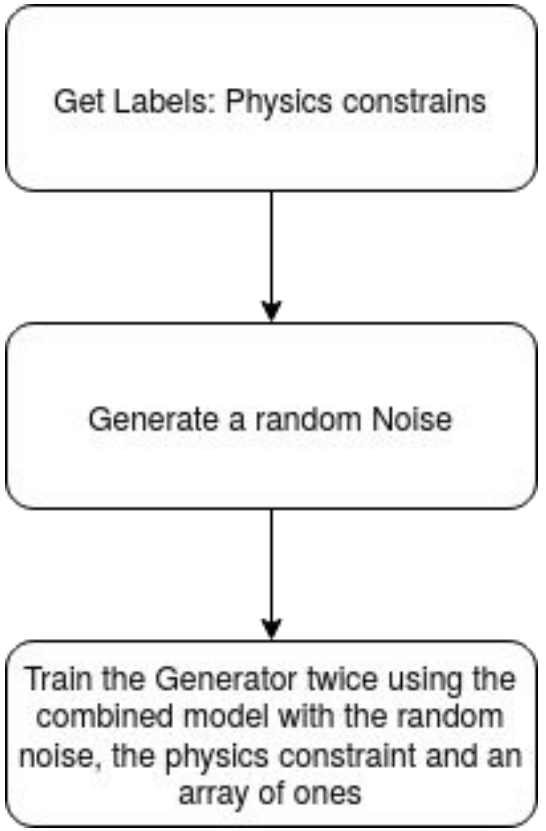
GAN Training - Discriminator

- The Discriminator has two steps for training, the Real data training and the Fake data training.
- In order to train the Discriminator on real data we use “1” labels + small fraction of “0” to introduce noise.
- For the fake data we use “0” labels + a small fraction of “1” to introduce noise.



GAN Training - Generator

- The Generator only needs to train using the random noise to generate the image and pass it to the Discriminator
- The combined model is responsible for generating the image and updating the weights of the generator
- The Generator is trained twice to balance the training of the Discriminator
- We use “1” labels in order to give the idea that the generated data is all real data.



Backup v3-32

		TPU v3-32 batch 128	TPU v3-8 batch 128
Discrimina tor Training	Initialization and image generation	18,1	5,3
	Real data train times	13,2	1,1
	Fake data train times	16,3	1,7
	Sum of times	47,6	8,2
Generator Training	Initialization	0,6	2,4
	Image generation	0,047	0,013
	Training	2,7	1,6
	Sum of times	14,894	5,626
Total training Time		62,494	13,626

Backup v2-8

		TPU v3-8 batch 128	TPU v2-8 batch 128
Discriminator Training	Initialization and image generation	5,3	5,2
	Real data train times	1,1	1,7
	Fake data train times	1,7	2,4
	Sum of times	8,2	9,3

Backup Strategy code

```
strategy = tf.distribute.MirroredStrategy()
BATCH_SIZE_PER_REPLICA = batch_size
batch_size = batch_size * strategy.num_replicas_in_sync

with strategy.scope():
    d=discriminator(xpower, dformat=dformat)
    g=generator(latent_size, dformat=dformat)

with strategy.scope():
    combined = Model(
        inputs=[latent],
        outputs=[fake, aux, ang, ecal, add_loss],
        name='combined_model'
    )

dataset = tf.data.Dataset.from_tensor_slices(dataset).batch(batch_size)


discriminator.train_on_batch
```

Backup TPU code

```
tpu_address = os.environ["TPU_NAME"]
cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver(tpu=tpu_address)
tf.config.experimental_connect_to_cluster(cluster_resolver)
tf.tpu.experimental.initialize_tpu_system(cluster_resolver)

strategy = tf.distribute.TPUStrategy(cluster_resolver)

BATCH_SIZE_PER_REPLICA = batch_size
batch_size = batch_size * strategy.num_replicas_in_sync
with strategy.scope():
    d=discriminator(xpower, dformat=dformat)
    g=generator(latent_size, dformat=dformat)

with strategy.scope():
    combined = Model(
        inputs=[latent],
        outputs=[fake, aux, ang, ecal, add_loss],
        name='combined_model'
    )

dataset = tf.data.Dataset.from_tensor_slices(dataset).batch(batch_size)

discriminator.train_on_batch
```

Full TPU Hardware