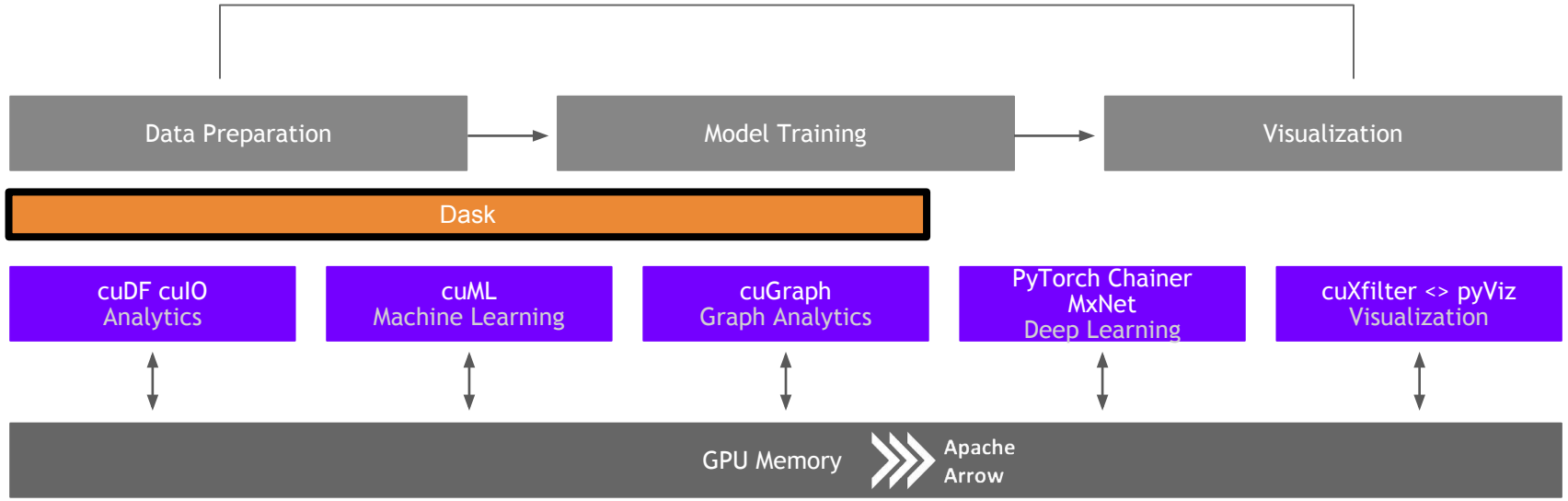
RAPIDS

Distributed GPU Computing with Dask

Nick Becker
RAPIDS Engineering

RAPIDS

Scaling RAPIDS with Dask



Dask

What is Dask?

- Distributed compute scheduler built to scale Python
- Scales workloads from laptops to supercomputer clusters
- Extremely modular: disjoint scheduling, compute, data transfer and out-of-core handling
- Multiple workers per node allow easier one-worker-per-GPU model

Why Dask?

PyData Native

- **Easy Migration:** Built on top of NumPy, Pandas Scikit-Learn, etc.
- **Easy Training:** With the same APIs
- **Trusted:** With the same developer community

Deployable

- **HPC:** SLURM, PBS, LSF, SGE
- **Cloud:** Kubernetes
- **Hadoop/Spark:** Yarn

Easy Scalability

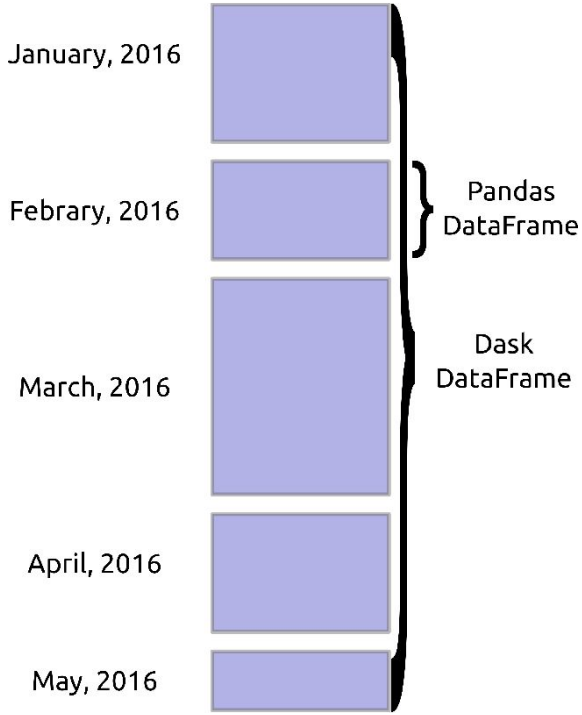
- Easy to install and use on a laptop
- Scales out to thousand-node clusters

Popular

- Most common parallelism framework today in the PyData and SciPy community

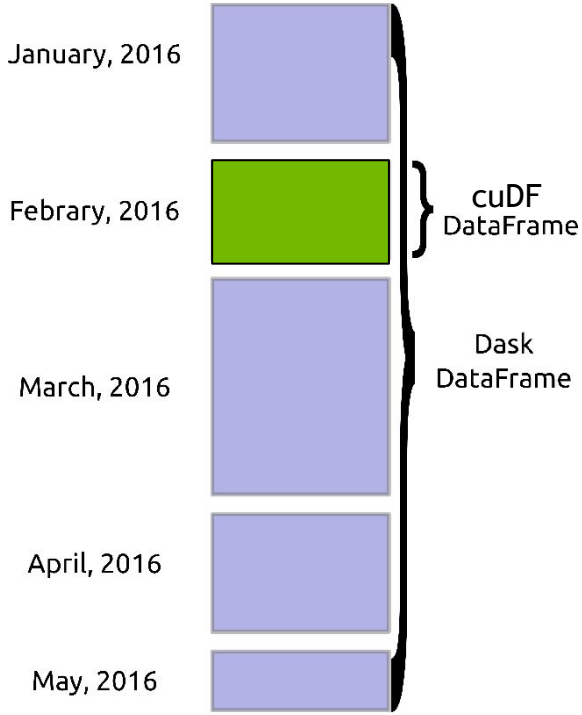
Combine Dask with cuDF

Many CPU DataFrames form a distributed CPU DataFrame



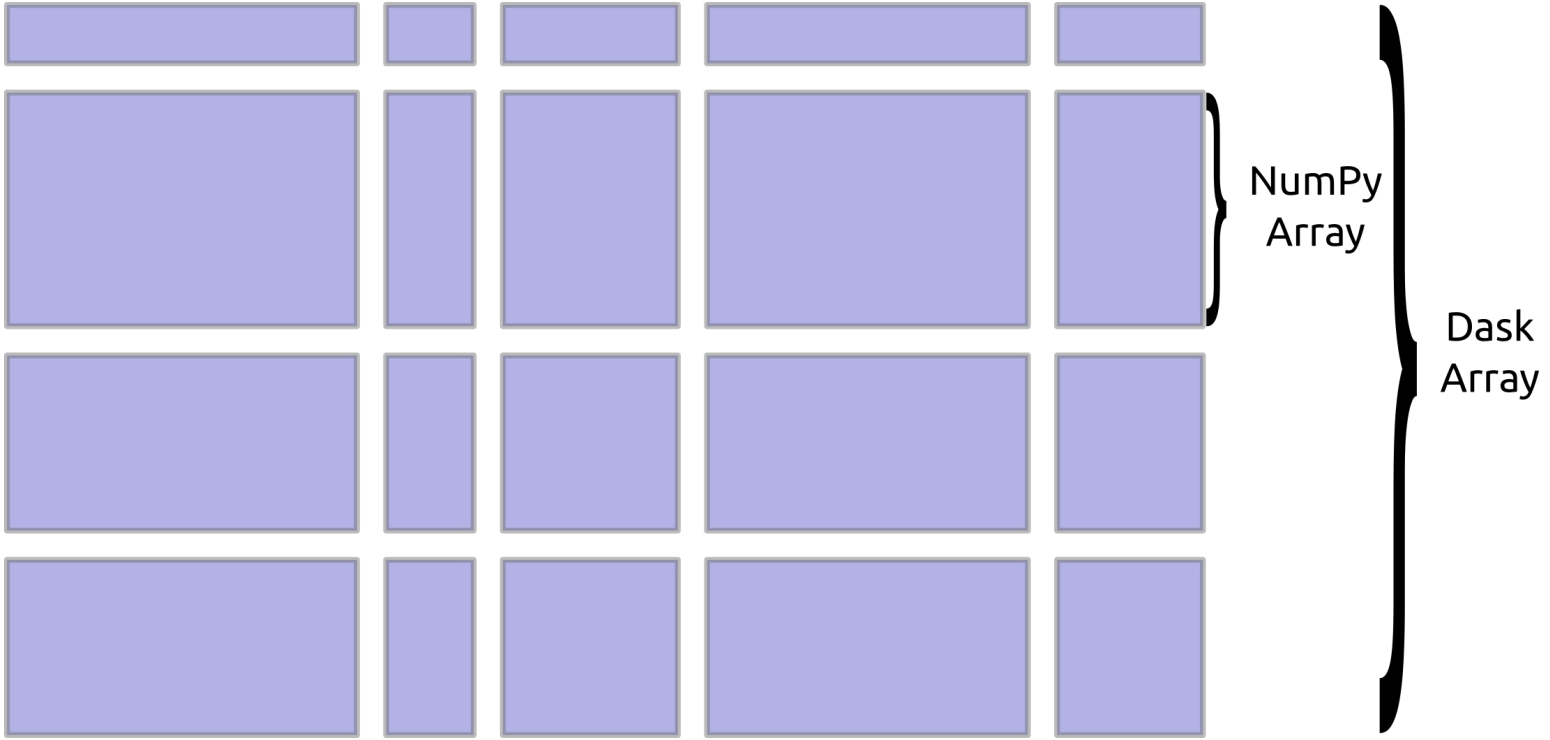
Combine Dask with cuDF

Many GPU DataFrames form a distributed GPU DataFrame



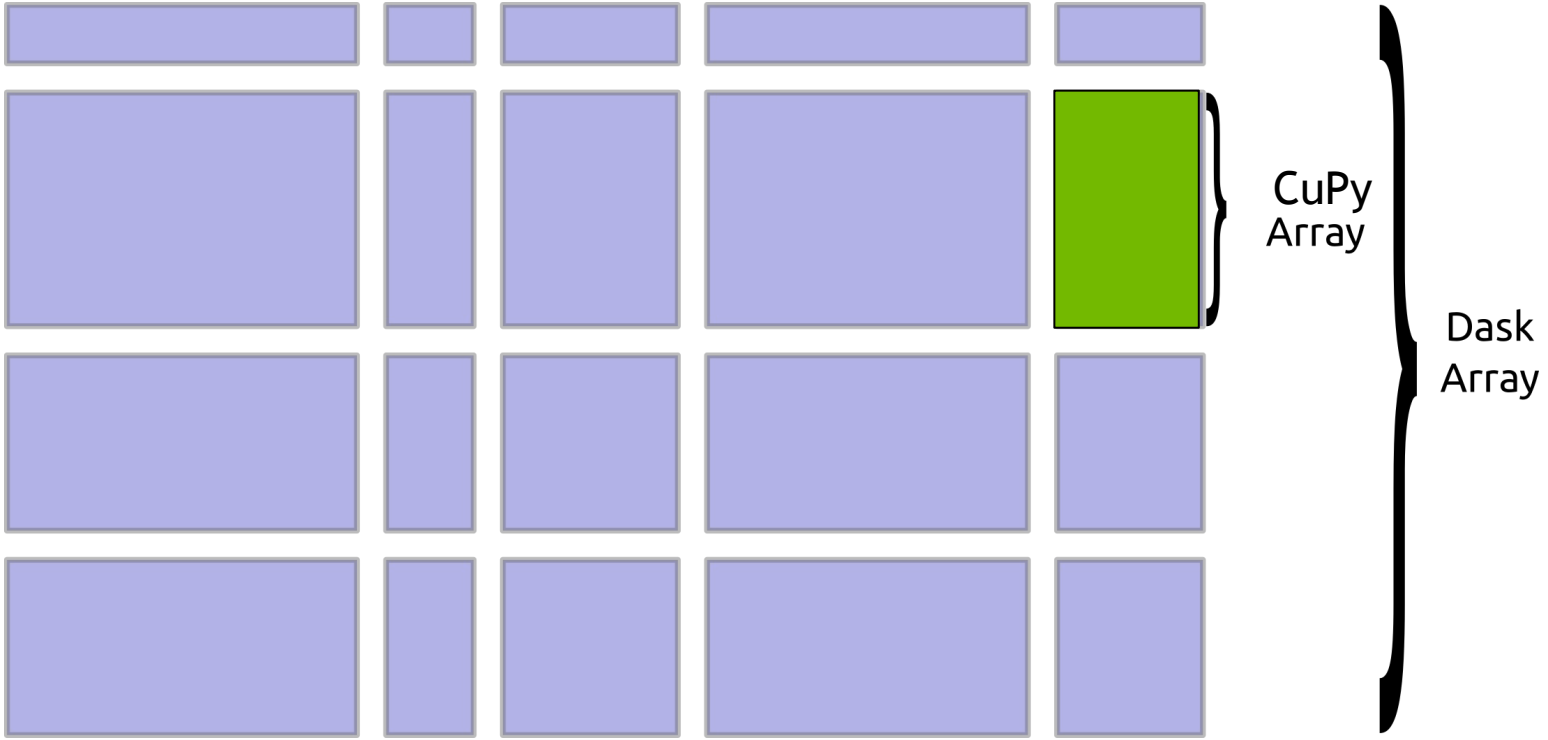
Combine Dask with CuPy

Many CPU arrays form a Distributed CPU array



Combine Dask with CuPy

Many GPU arrays form a Distributed GPU array



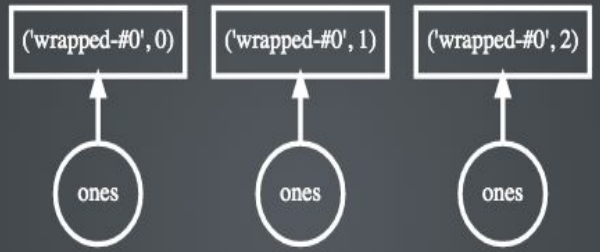
DASK

Dask APIs Produce Task Graphs

Dask Schedulers Execute Task Graphs

DASK

1D-Array

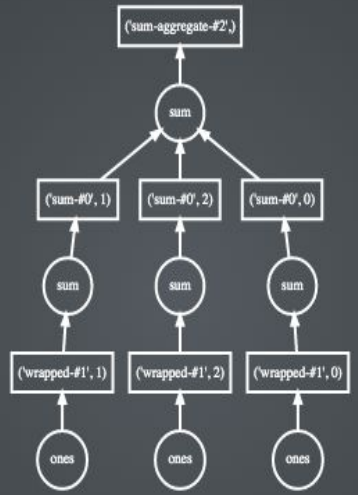



```
>>> np.ones((15,))
array([ 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

>>> x = da.ones((15,), chunks=(5,))
```

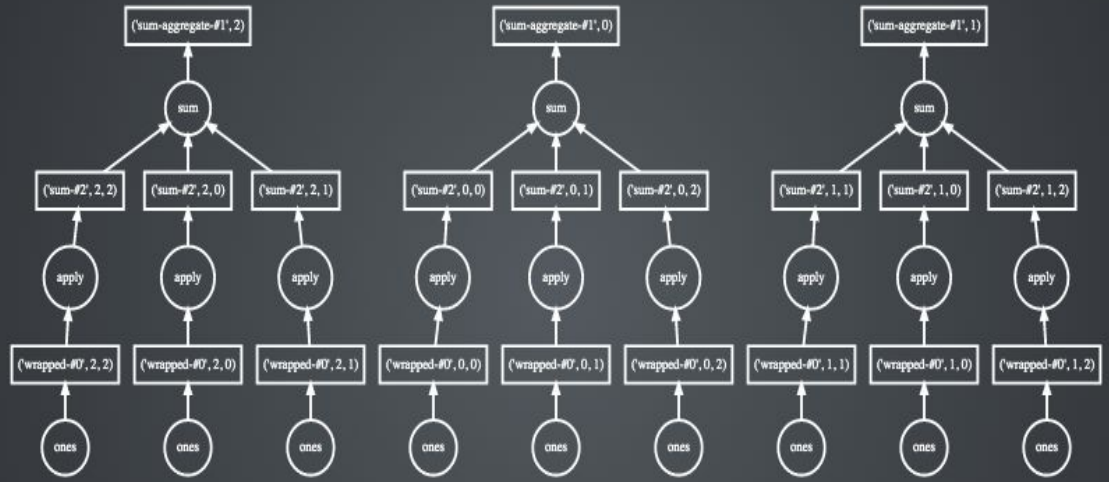
DASK

1D-Array




```
x = da.ones((15,), chunks=(5,))
x.sum()
```

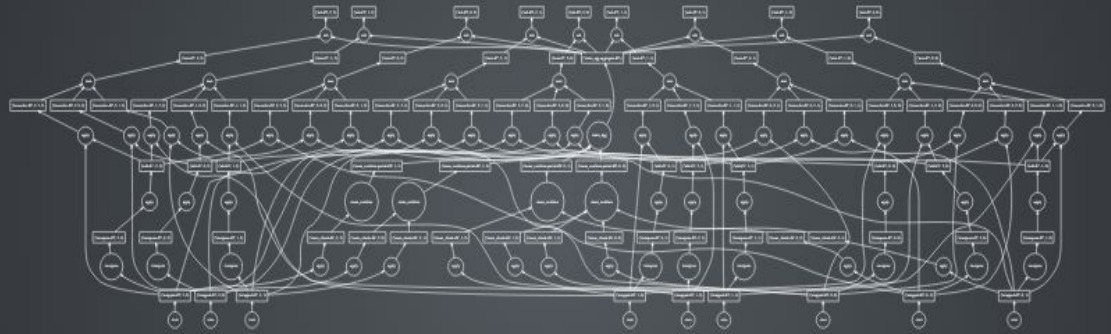
ND-Array - Sum




```
x = da.ones((15, 15), chunks=(5, 5))
x.sum(axis=0)
```

DASK

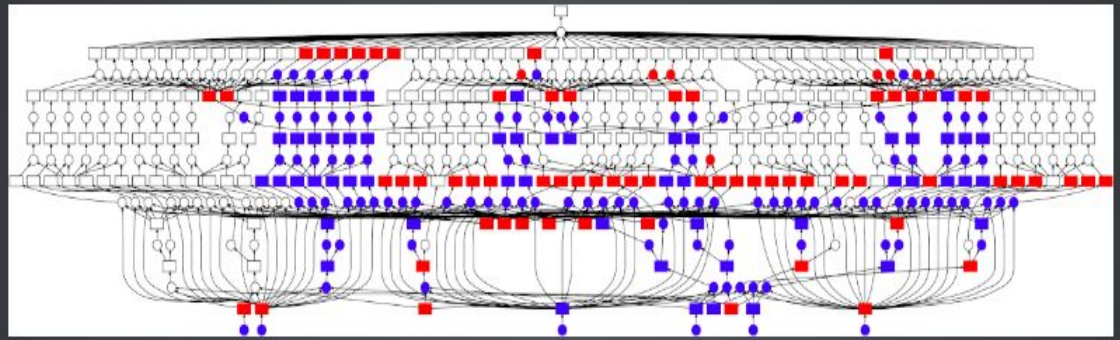
ND-Array - Compound Operations



```
x = da.ones((15, 15), chunks=(5, 5))
x.dot(x.T + 1) - x.mean()
```

DASK

Dask.array/dataframe/delayed author task graphs



Now we need to run them efficiently

Dask Deployments

General Pattern

```
cluster = Cluster(ncores=..., memory=...,
                  attr1=...,
                  attr2=...,
                  ...
                  )
cluster.scale(n_workers)
```

K8s Native API

Quickstart

```
from dask_kubernetes import KubeCluster

cluster = KubeCluster.from_yaml('worker-spec.yml')
cluster.scale_up(10)  # specify number of nodes explicitly

cluster.adapt(minimum=1, maximum=100)  # or dynamically scale based on current workload
```

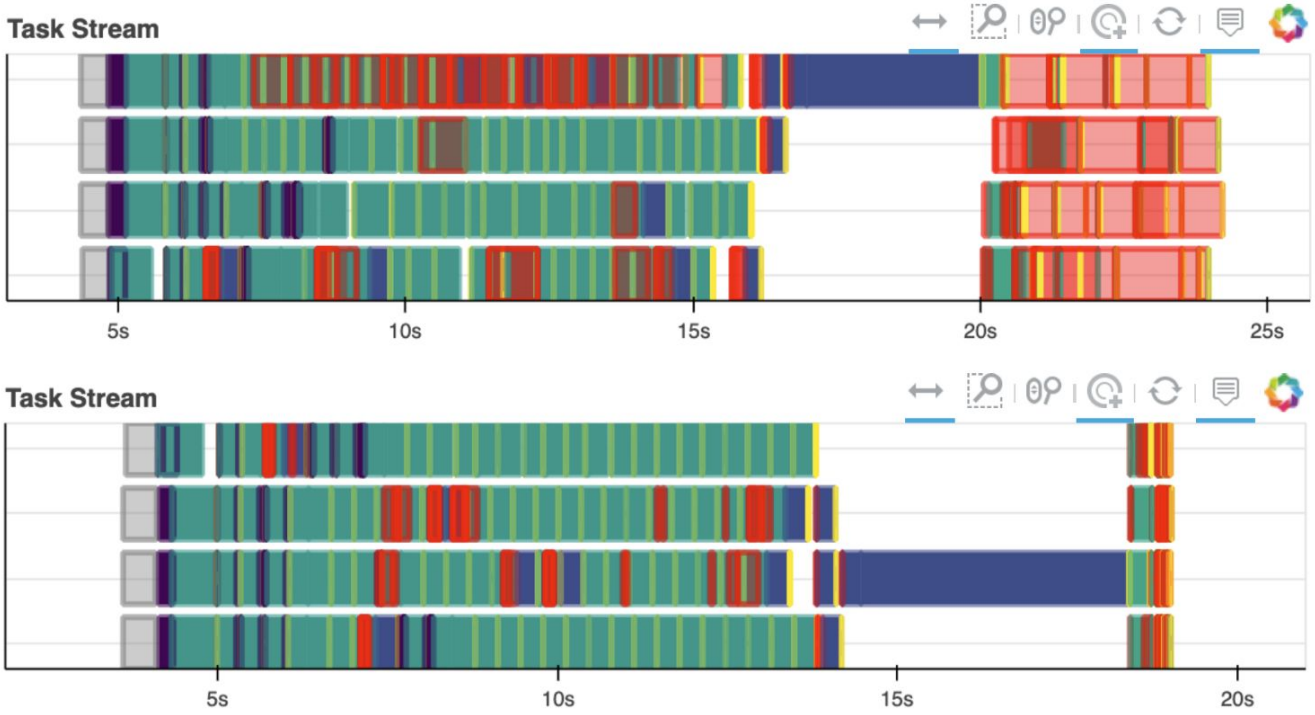
OpenUCX

Bringing hardware accelerated communications to Dask

- TCP sockets are slow!
- UCX provides uniform access to transports (TCP, InfiniBand, shared memory, NVLink)
- Python bindings for UCX (ucx-py): https://github.com/rapidsai/ucx-py
- Will provide best communication performance to Dask based on available hardware on nodes/cluster

OpenUCX

Dask Array SVD + CuPy Experiment with and without UCX



Scale up with RAPIDS

Scale Up / Accelerate

RAPIDS and Others

Accelerated on single GPU

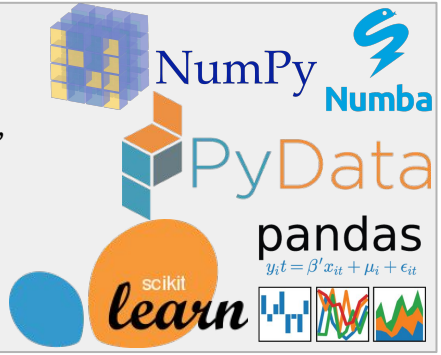
NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS

PyData

NumPy, Pandas, Scikit-Learn, Numba and many more

Single CPU core
In-memory data



Scale out with RAPIDS + Dask with OpenUCX

Scale Up / Accelerate

Scale out / Parallelize

RAPIDS and Others

Accelerated on single GPU

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS

RAPIDS + Dask with OpenUCX

Multi-GPU
On single Node (DGX)
Or across a cluster

RAPIDS

DASK

PyData

NumPy, Pandas, Scikit-Learn, Numba and many more

Single CPU core
In-memory data

NumPy

Numba

PyData

pandas

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

scikit learn

Dask

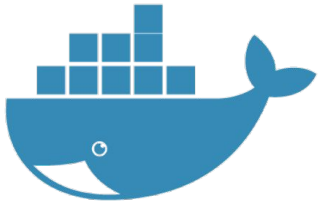
Multi-core and Distributed PyData

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

DASK

RAPIDS

How do I get the software?



- https://github.com/rapidsai
- https://anaconda.org/rapidsai/
- https://dask.org/

- https://ngc.nvidia.com/registry/nvidia-rapidsai-rapidsai
- https://hub.docker.com/r/rapidsai/rapidsai/

THANK YOU

Nick Becker
nicholasb@nvidia.com

RAPIDS