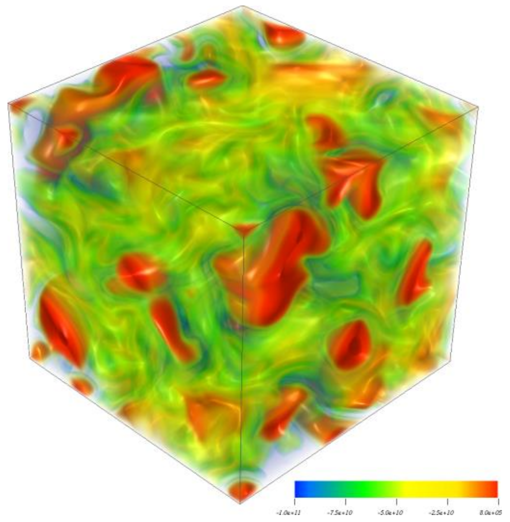
LEGATE: HIGH PRODUCTIVITY HIGH PERFORMANCE COMPUTING

Manolis Papadakis

NERSC Data Seminar, 2021-08-03

MOTIVATION

WRITING SINGLE-CORE PYTHON APPLICATIONS



Direct Numerical Simulation of
3D Combustion with Multi-Physics Chemistry

NumPy

Cells: 4D Arrays: [X,Y,Z,Chemical Species]
Fluxes: 5D Arrays [X,Y,Z,Chemical Species,Face]

```
cells[:] += alpha * np.sum(fluxes, axis=4)
```

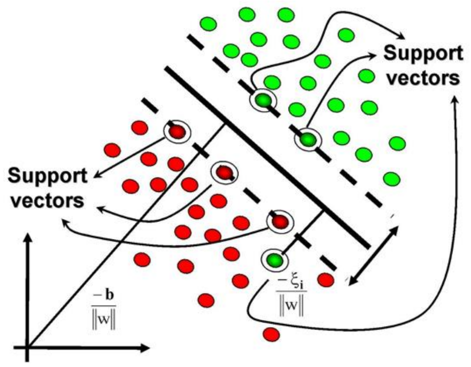
Uncertainty
Quantification

Pandas

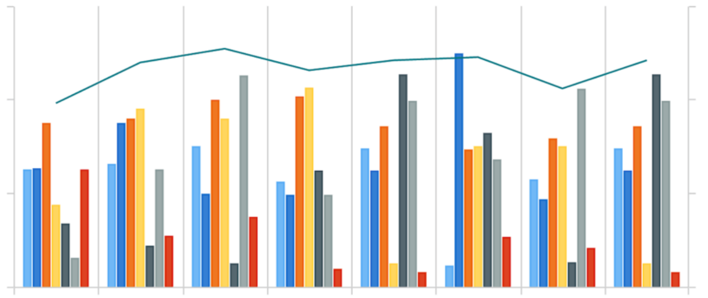
<u>Statistical Analyses</u>
Group-by over chemical
species with different
compositions

scikit learn

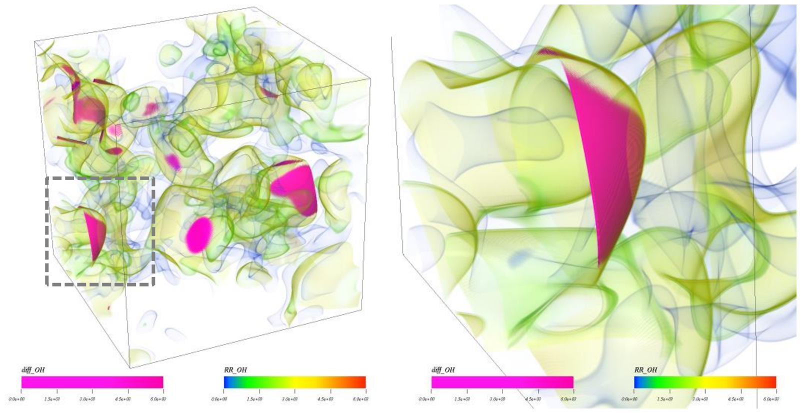
<u>SVM Classifier</u>
Recognize flame wavefronts



matplotlib

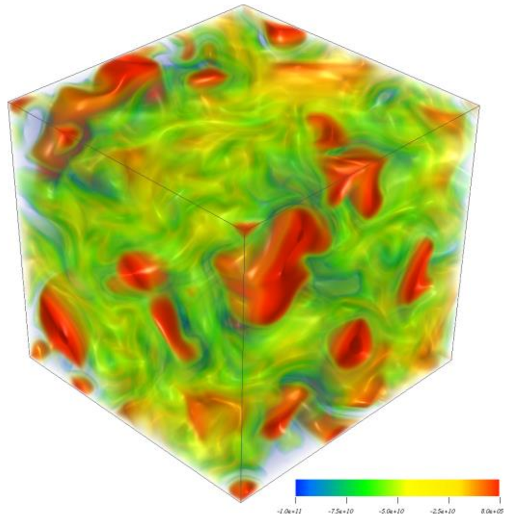


vtk



NVIDIA

WRITING HPC PYTHON APPLICATIONS TODAY



Enter mpi4py…

Run on N nodes

Uncertainty Quantification

NumPy

CuPy

Manage explicit ghost cells

```
cells[:] += alpha * np.sum(fluxes, axis=4)
```

1 Process/Node
Use NVLink
Cannot use drop-in CuPy

1 Process/GPU
Mpi4py has no GPUDirect
Drop-in CuPy for NumPy

Write/Read to/from Disk

I/O Bound

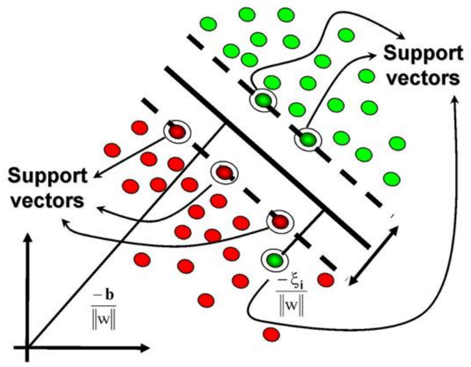
Run on M nodes

Pandas

Statistical Analyses
Group-by over chemical species with different compositions

scikit learn

SVM Classifier
Recognize flame wavefronts

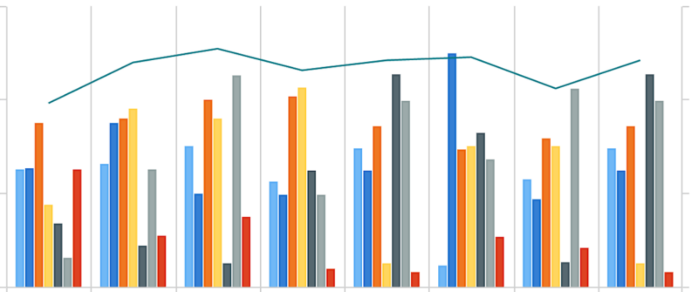


Write/Read to/from Disk

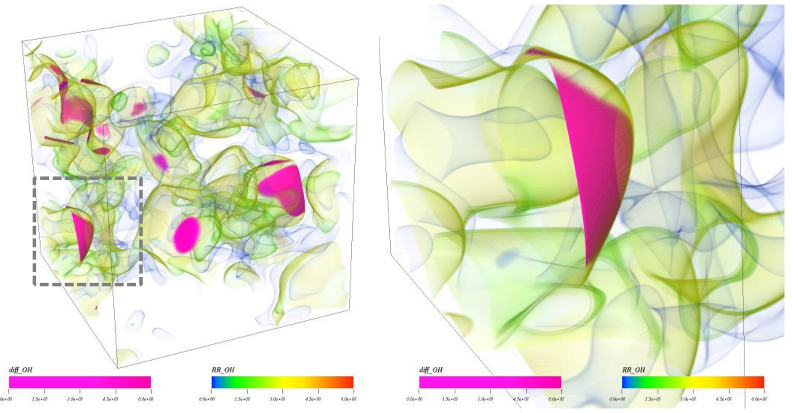
I/O Bound

Run on 1 node

matplotlib



vtk



GOAL OF THE LEGATE PROJECT

Enable users to program these:

The same way they program this:

Guarantee:

1. Sequential Semantics
2. Software Interoperability
3. Portability

Deliver:

1. Implicit Parallelism
2. Composability
3. Scalability

OVERVIEW OF LEGATE

DISTRIBUTED IMPLEMENTATIONS OF FAMILIAR SEQUENTIAL APIS

Require only an import statement change
- no API changes
- no user management of data distribution

Same code runs on everything

Maintain sequential semantics of original APIs
Discover inherent parallelism automatically

```
import legate.numpy as np

def cg_solve(A, b, tol=1e-10):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / p.dot(Ap)
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < tol:
          break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
     return x
```

LEGATE PROJECT STATUS

Open-source as of Apr 2021 (https://github.com/nv-legate)

Legate NumPy	Legate Pandas
Basic and advanced indexing	Join, groupby reduction, sorting
Broadcasting/reshaping/transpose	Arithmetic, cumulative, reduction operators
Arithmetic operators	CSV and Parquet file formats
Dot (vector-vector, matrix-vector, and matrix-matrix)	Queries with user defined predicates (via Numba)
Random array creation	String and dictionary types

LEGATE STACK

Familiar Domain-Specific Interfaces

- Legate NumPy
- Legate Pandas
- Legate SciPy

Legate Core

Runtime System for Scalable Execution

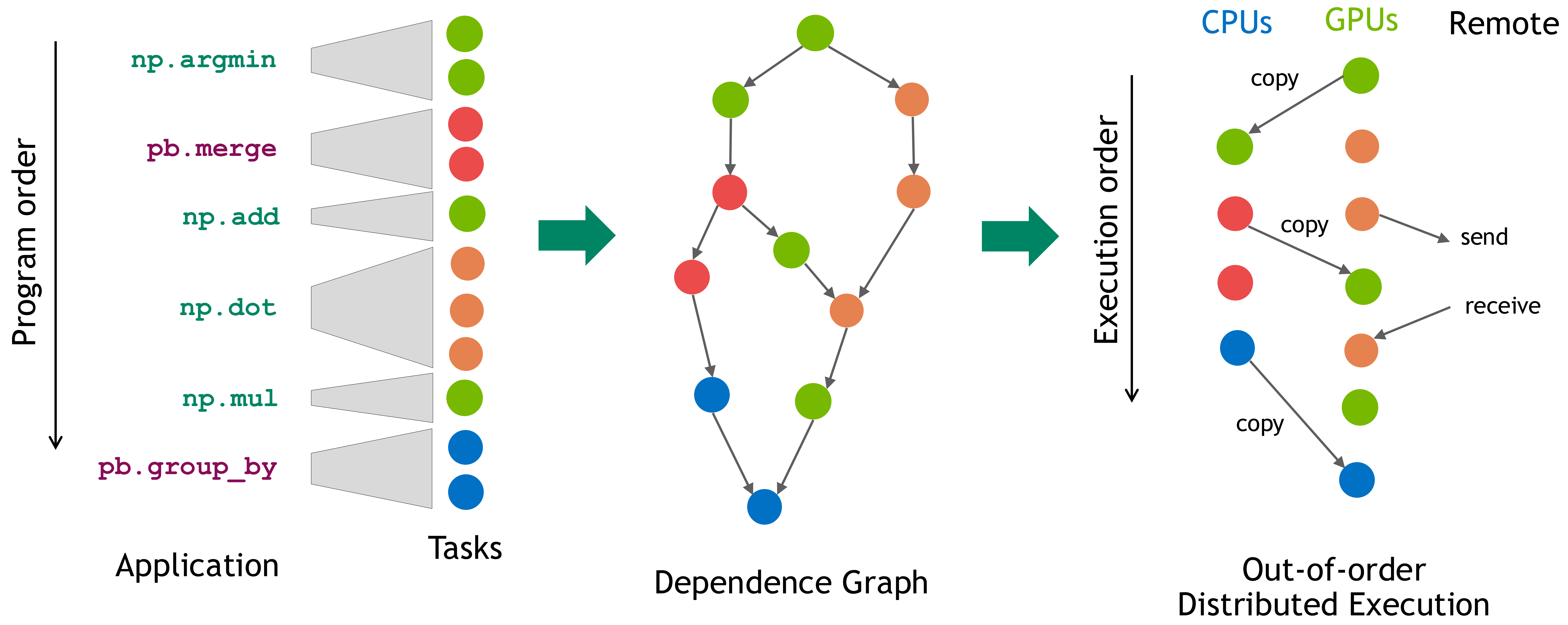
Legion

GPU-Accelerated Domain Libraries

cuBLAS, cuDF, NCCL, cuTENSOR, cuML, …

PARALLEL EXECUTION ACROSS LIBRARIES

Legion runtime inserts required ordering, sync, copies



COMMON DISTRIBUTED DATA MODEL

Facilitates data exchange between Legate libraries

Libraries can reuse the current partitioning, or request a different partition to fit their needs

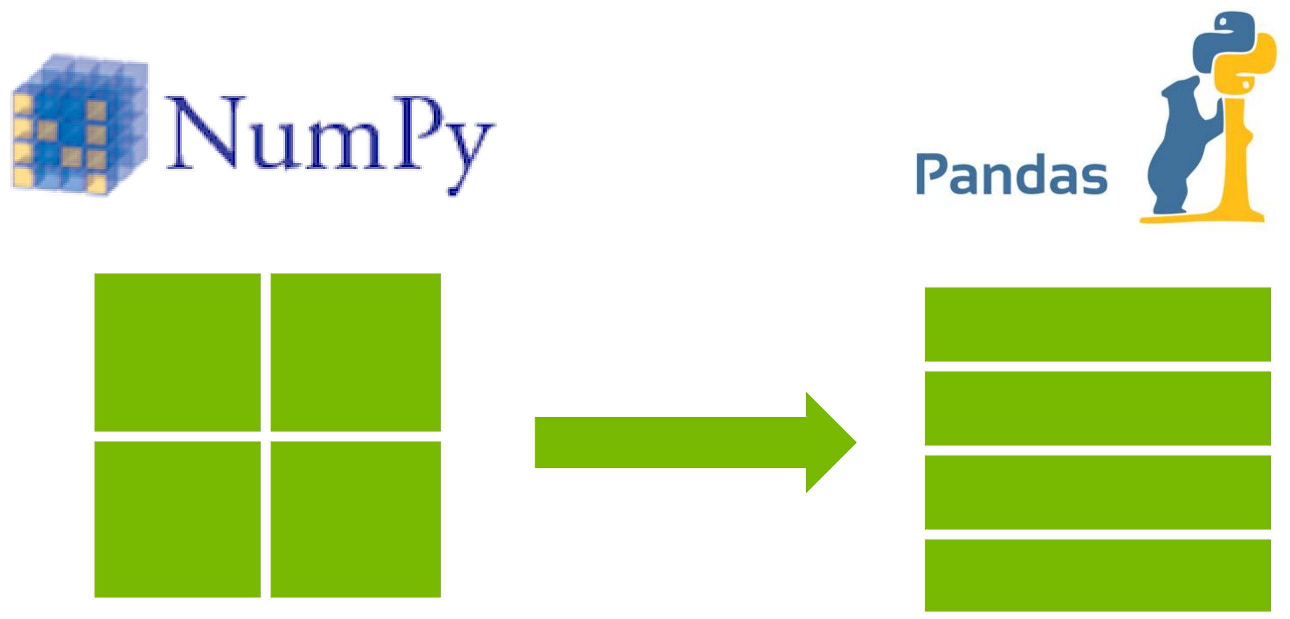
Multiple partitions allowed to coexist, kept in sync automatically

Legion copies/reformats data only when needed

Legion combines partition info to move minimal amount of data on partition switch.

Aliases to the same distributed collection

```
import legate.numpy as np
import legate.pandas as pd

a1 = np.random.randn(N)
sr = pd.Series(a1)
a2 = np.asarray(sr)
```



Spatial Decomposition

Partition by Chemical Species

DEMO (CORI-GPU)

JACOBI STENCIL ON 1 GPU

1/8 of a cori-gpu node

```
nx, ny = 100, 100
...
while l2 > l2_target and icount < max_iter:
  icount += 1
  pn = p.copy()
  p[1:-1,1:-1] = .25 * (
    p[1:-1, 2:] + p[1:-1, :-2] +
    p[2:, 1:-1] + p[:-2, 1:-1]
  )
  l2 = np.sqrt(np.sum((p-pn)**2)/np.sum(pn**2))
  if icount % output_freq == 0:
    print(icount, l2)
```

```
$ quickstart/run.sh 1/8 a.py
```

```
100 0.003264076742254255
200 0.0015602183973943686
300 0.0009900220308147304
400 0.0007072288110380173
500 0.0005398367142183384
600 0.0004301171625455644
700 0.0003532454152280064
800 0.0002967904950595478
900 0.00025384486510825514
1000 0.00022026753267800596
l2 = 0.00022026753267800596, icount = 1000, time =
1.5601110458374023 s
```

Full example: https://gist.github.com/manopapad/0042aec3815fb48b55cd7693060618b4

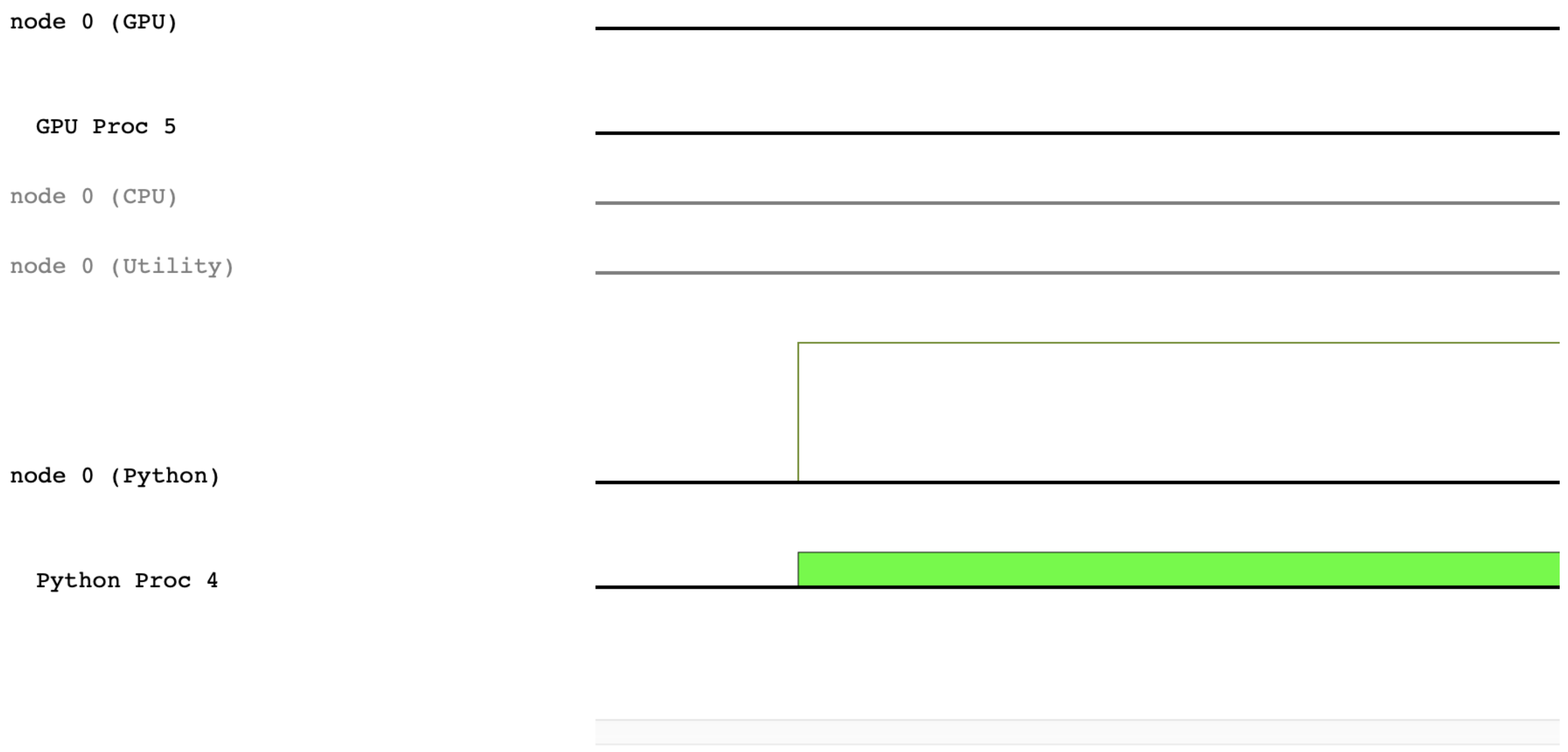
based on https://barbagroup.github.io/essential_skills_RRC/laplace/1/

ADD LEGION-LEVEL PROFILING

Legate's heuristics decided problem size too small for GPU execution

```
$ quickstart/run.sh 1/8 a.py --profile
```

```
nx, ny = 100, 100
...
while l2 > l2_target and icount < max_iter:
  icount += 1
  pn = p.copy()
  p[1:-1,1:-1] = .25 * (
    p[1:-1, 2:] + p[1:-1, :-2] +
    p[2:, 1:-1] + p[:-2, 1:-1]
  )
  l2 = np.sqrt(np.sum((p-pn)**2)/np.sum(pn**2))
  if icount % output_freq == 0:
    print(icount, l2)
```



Full example: https://gist.github.com/manopapad/0042aec3815fb48b55cd7693060618b4

based on https://barbagroup.github.io/essential_skills_RRC/laplace/1/

FORCE GPU EXECUTION

Slower than executing in Python

```
nx, ny = 100, 100
...
while l2 > l2_target and icount < max_iter:
  icount += 1
  pn = p.copy()
  p[1:-1,1:-1] = .25 * (
    p[1:-1, 2:] + p[1:-1, :-2] +
    p[2:, 1:-1] + p[:-2, 1:-1]
  )
  l2 = np.sqrt(np.sum((p-pn)**2)/np.sum(pn**2))
  if icount % output_freq == 0:
    print(icount, l2)
```

```
$ quickstart/run.sh 1/8 a.py --profile -lg:numpy:test
```

```
100 0.0032640767422542506
200 0.0015602183973943682
300 0.0009900220308147309
400 0.00070728811038017
500 0.0005398367142183388
600 0.0004301171625455644
700 0.0003532454152280063
800 0.00029679049505954713
900 0.00025384486510825525
1000 0.00022026753267800596
l2 = 0.00022026753267800596, icount = 1000, time =
4.429934740066528 s
```

Full example: https://gist.github.com/manopapad/0042aec3815fb48b55cd7693060618b4

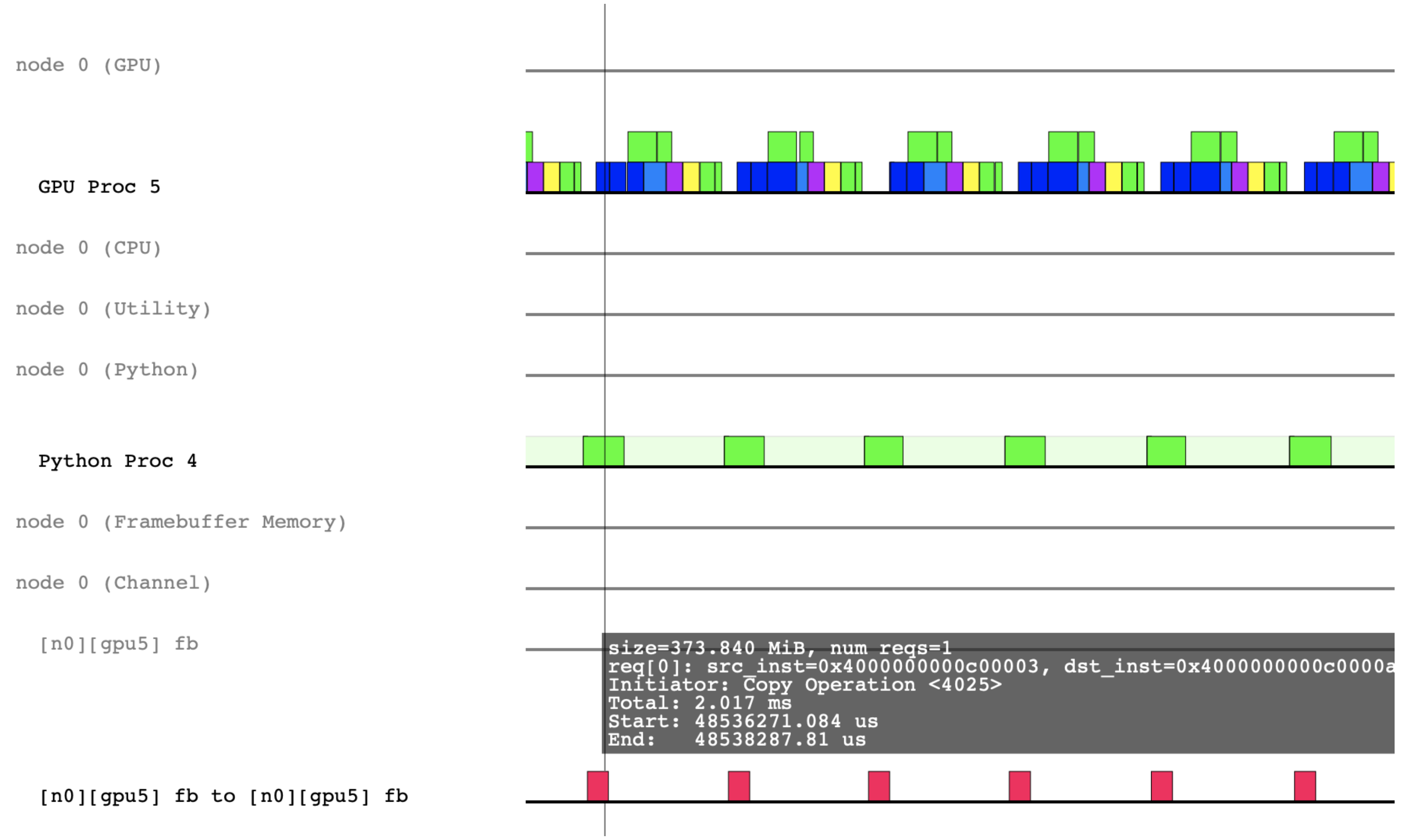
based on https://barbagroup.github.io/essential_skills_RRC/laplace/1/

INCREASE DATA SIZE

Enough work for the GPU, but gaps due to blocking in control program

```
$ quickstart/run.sh 1/8 b.py --profile
```

```python
nx, ny = 7000, 7000
...
while l2 > l2_target and icount < max_iter:
  icount += 1
  pn = p.copy()
  p[1:-1,1:-1] = .25 * (
    p[1:-1, 2:] + p[1:-1, :-2] +
    p[2:, 1:-1] + p[:-2, 1:-1]
  )
  l2 = np.sqrt(np.sum((p-pn)**2)/np.sum(pn**2))
  if icount % output_freq == 0:
    print(icount, l2)
```



Full example: https://gist.github.com/manopapad/6a3928868bb020736028a4b30193bd29

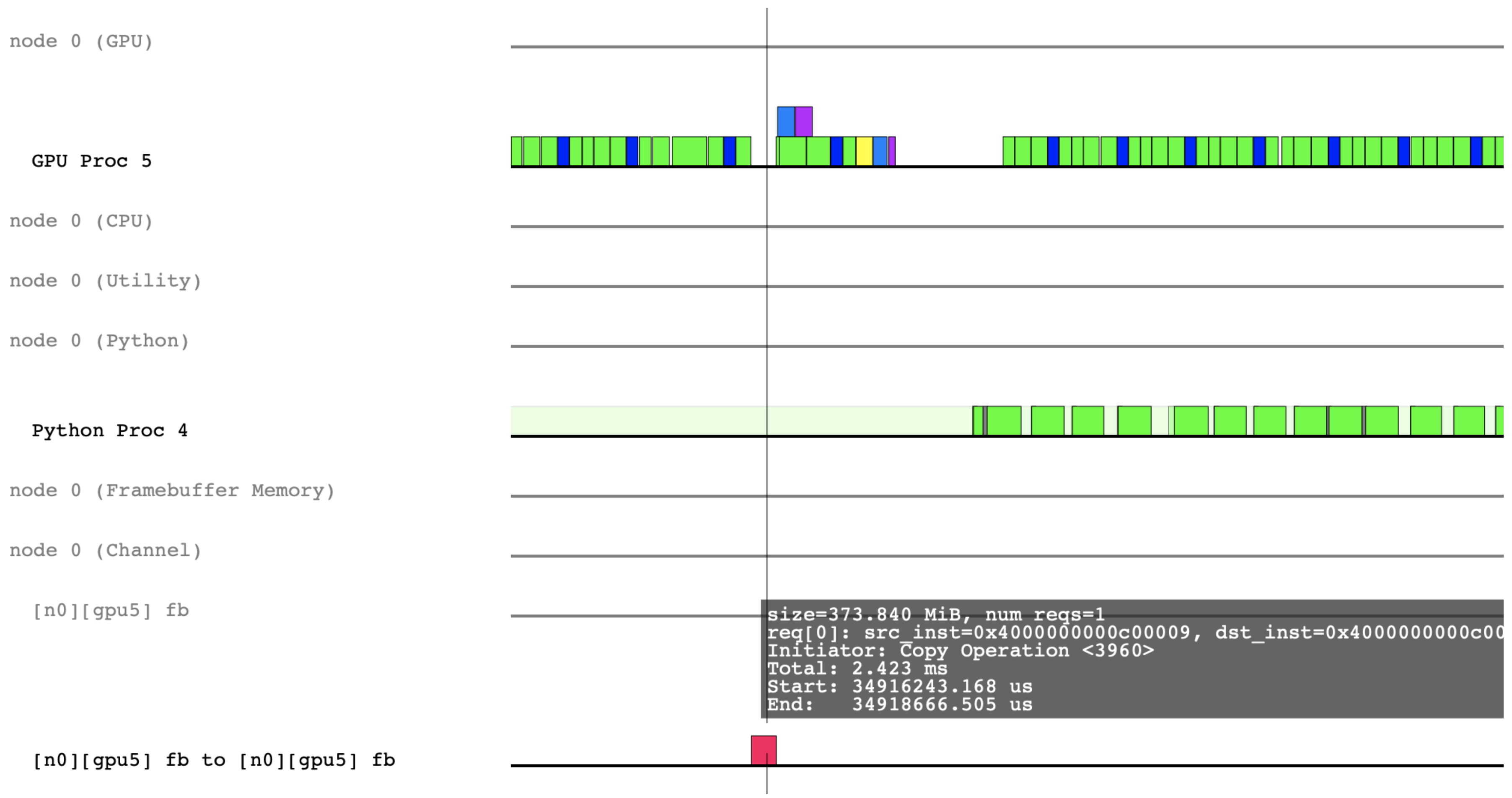
based on https://barbagroup.github.io/essential_skills_RRC/laplace/1/

DON'T CHECK FOR CONVERGENCE EVERY ITERATION

Lets control program run ahead to generate work for the GPU

```
$ quickstart/run.sh 1/8 c.py --profile
```

```python
nx, ny = 7000, 7000
...
while l2 > l2_target and icount < max_iter:
  icount += 1
  if icount % output_freq == 0:
    pn = p.copy()
  p[1:-1,1:-1] = .25 * (
    p[1:-1, 2:] + p[1:-1, :-2] +
    p[2:, 1:-1] + p[:-2, 1:-1]
  )
  if icount % output_freq == 0:
    l2 = np.sqrt(np.sum((p-pn)**2)/np.sum(pn**2))
    print(icount, l2)
```



Full example: https://gist.github.com/manopapad/69e95ec85264164ca7bebf6e932375d8

based on https://barbagroup.github.io/essential_skills_RRC/laplace/1/

RUN ON 2 GPUS

Slightly faster, only boundary cells are traded

```
$ quickstart/run.sh 1/4 c.py --profile
```


```python
nx, ny = 7000, 7000
...
while l2 > l2_target and icount < max_iter:
  icount += 1
  if icount % output_freq == 0:
    pn = p.copy()
  p[1:-1,1:-1] = .25 * (
    p[1:-1, 2:] + p[1:-1, :-2] +
    p[2:, 1:-1] + p[:-2, 1:-1]
  )
  if icount % output_freq == 0:
    l2 = np.sqrt(np.sum((p-pn)**2)/np.sum(pn**2))
    print(icount, l2)
```

Full example: https://gist.github.com/manopapad/69e95ec85264164ca7bebf6e932375d8

based on https://barbagroup.github.io/essential_skills_RRC/laplace/1/

TRY IT YOURSELF!

https://github.com/nv-legate/quickstart

github.com/nv-legate/quickstart

nv-legate/quickstart: Docker images and cluster-specialized build/run scripts

README.md

CoriGPU @ LBL

Add to `~/.bashrc`:

```
# Cori runs even sub-shells in login mode, so guard these from running more
if [[ -z $CONDA_PREFIX ]]; then
    module purge
    module load cgpu esslurm cudatoolkit/10.2.89_3.28-7.0.1.1_2.1__g88d3d59
    eval "$(conda shell.bash hook)"
    conda activate legate
fi
```

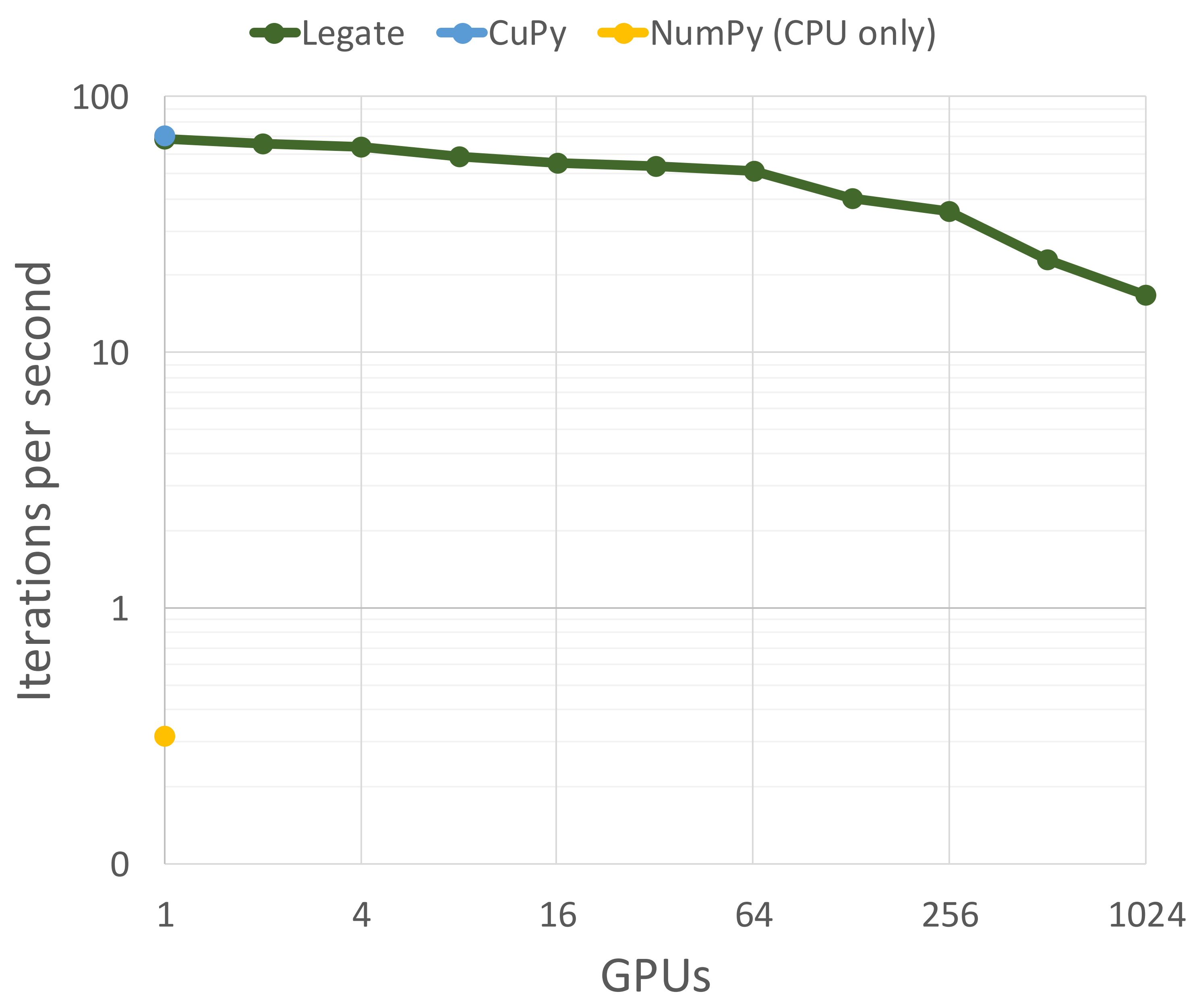
Log out and back in, then run:

BENCHMARK RESULTS

PROGRAMING 1024 GPUS WITH NUMPY

```python
import legate.numpy as np

def cg_solve(A, b, conv_iters):
    x = np.zeros_like(b)
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    converged = False
    max_iters = b.shape[0]

    for i in range(max_iters):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)

        if i % conv_iters == 0 and \
            np.sqrt(rsnew) < 1e-10:
            converged = i
            break

        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
```

Conjugate Gradient example



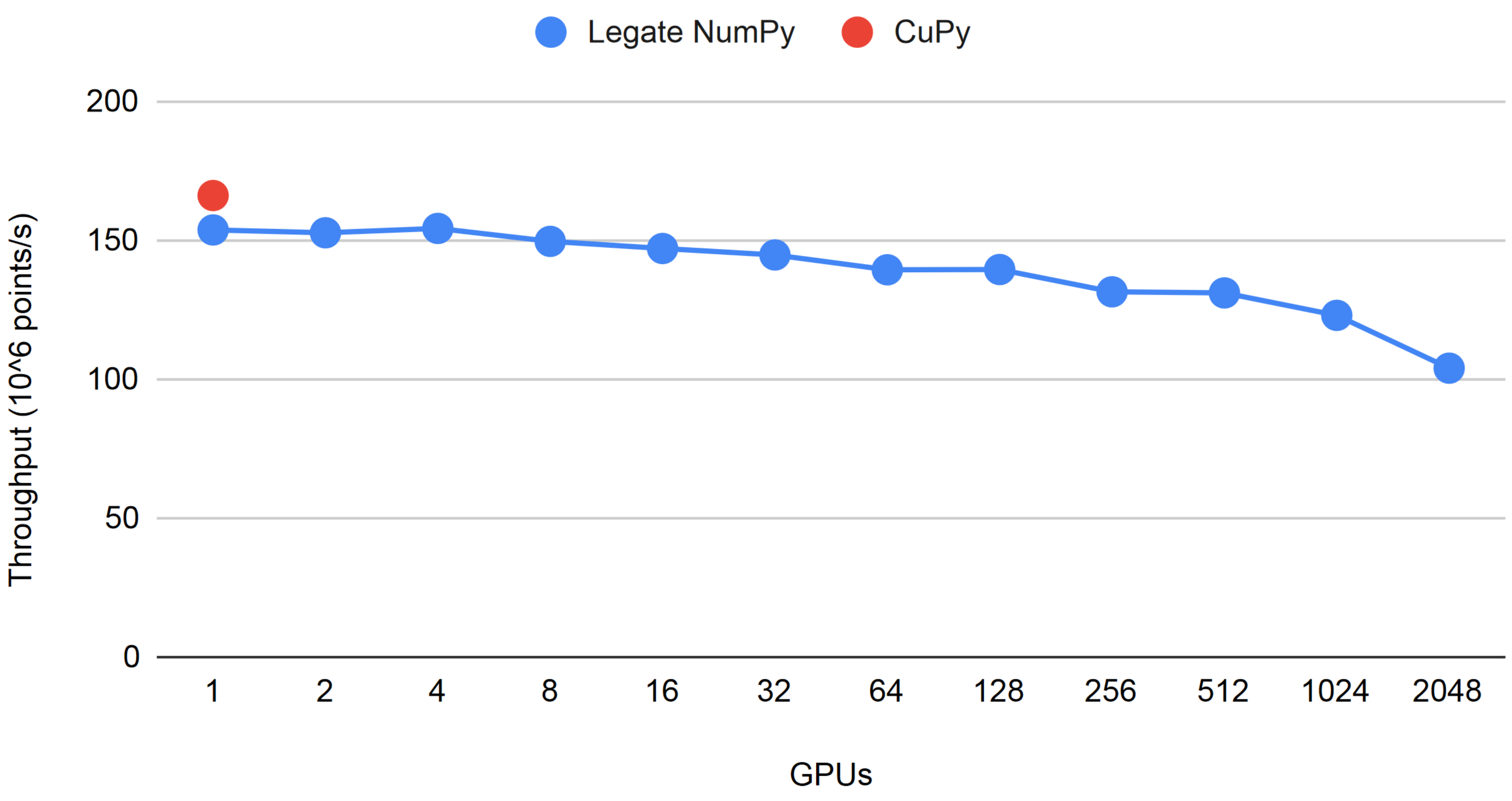
Machine: DGX SuperPOD with A100-80GB GPUs

FROM PEDAGOGY TO SUPERCOMPUTING

```
import legate.numpy as np

for _ in range(iter):
    un = u.copy()

    vn = v.copy()
    b = build_up_b(rho, dt, dx, dy, u, v)
    p = pressure_poisson_periodic(b, nit, p, dx, dy)

    u[1:-1, 1:-1] = (
        un[1:-1,1:-1]
        - un[1:-1,1:-1] * dt / dx * (un[1:-1,1:-1] - un[1:-1,0:-2])
        - vn[1:-1,1:-1] * dt / dy * (un[1:-1,1:-1] - un[0:-2,1:-1])
        - dt / (2 * rho * dx) * (p[1:-1,2:] - p[1:-1,0:-2])
        + nu
        * (
            dt
             / dx ** 2
            * (un[1:-1,2:] - 2 * un[1:-1,1:-1] + un[1:-1,0:-2])
            + dt
             / dy ** 2
            * (un[2:,1:-1] - 2 * un[1:-1,1:-1] + un[0:-2,1:-1])
        )
        + F * dt
    )
```

Code excerpted from “CFD Python”

https://github.com/barbagroup/CFDPython



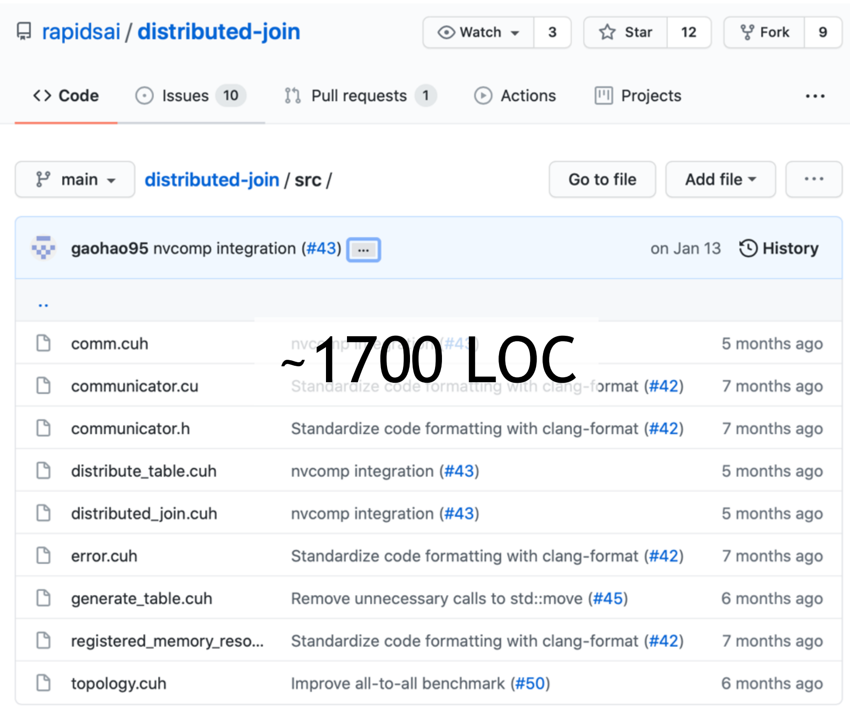
Machine: DGX SuperPOD with A100-80GB GPUs

Extracted from “CFD Python” course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, **1**(9), 21, https://doi.org/10.21105/jose.00021

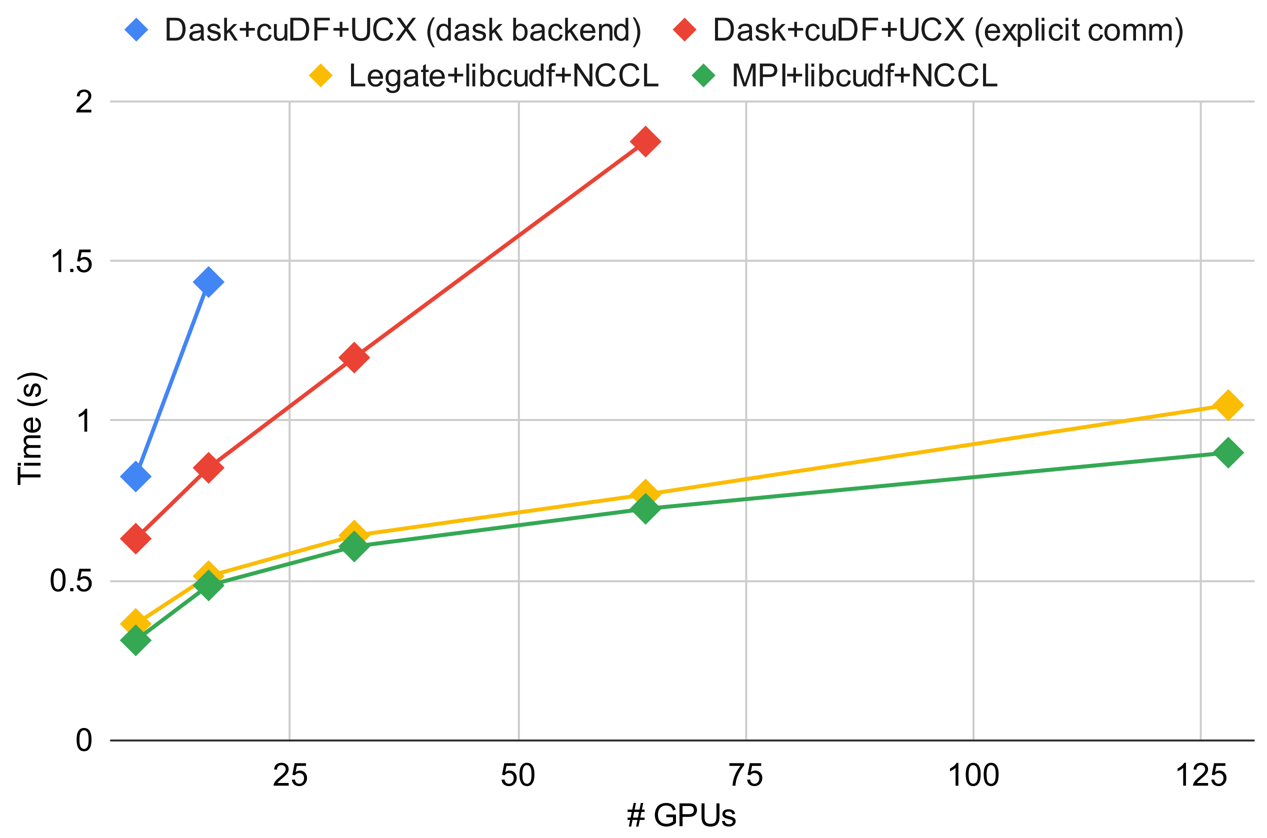
NEAR-MPI PERFORMANCE IN ~10 LINES OF PANDAS

```python
import legate.numpy as np
import legate.pandas as pd

size = num_rows_per_gpu * num_gpus

key_l = np.arange(size)
val_l = np.random.randn(size)
lhs = pd.DataFrame({ "key": key_l, ”val": val_l })

key_r = key_l // 3 * 3     # selectivity: 0.33
payload_r = np.random.randn(size)
rhs = pd.DataFrame({ "key": key_r, ”val": val_r })

out = lhs.merge(rhs, on="key")
```

vs.



Join microbenchmark (300M rows/GPU)



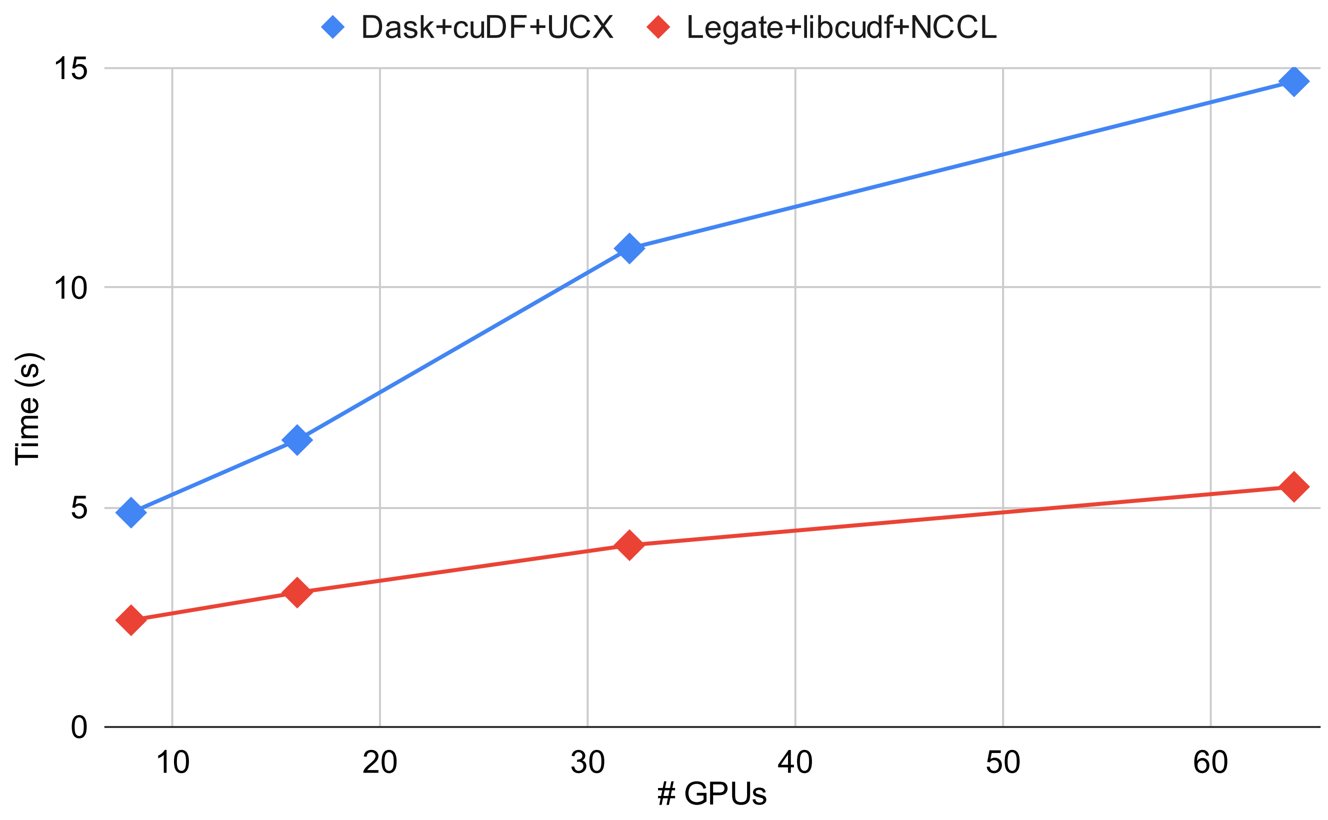
Same level of performance as hand-written MPI code
2.4X faster than the best Dask+cuDF code

Machine: DGX SuperPOD with A100-80GB GPUs

SCALABLE DATA PROCESSING WITH PANDAS

```python
def create_12_mon_features(lib, joined_df):
    testdfs = []
    n_months = 12
    for y in range(1, n_months + 1):
        tmpdf = joined_df[
            [
                "loan_id",
                "timestamp_year",
                "timestamp_month",
                "delinquency_12",
                "upb_12",
            ]
        ]
        tmpdf["josh_months"] = (
            tmpdf["timestamp_year"] * 12 + tmpdf["timestamp_month"]
        )
        tmpdf["josh_mody_n"] = (
            (tmpdf["josh_months"].astype("float64") - 24000 - y) / 12
        ).astype("int64")
        tmpdf = lib.group_and_apply(
            tmpdf,
            ["loan_id", "josh_mody_n"],
            "agg",
            {"delinquency_12": "max", "upb_12": "min"},
        )
        tmpdf["delinquency_12"] = (tmpdf["delinquency_12"] > 3).astype("int32")
        tmpdf["delinquency_12"] += (tmpdf["upb_12"] == 0).astype("int32")
        tmpdf["timestamp_year"] = (
            ((tmpdf["josh_mody_n"] * n_months) + 24000 + (y - 1)) / 12
        ).astype("int16")
        tmpdf["timestamp_month"] = np.int8(y)
        del tmpdf["josh_mody_n"]
        testdfs.append(tmpdf)
        del tmpdf
    del joined_df
    return lib.concat(testdfs)


def combine_joined_12_mon(joined_df, testdf):
    del joined_df["delinquency_12"]
    del joined_df["upb_12"]
    joined_df["timestamp_year"] = joined_df["timestamp_year"].astype("int16")
    joined_df["timestamp_month"] = joined_df["timestamp_month"].astype("int8")
    result = joined_df.merge(
        testdf, how="left", on=["loan_id", "timestamp_year", "timestamp_month"]
    )
    return result
```

Mortgage data example



2.7X faster than Dask+cuDF
(~5X faster on computation tasks)

Machine: DGX SuperPOD with A100-80GB GPUs

SUMMARY

CALL TO ACTION!

Try it out on your own! https://github.com/nv-legate/quickstart#corigpu--lbl
Please report issues on github

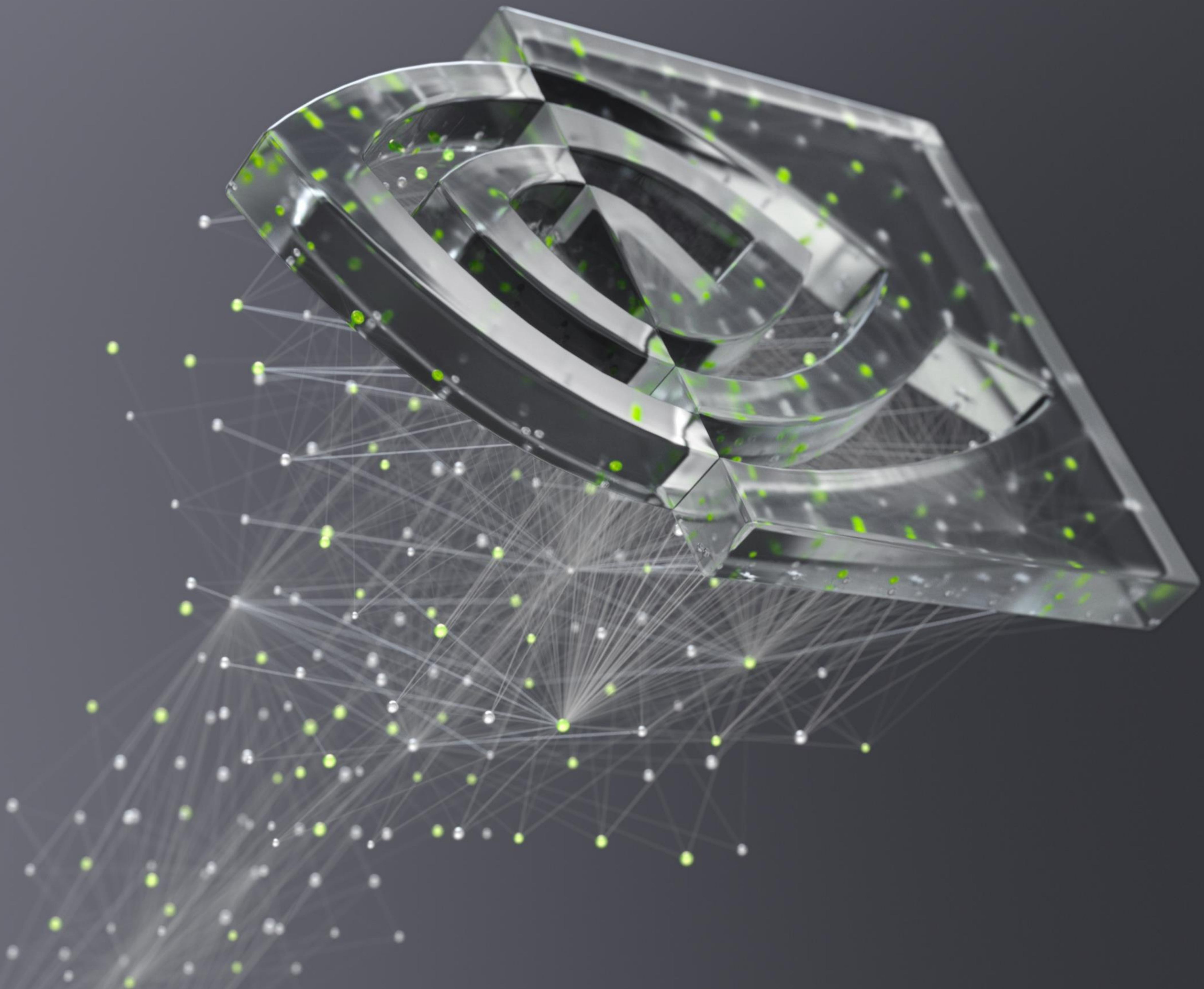
Have an existing NumPy/Pandas code you want to scale? Become an early user!
Legate works best for pure Python programs that operate on large arrays/dataframes
(vs spawning independent work on a lot of small collections)
Help us prioritize work on missing features

Have other Python libraries you want to use? Let us know!
Help us prioritize work on other libraries

Tell us: How are you scaling Python programs today? Are you happy with your setup?

Tell us: What kind of deployment works best for you? (modules? conda? containers? other?)

Contact us at legate@nvidia.com for questions or comments



NVIDIA