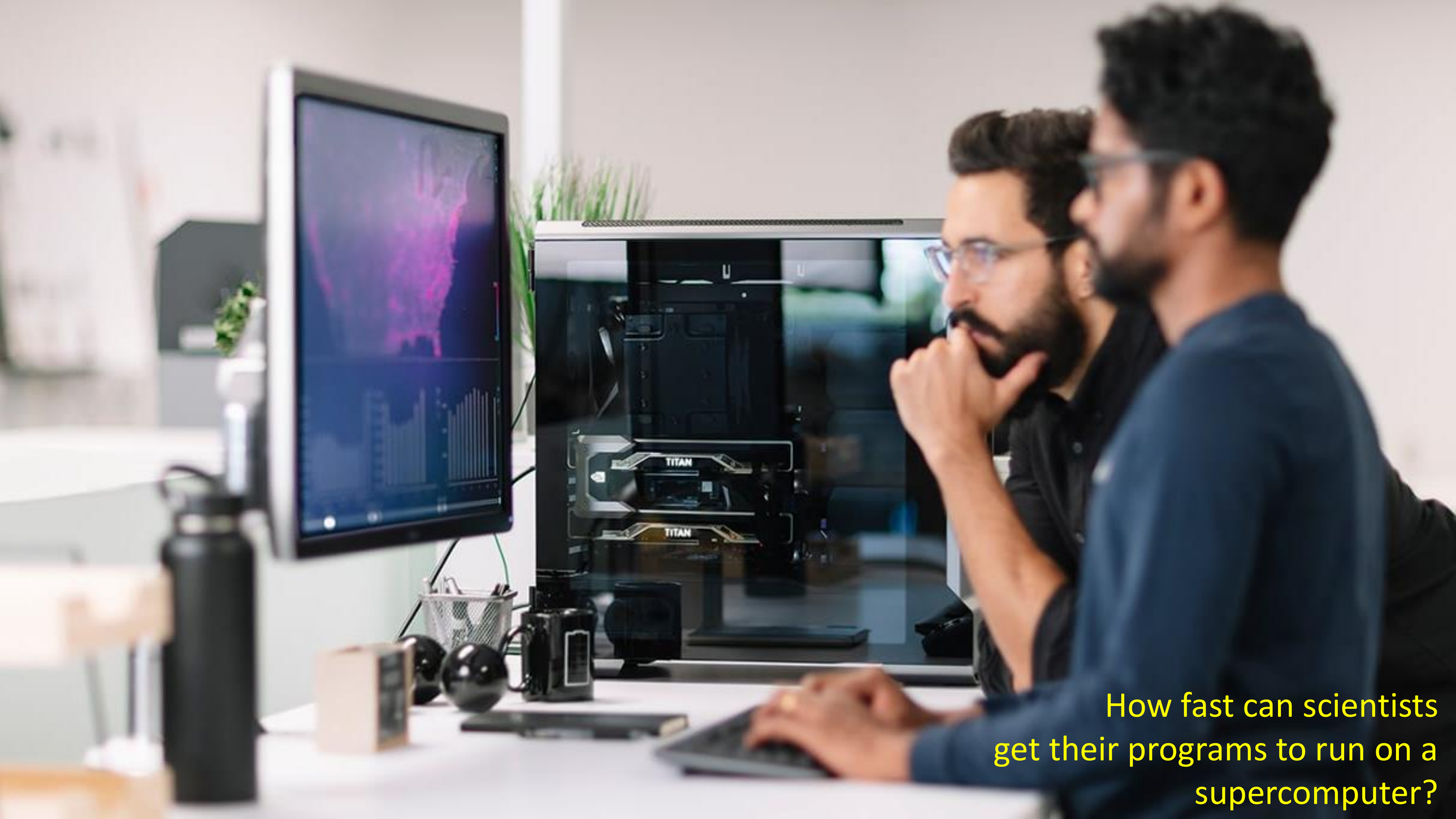
cuPyNumeric, Zero code change scaling of NumPy code

Manolis Papadakis, Software Engineering Manager



How fast can scientists get their programs to run on a supercomputer?

Currently using NumPy extensively + some dask. "Big gap" between what they're doing now and where they want to be. Ultimate goal is to look at the goldmine of data, many TB's and analyze with their methods.

Our objective is not to reach 100% performance in ALL cases (which is anyways not possible), but to provide fast scripting + reasonable performance for complex SciML workflows.

A scientist can only get 2-3 days of beam time each year for experiments. They must run the experiments, analyze the data, adjust and run experiments again. They are in the mode of using libs like NumPy to write their analysis and very unfamiliar with writing code to handle the complexity of GPU or distributed systems.

The use case is to process x-ray CT data. With NumPy on the CPU he cannot go beyond 500 cubes. However, the problem space could require the whole Venado to run. The top challenge is how to scale from CPU to GPU to multi-GPU to multi-node.

“

With Venado online, researchers will have much more GPU resources available. We need to enable them to fully utilize the resources.

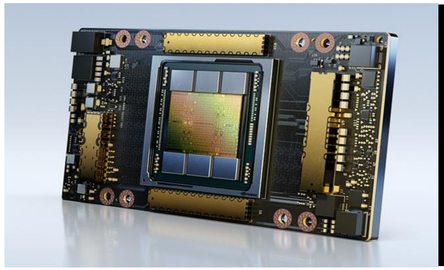
”

What is cuPyNumeric?

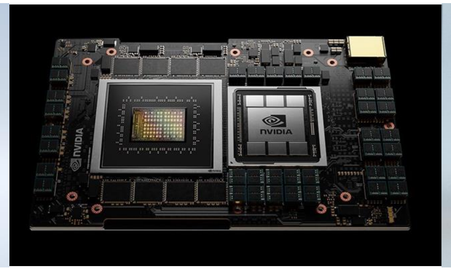
- **Standard Python programming w/o constraints.**
- **Transparent scaling from single CPU core to a multi-GPU multi-Node supercomputer.**

Supercomputer

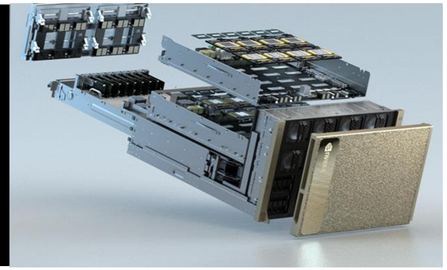
DGX Cloud



GPU



Grace Hopper Superchip



DGX

interchangeable

```
import cupynumeric as np
#import numpy as np

A = np.random.rand(M, K)
B = np.random.rand(K, N)
C = A @ B

print(C)
print(type(C))
```

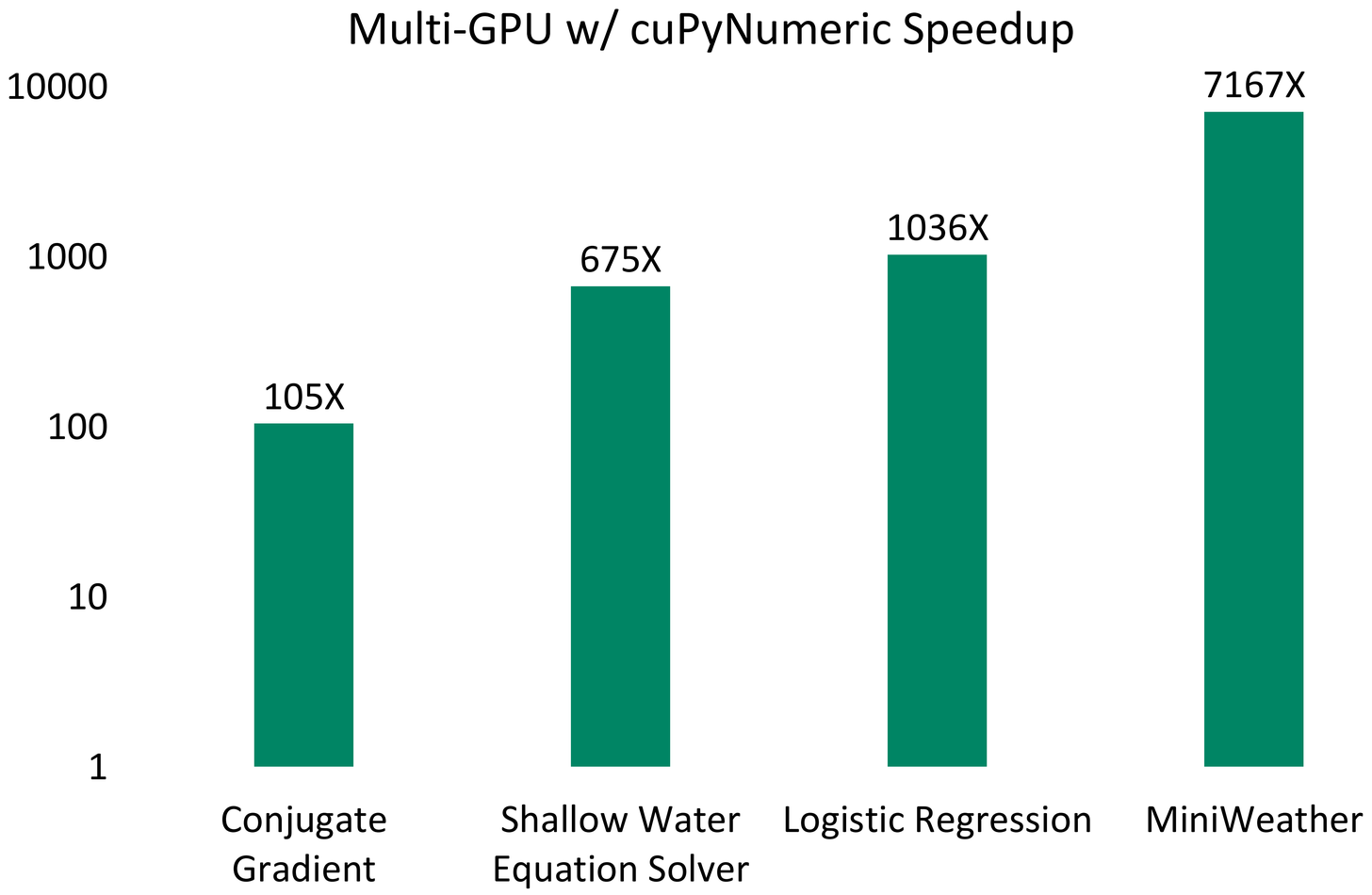
```
$ legate --gpus 1 foo.py
```

No code change run on Multi-GPU Multi-Node

```
$ legate --gpus 8 foo.py
```

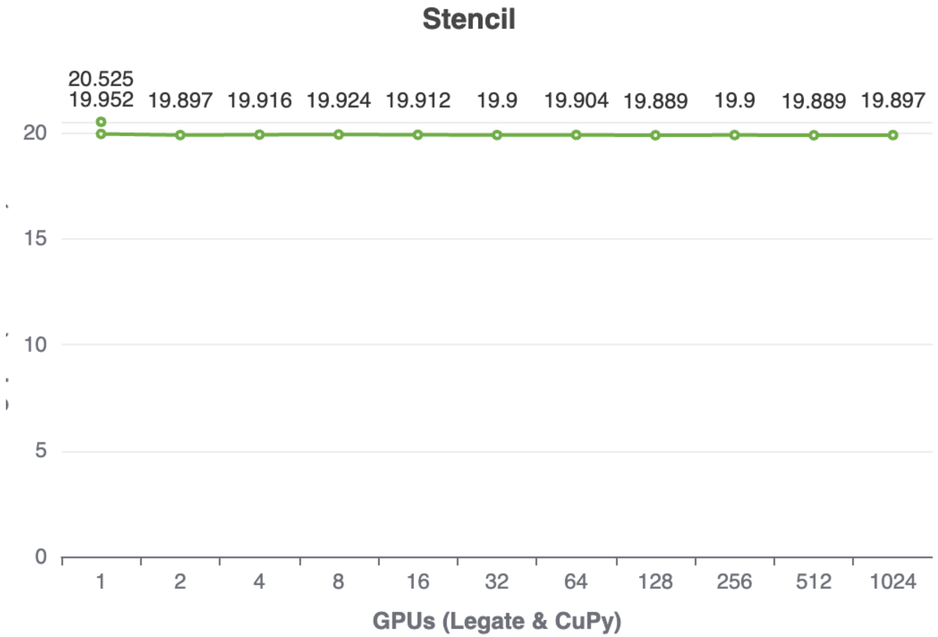
```
$ legate --nodes 64 --gpus 8 foo.py
```

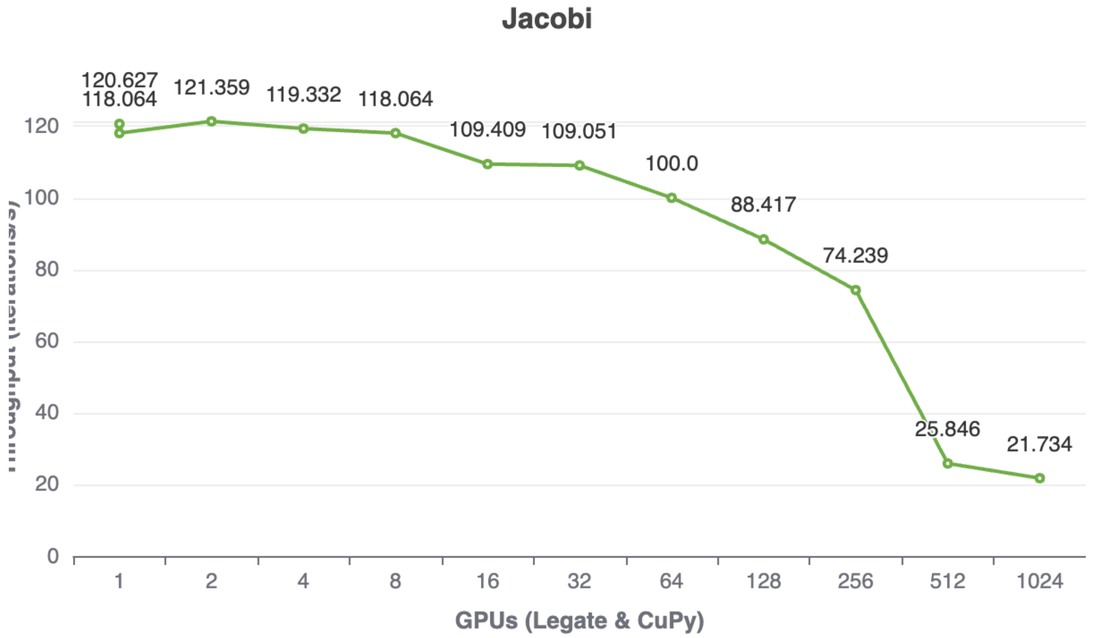
A Not So Fair Comparison

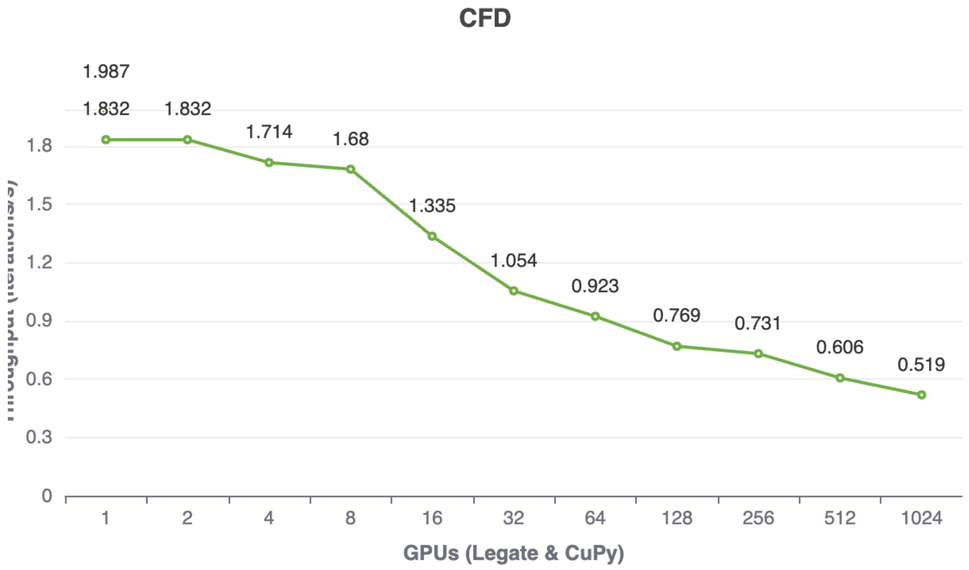


- It’s unfair because the number is single CPU vs 8 GPUs.
- It’s also workload dependent.
- In the real-world use cases the speedup will be less end to end.
- It’s still relevant because:
  - The runs are same code w/o any change.
  - NumPy can only one CPU and cuPyNumeric scales.

Scale to 1000+ GPUs



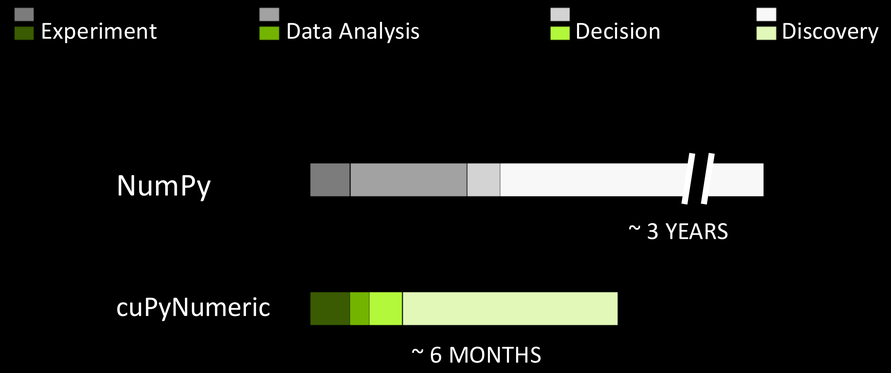




Growing the input size each time such that each GPU has the same working set size.
On EOS w/ 8 H100 per node.

EARLY CUPYNUMERIC ADOPTERS

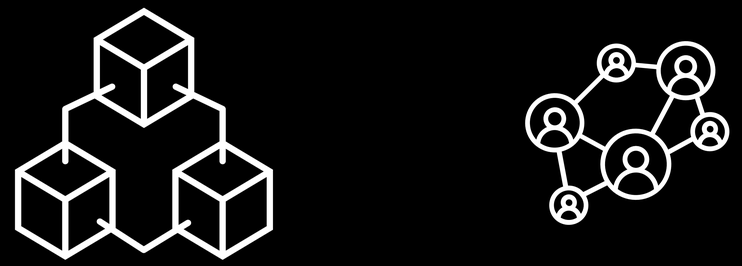
LCLS Experiment Workflow



6X
Faster Analysis

Accelerating Discovery From 3 years to 6 months

Suspect Money Laundry Detection by Transaction Path Tracking



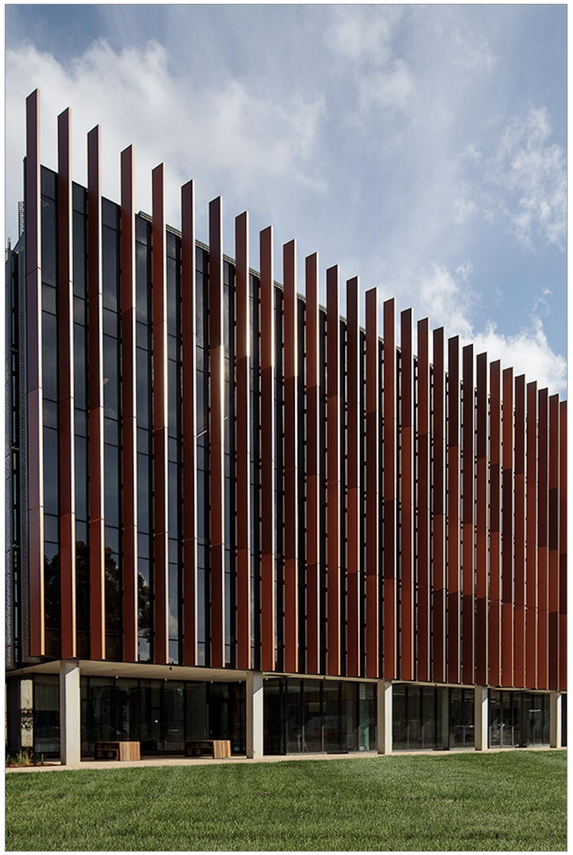
50X
Faster Computation

National Payments Corporation of India

Speedup Computation from 5 hours to minutes

EARLY CUPYNUMERIC ADOPTERS

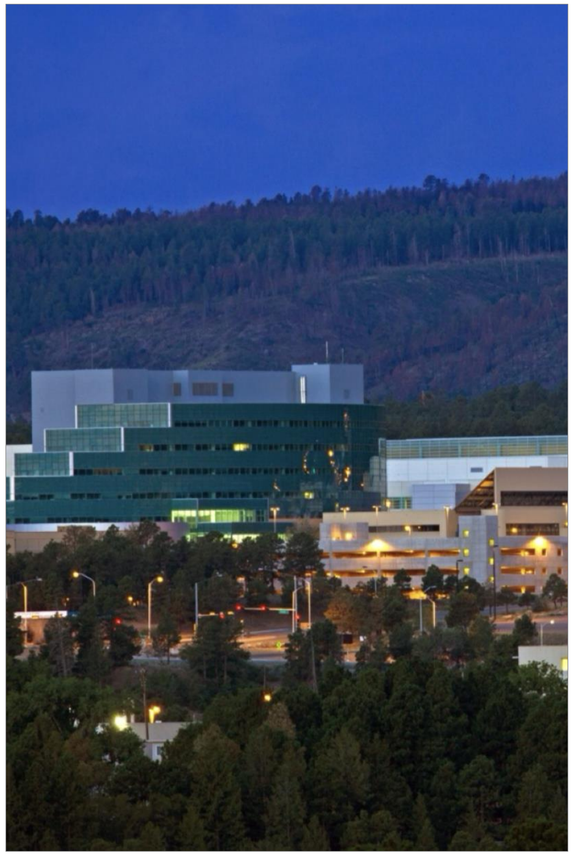
Get Science Done Faster



Australia National University

CTR at Stanford University

UMass Boston



Los Alamos National Lab

Stanford University image by Pere Joan | Los Alamos image by U.S. Department of Energy from United States | Australia National University image by Pascal Vuylsteke | University of Massachusetts Boston Image by Bill Ilott

cuPyNumeric is Part of Legate Ecosystem

Composable libraries built on a common runtime stack

Foundational Legate Libraries

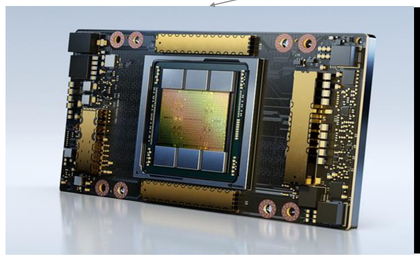
cuPyNumeric | Legate JAX | Legate Sparse | Legate ML | Legate Dataframe | Legate IO | . . .

Legate Runtime Stack

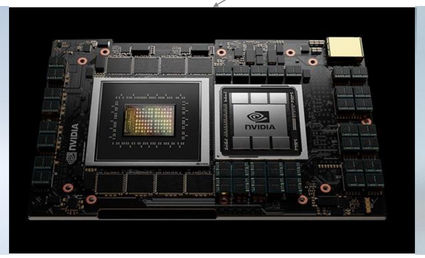
Legate - Productivity & Composability Layer

Legion - Implicit Parallelism Layer

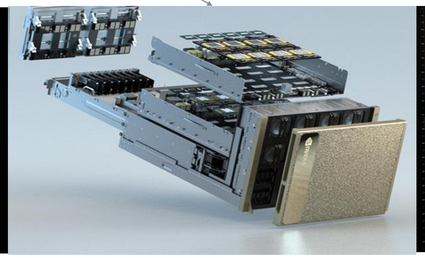
Realm - Runtime for Scalable and Portable Execution



GPU



Grace Hopper Superchip

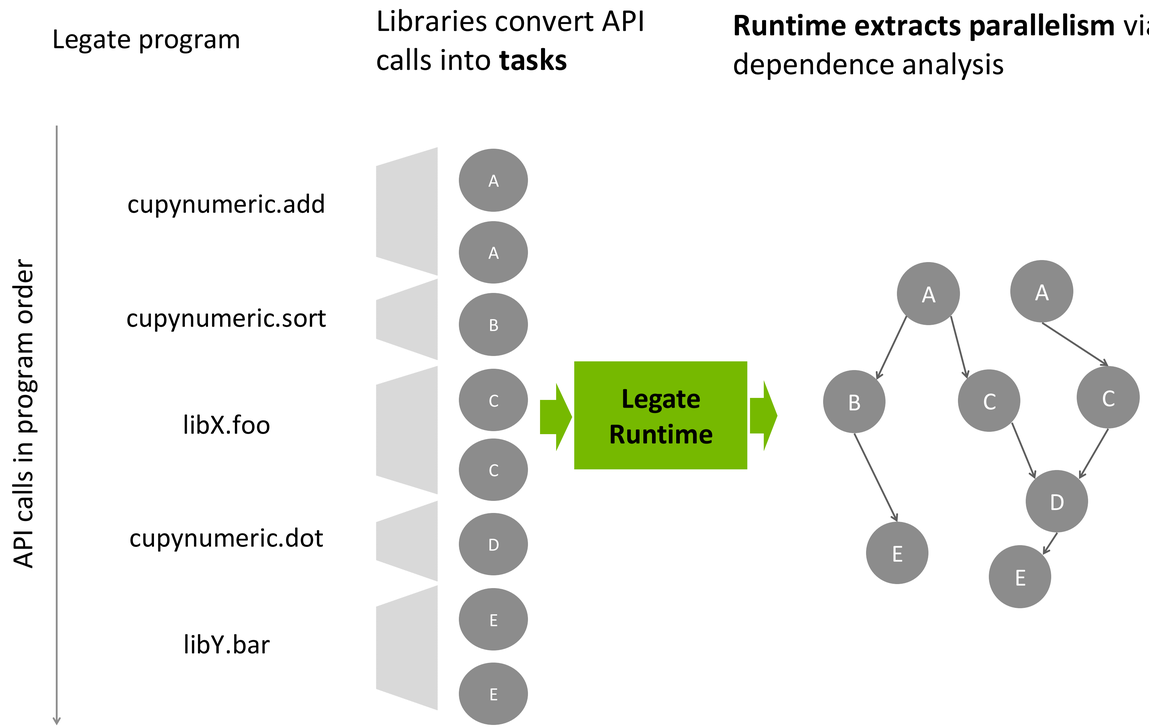


DGX

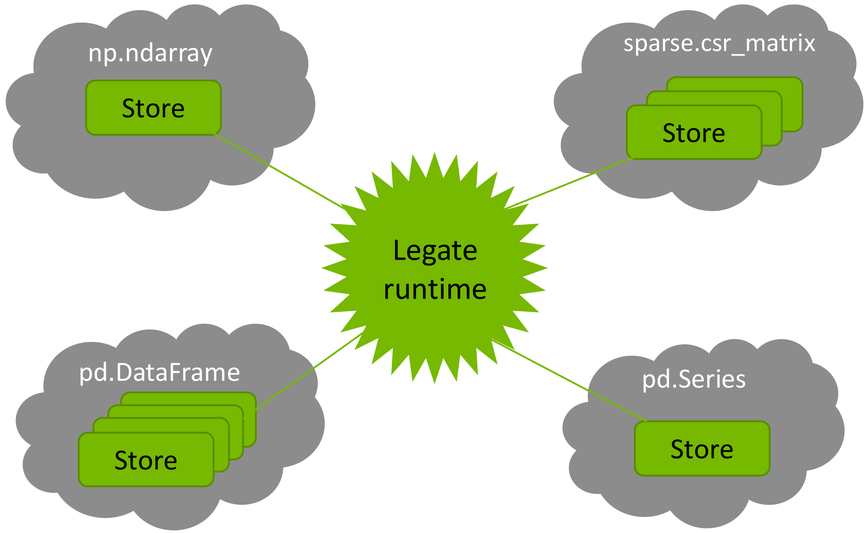
DGX Cloud

Legate - cuPyNumeric's Secret Sauce

Extract Implicit Parallelism for Effortless Scaling



Unified Data Abstraction for Composability



What cuPyNumeric Brings to Scientists

Get your science done faster!

Easy

- Compose script just like using NumPy, SciPy and native Python.
- Data-center scale accelerated computing as simple as programming a single core CPU

Flexible

- Single script runs on both GPU and CPU.
- Dynamically determine CPU or GPU based on the availability of resources.

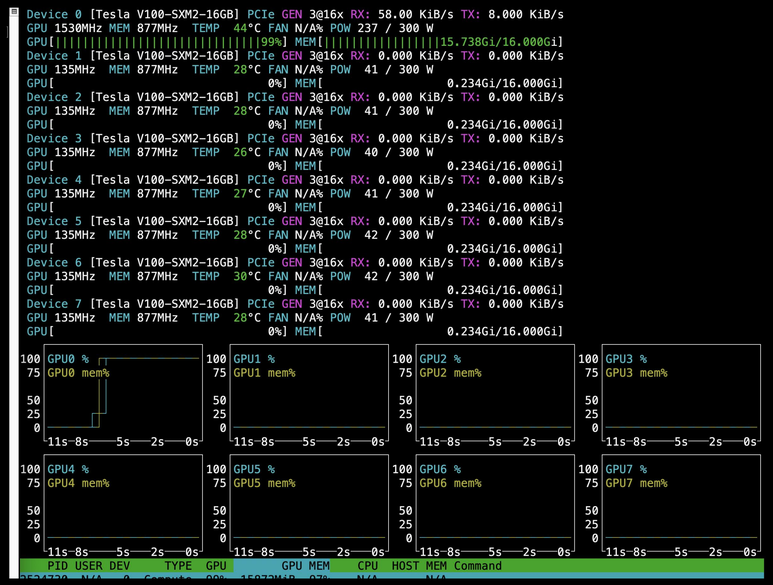
Scalable

- Zero code change to scale from single CPU to multi-GPU, multi-node compute.
- Good scaling efficiency. (goal)

Same Code, Any Number of GPUs

```
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
> CUDA_VISIBLE_DEVICES=0 python gemm.py -n 10000
Problem Size:     M=10000 N=10000 K=10000
Total Iterations: 100
Total Flops:      1999.9 GFLOPS/iter
Total Size:       1200.0 MB
Elapsed Time:     13722.571 ms
Average GEMM:     137.22571 ms
FLOPS/s:          14573.799618161931 GFLOPS/s
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
>
```

1 GPU
14.6 TFLOPS/s

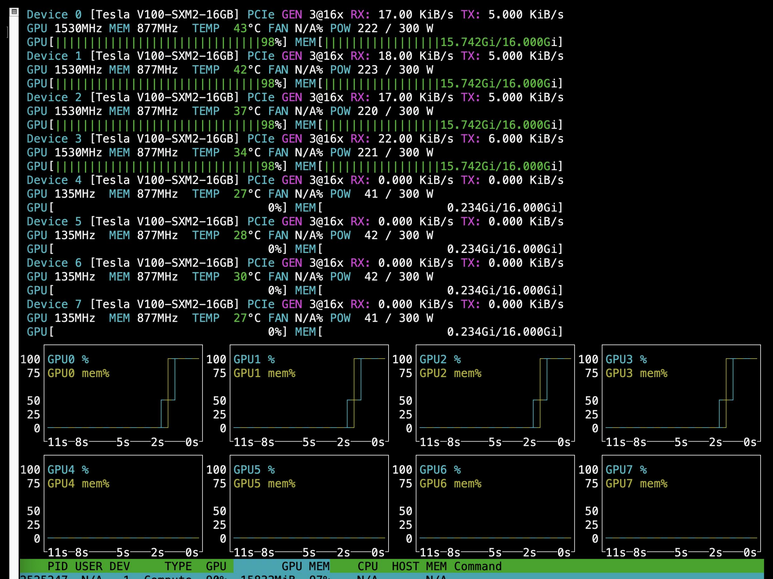



```
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
> CUDA_VISIBLE_DEVICES=0,1 python gemm.py -n 10000
Problem Size:     M=10000 N=10000 K=10000
Total Iterations: 100
Total Flops:      1999.9 GFLOPS/iter
Total Size:       1200.0 MB
Elapsed Time:     7064.53 ms
Average GEMM:     70.64529999999999 ms
FLOPS/s:          28309.031174048385 GFLOPS/s
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
>
```

2 GPUs
28.3 TFLOPS/s


```
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
> CUDA_VISIBLE_DEVICES=0,1,2,3 python gemm.py -n 10000
Problem Size:     M=10000 N=10000 K=10000
Total Iterations: 100
Total Flops:      1999.9 GFLOPS/iter
Total Size:       1200.0 MB
Elapsed Time:     3542.976 ms
Average GEMM:     35.42976 ms
FLOPS/s:          56446.89662024242 GFLOPS/s
```

4 GPUs
56.4 TFLOPS/s




```
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
> python gemm.py -n 10000
Problem Size:     M=10000 N=10000 K=10000
Total Iterations: 100
Total Flops:      1999.9 GFLOPS/iter
Total Size:       1200.0 MB
Elapsed Time:     1841.133 ms
Average GEMM:     18.41133 ms
FLOPS/s:          108623.33139430992 GFLOPS/s
(demo) [wonchanl@prm-dgx-02:~/cupynumeric]
>
```

8 GPUs
108.6 TFLOPS/s

Please give it a try!

cuPyNumeric Tutorials
(https://nvda.ws/3AVtGtx)

Your feedback to
legate@nvidia.com
is highly appreciated.