Comparing GPU and TPU in an Iterative Scenario: A Study on Neural Network-based Image Generation

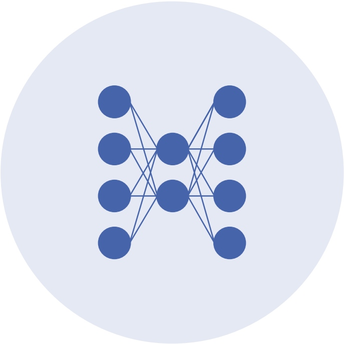
Roman Lehmann, Paul Schaarschmidt, Wolfgang Karl

Motivation

Flow Simulation

δ

VS



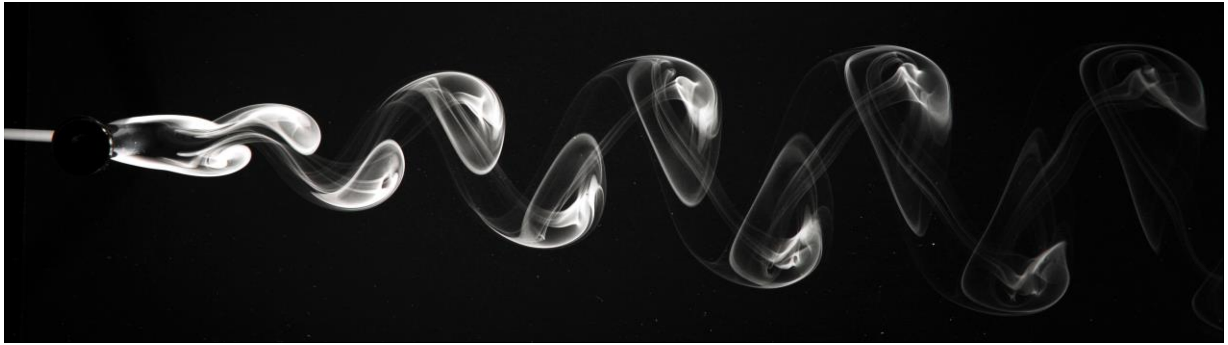
Numerical Methods

ML Approach

e.g. Navier-Stokes Equations

e.g. Image Generation Approach

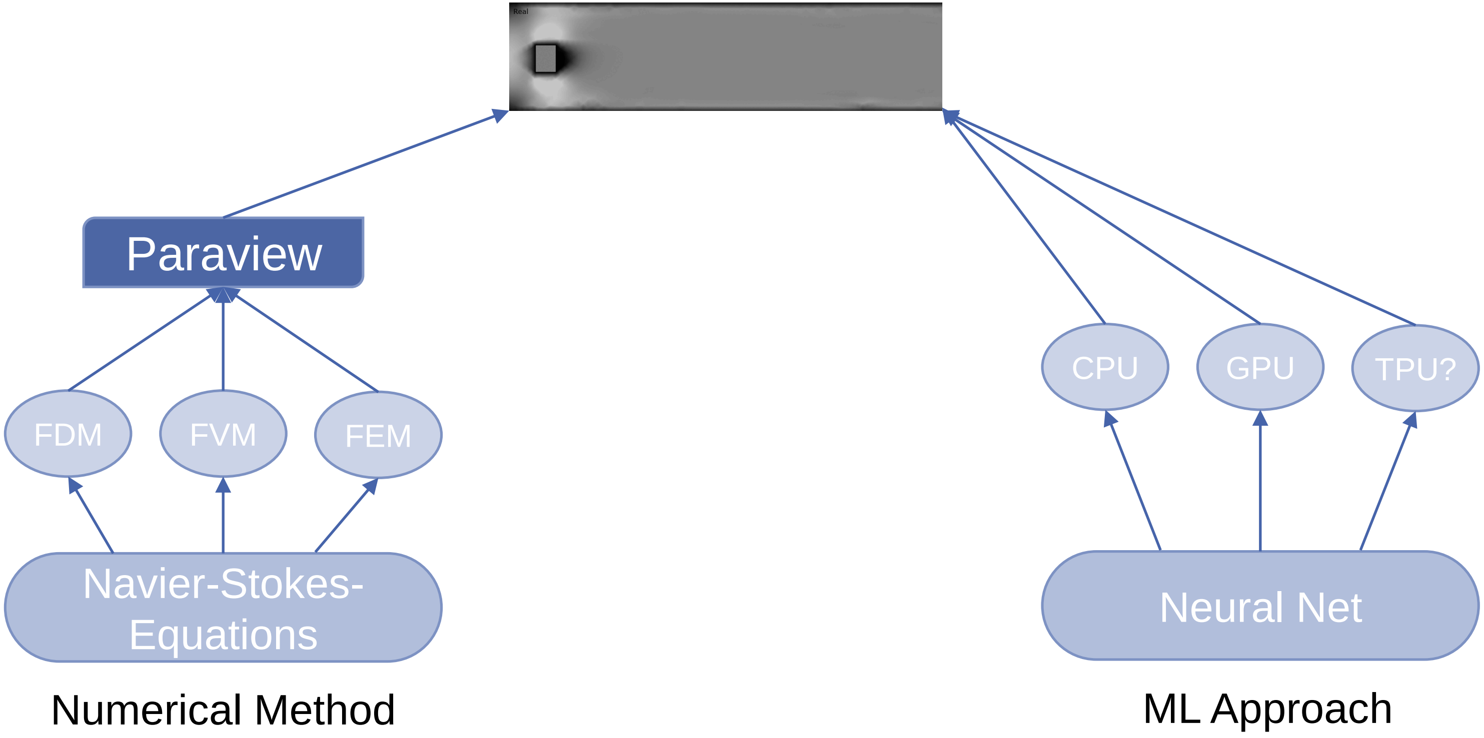
Problem: Simulation of Kármán Vortex Street



Visualization of the vortex street behind a circular cylinder in air [1].

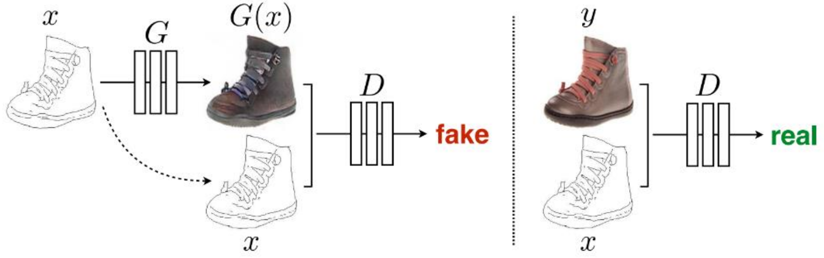
Numerical simulation of the vortex street behind a rectangular obstacle.

Flow Simulation



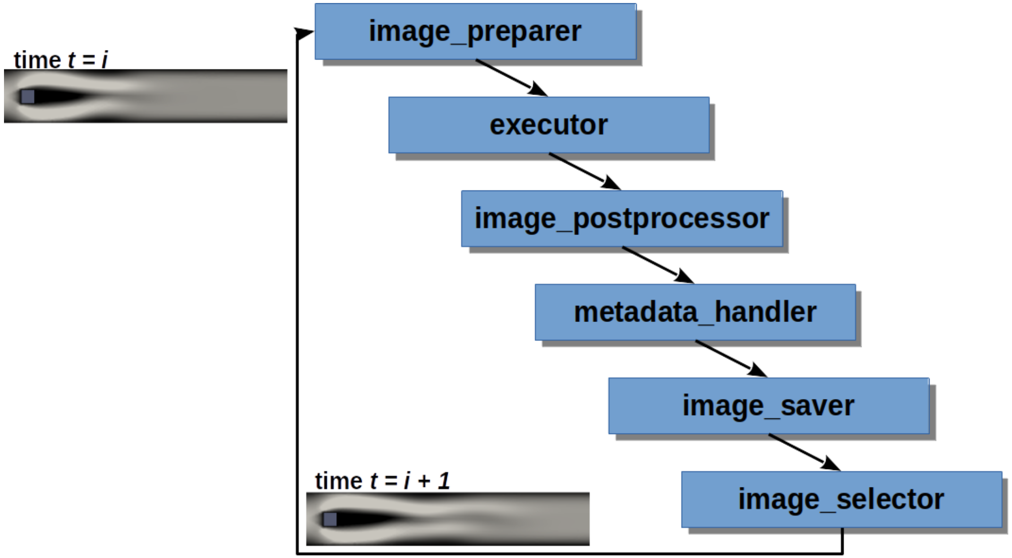
Our Approach

- Goal is to simulate the Kármán vortex street with an image-to-image (Pix2Pix) approach based on NN.
- “Conditional Generative Adversarial Network” (cGAN)
- Generator and discriminator net



Training: D learns to classify between fake and real images. G learns to fool the D [2].

Method for image generation

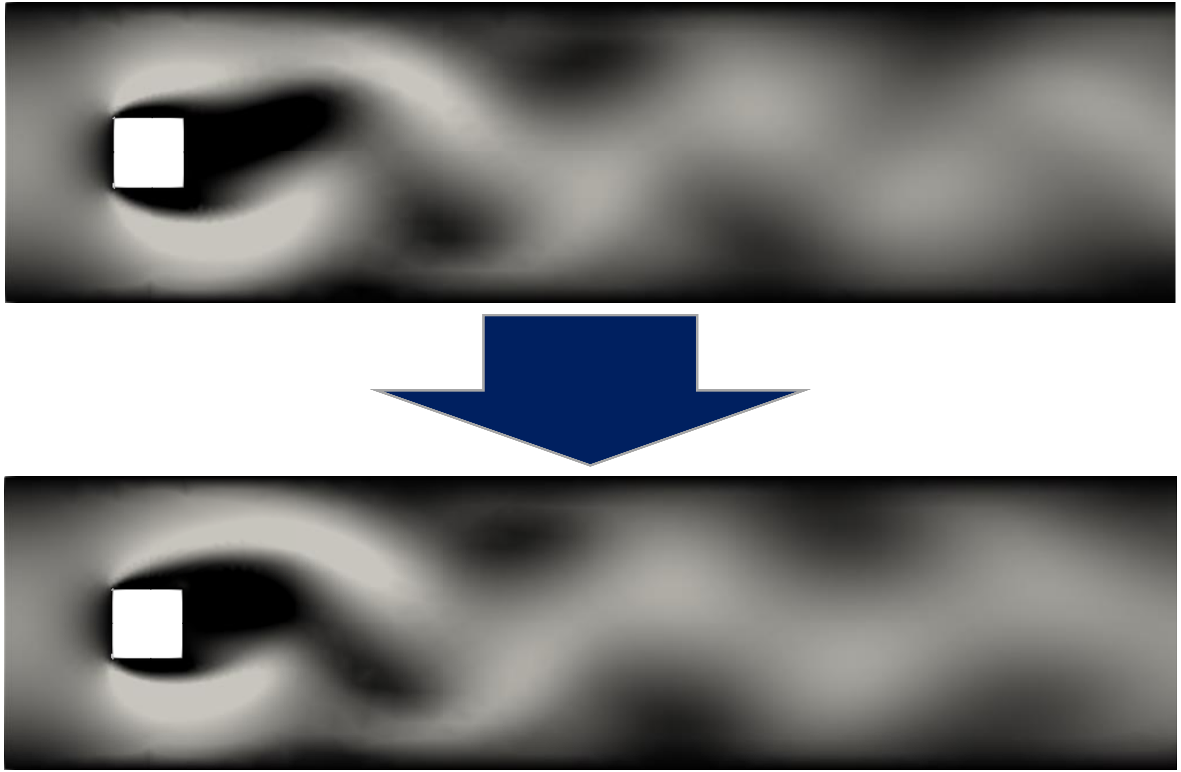


- The trained generator is placed as executor
- A U-Net is used as net architecture
- Iterative application

Results

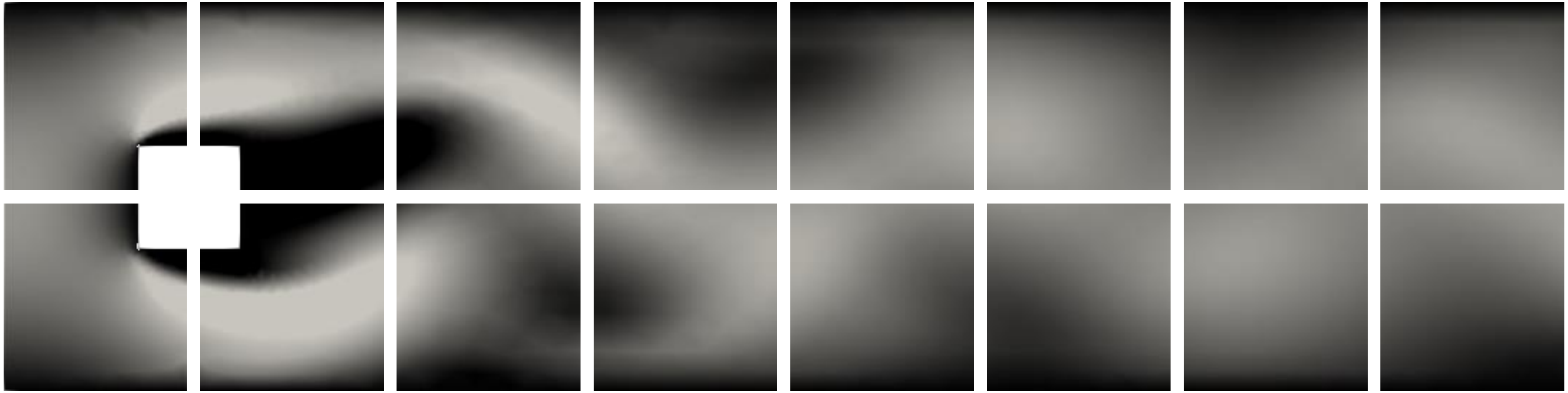
- Results are usable
- Tends to overfitting
- Map-Reduce approach for further improvement

»Map-Reduce« Approach (1/2)

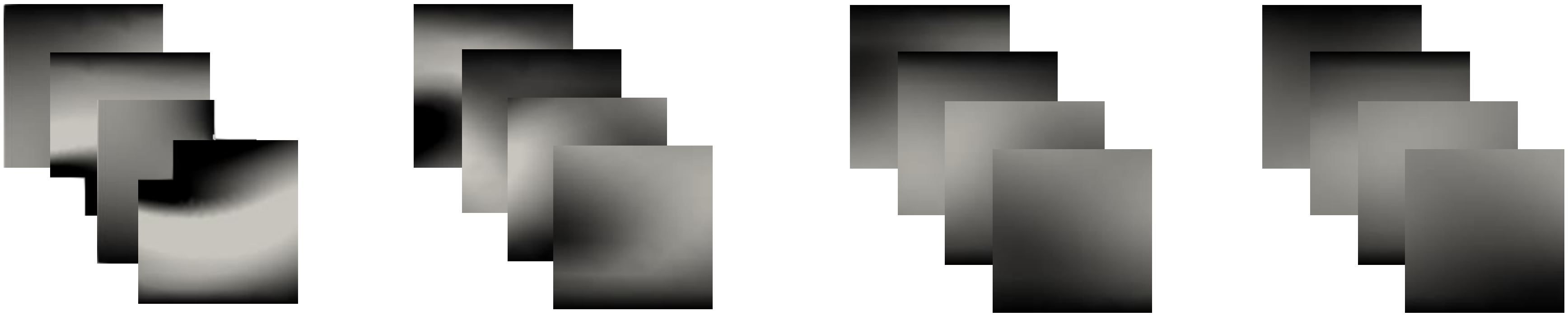


Input image

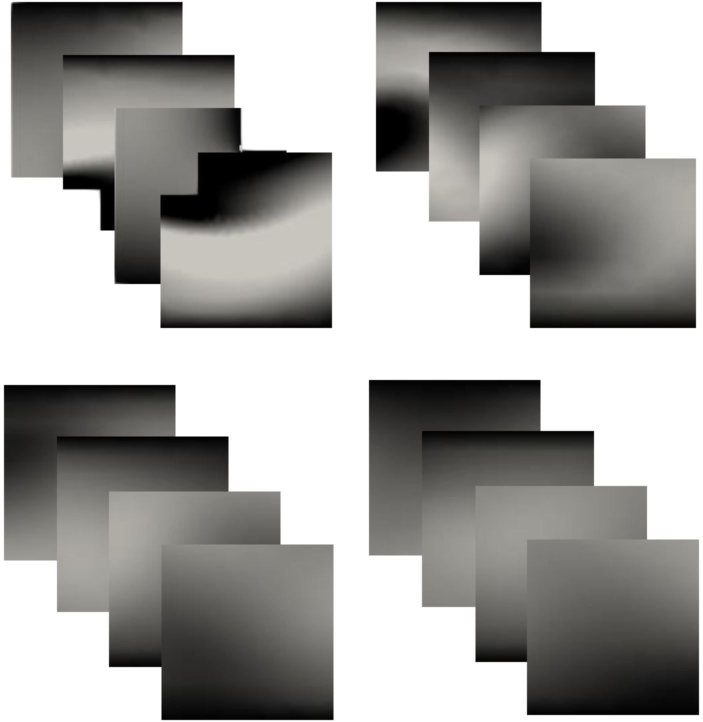
Devide into tiles

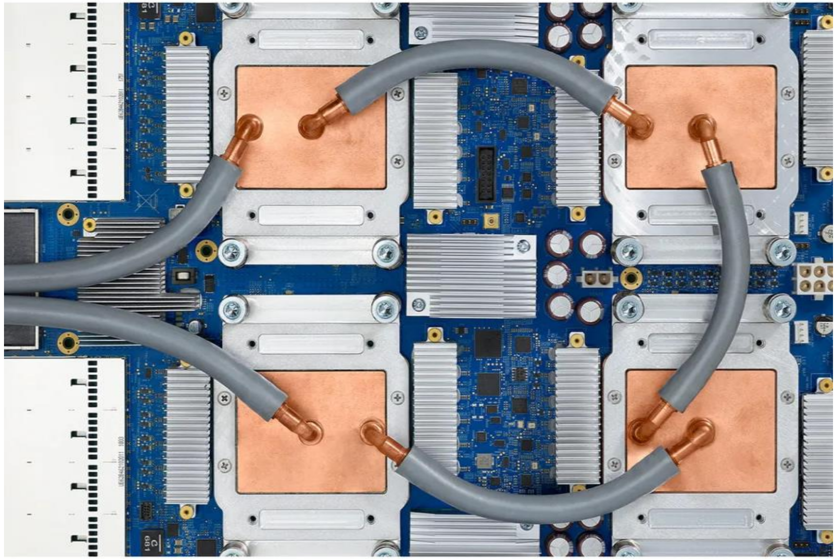


Grouping

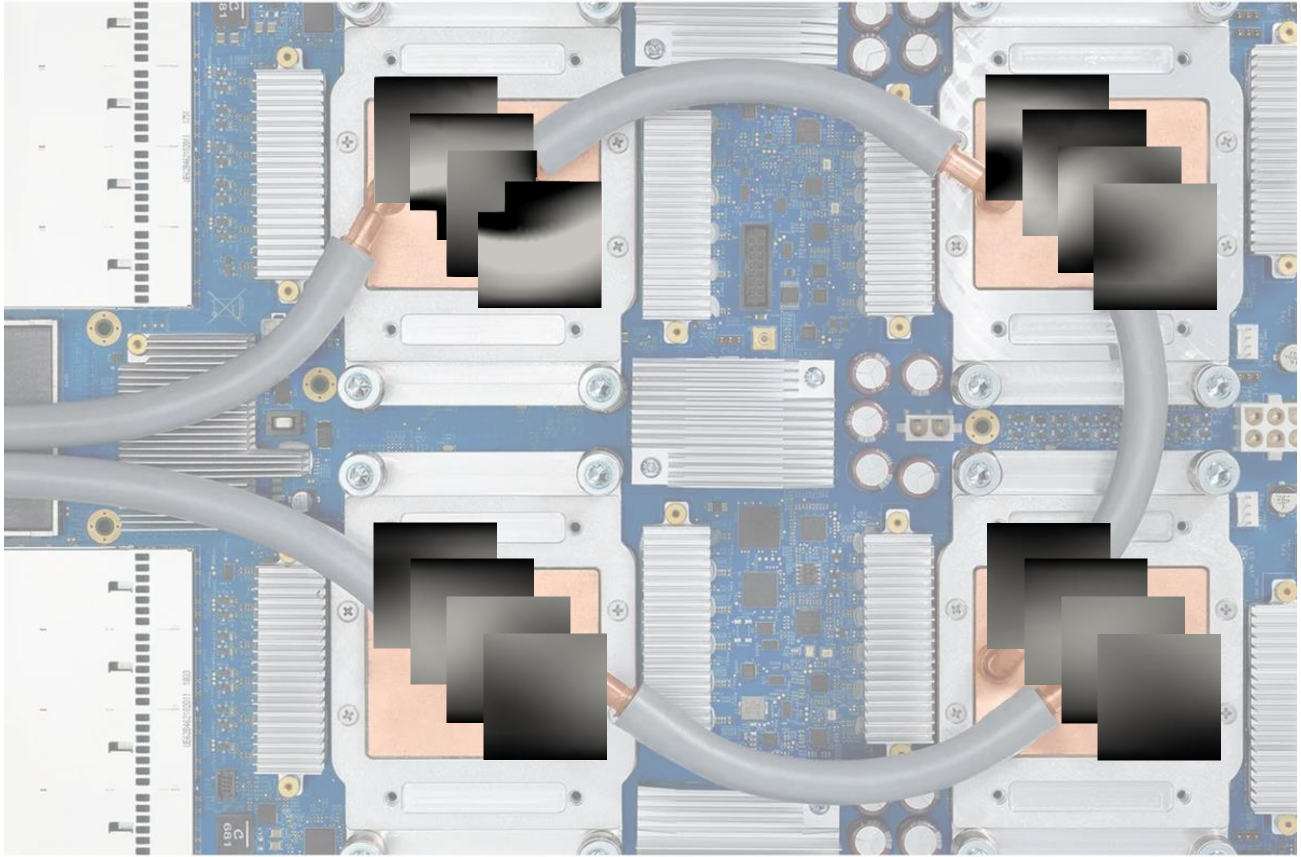


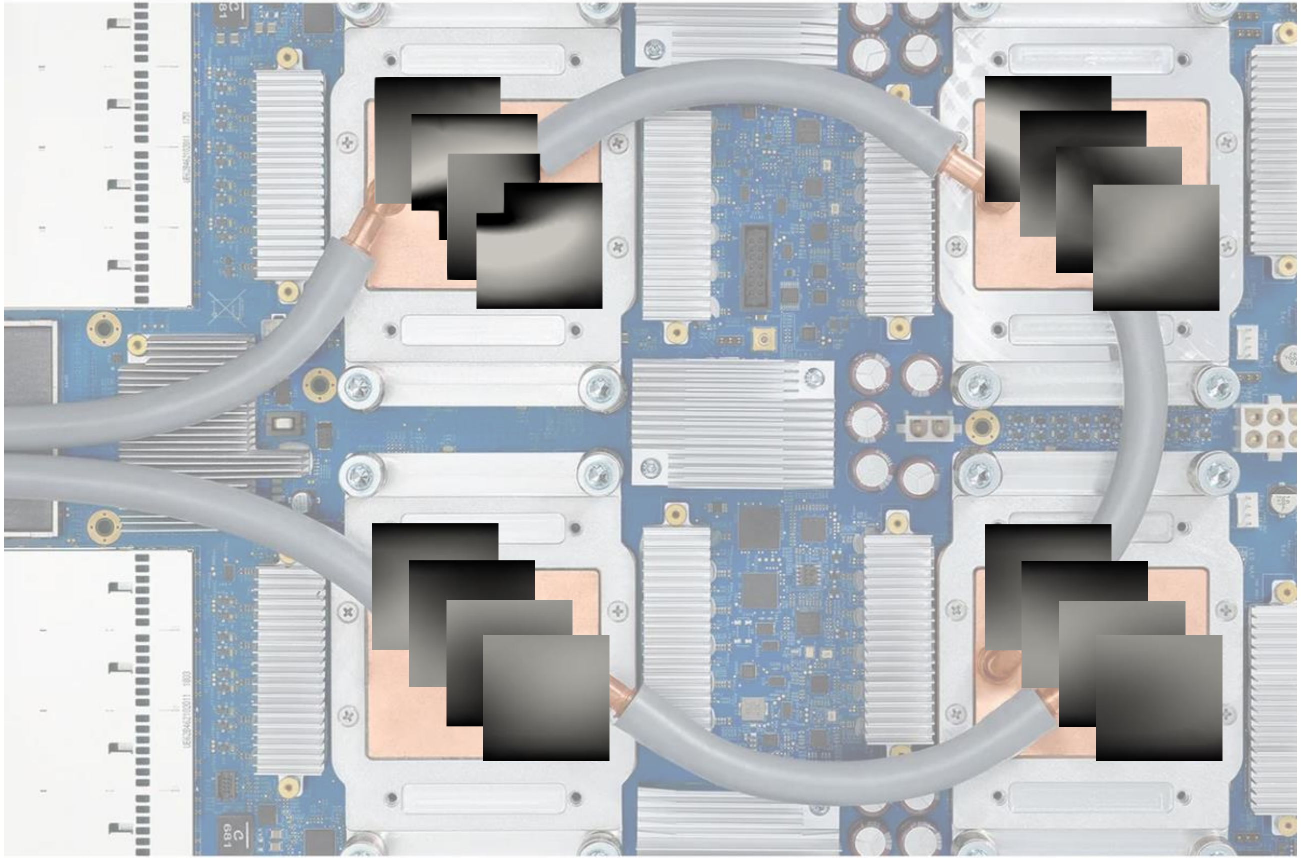
Assignment to TPU



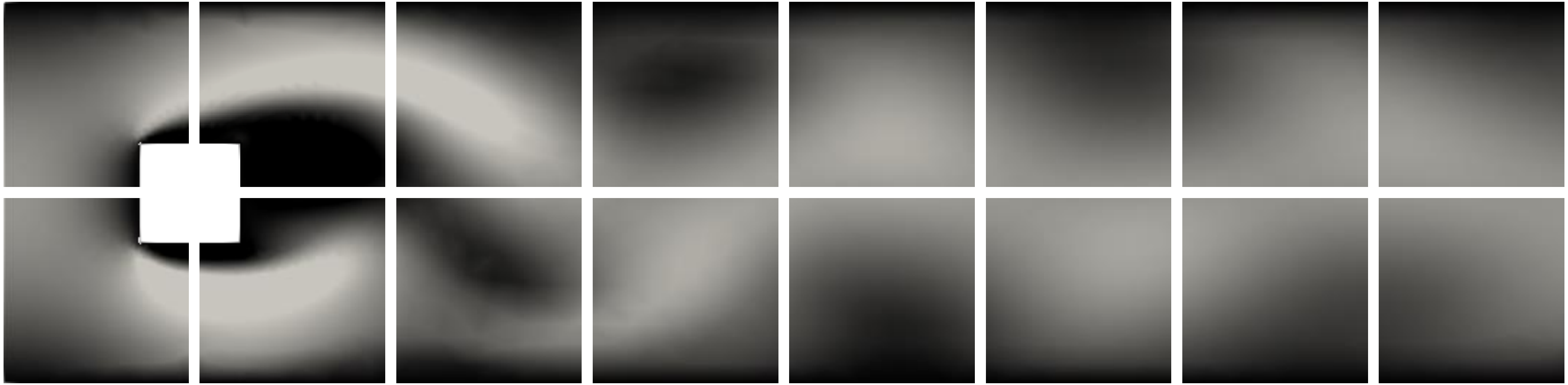


[3]

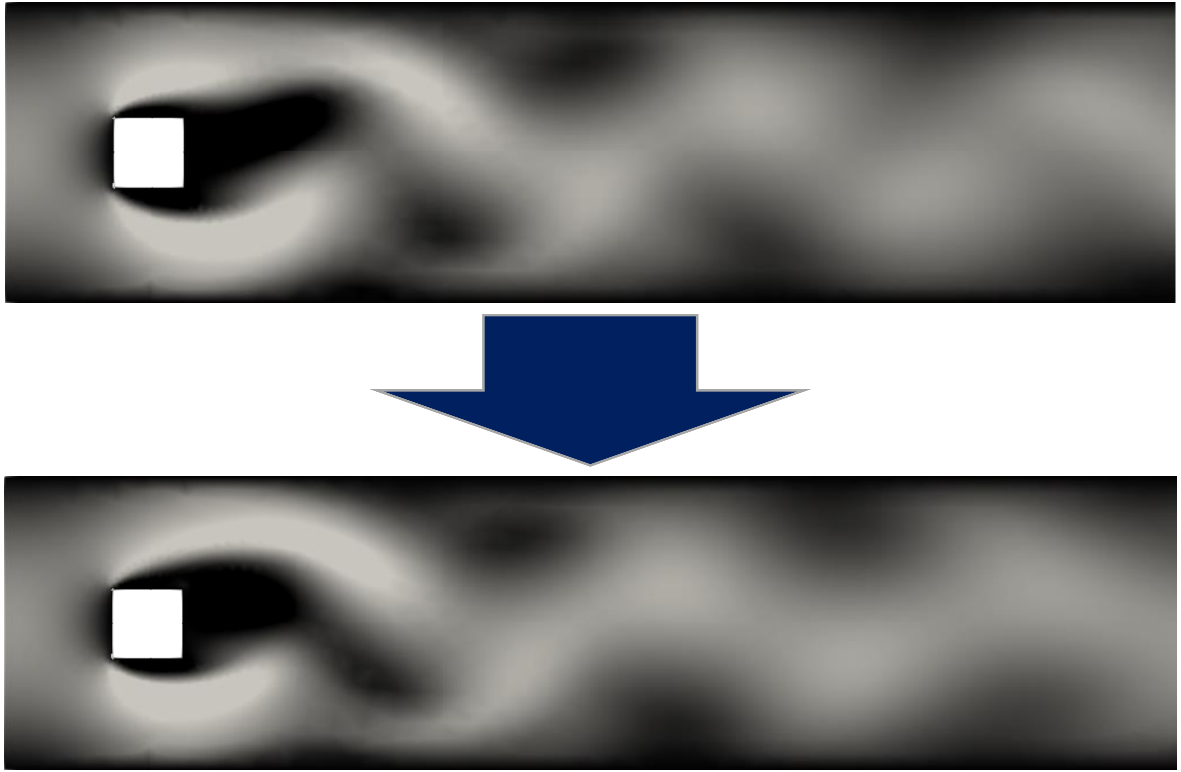




Merge tiles

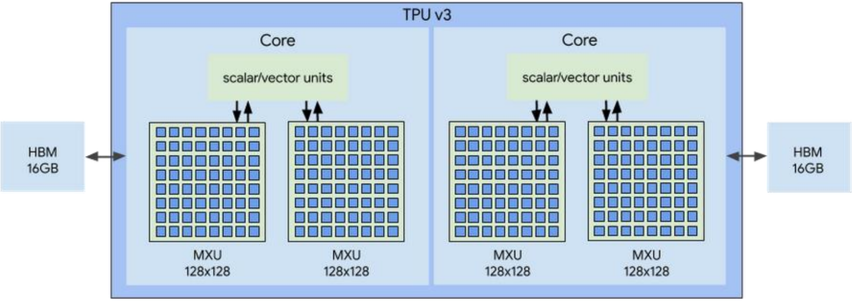


»Map-Reduce« Approach (2/2)

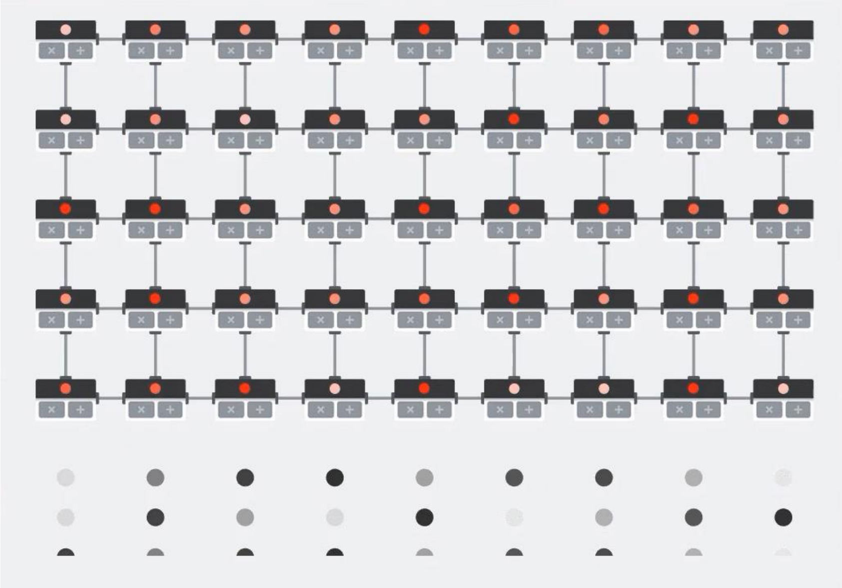


TPU Architecture

TPUv3 Architecture [4]

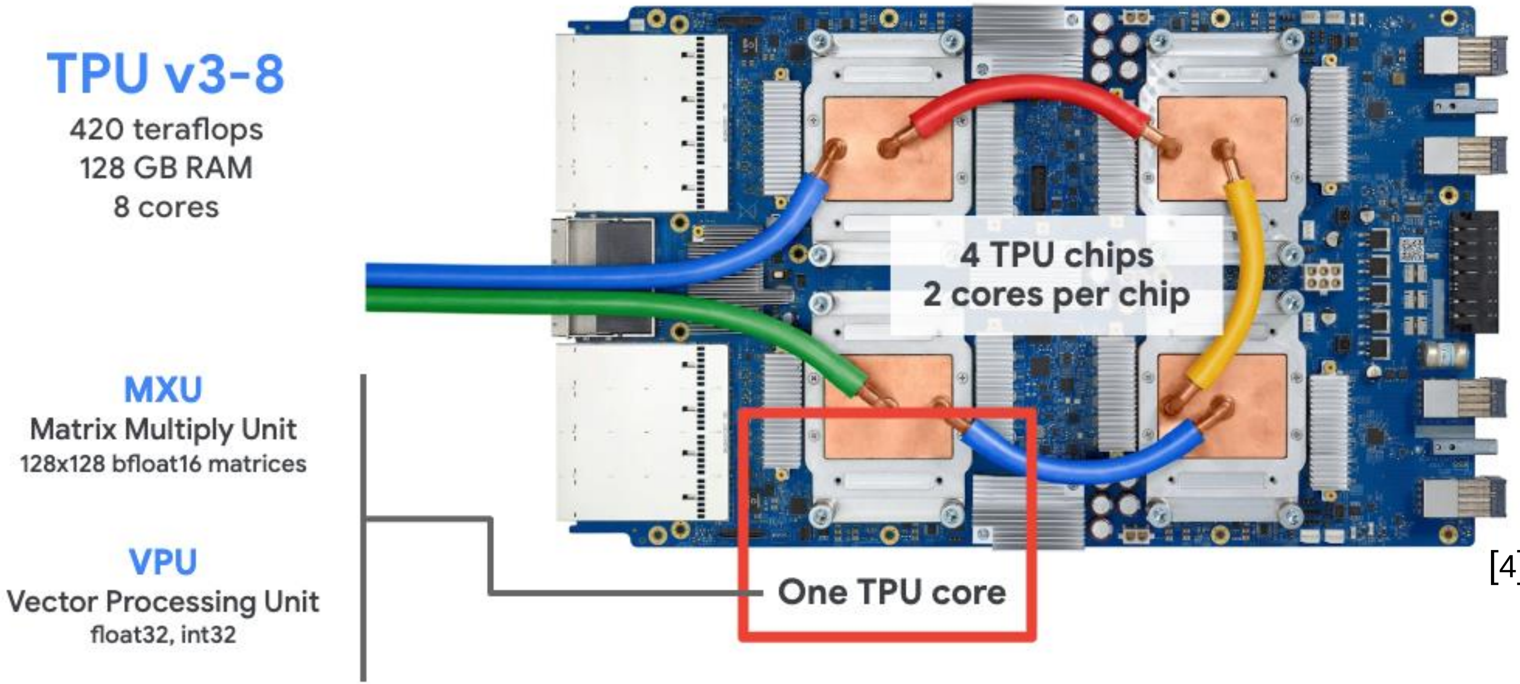


- Each Core consists of MXU and VPU
- MXU is realized according to a systolic array



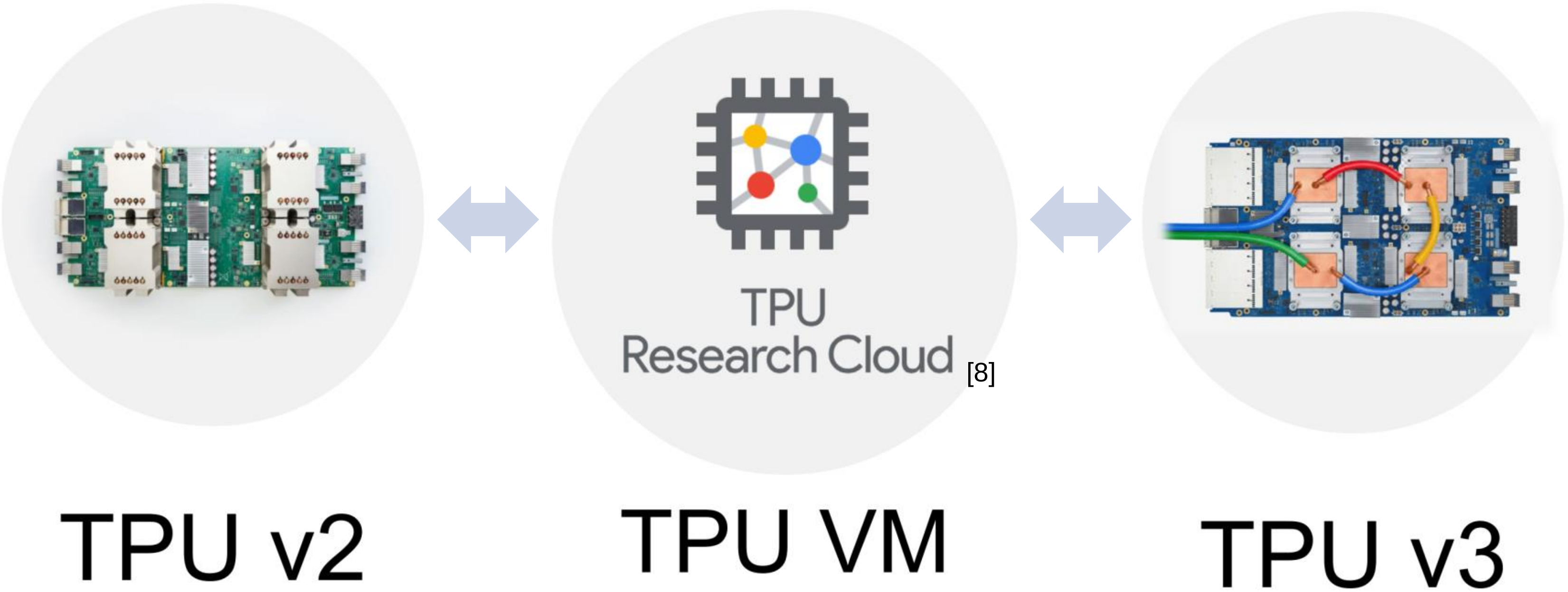
Scheme of a systolic array [4]

TPU v3


TPU v3-8
420 teraflops
128 GB RAM
8 cores
MXU
Matrix Multiply Unit
128x128 bfloat16 matrices
VPU
Vector Processing Unit
float32, int32
4 TPU chips
2 cores per chip
One TPU core

[4]

TPU Research Cloud

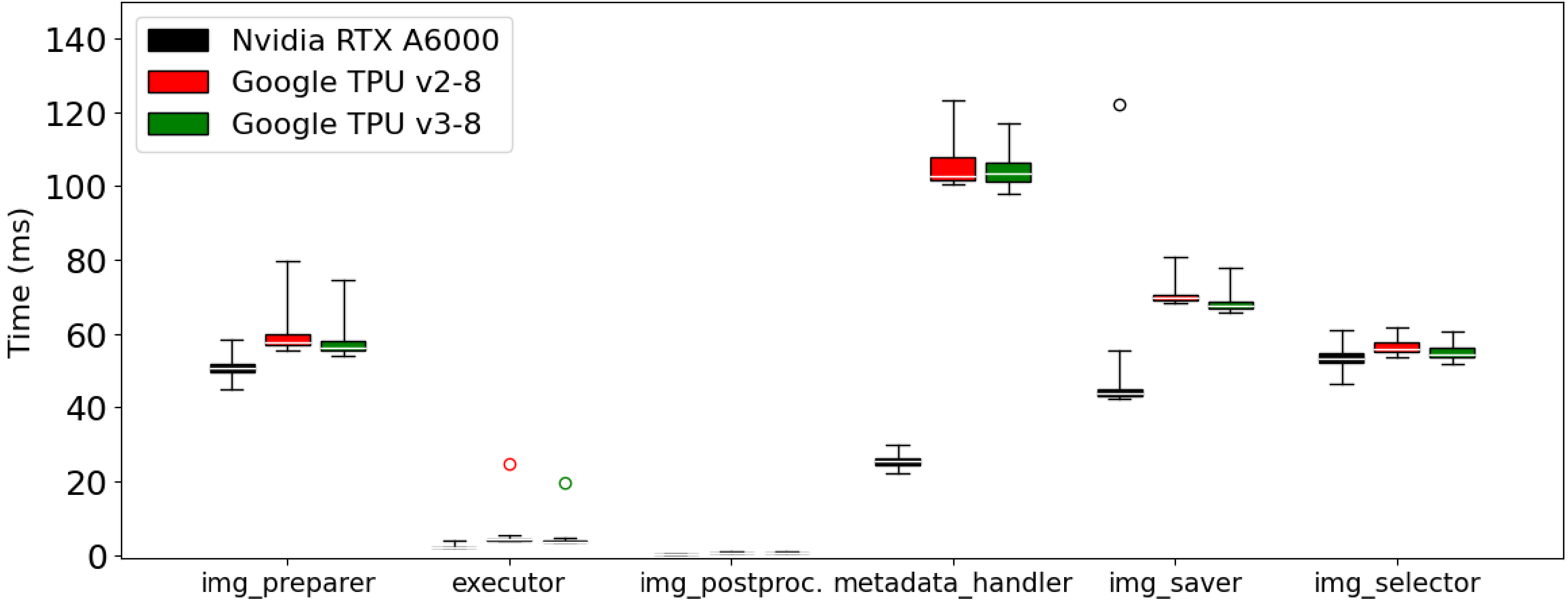


[8]

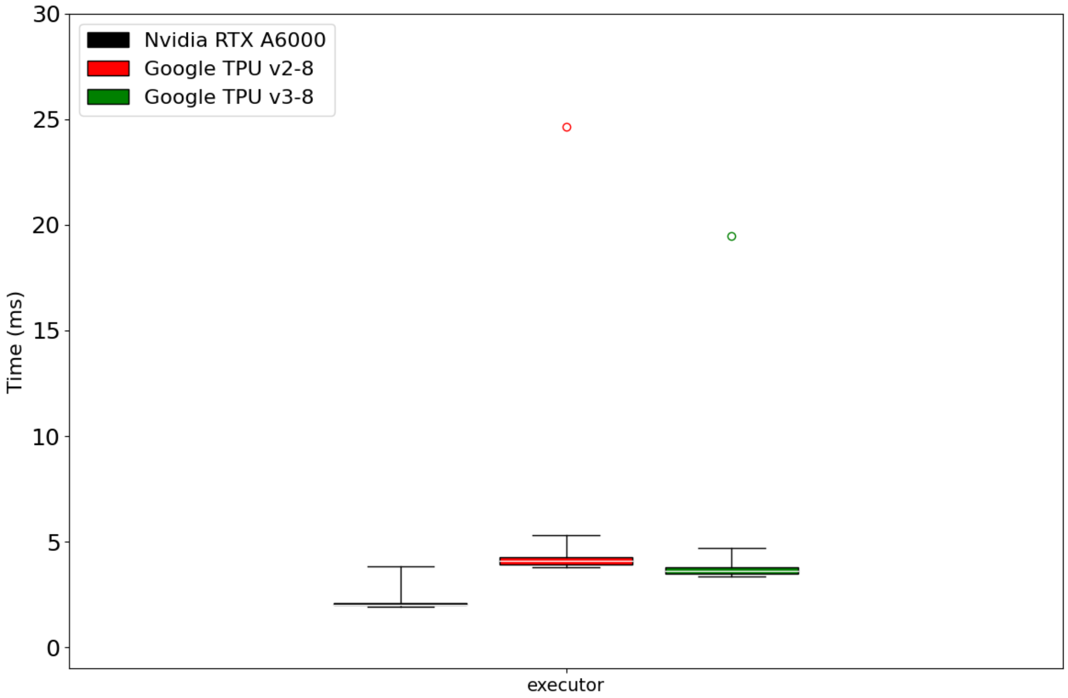
Used Systems

- 1st System
  - TPU Research Cloud with two Intel® Xeon® Gold 5418Y CPUs (24 Cores each) and connection to TPUv2 and TPUv3 with 8 or 32 Cores each
- 2nd System
  - Institute server with two AMD Epyc™ 7F32 CPUs (8 Cores each) and one Nvidia RTX™ A6000 GPU.
- PyTorch was used as ML-Framwork with XLA compiler to support XLA devices (TPU)

Analysis of the generation steps

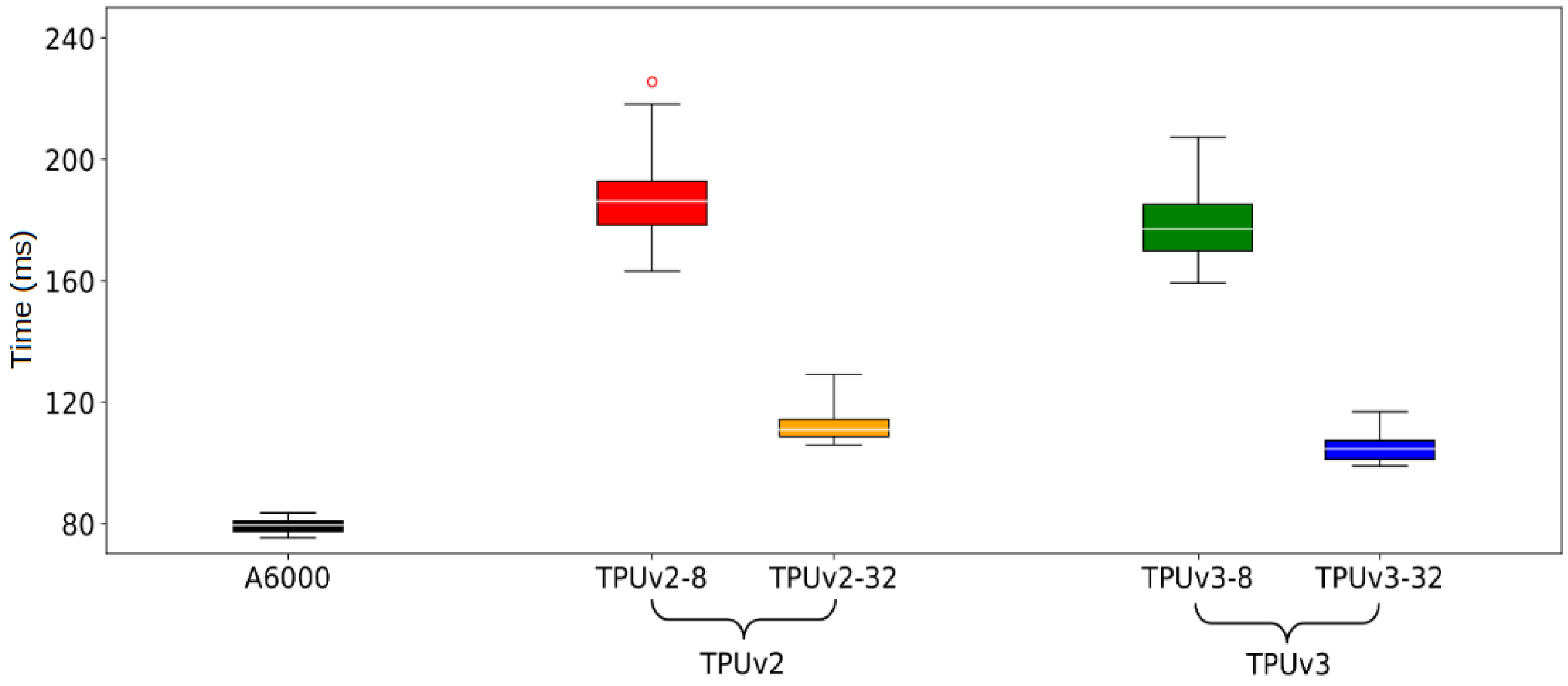


Executor



30
25
20
15
10
5
0
Time (ms)
Nvidia RTX A6000
Google TPU v2-8
Google TPU v3-8
executor

Executor in map-reduce



Explanation

- TPU are specifically designed for neural networks
  - But TPU cannot be used directly
  - XLA (Accelerated Linear Compiler) transforms the graph of computation nodes into TPU machine code
  - Costly for small problem sizes
  - Designed to reuse the graph

Explanation (2)

- TPU utilization is insufficient
  - Only the executor step can use the TPU
  - Naive approach can only use one core → map-reduce
  - Map-Reduce: XLA step for every core
    - Increased time consumption for loop fusions
    - Probably because of optimization for saving memory accesses

Training

Batch size \ Device	A6000	TPUv2-8	TPUv3-8	TPUv4-8
8	3451s	2518s	2383s	2130s
16	3385s	1525s	1438s	1369s
32	3133s	-	-	1092s
64	2790s	-	-	1077s

- Training with data parallelism
- Computational graph can be reused while training
- Average training time over 45 epochs
- Speedup between 1.4 – 2.6

Conclusion

- Carefully considering specific nature of iterative application
- No benefits from using TPUs in inference in this use case
- TPUs have advantages in training with higher batch sizes

References

[1] Kármán vortex street. (2023, September 2). In *Wikipedia*. https://en.wikipedia.org/wiki/K%C3%A1rm%C3%A1n_vortex_street

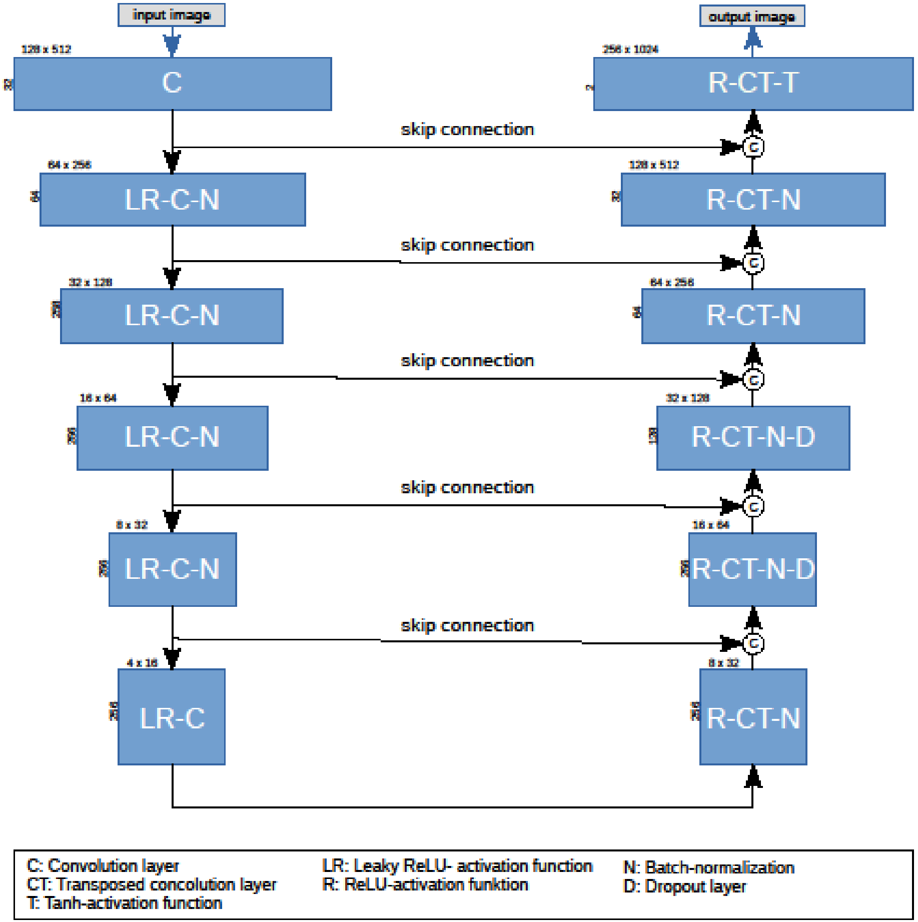
[2] "Image-to-Image Translation with Conditional Adversarial Networks" from **Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A.** (2017)

[3] "Image Classification at Supercomputer Scale" from **Ying, Chris et al.** (2018)

[4] Keras and modern convnets, on TPU, (2023, September 13). In CodeLabs. https://codelabs.developers.google.com/codelabs/keras-flowers-tpu#2

[5] TPU Research Cloud. (2023, September 13). In Google Research. https://sites.research.google/trc/about/

U-Net


input image
output image
128 x 512
C
256 x 1024
R-CT-T
skip connection
64 x 256
LR-C-N
128 x 512
R-CT-N
skip connection
32 x 128
LR-C-N
64 x 256
R-CT-N
skip connection
16 x 64
LR-C-N
32 x 128
R-CT-N-D
skip connection
8 x 32
LR-C-N
16 x 64
R-CT-N-D
skip connection
4 x 16
LR-C
8 x 32
R-CT-N
C: Convolution layer
CT: Transposed concolution layer
T: Tanh-activation function
LR: Leaky ReLU- activation function
R: ReLU-activation funktion
N: Batch-normalization
D: Dropout layer