HELMHOLTZ

RESEARCH FOR
GRAND CHALLENGES

HEAT

Helmholtz Analytics Toolkit

Claudia Comito, Björn Hagemeier

Jülich Supercomputing Center

September 9th, 2021

Getting started on JUWELS

Part 1

>> LOGIN (instructions from Björn)

```
>> jutil env activate -p training2121

>> module load GCC Python CUDA ParaStationMPI SciPy-Stack NCCL mpi4py

>> cd $PROJECT/users/

>> mkdir [YOUR_USER_NAME]

>> cd [YOUR_USER_NAME]
>> cp -r $PROJECT/users/comito1/create_heat_venv.sh .

>> ./create_heat_venv.sh

>> source $PROJECT/users/[YOUR_USER_NAME]/heat-1.1/bin/activate
```

Getting started on JUWELS

Part 2

```
>> cp -r ../comito1/training*

>> cd training/examples

EDIT sbatch_run.sh to activate your virtual environment! (Replace
‘comito1’ with your username)

>> sbatch sbatch_run.sh [py FILE]


(Suggestion) ADD TO YOUR .bashrc:

>> jutil env activate -p training2121

>> module load GCC Python CUDA ParaStationMPI SciPy-Stack NCCL mpi4py

>> source $PROJECT/users/[YOUR_USER_NAME]/heat-1.1/bin/activate
```

Getting started on your machine

Requirements:

- Python >= 3.7
- MPI
- (CUDA)

>> pip install heat[hdf5,netcdf]

This talk

- Background
- Heat: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

- Cross-discipline pilot project: neuroscience, aerospace, biology, earth science, …
- Provide general methods for Scientific Big Data Analytics (incl. ML/NN)
- Heat target: SciPy/NumPy users
  - overcome single-node memory bottlenecks
  - parallelise calculations transparently
  - https://github.com/helmholtz-analytics/heat

This talk

- Background
- HeAT: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

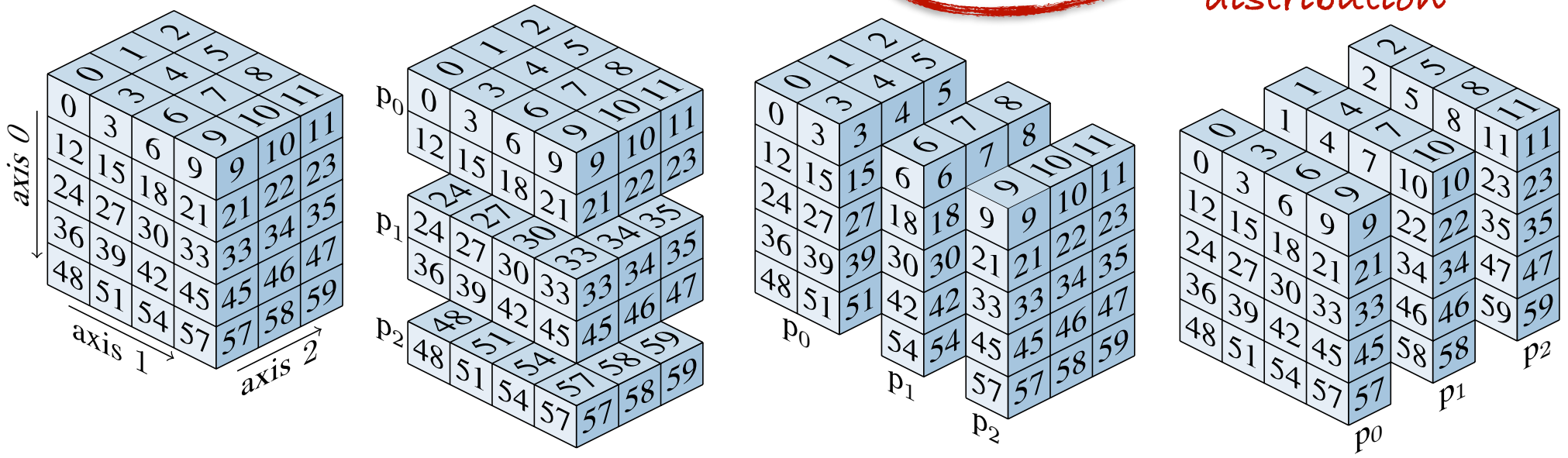
- open-source Python library
- PyTorch-based (eager execution)
- Parallelisation under the hood
- Multi-node (MPI)
- Multi-GPU
- https://heat.readthedocs.io/

HeAT data objects: DNDarrays

(Distributed N-Dimensional Array, on 3 processes)

```
import heat as ht
data = ht.arange(60).reshape(5,4,3)
data = ht.array(data, split=0)
```

Enable data distribution

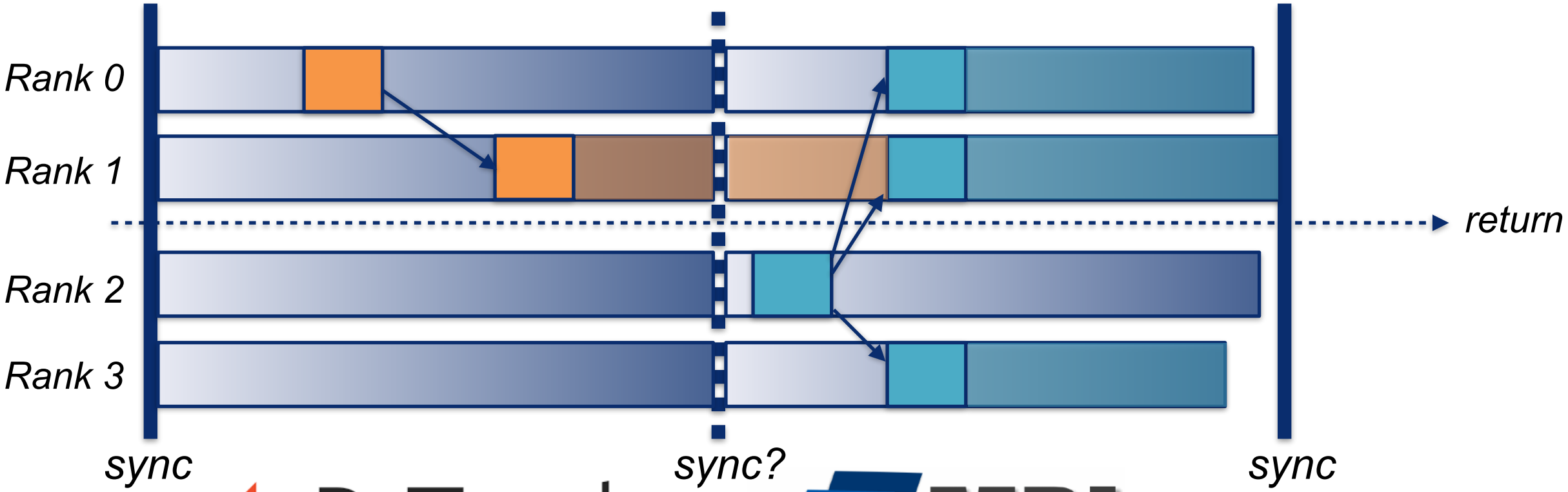


(a) `split=None` (b) `split=0` (c) `split=1` (d) `split=2`

[Götz et al., IEEE BigData 2020]

Programming model

Heat: Bulk-Synchronous + Asynchronous



PyTorch

MPI

mpi4py

This talk

- Background
- Heat: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

- NumPy API, ideally:

```
import heat as np
```

- Distributed linear algebra, reductions, binary ops, manipulations…
- Parallel I/O
- (multi-)GPU support

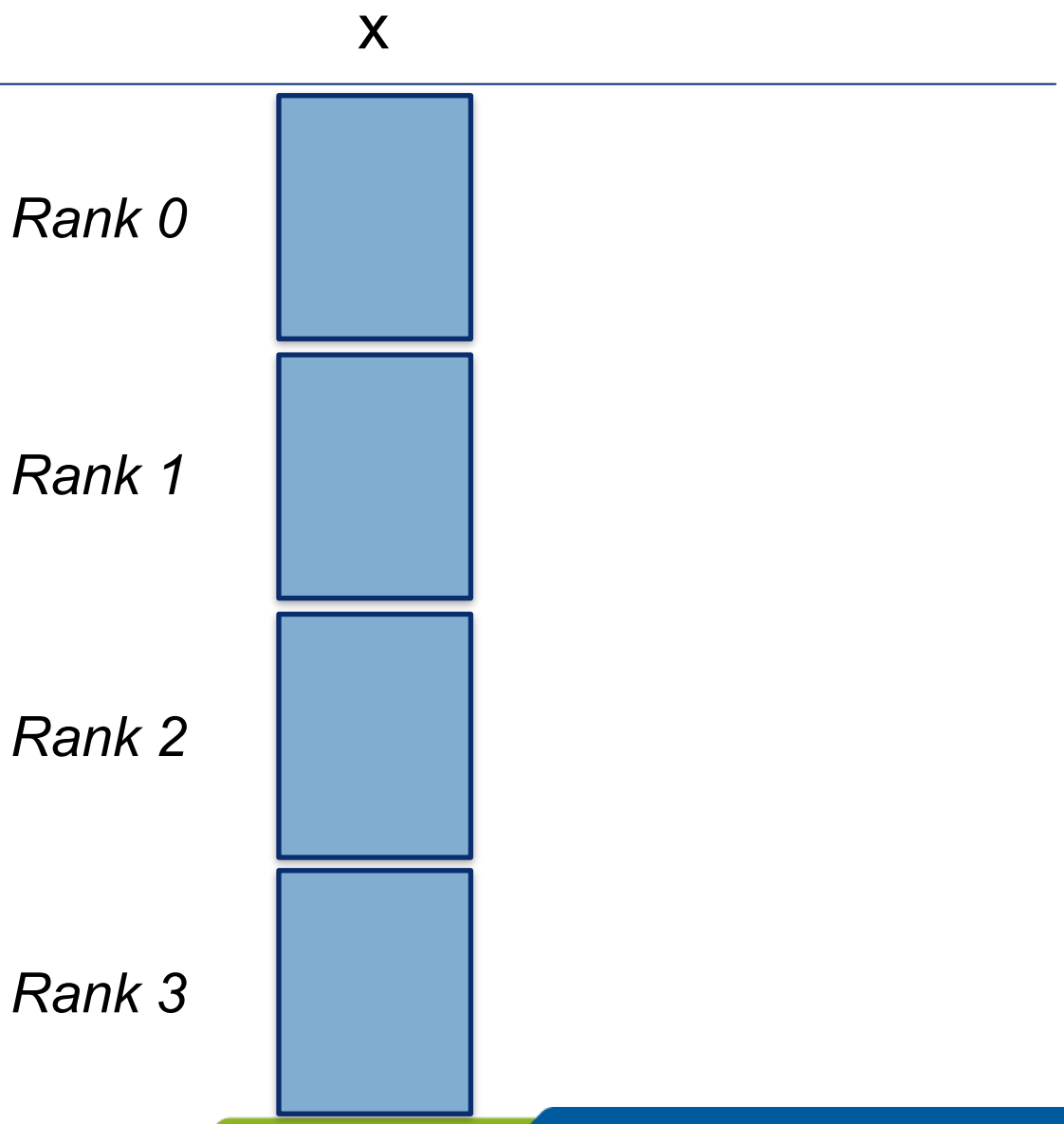
```
data = ht.array(data, device=“gpu”)
```

Examples:

4 processes

```
import heat as ht

# x from file/array/tensor
x = ht.array(x)
```



Examples:

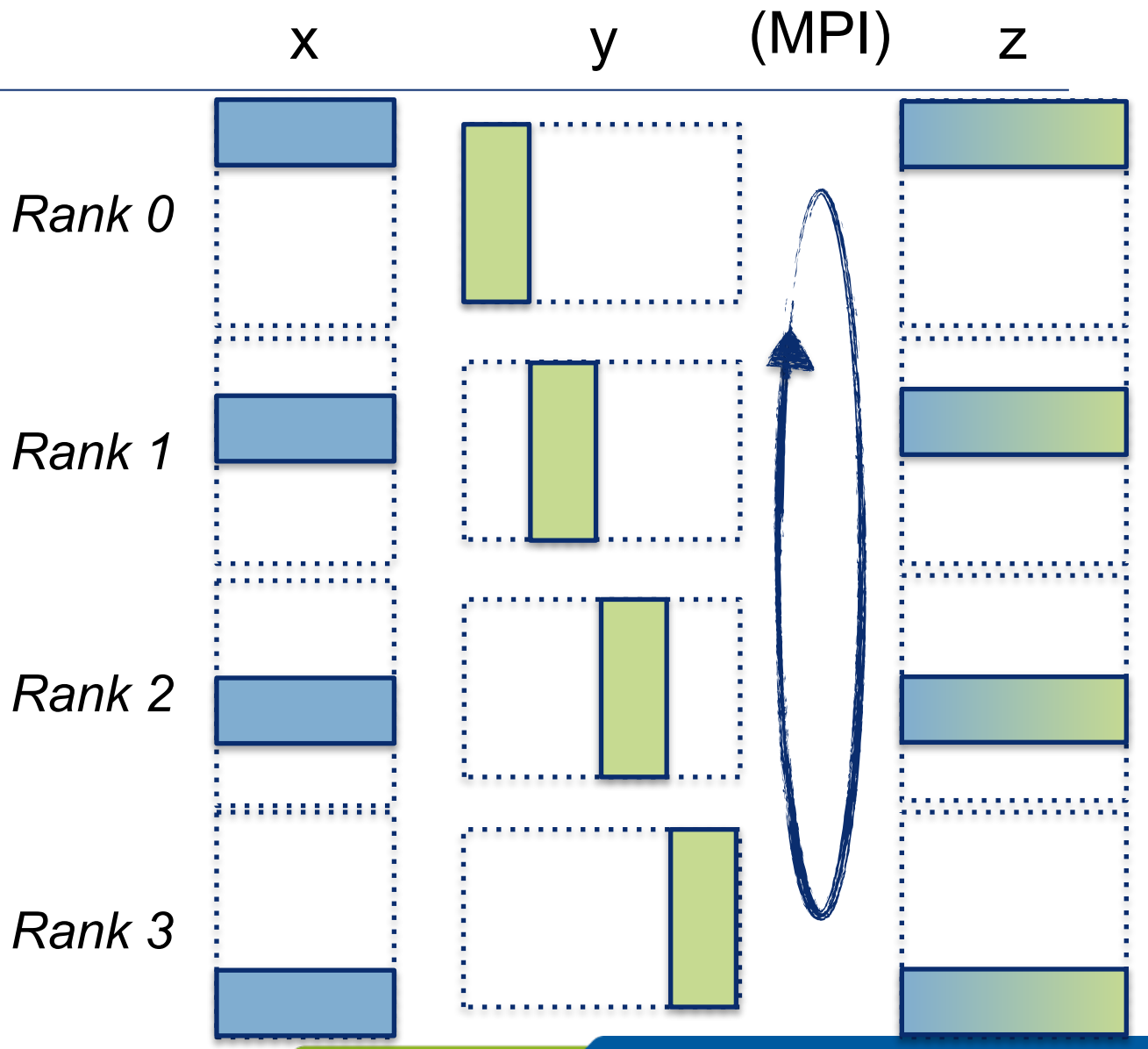
4 processes

```
import heat as ht

# x from file/array/tensor
x = ht.array(x, split=0)

y = ht.array(y, split=1)

z = ht.dot(x, y)
```



Examples:

4 processes

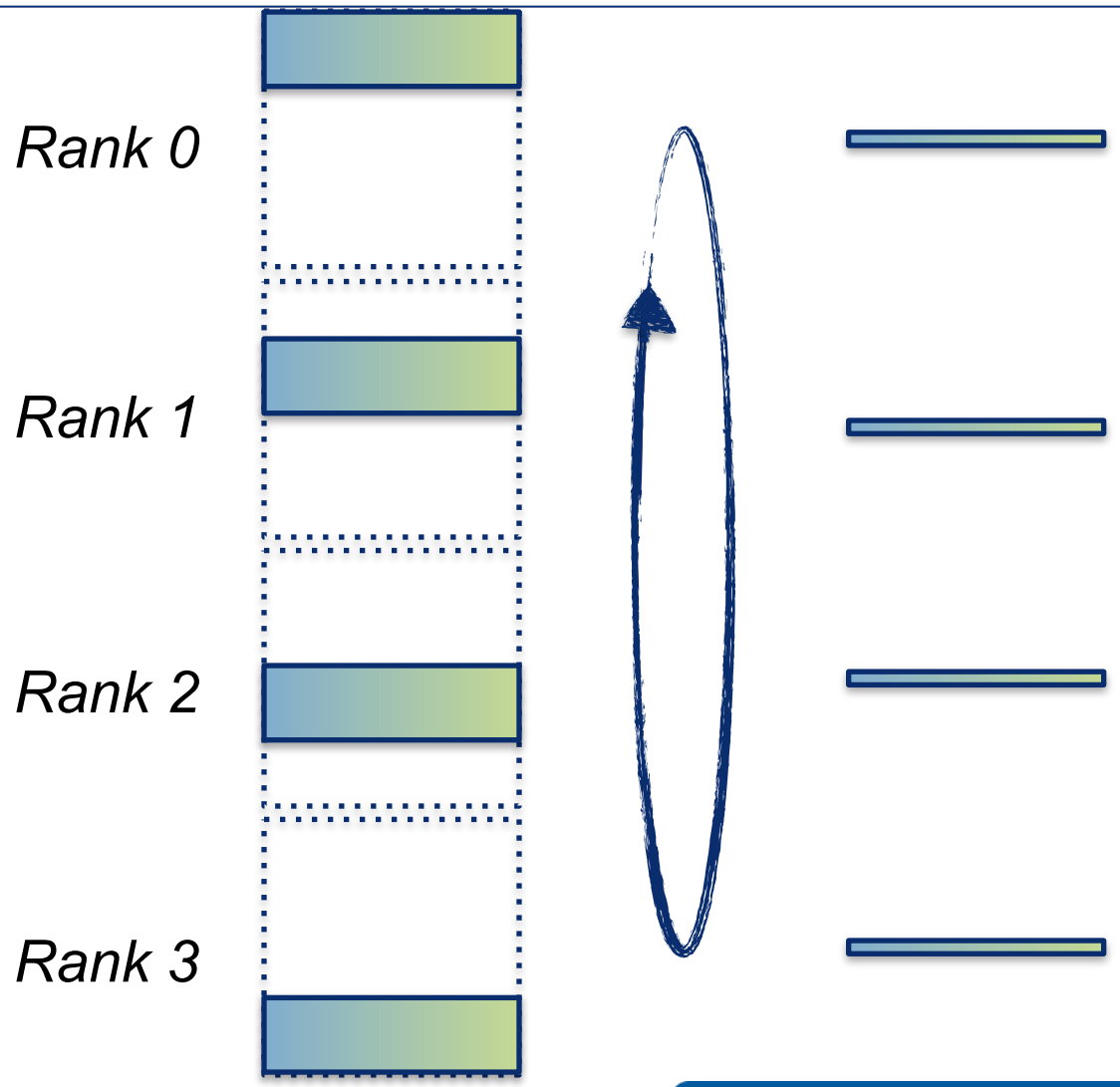
```
import heat as ht

# x from file/array/tensor
x = ht.array(x, split=0)

y = ht.array(y, split=1)

z = ht.dot(x, y)

t = z.sum(axis=0)
z.reshape()
z.transpose()
sort()
unique()
std()
percentile()
x*y
```



Examples

DNDarrays

INTERACTIVE SESSION:

```
>> salloc --account=training2121 --nodes=1 --time=00:30:00 --gres=gpu:4
>> cd training/examples
>> source $PROJECT/users/[USER_NAME]/heat-1.1/bin/activate
>> srun --ntasks-per-node=4 --gres=gpu:4  python 1_arrays.py
>> srun --ntasks-per-node=4 --gres=gpu:4  python 2_distributed_arrays.py
```

SBATCH:

```
>> cd training/examples
>> sbatch sbatch_run 1_arrays.py
(Output in slurm….out)
```

training/examples/1_arrays.py

```python
import numpy as np
import torch
import heat as ht


np_data_3d = np.arange(60).reshape(5,4,3)
torch_data_3d = torch.arange(60).reshape(5,4,3)

# wrap data into a DNDarray
ht_data_3d = ht.array(np_data_3d)

# DNDarray is MPI-aware
rank = ht_data_3d.comm.rank
size = ht_data_3d.comm.size

print(f"On rank {rank} of {size}:")
print(f"DNDarray global shape: {ht_data_3d.shape}")
print(f"DNDarray  local shape: {ht_data_3d.lshape}")
```

```
On rank 3 of 4:
DNDarray global shape: (5, 4, 3)
DNDarray  local shape: (5, 4, 3)
On rank 0 of 4:
DNDarray global shape: (5, 4, 3)
DNDarray  local shape: (5, 4, 3)
On rank 2 of 4:
DNDarray global shape: (5, 4, 3)
DNDarray  local shape: (5, 4, 3)
On rank 1 of 4:
DNDarray global shape: (5, 4, 3)
DNDarray  local shape: (5, 4, 3)
```

training/examples/2_distributed_arrays.py

```python
import numpy as np
import torch
import heat as ht


np_data_3d = np.arange(600).reshape(-1,4,3)
torch_data_3d = torch.arange(600).reshape(-1,4,3)

# wrap data into a DNDarray
ht_data_3d = ht.array(torch_data_3d, split=0)

# DNDarray is MPI-aware
rank = ht_data_3d.comm.rank
size = ht_data_3d.comm.size

print(f"On rank {rank} of {size}:")
print(f"DNDarray global shape: {ht_data_3d.shape}")
print(f"DNDarray  local shape: {ht_data_3d.lshape}")
```

```
On rank 0 of 4:
DNDarray global shape: (50, 4, 3)
DNDarray  local shape: (13, 4, 3)
On rank 3 of 4:
DNDarray global shape: (50, 4, 3)
DNDarray  local shape: (12, 4, 3)
On rank 2 of 4:
DNDarray global shape: (50, 4, 3)
DNDarray  local shape: (12, 4, 3)
On rank 1 of 4:
DNDarray global shape: (50, 4, 3)
DNDarray  local shape: (13, 4, 3)
```

training/examples/3_linalg.py

(See linalg module for more distributed operations)

```
import heat as ht
import time

n = 10000
m = 9000
device = "cpu"
ht.random.seed(1)
x = ht.random.randn(n, m, dtype=ht.float64, split=0, device=device)
y = ht.random.randn(m, n, dtype=ht.float64, split=0, device=device)
rank = x.comm.rank
size = x.comm.size

# DOT PRODUCT
start = time.time()
dot = ht.dot(x, y)
end = time.time()

#1. OPTIMIZE SPLIT AXES FOR OPERATION
#2. PERFORM OPERATION ON GPU
#3. TRY DIFFERENT NUMBERS OF TASKS (srun –ntasks-per-node parameter)
```

training/examples/4_sandbox.py

```python
import heat as ht

x = ht.arange(80, dtype=ht.float32)
#print(x)
rank = x.comm.rank
size = x.comm.size
# change data distribution in place
x.resplit_(axis=0)
# reshape distributed data
x = x.reshape(10, 4, -1)
# reduction operations
sum_along_0 = x.sum(axis=0)
global_sum = x.sum()
sum_along_1 = x.sum(axis=1)
# print local tensors
print(f"On rank {rank}, sum along axis 1 = {sum_along_1.larray}")
# statistics
avg_along_0 = x.average(axis=0)
print(f"On rank {rank}, average along axis 0 = {avg_along_0.larray}")
```

This talk

- Background
- Heat: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

Heat Machine Learning

Example: KMeans

```
# same as scikit-learn
data = ht.load(my_datafile, dataset=my_dataset, split=0)
kmeans = ht.cluster.KMeans(n_clusters, max_iter)
kmeans.fit(data)
```

- **Optimised for HPC (“SCALABLE”?)**
- **Clustering/classification: K-means, k-nn, LASSO regression, Gaussian Naive-Bayes…**
- **Data-parallel Neural Networks: DASO**
- **Ongoing efforts: SVD, PCA, distributed AD**
- **Check out our GitHub repo!**

Clustering tutorial

https://heat.readthedocs.io/en/latest/tutorial_clustering.html

https://github.com/helmholtz-analytics/heat

- Background
- Heat: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

- HPC implementation = attention to communication/memory details: “does it scale?”
- Significant speed-up vs. Dask/ CuPy
- Check out our paper:

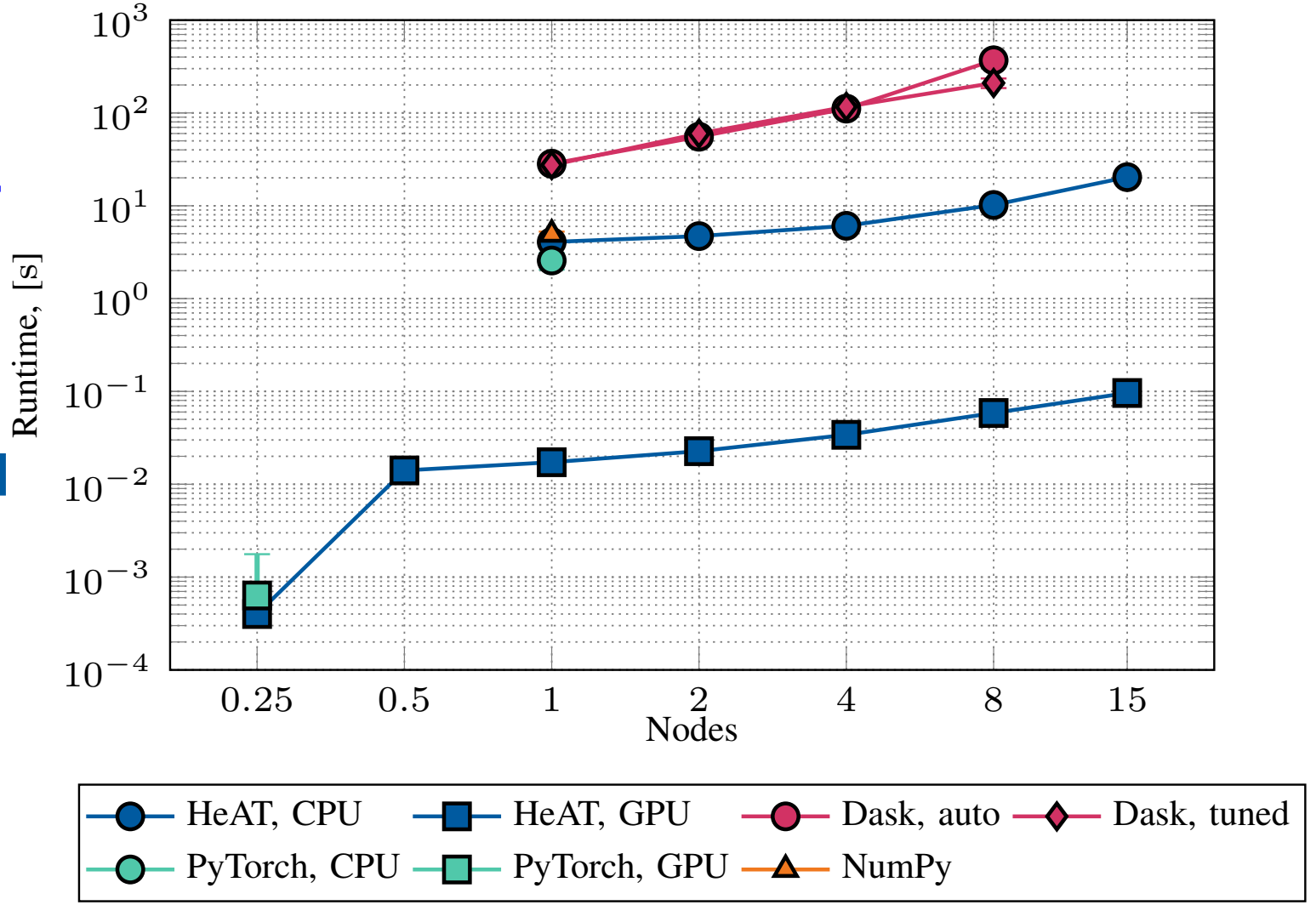
Götz et al. 2020, IEEE BigData

- Check out our benchmarks on https://github.com/helmholtz-analytics/heat

Benchmarks vs. Dask/CuPy: weak scaling

i.e. constant workload on each MPI process

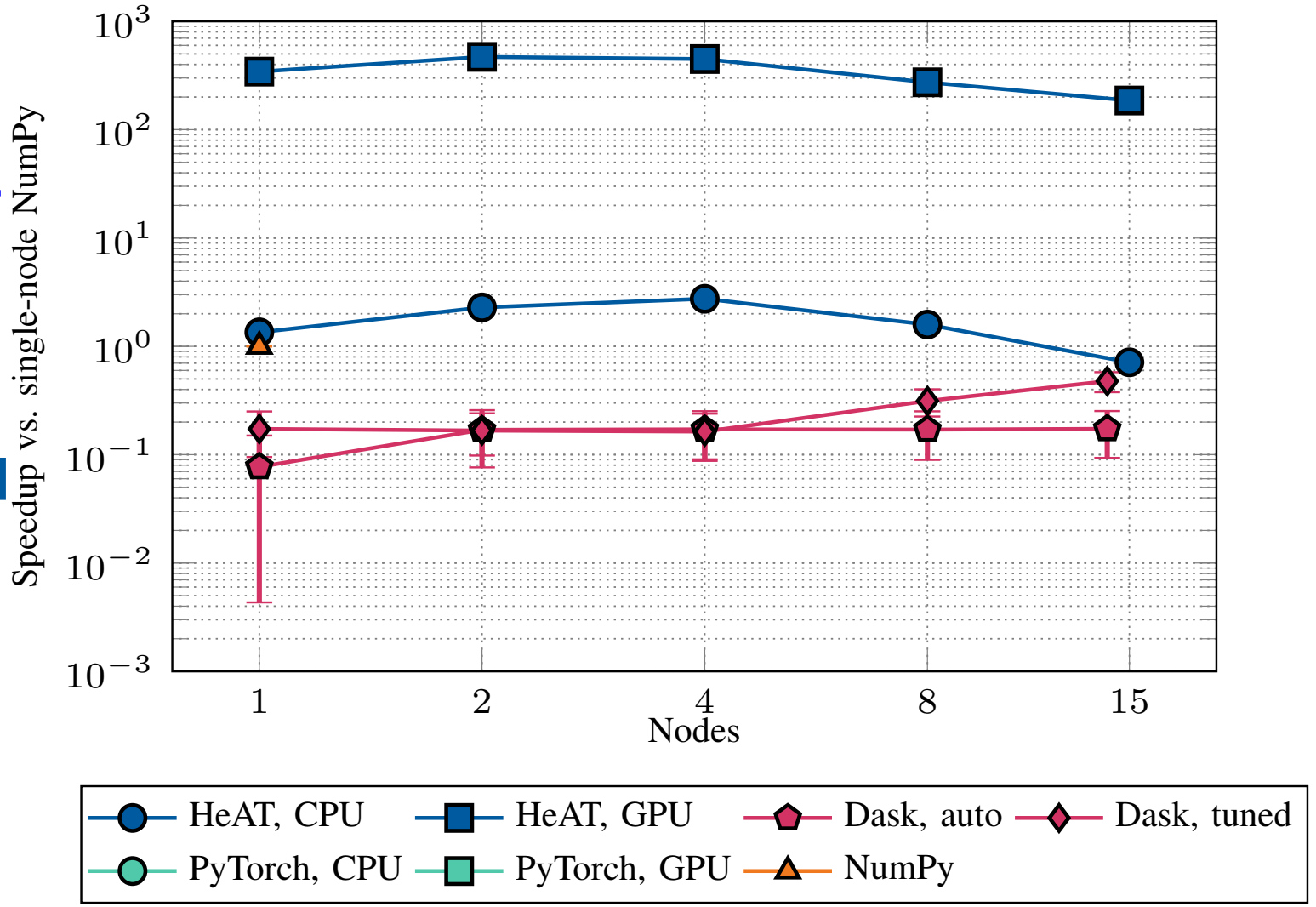
- Pairwise Euclidean distances on SUSY dataset
- 5,000,000 entries, 18 features
- On each node:
  - 2x12-core Intel Xeon Gold 6126
  - 4xTesla V100 GPUs



Benchmarks vs. Dask/CuPy: strong scaling

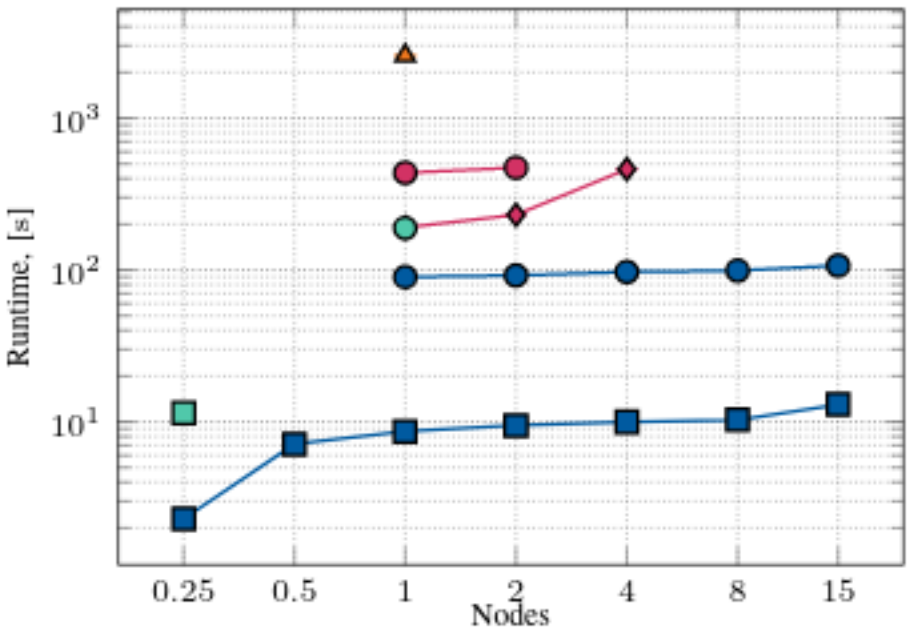
i.e. constant workload on system

- Pairwise Euclidean distances on SUSY dataset
- 5,000,000 entries, 18 features
- On each node:
  - 2x12-core Intel Xeon Gold 6126
  - 4xTesla V100 GPUs

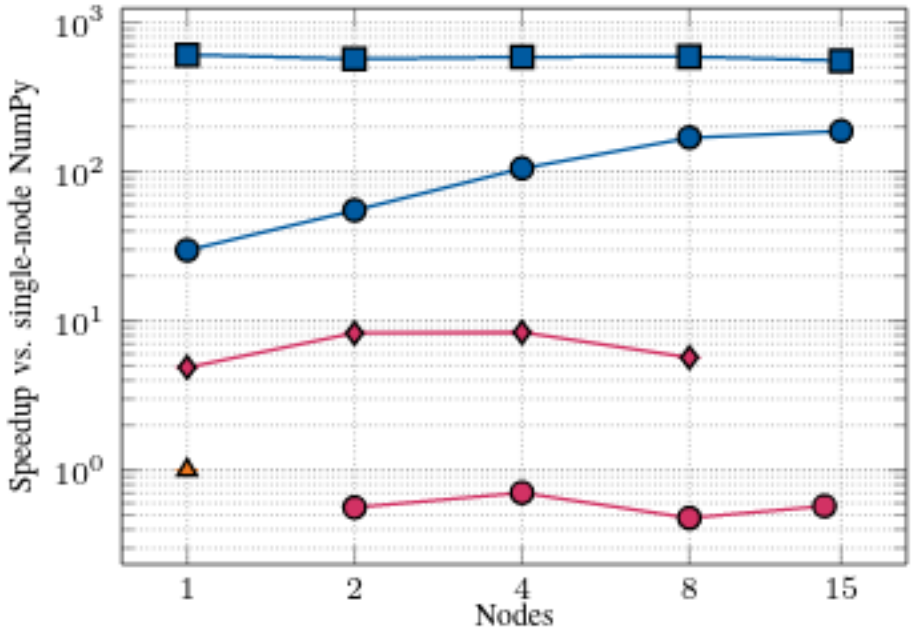


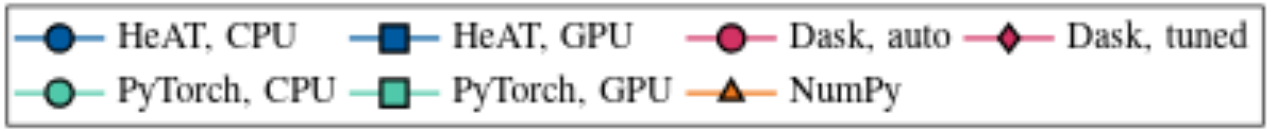
K-means clustering

Weak Scaling



Strong Scaling





DATA PARALLEL NEURAL NETWORKS

Coquelin et al. 2021

helmholtz-analytics / heat Public

Code | Issues 113 | Pull requests 21 | Discussions | Actions | Pr

master | 60 branches | 17 tags | Go to file | Add file

coquelin77 removed typing.Literal from linalg.basics.py for py3.7... ✓ acb26f7 13 days ago

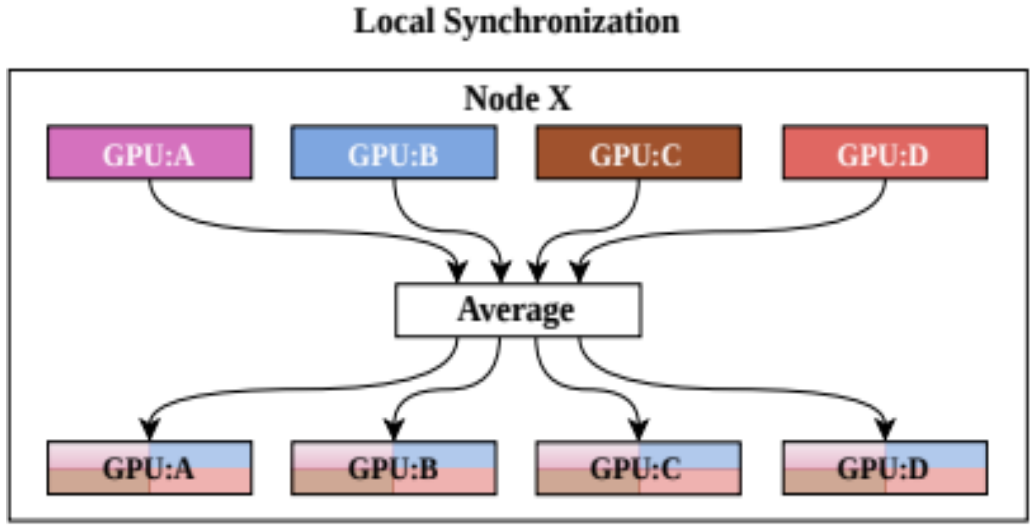
.github	newline fix
benchmarks	changes for distance benchmarks
doc	updated print statements in tutorial that i missed during my ...
examples	Reworked docstrings and examples for knn, modified code ...
heat	removed typing.Literal from linalg.basics.py for py3.7 compa...
scripts	fix tutorial

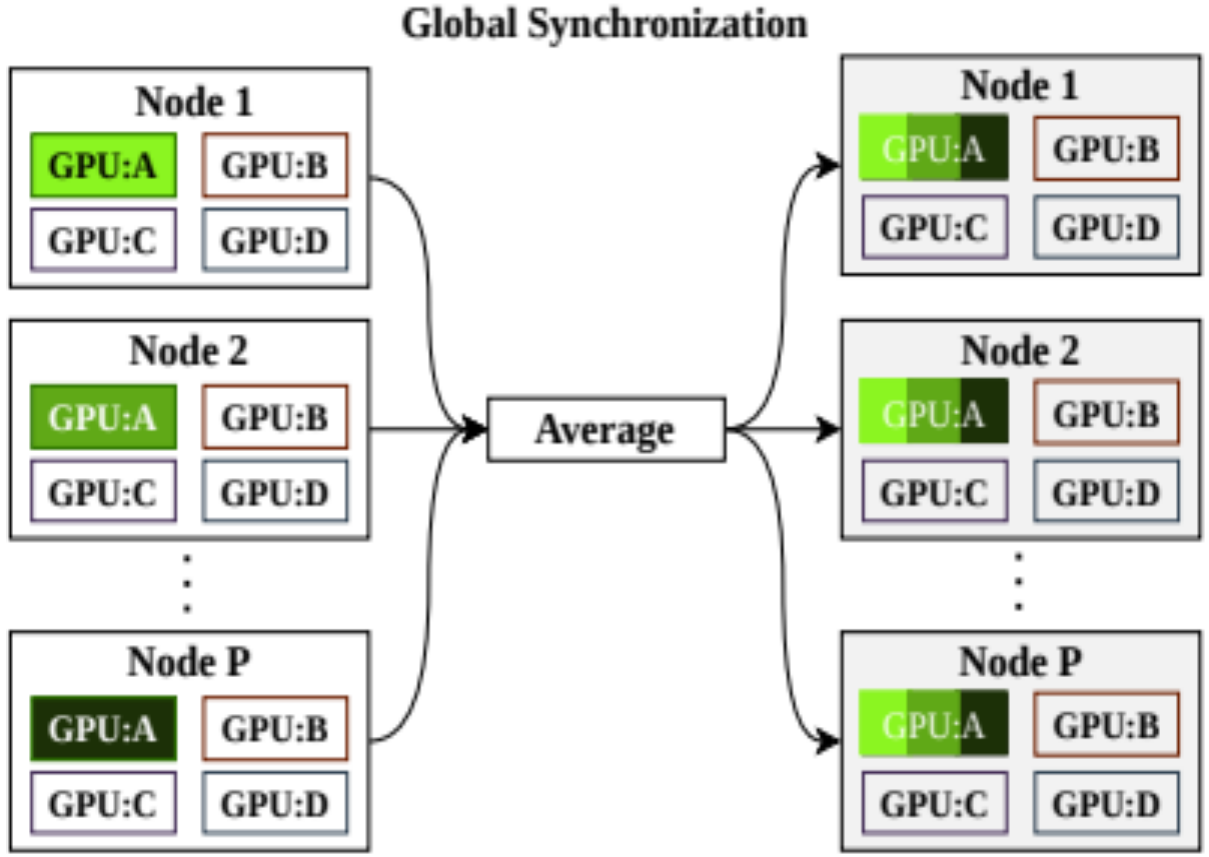
examples

Distributed Asynchronous and Selective Optimization

DASO

- DPNN training with NCCL and MPI Groups
  - Requires multiple nodes each with multiple GPUs
- Reduce communication overhead + increase speed with **selective** global updates while keeping accuracy
- Expensive global synch. avoided when possible





Code Sample

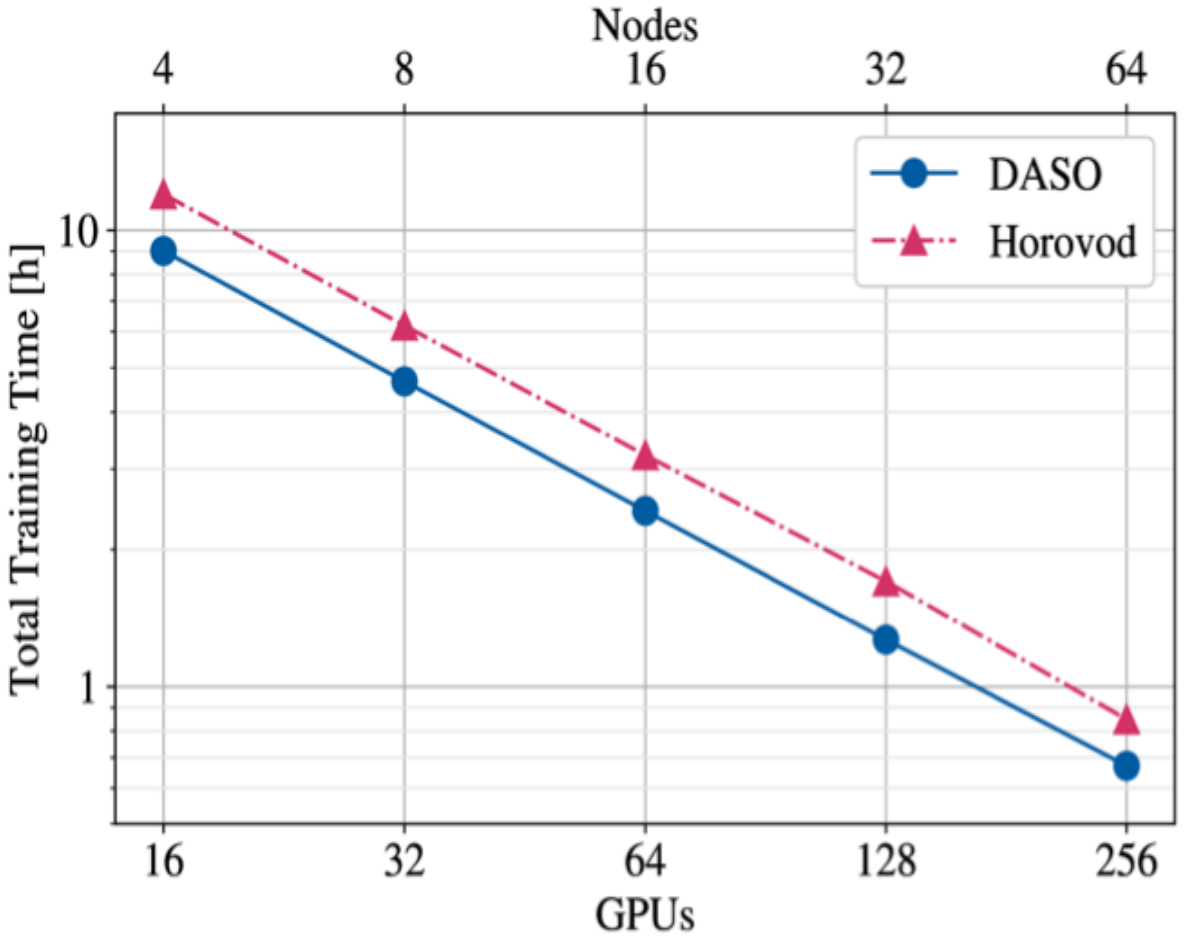
DASO - Distributed Asynchronous and Selective Optimization

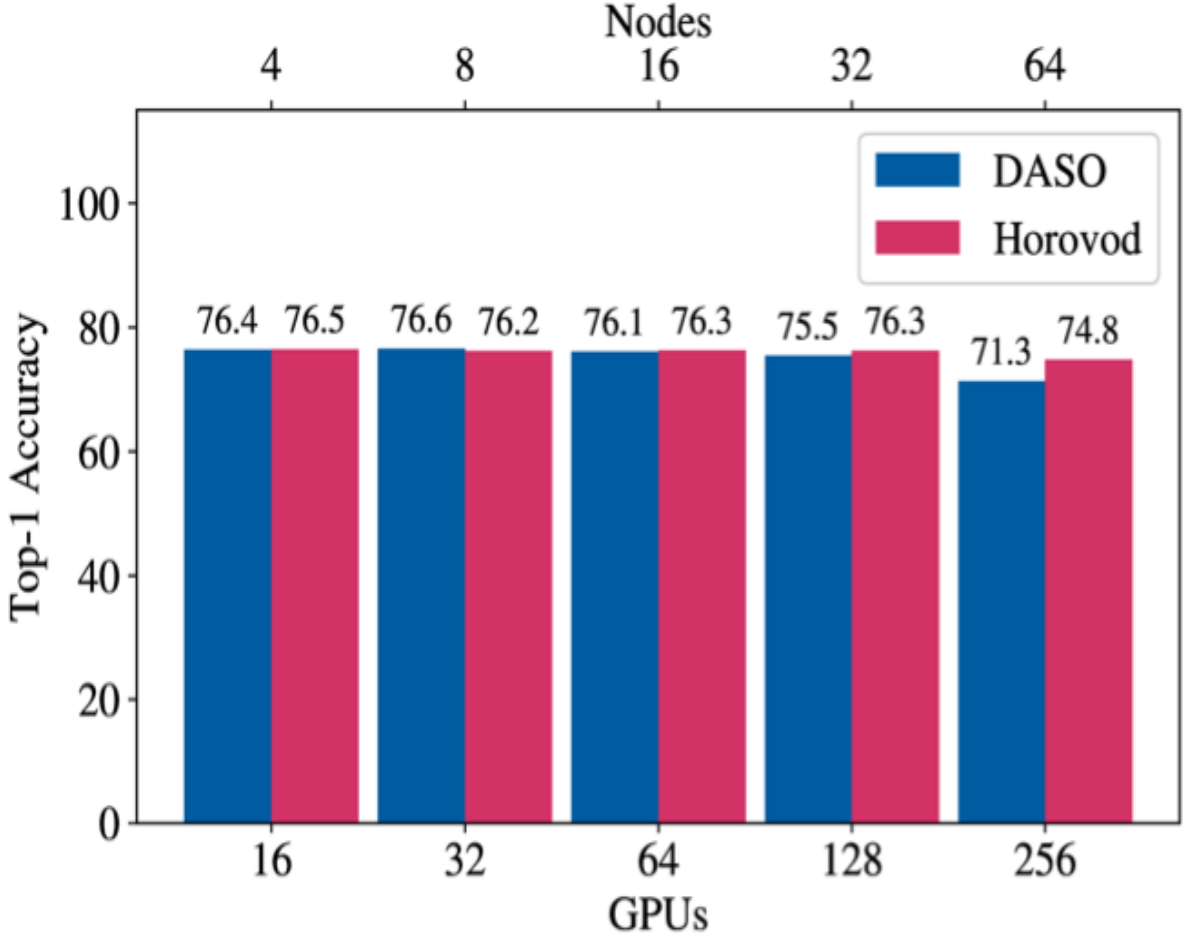
- Required steps to use DASO:
    1) Create PyTorch distributed Group (node local)
    2) Define dataloaders to distribute data between all processes (not shown)
    3) Define the DASO optimizer with a PyTorch optimizer instance
    4) Create the hierarchical network in HeAT
- Step functions are handled internally, `daso_optimizer.step()` replaces PyTorch optimizer.step()

```
 1 import heat as ht
 2 import torch
 3 ...
 4 # create PyTorch distributed group
 5 world_size = ht.MPI_WORLD.size
 6 rank = ht.MPI_WORLD.rank
 7 local_rank = rank % num_local_gpus
 8 torch.distributed.init_process_group(
 9     backend="nccl",
10     rank=local_rank,
11     world_size=world_size
12 )
13 ...
14 # the DASO optimizer is created
15 daso_optimizer = ht.optim.DASO(
16     local_optimizer=optimizer,
17     total_epochs=num_epochs
18 )
19 ...
20 # the hierarchical network is created
21 ht_model = ht.nn.DataParallelMultiGPU(
22     net,
23     daso_optimizer
24 )
```

DASO vs Horovod – ImageNet Training with ResNet-50

DASO - Distributed Asynchronous and Selective Optimization





DASO vs Horovod – Takeaways

DASO - Distributed Asynchronous and Selective Optimization

- DASO takes ~34% less time than Horovod while maintaining accuracy!
- Implementation changes are extremely similar
  - Optimizer wrapper
  - Dataloader updates
  - Network wrapper
- DASO is slurm+AMP compatible
  - hvd is NOT!
- Horovod has issues running with slurm
  - Specifically with SyncBatchNorm and Automatic Mixed Precision

This talk

- Background
- HeAT: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

- Brute force: `import heat as np`
- Functions not available: request features!
- (PCA is already high on our list)
- Double-check your dtypes
- …and your split/device attributes!
- Get in touch/get help!

Applications so far

- Earth System Modeling: TerrSysMP with heat-based post-processing library for ParFlow
- Aerospace: “High-performance data analytics of hybrid rocket fuel combustion”, C. Debus et al.
- Neuroscience: ASSET (“Analysis of Sequences of Synchronous Events in Massively Parallel Spike Trains”) within the elephant library

Heat4eflows?

- Distributed N-dimensional array operations, machine learning, (data parallel) neural networks
- Multi-CPU, multi-GPU support
- Mid-term: split low-level Heat from ML/NN (Intel collaboration)
- Next round of funding proposals
- Pillar I? (e.g. k-means)
- Pillar II?
- Pillar III?

This talk

- Background
- HeAT: programming model
- Low-level operations
- High-level algorithms
- On performance
- “heatifying” your code
- Outlook

```
pip install heat
```

- Heat v1.1.0 just released
- v1.2.0 in the works with major performance enhancement
- Ongoing: SVD, PCA, parallel AD (`mpi4torch`)
- Head to GitHub  for issues and feature requests
- Want to contribute? Get in touch!
- Building up Horizon Europe collaboration (“analysis, precision, decision support”)

Thank you so much!

HEAT

Helmholtz Analytics Toolkit

Markus Götz,
Daniel Coquelin (IBG-3),
Charlotte Debus (DLR)
Karlsruhe Institute
of Technology

Claudia Comito,
Björn Hagemeier,
Kai Krajsek,
Michael Tarnawa,
Lena Blind
Jülich Supercomputing Center

Philipp Knechtges,
Martin Siggel,
German Aerospace Center,
Cologne