RAISE Center of Excellence

Performance analysis and code optimizations for distributed training of autoencoders

ISC 2022 - 02.06.2022

Rakesh Sarma (Forschungszentrum Jülich, Germany)

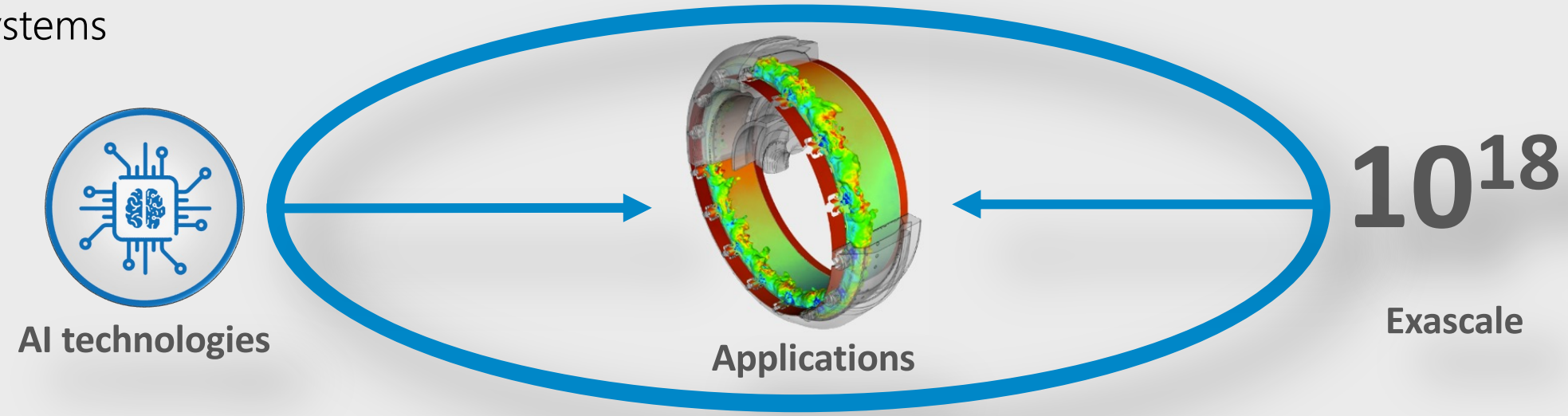
Contributors: E. Inanc, M. Albers, M. Aach, W. Schröder, A. Lintermann

Outline

- Motivation and Introduction
- Use-case on Turbulent Boundary Layer (TBL) flows
- Autoencoders for flow reconstruction
- Code Optimizations

Motivation

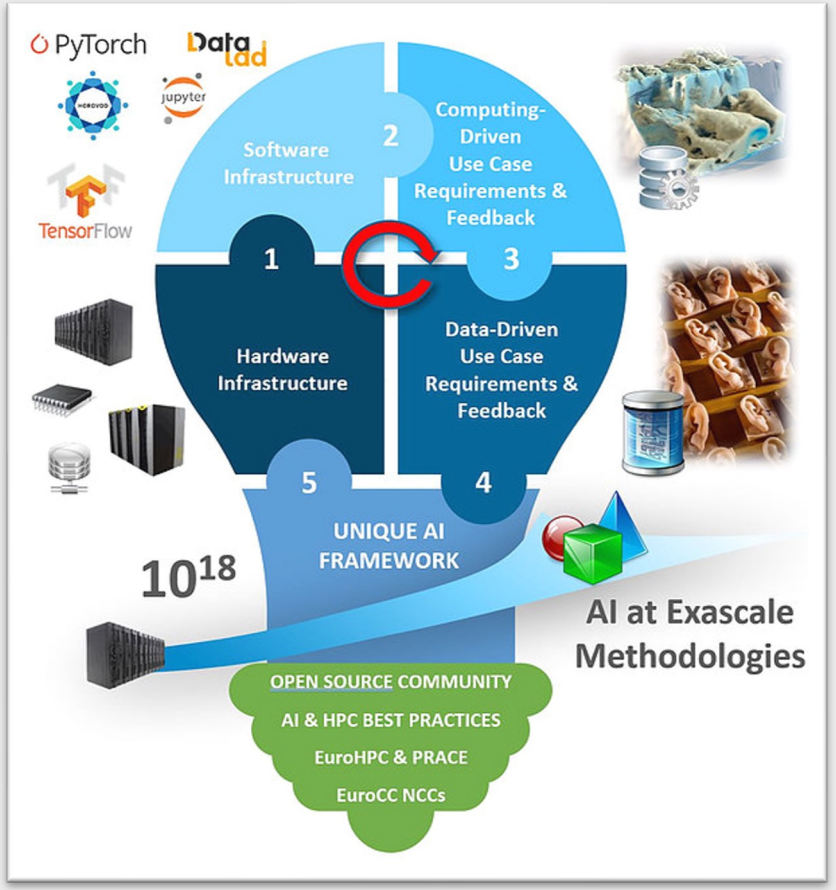
- AI technologies are key to
  - extract knowledge from big data collections
  - reason from existing knowledge
  - find hidden features and detect unseen correlations in massively large data sets
- High-Performance Data Analytics (HPDA) requires
  - intelligent analytic tools
  - scalable systems



CoE RAISE's Major Objectives

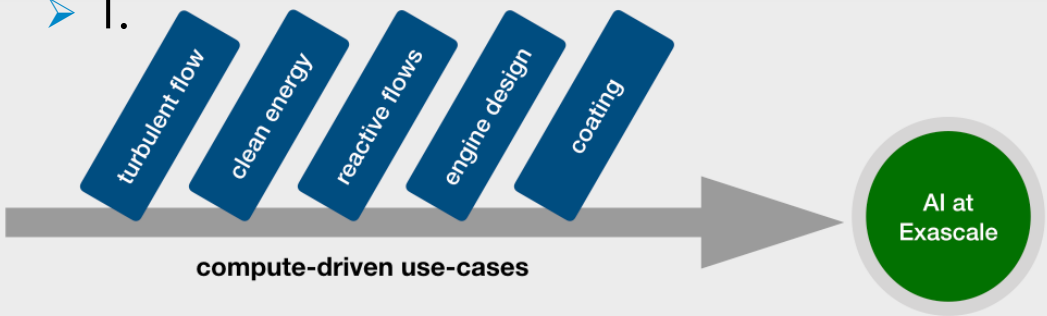
- Development of AI methods towards Exascale
- Connect
  - hardware infrastructure,
  - software infrastructure,
  - compute-driven use cases,
  - and data-driven use cases

to create <u>Unique AI framework</u> for academia and industry

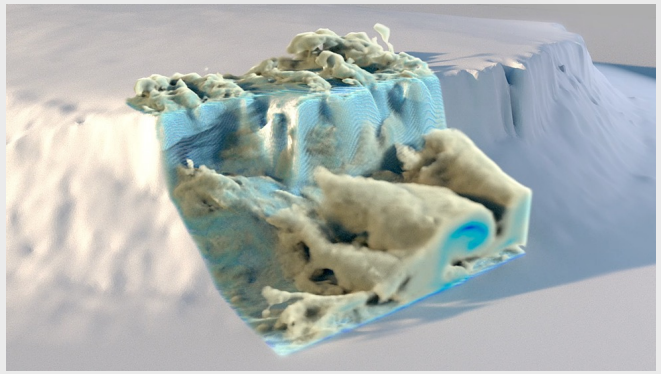


Use Cases in CoE RAISE

- Two kinds of use cases:
- 1.

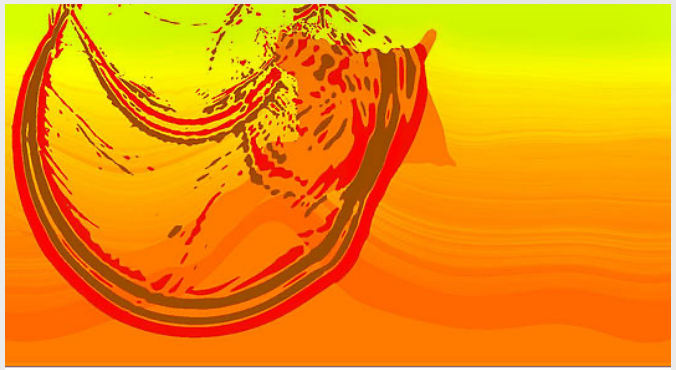


e.g, AI for wind farm layout



- 2.

e.g, seismic imaging with remote sensing

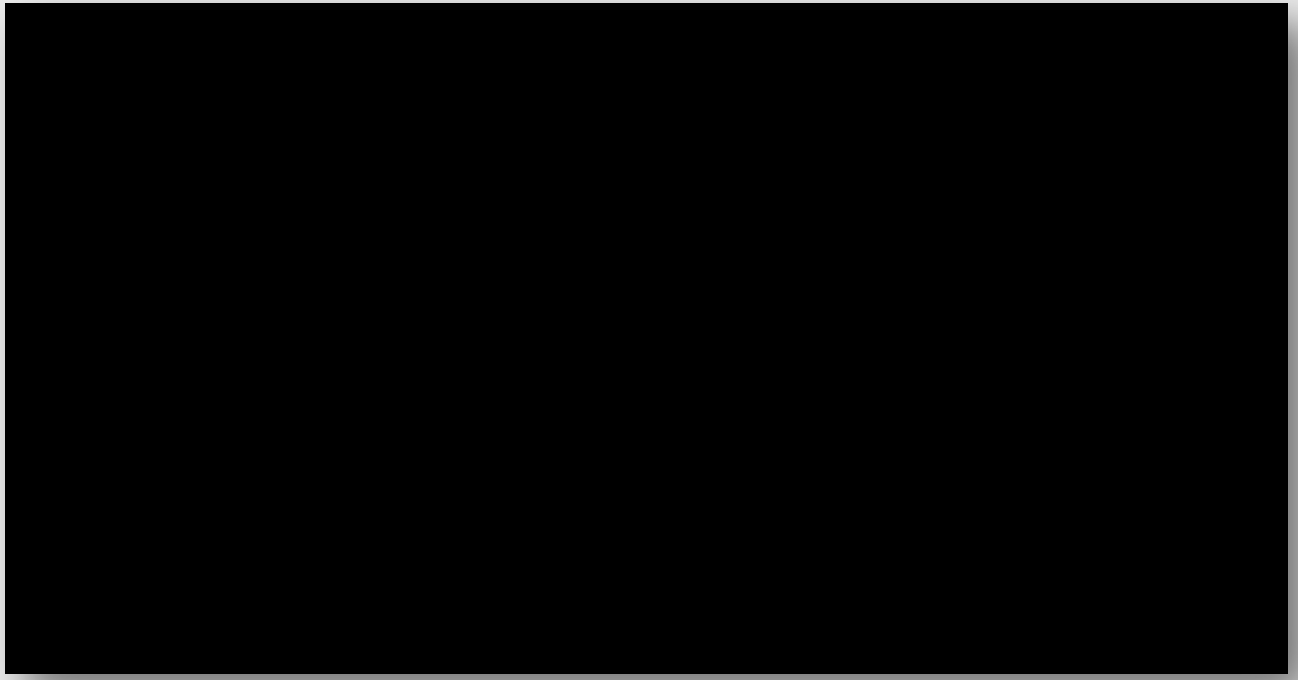


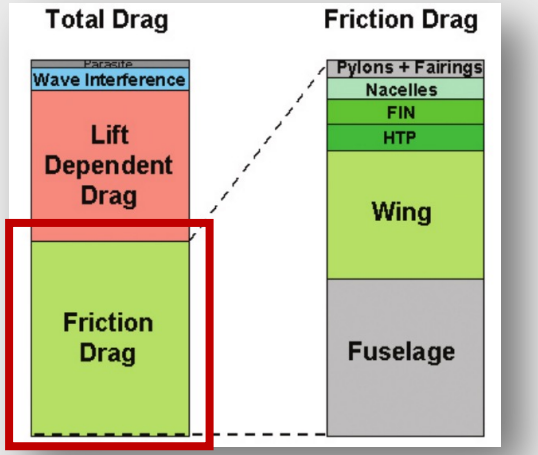
RAISE Partners:

SAFRAN
JÜLICH Forschungszentrum
CERFACS
Atos
1862 RĪGAS TEHNISKĀ UNIVERSITĀTE
Delphi Consortium
CERN
UNIVERSITY OF ICELAND
FLANDERS MAKE
ParTec Modular Supercomputing
BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación
RWTH AACHEN UNIVERSITY
The Cyprus Institute RESEARCH·TECHNOLOGY·INNOVATION

Example: Use Case on Turbulent Boundary Layer Flows

- About 50% of aircraft drag is friction drag, mostly due to turbulent boundary layers (TBL)
- Drag reduction of TBL via passive or active measures





Schrauf (Aeronaut. J., 2005)

- <u>Goal:</u> High drag reduction Δc_d and net power saving ΔP_{net}

Example: Use Case on Turbulent Boundary Layer Flows

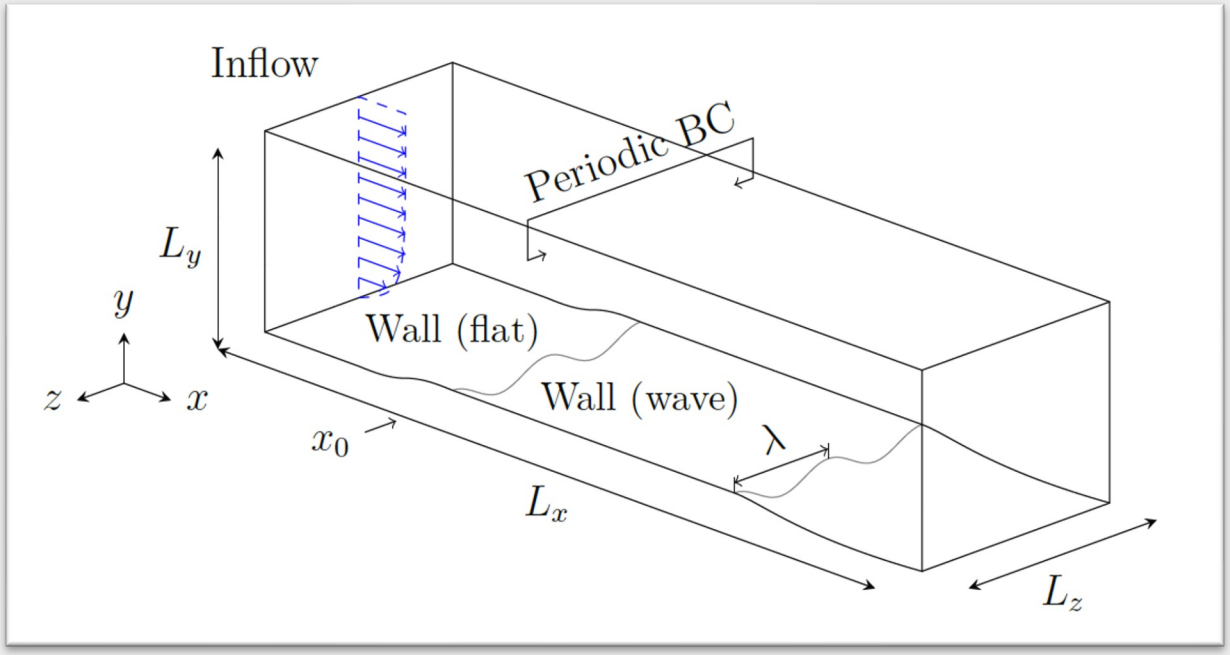
RAISE Center of Excellence

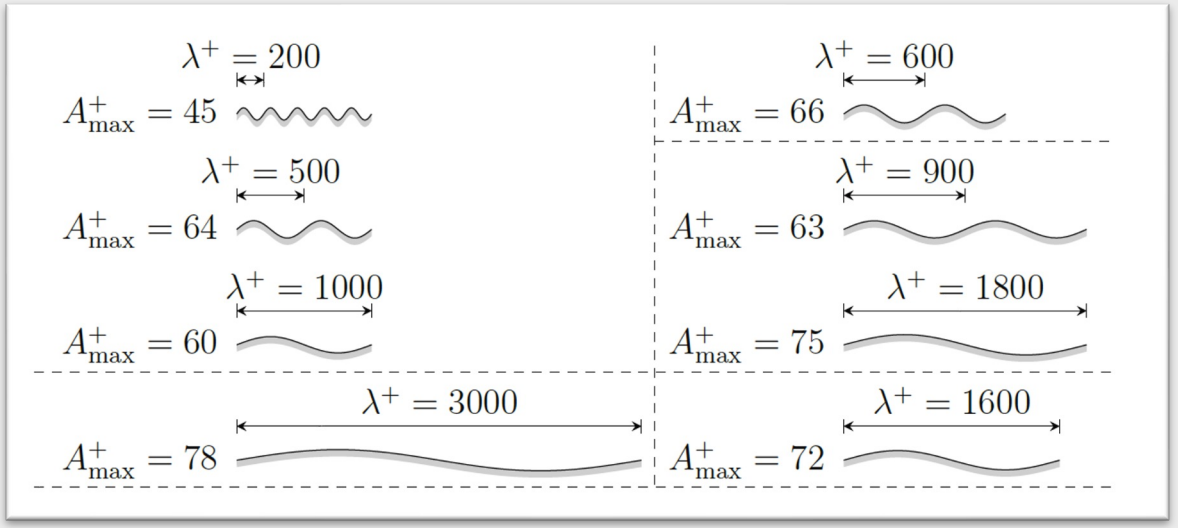
- High-fidelity simulations required:

- AI can help!
  - Data mining
  - Surrogate models
  - Physics modelling
  - Super-resolution
  - Simulation acceleration

Example: Use Case on Turbulent Boundary Layer Flows

- Research Question: Drag/Net Power dependence on actuation parameters





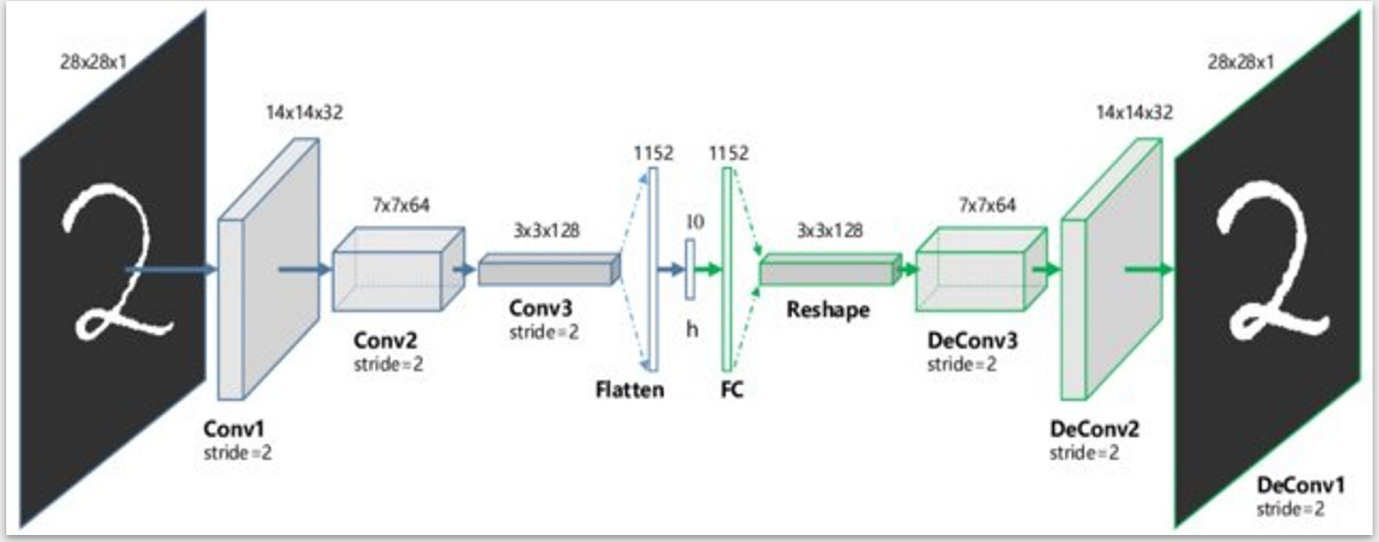
$$y^+|_{\text{wall}}(z^+, t^+) = A^+ \cos\left(\frac{2\pi}{\lambda^+} z^+ + \frac{2\pi}{T^+} t^+\right)$$

Example: AI/HPC methods for investigating TBL Flows

- Distributed learning methods
- Investigation of highly scalable frameworks
- Dimensionality reduction - ROMs, Autoencoders, Variational Autoencoders, Autoencoder-Generative networks
- Sequential models - temporal prediction, online learning
- Physics-informed Neural Networks - physical constraints in loss function, learning PDEs from data

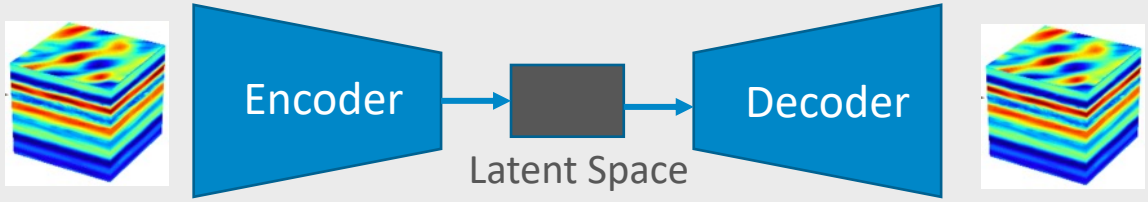
Autoencoders

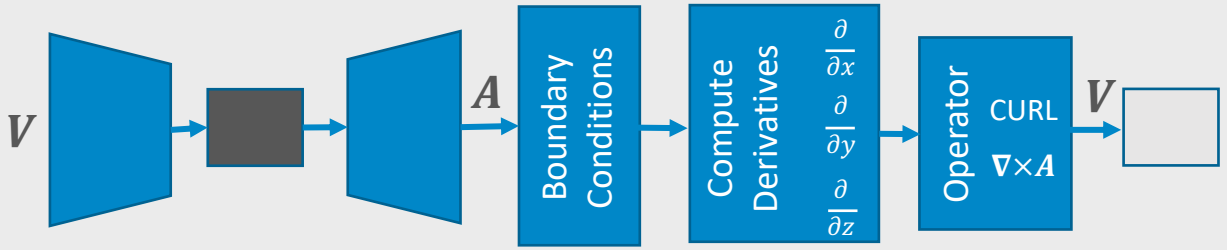
- Unsupervised, deterministic network
- Compression, denoising, segmentation, neural inpainting
- Code tensor or the latent space is typically obtained by flattening
- **Reconstruction loss** - e.g. in case of images, pixel-wise MSE between original and reconstructed images



Source: Guo, Xifeng & Liu, Xinwang & Zhu, En & Yin, Jianping. (2017). Deep Clustering with Convolutional Autoencoders. NeurIPS 2017, 373-382. 10.1007/978-3-319-70096-0_39.

Autoencoders for Turbulent Boundary Layer Flows





Plain autoencoders (Plain CAE)

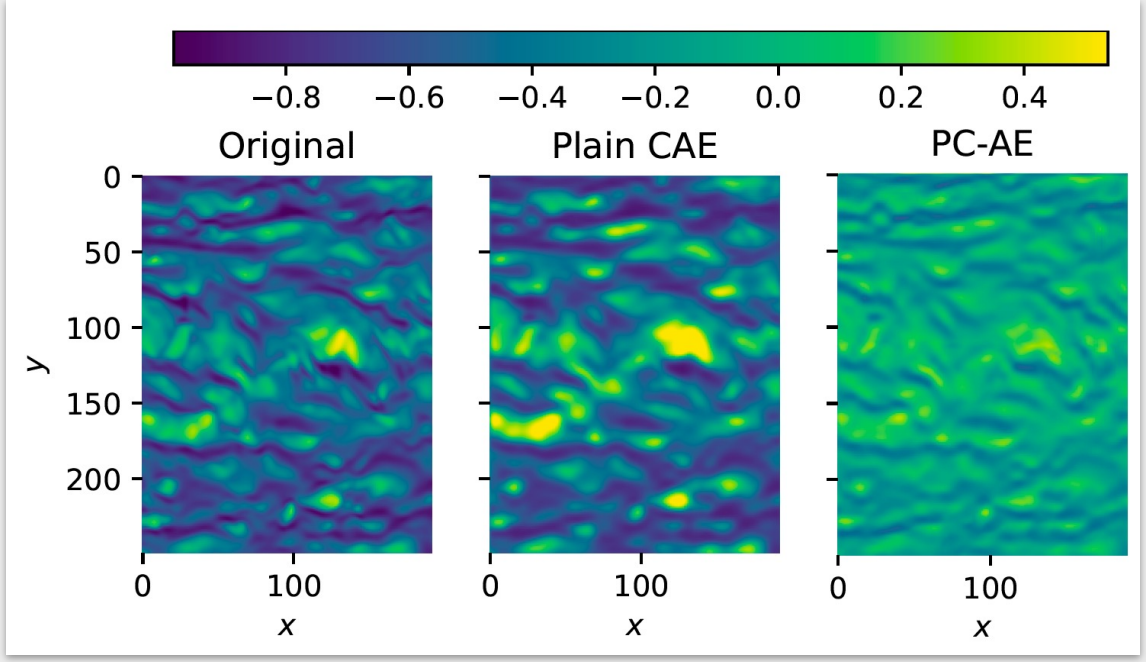
- 2-d convolutions with 4 hidden layers and decoding layers based on upsampling
  - Custom dataloader for handling non-uniform dimensions of dataset
  - Good accuracy on TBL dataset
  - Good parallel performance (results to follow)

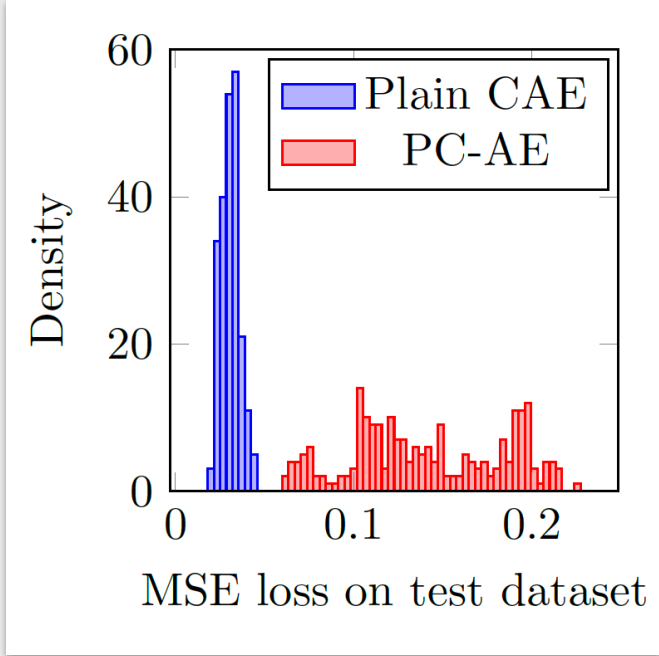
Physics-constrained autoencoders (PC-AE)*

- Trainable encoder-decoder part outputs velocity potential A
- FD stencils are CNN kernels with fixed, non-trainable weights
- Divergence free condition imposed in the final layer
- For the TBL data, slow convergence compared to plain autoencoder.

**Mohan et.al., 2020, Embedding Hard Physical Constraints in Convolutional Neural Networks for 3D Turbulence, ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*

Autoencoders for Turbulent Boundary Layer Flows



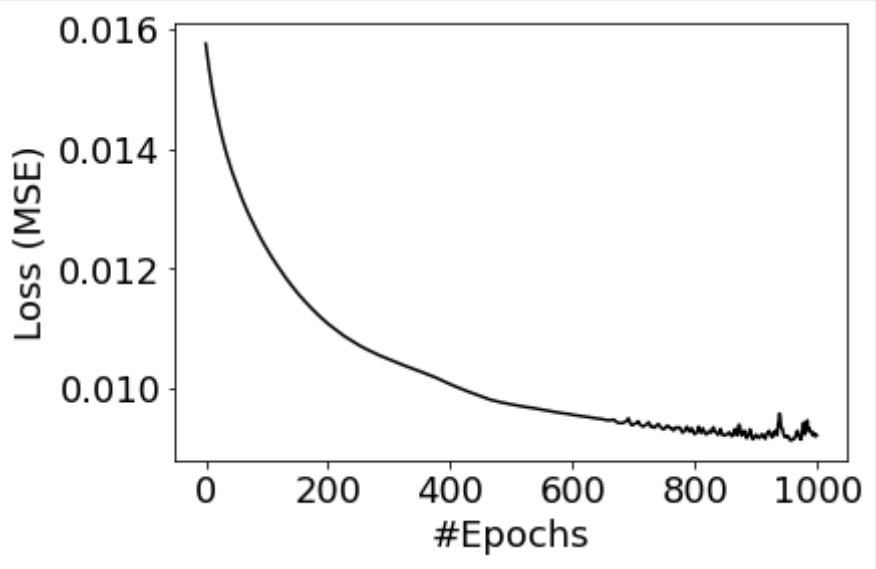


- Plain CAE provides good reconstructions compared to PC-AE
- Incompressibility condition in PC-AE is probably not valid in the entire domain – another physical constraint required

Why distributed training?

Training using 1 GPU:*

- 7 hours per epoch**
- 1000 epochs takes 10 months
- many runs for hyperparameter tuning ☹

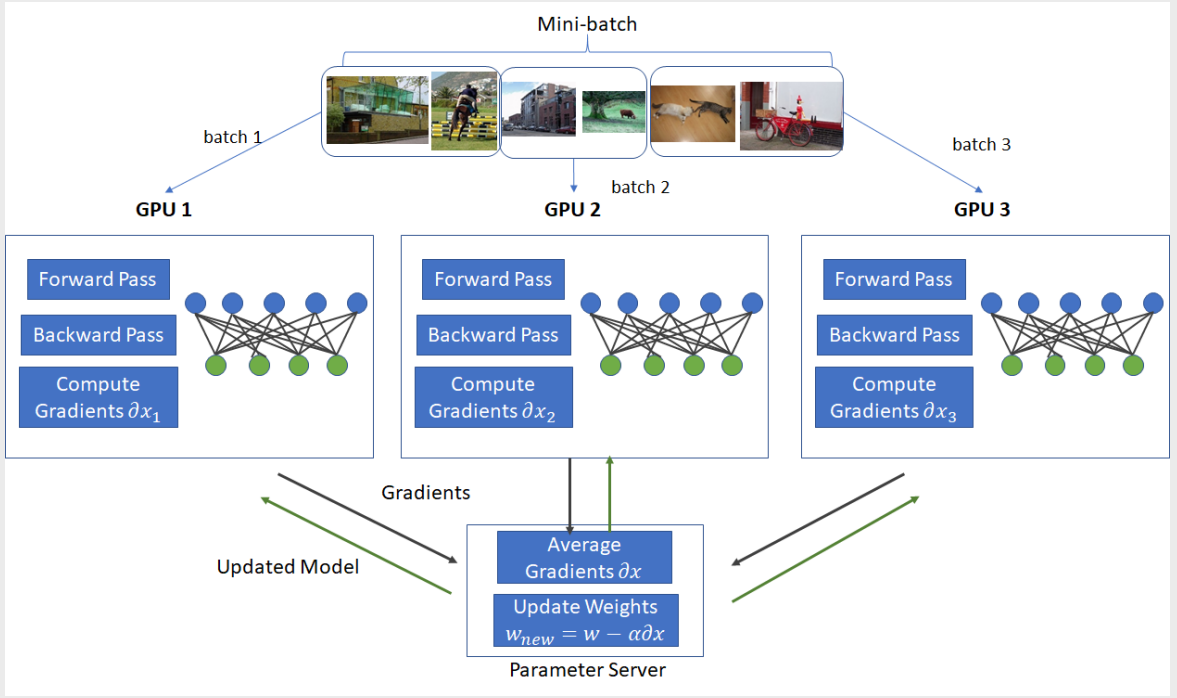


a solution: distributed data parallelism (DDP)

*NVIDIA A100 on JURECA DC
**extrapolated from four A100 runs assuming perfect scaling

Distributed Data Parallelism (DDP)

- mini-batch is split to smaller batches
- Identical network
- Server gathers, updates and sends parameters



https://www.telesens.co/2017/12/25/understanding-data-parallelism-in-machine-learning/

- Intercomm depends on:
  - Network and/or
  - Data loading
- Mini-batch size becomes large for many GPUs
  - Accuracy changes
  - Tuning required

Hardwares

- Tier-0/1 + Prototype HPCs
- *FPGA on the way*

Location	System name	CPU	Accelerators
CINECA	Marconi100	1,960 IBM POWER9	3,920 NVIDIA V100
FZJ	Juwels Cluster + Booster	2,560 Intel Xeon	3,774 NVIDIA A100
FZJ	Jureca DC	1,536 AMD EPYC	768 NVIDIA A100
FZJ	DEEP-EST	147 Intel Xeon	75 NVIDIA V100
FZJ	JUAWEI	11 ARM HiSilicon	
HLRS	Hawk	11,264 AMD EPYC	64 NVIDIA V100*
CSCS	Piz Daint	9,330 Intel Xeon	68,448 NVIDIA P100
BSC	Marenostrum4	6,912 Intel Xeon	
BSC	CTE-AMD	33 AMD EPYC	
BSC	CTE-ARM	192 ARM A64FX	
BSC	HUAWEI	16 ARM Kunpeng 92	
CEA	Joliot-Curie	2,484 Intel Xeon + 2,	
LRZ	SuperMUC-NG	6,480 Intel Xeon	64 NVIDIA V100

Individual care:

- Software stack
- Code optimization
- Node communication

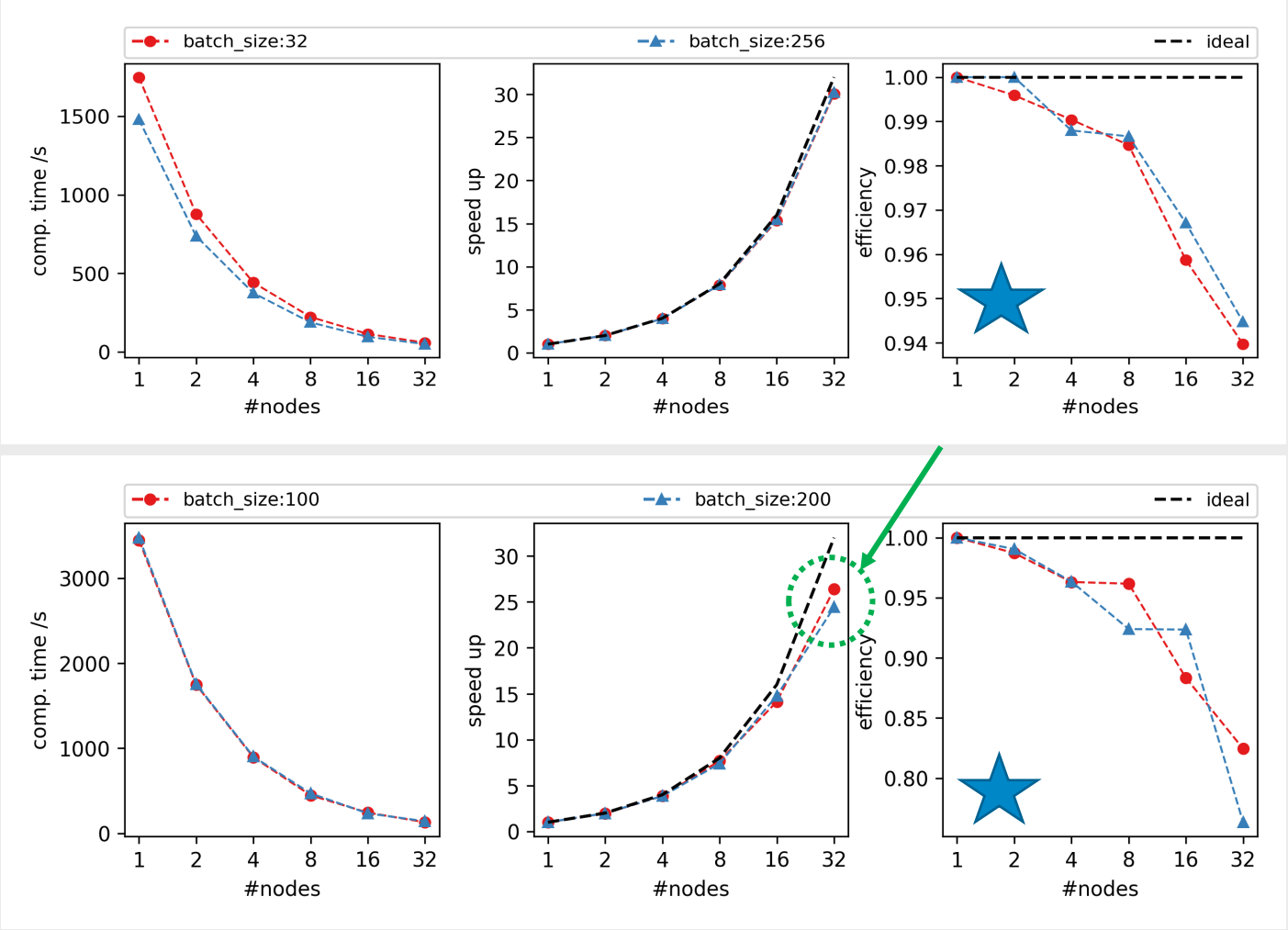
*cloud nodes

Setup

- Frameworks:
  1. Distributed Data Parallel (DDP) - PyTorch
  2. Horovod – Uber
  3. HeAT – Helmholtz Analytics Framework
  4. DeepSpeed - Microsoft
- Datasets:
  1. MNISTx100 (MNIST concatenated 100 times)
  2. ImageNet
  3. TBL (TBL-large) full 8.3TB
  4. TBL (TBL-small) 0.25% of TBL-large (22GB/8.3TB)

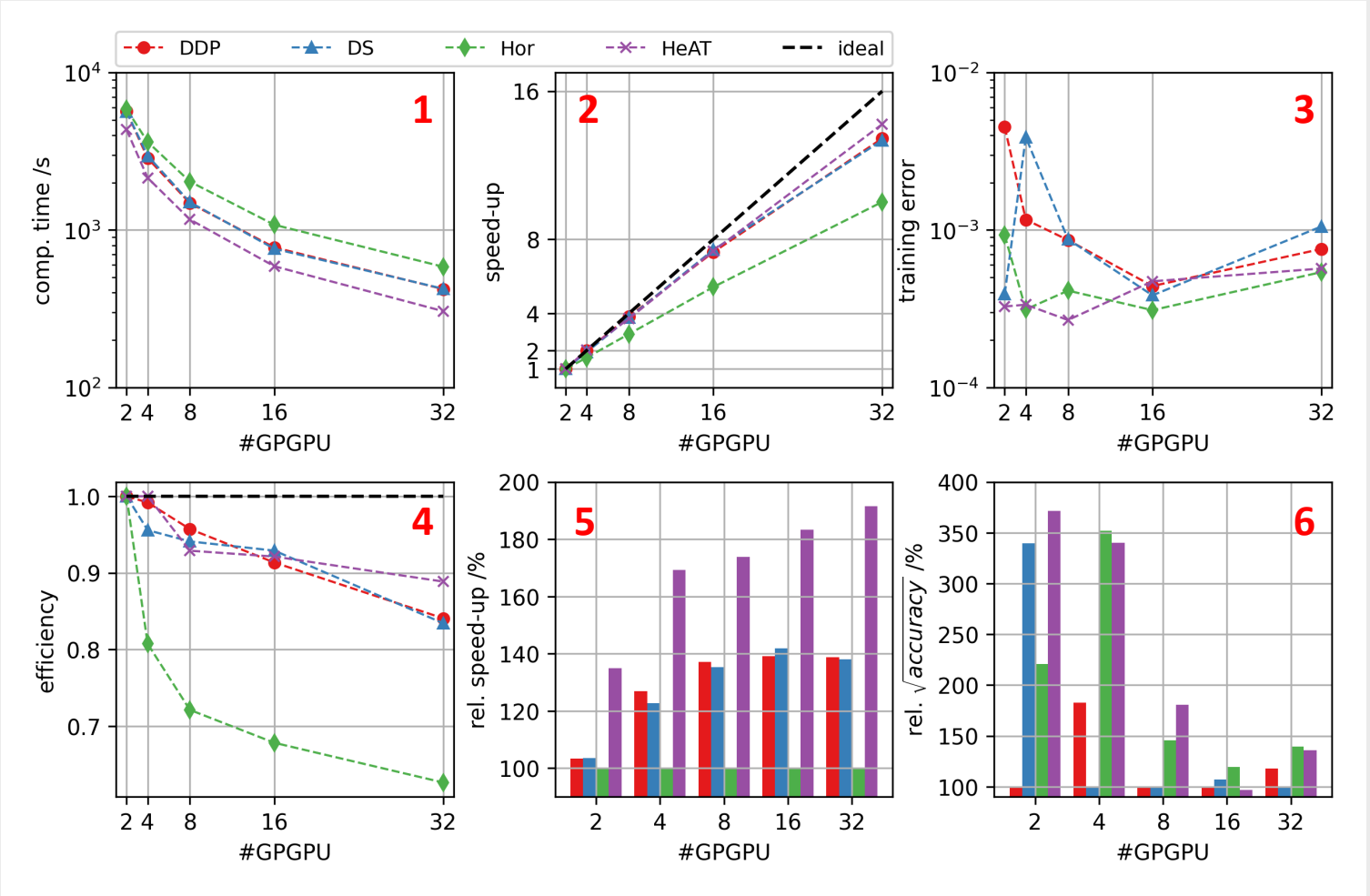
First tests (prototype DEEP-EST in FZJ)

- FW=PyTorch-DDP
  Dataset=MNISTx100
  Epoch=10
  Learning Rate=0.01
  Batch Size=[32,256]
  CNN training
  22K parameters
- FW=PyTorch-DDP
  Dataset=TBL-small
  Epoch=10
  Learning Rate=0.01
  Batch Size=[100,200]
  CAE training
  10K parameters



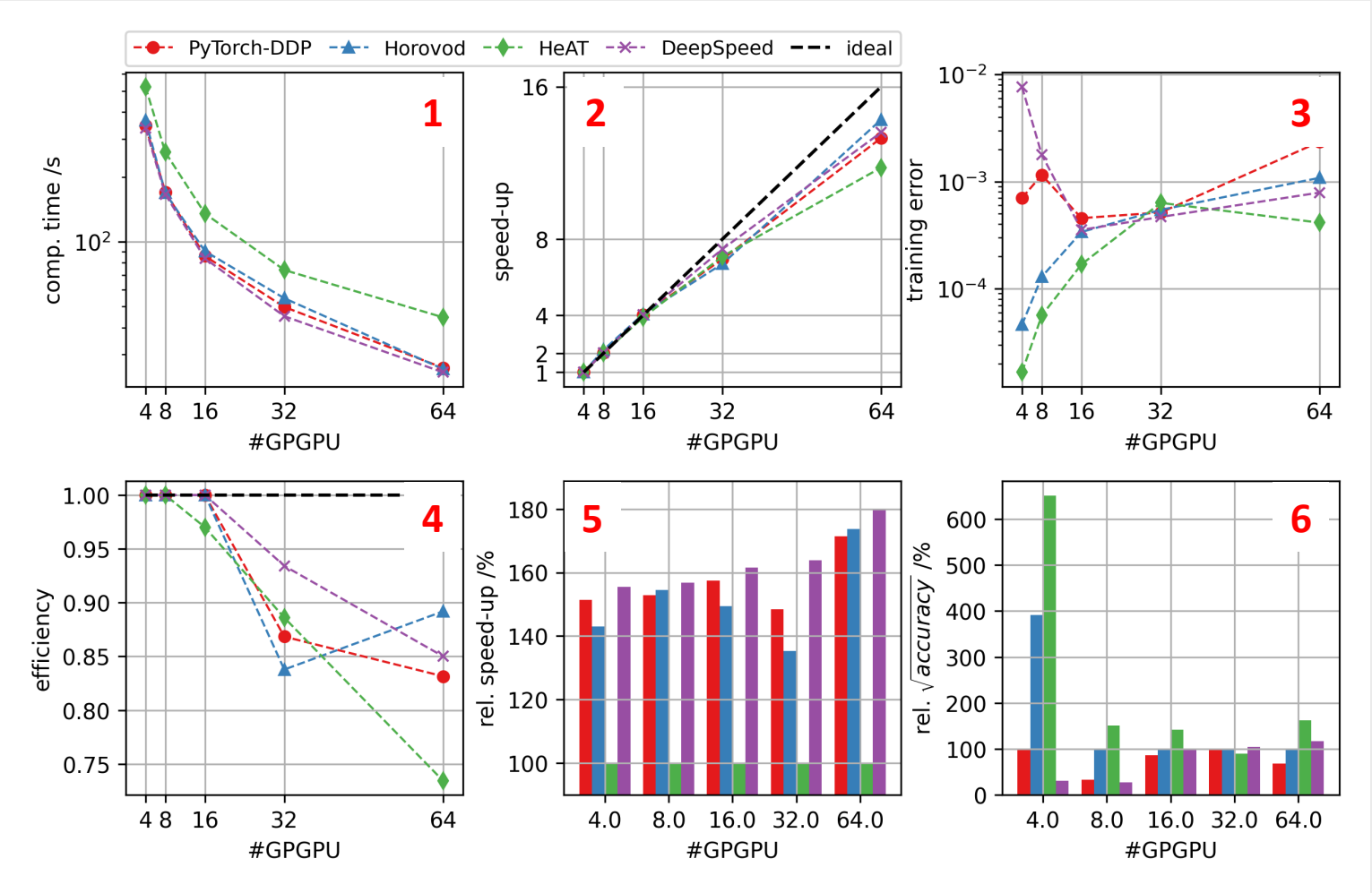
All frameworks (prototype CTE-AMD in BSC)

Dataset=TBL-small
Epoch=10
Learning Rate=0.01
Batch size=96
CAE training
10K parameters



Frameworks (TIER 0/1 Jureca in FZJ)

Dataset=TBL-small
Epoch=10
Learning Rate=0.01
Batch size=96
CAE training
10K parameters

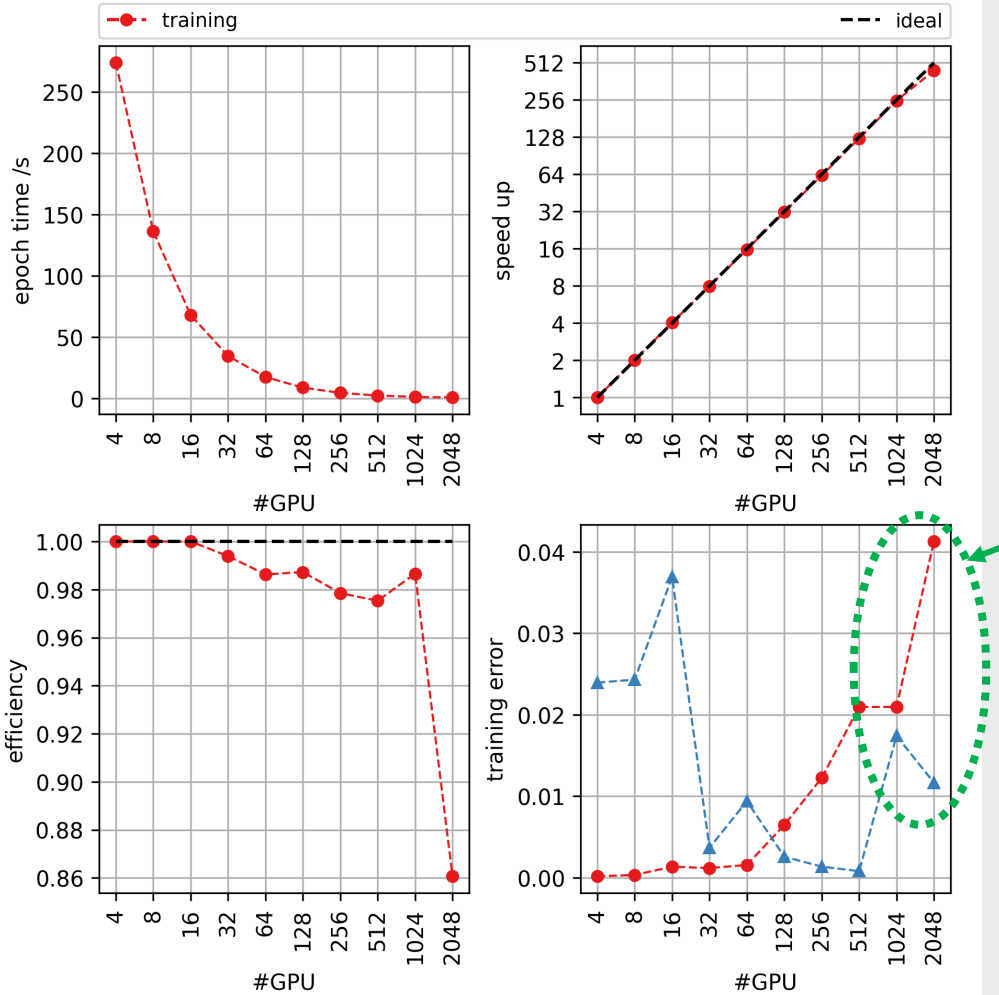


First test of a lot of GPUs (TIER 0/1 Juwels in FZJ)

Dataset=TBL-small
Epoch=10
Batch size=96
CAE training
10K parameters

Two runs:

- Learning Rate=1e-4
- Learning Rate=1e-4*#GPU



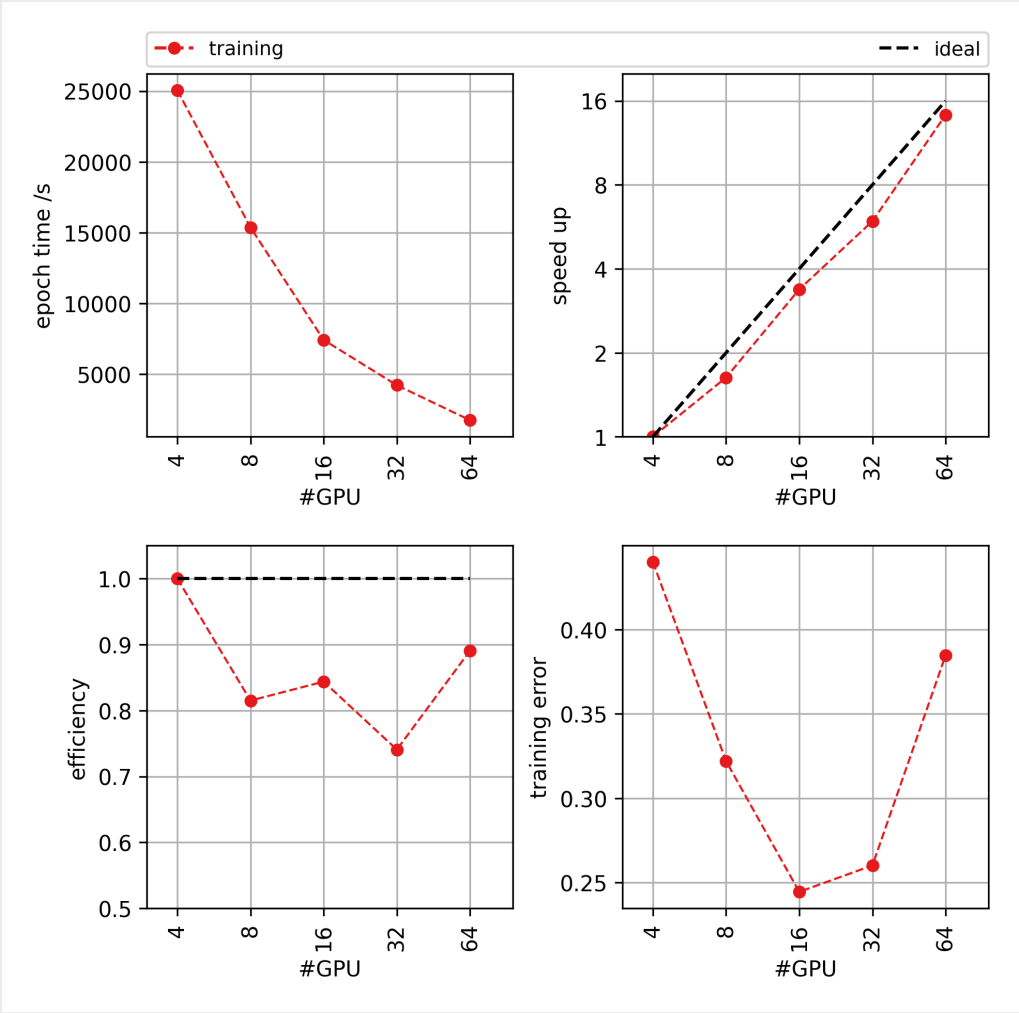
- Perfect scaling for:
  - small networks – less comm
  - Small data set – fast data transfer
- Hyperparameter tuning helps

Full dataset (TIER 0/1 Jureca in FZJ)

FW=PyTorch-DDP
Dataset=*TBL-large*
Epoch=3
Batch size=6
Learning rate=1e-4

CAE training:

- 66K parameters
- 39GB/40GB GPU memory



- Good scaling performance
- 25000 sec to 1750 sec per epoch
- Code optimization?

Code-optimization

What to focus?

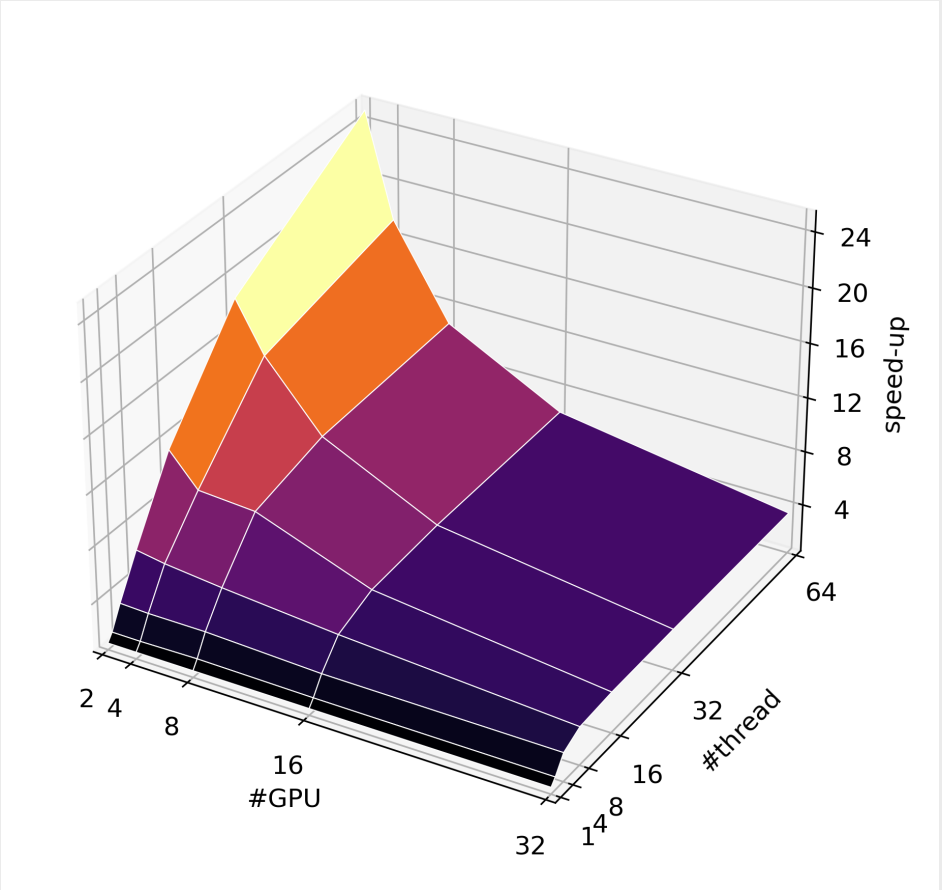
1. Data loader *(large data sets)*
   - Loader libraries (ex. DALI)
   - Multi-thread
   - Preload algorithms
   - Dataset type precision
   - Memory placement
2. Training *(large networks)*
   - *cuDNN* algorithms
   - Warm-up
   - Network parameter precision (ex. AMP)
   - Model parallelism

PyTorch Profiler:

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	Self CUDA	Self CUDA %
enumerate(DataLoader)#_SingleProcessDataLoaderIter._...	25.08%	69.389s	47.92%	132.573s	12.052s	69.420s	48.88%
aten::min	5.58%	15.426s	6.49%	17.955s	99.747ms	15.460s	10.89%
aten::sub	4.88%	13.498s	4.89%	13.524s	50.091ms	13.509s	9.51%
aten::copy_	3.88%	10.720s	4.58%	12.661s	10.464ms	12.534s	8.83%
aten::max	2.81%	7.764s	3.24%	8.970s	99.663ms	7.768s	5.47%
aten::_cat	2.56%	7.094s	2.59%	7.167s	79.637ms	7.114s	5.01%
aten::div	1.98%	5.470s	1.98%	5.473s	60.806ms	5.472s	3.85%
aten::mul	1.91%	5.284s	1.93%	5.349s	17.254ms	5.284s	3.72%
aten::cudnn_batch_norm_backward	0.00%	3.414ms	0.02%	60.089ms	858.414us	1.378s	0.97%
aten::cudnn_batch_norm	0.01%	22.138ms	0.02%	46.367ms	662.386us	683.501ms	0.48%
aten::cudnn_convolution_backward_weight	0.00%	9.100ms	0.14%	374.162ms	4.677ms	570.337ms	0.40%
aten::cudnn_convolution	0.00%	11.300ms	0.05%	136.744ms	1.709ms	530.339ms	0.37%
aten::cudnn_convolution_backward_input	0.00%	9.412ms	0.02%	67.324ms	961.771us	466.414ms	0.33%
aten::upsample_bilinear2d_backward	0.00%	546.000us	0.00%	2.621ms	65.525us	393.846ms	0.28%
aten::upsample_bilinear2d	0.00%	802.000us	0.01%	39.237ms	980.925us	383.745ms	0.27%
aten::leaky_relu_backward	0.00%	1.385ms	0.00%	2.025ms	28.929us	137.264ms	0.10%
aten::fill_	0.02%	51.785ms	0.05%	137.154ms	253.989us	136.208ms	0.10%
aten::empty	0.04%	102.167ms	0.10%	278.413ms	165.427us	125.344ms	0.09%
aten::leaky_relu	0.01%	30.508ms	0.02%	42.224ms	603.200us	114.008ms	0.08%
aten::empty_strided	0.00%	10.856ms	0.03%	70.465ms	140.930us	65.535ms	0.05%
aten::resize_	0.00%	6.681ms	0.13%	353.828ms	643.324us	51.134ms	0.04%
aten::view	0.00%	5.323ms	0.02%	59.511ms	85.016us	47.758ms	0.03%
DistributedDataParallel.forward	0.01%	28.707ms	0.17%	471.766ms	47.177ms	42.159ms	0.03%
aten::clone	0.02%	48.276ms	1.29%	3.565s	12.731ms	41.040ms	0.03%
aten::contiguous	0.00%	12.494ms	1.29%	3.578s	13.250ms	36.747ms	0.03%
aten::narrow	0.01%	23.243ms	0.02%	62.542ms	195.444us	35.270ms	0.02%
aten::to	0.01%	26.056ms	2.35%	6.498s	13.539ms	34.515ms	0.02%
aten::slice	0.01%	25.389ms	0.03%	91.253ms	90.350us	32.823ms	0.02%
aten::as_strided	0.01%	31.250ms	0.02%	60.293ms	44.009us	26.657ms	0.02%
aten::empty_like	0.01%	33.827ms	0.03%	87.175ms	235.608us	18.955ms	0.01%
aten::_to_copy	0.00%	7.923ms	2.31%	6.377s	13.286ms	11.047ms	0.01%
aten::mse_loss	0.00%	355.000us	0.00%	3.003ms	150.150us	10.264ms	0.01%

Optimization – ex. multi-threads

FW=PyTorch-DDP
Dataset=TBL-small
(22 GB)
Epoch=10
Learning Rate=1e-4
Batch size=96
10K parameters

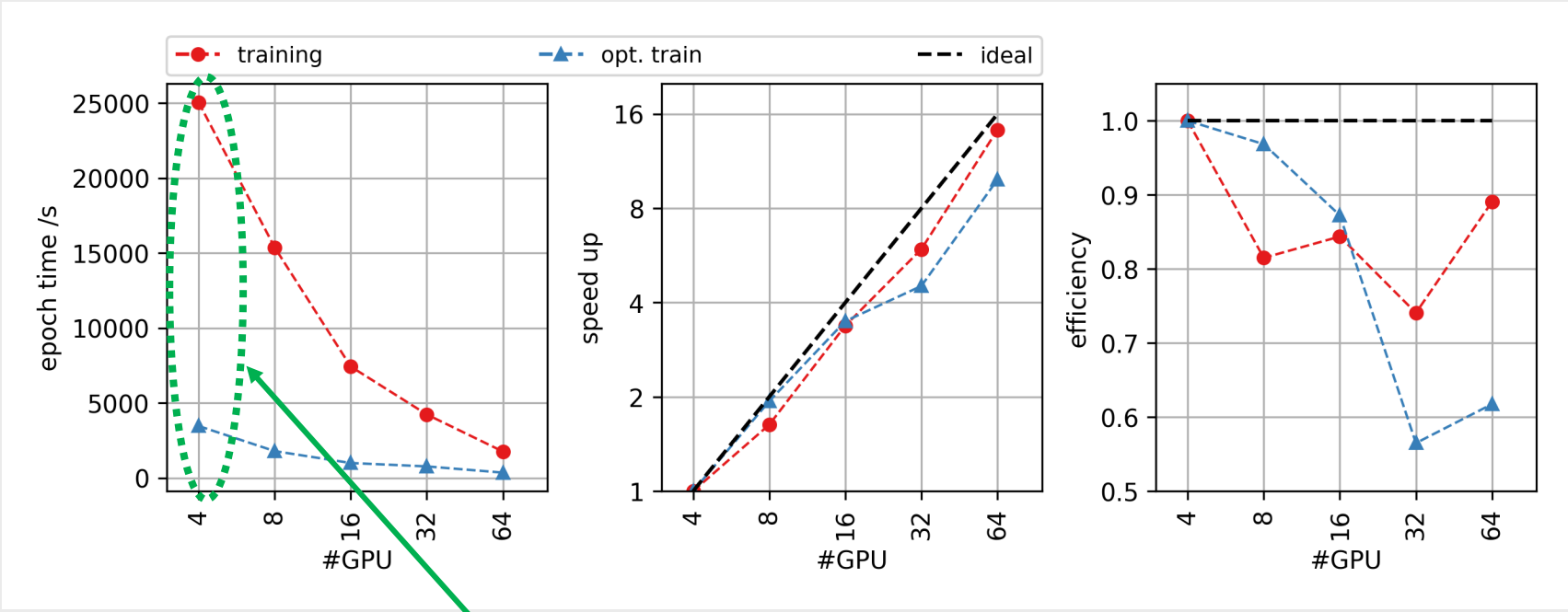


- Bottleneck: large data sets
- Unstable if not correctly configured

Optimization – results

FW=PyTorch-DDP
Dataset=*TBL-large*
Epoch=3
Batch size=3
Learning rate=1e-4

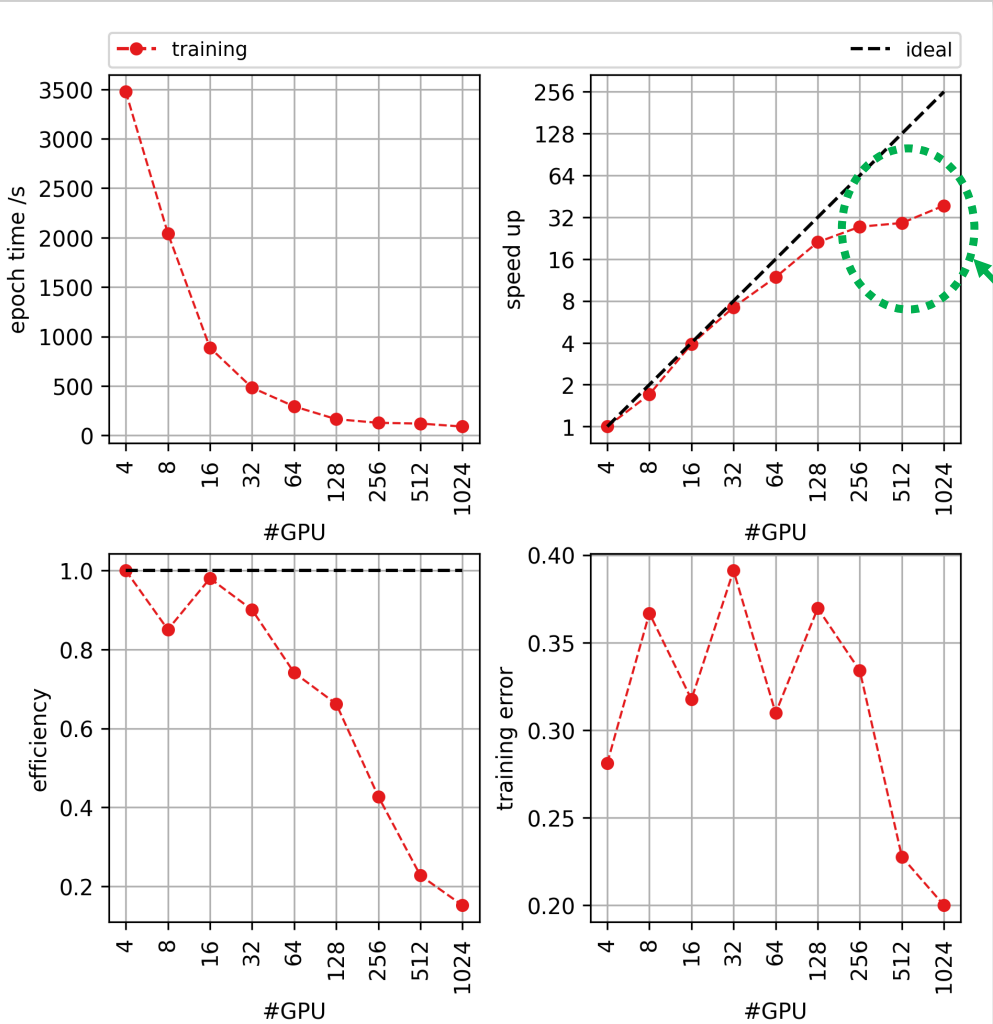
32 threads



- 25000 sec to 3450 sec per epoch

Back to Juwels in FZJ w/ optimizations

FW=PyTorch-DDP
Dataset=*TBL-large*
Epoch=3
Batch size=3
Learning rate=1e-4
CAE training
66K parameters



- scaling up to 128 GPUs
  - Need larger dataset to test 256+ GPUs
- 100000 sec to 160 sec per epoch
  (1 GPU non-optimized vs 128 GPU optimized)

Conclusion & outlook

- Distributed data parallel approach fits to CFD
- Existing frameworks perform different
- Individual optimizations required – data set size / network size

Outlook:

- Data parallel with model parallelism
- Automation
- Tailored APUs

Thank you for your attention

drive. enable. innovate.

RAISE Center of Excellence

The CoE RAISE project have received funding from the European Union's Horizon 2020 – Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733

Follow us: