Legate & cuPyNumeric

Wonchan Lee | Legion Retreat 2024 | Dec 4, 2024

Scaling with Zero Code Change

Author in familiar APIs, scale to any machines

Human productivity is the biggest bottleneck for parallel/distributed programming

- Prototypes written in “easy” programming languages/APIs (e.g., Python, MATLAB, etc.) are often re-implemented using “more serious” frameworks for scaling

“Zero code change” scaling for better productivity

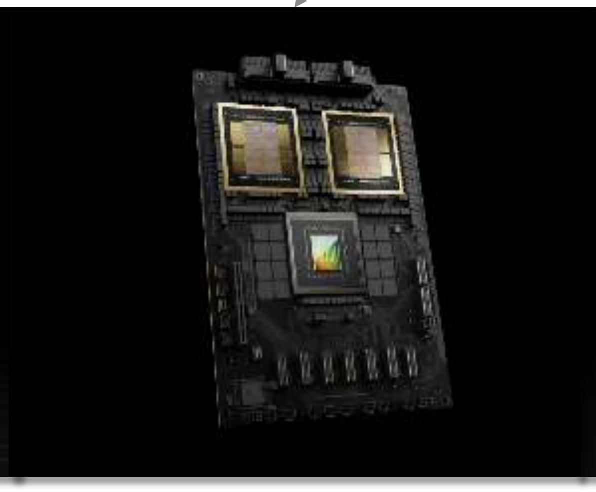
- Author programs in familiar APIs, scale them to any machines
- Easy transition from prototyping to production

NumPy SciPy pandas scikit learn dmlc XGBoost HDF . . .

```
# Generate a random positive semi-definite matrix
A = scipy.sparse.random(n, n, format=“csr”)
A = 0.5 * (A + A.T) + n * scipy.sparse.eye(n)

# Estimate the maximum eigenvalue of A
x = numpy.random.rand(A.shape[0])
for _ in range(iters):
  x = A @ x
  x /= numpy.linalg.norm(x)
result = numpy.dot(x.T, A @ x)
```

Single GPU



Mixed CPU/GPU

Single-Node Multi-GPU

Multi-Node Multi-GPU Cloud & Supercomputer

Introducing Legate

Programming framework for scalable and composable software

Provides uniform solutions to common scaling problems

- Data partitioning and coherence management
- Compute partitioning and distribution
- Scalable execution on distributed memory machines
- **Composability/interoperability between libraries**

Built on years of research on Legion and Realm

- Scalable task-based implicit parallelism
- First-class data partitions with coherence management
- Constraint-based partitioning facilitating library-local reasoning for data partitions

NumPy SciPy pandas learn dmlc XGBoost . . .

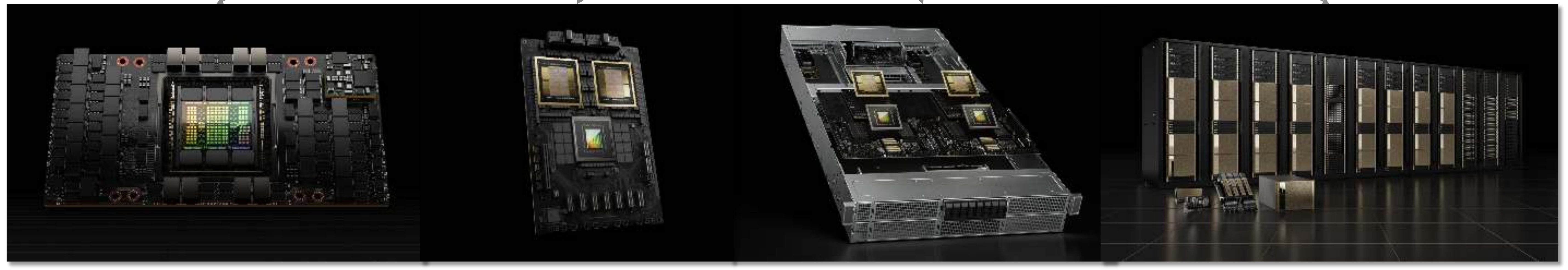
cuPyNumeric | Legate Sparse | Legate Dataframe | Legate RAFT | Legate Boost | Legate IO

Legate Framework

Legate Core - Productivity & Composability Layer

Legion - Implicit Parallelism Layer

Realm - Runtime for Scalable and Portable Execution



Single GPU | Mixed CPU/GPU | Single-Node Multi-GPU | Multi-Node Multi-GPU Cloud & Supercomputer

How Legate Works

Implicit parallelism via “scale-free” tasking

1. Legate program makes API calls

2. Legate libraries issue **“scale-free” tasks**

3. Legate core converts each scale-free task to parallel tasks

4. Legion analyzes data dependencies and constructs a Realm task graph

5. Realm executes the task graph on a machine in a scalable manner

Legate Framework

Legate libraries are **free of any explicit parallelization or synchronization/data movement**, making them **composable and transparently scalable** by construction

Legate Programming Model

Index launch[1] + constraint-based auto-partitioning[2] = scale-free tasking

Example: Multi-GPU Batched FFTs

Declare a **task**, a unit of computation applied to a subset of data

Specify desirable data partitions using **partitioning constraints**

Specify how data is used by the task via type annotations

Task bodies can **reuse the existing single-GPU libraries**

Tasks are **invoked like Python functions**, and the runtime **automatically parallelizes** them **by partitioning inputs and outputs**

```
import legate.core as lg

@lg.task(
  variants=(“gpu”,)
  constraints=(
    lg.align(”input", ”output"),
    lg.broadcast(”input", (0,)),
  ),
)
def batched_fft(
  output: lg.OutputArray, input: lg.InputArray,
):
  cu_input     = cupy.asarray(input)
  cu_output    = cupy.asarray(output)
  cu_output[:] = cupy.fft.fftn(cu_input, axes=(0,))


batched_fft(
  lg.create_array(shape=(M,N), dtype=lg.complex64),
  cupynumeric.random.rand(M,N).astype(np.complex64),
)
```

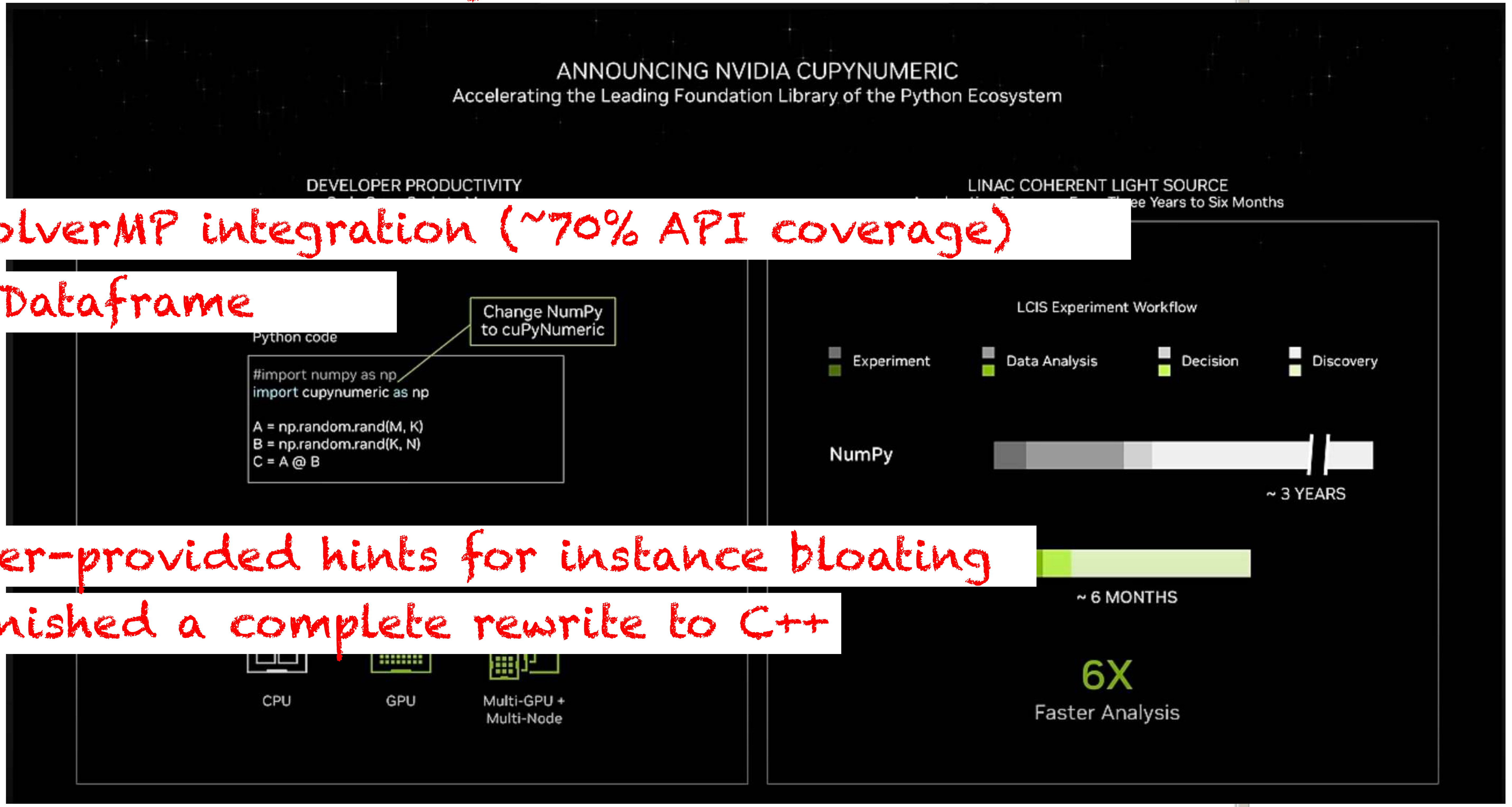
1. Rupanshu Soi, Michael Bauer, Sean Treichler, Manolis Papadakis, Wonchan Lee, Patrick S. McCormick, Alex Aiken, Elliott Slaughter, *Index launches: scalable, flexible representation of parallel task groups*. SC 2021
2. Wonchan Lee, Manolis Papadakis, Elliott Slaughter, Alex Aiken, *A constraint-based approach to automatic data partitioning for distributed memory execution*. SC 2019

Revisiting the Plan From 2022

Updates since the last retreat

Plan

Grinding towards a beta-release — cuPyNumeric announcement @ SC'24

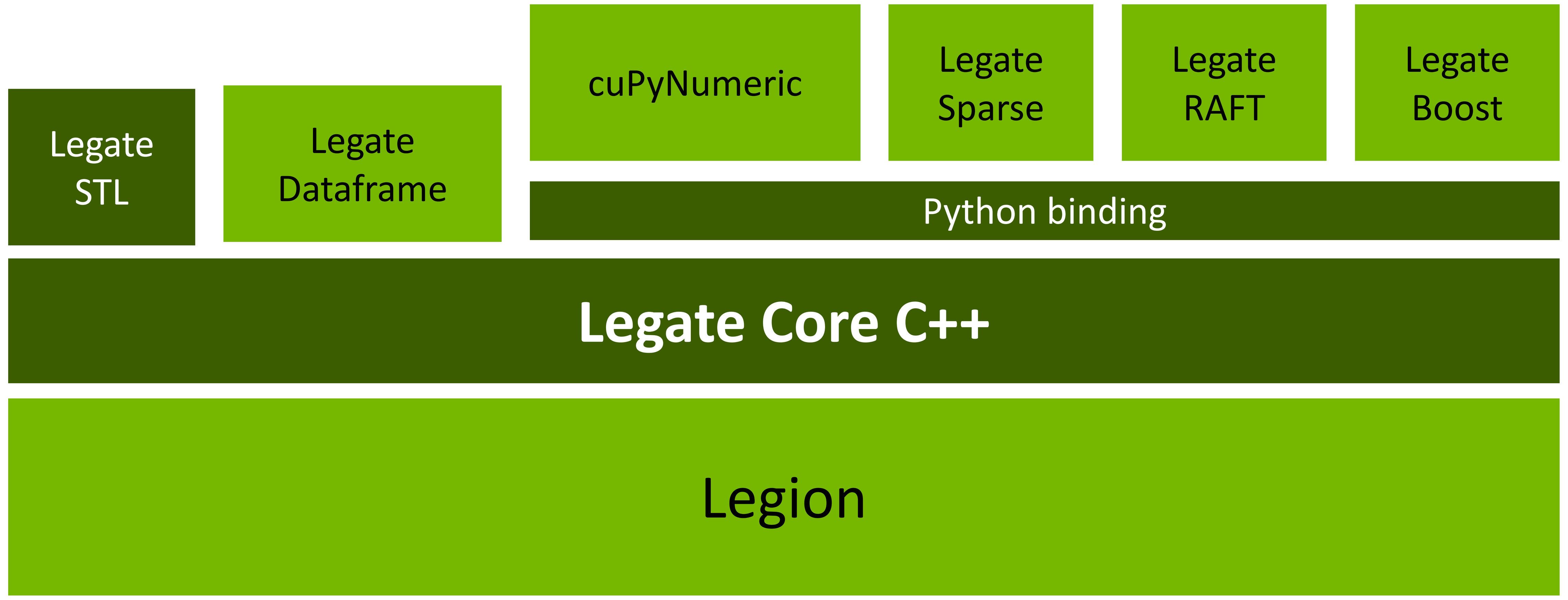


- More API coverage
  - FFT and linear algebra — cuFFT/cuSolver/cuSolverMP integration (~70% API coverage)
  - IO functions — Legate IO & Legate Dataframe
- Performance improvements
  - Better mapping heuristics — User-provided hints for instance bloating
  - Reduce overhead by porting Legate to C++ — Finished a complete rewrite to C++
- Usability
  - Automatic machine configuration — Machine flags are optional now
  - Compatibility with the canonical Python interpreter — Python is preferred to the Legate launcher
  - Multi-node capable conda packages — Conda packages are shipped with network support

Rewriting Legate Core in C++

A complete rewrite of the core framework in C++

- Allows Legate libraries to be written in languages other than Python (most importantly C++)
  - Libraries written in different languages can talk to each other
- Greatly reduces the framework overhead
- Is still a drop-in replacement of the old Python implementation



Has enabled a new opportunity: **Legate STL**

- STL-like template library of high-level, reusable, generic algorithms
- Scales functional C++ programs to MNMG systems via the Legate runtime
- Easy for users to extend Legate-based applications with custom algorithms

```
namespace stl = legate::experimental::stl;

// Create a 2-D Legate store and initialize it
auto store = stl::create_store<int64_t>({4, 5});
stl::fill(store1, 1.);

// Operate on a row of data at a time, accessible via mdspan
struct MungeRow {
  template <class Elem, class Ext, class Map, class Acc>
  void operator()(std::mdspan<Elem, Ext, Map, Acc> row);
};
stl::for_each(MungeRow{}, stl::rows_of(store));

// Transform the store with a pre-defined operator and
// reduce the result with a custom operator, in a single call
auto result =
stl::transform_reduce(store,
                      stl::scalar{std::int64_t{0}},
                      std::plus{},
                      [](auto x) const { return x * x; });
```

Python Task Support

Enable pure Python-based library development with Legate

- Tasks can be defined as decorated Python functions
- Legate data containers implement common array protocols, allowing task bodies to reuse existing single processor Python libraries (e.g., NumPy, CuPy, or Numba)

Eliminate the boilerplate code for task launch

- Function call = task launch
- Task launch boilerplate is synthesized from the type signature and decorator

```
import legate.core as lg

@lg.task(
  variants=(“gpu”,)
  constraints=(
    lg.align(”input", ”output"),
    lg.broadcast(”input", (0,)),
  ),
)
def batched_fft(
  output: lg.OutputArray, input: lg.InputArray,
):
  cu_input      = cupy.asarray(input)
  cu_output     = cupy.asarray(output)
  cu_output[:] = cupy.fft.fftn(cu_input, axes=(0,))


batched_fft(
  lg.create_array(shape=(M,N), dtype=lg.complex64),
  cupynumeric.random.rand(M,N).astype(np.complex64),
)
```

Legate RAPIDS: End-to-end Acceleration for Data Analytics

RAPIDS + **Legate**

Collection of optimized GPU kernels for ETL & ML

Runtime system for HPC-grade scalability for Python libraries

Target benchmark: RAPIDS GPU-XB-AI (derived from TPCx-AI)

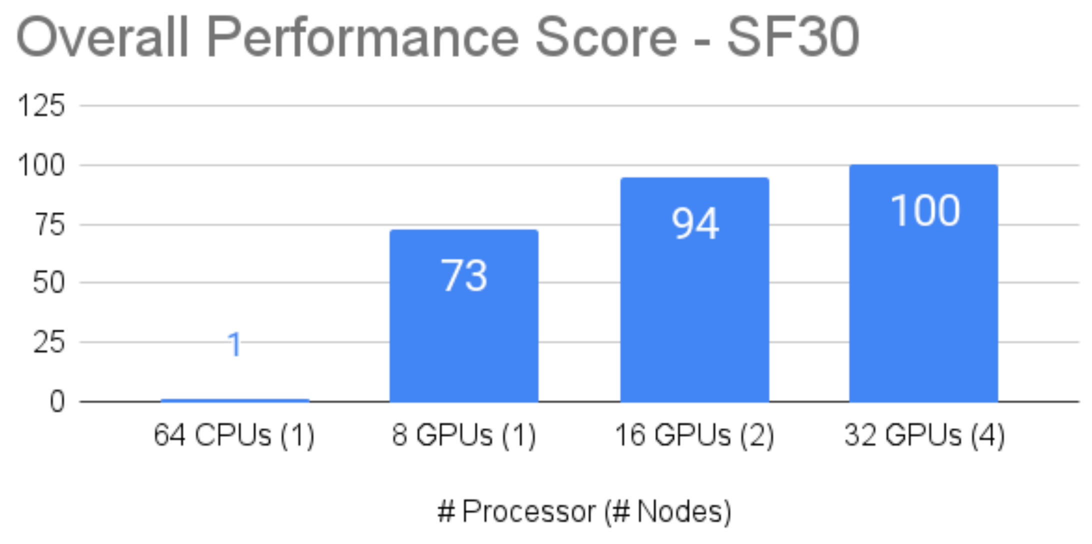
Data Loading → ETL → ML Training → ML Serving

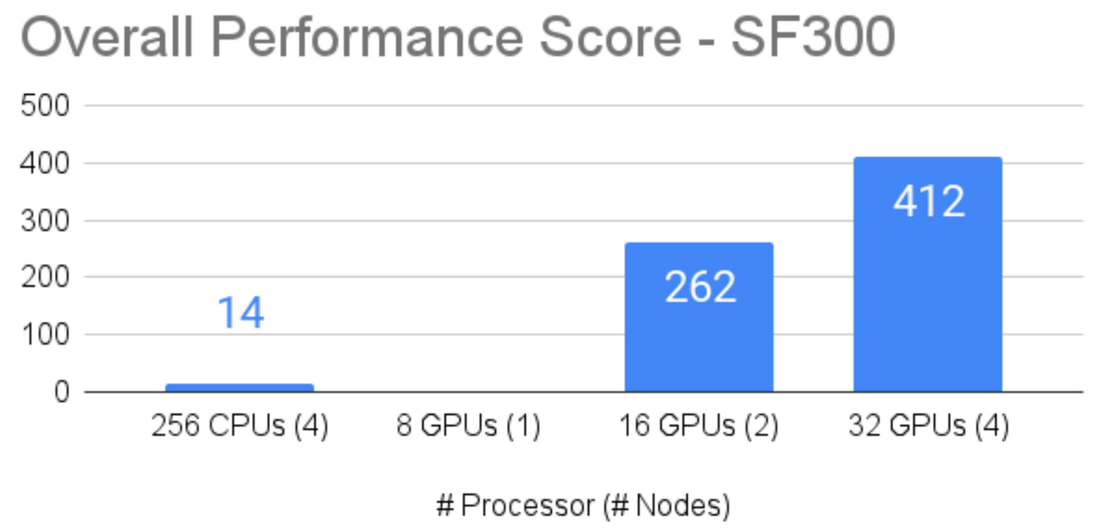
Workload	ETL Legate Dataframe + cuPyNumeric	ML Legate RAFT + Legate Boost
UC01	Hash join, hash aggregate, null handling, element-wise ops	K-means
UC04	Drop duplicates, cast to string, functions to support TF-IDF	Naive Bayes, TF-IDF
UC10	Hash join, datetime support, element-wise arithmetic	Logistic Regression

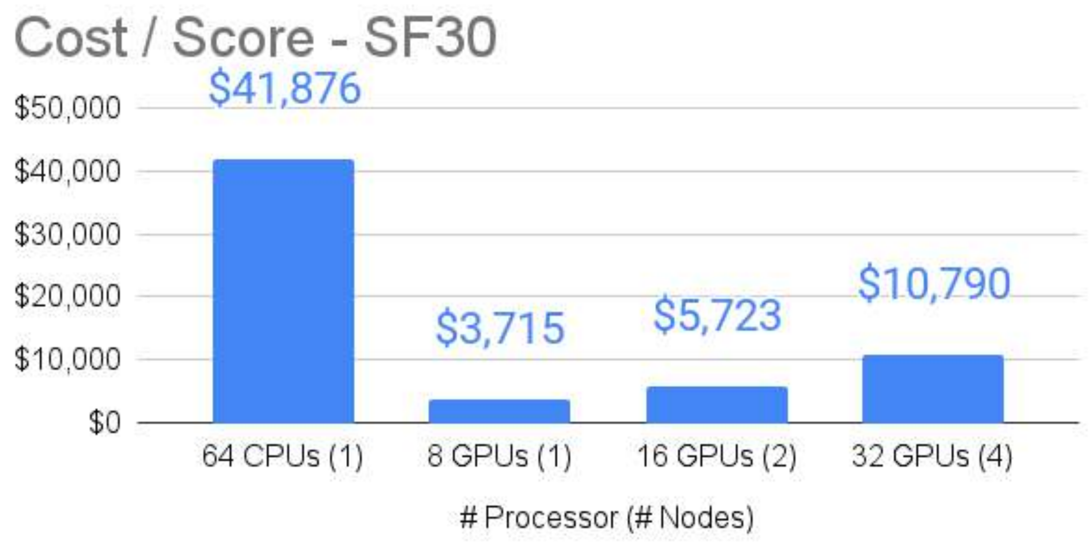
Preliminary comparison* with official TPCx-AI results

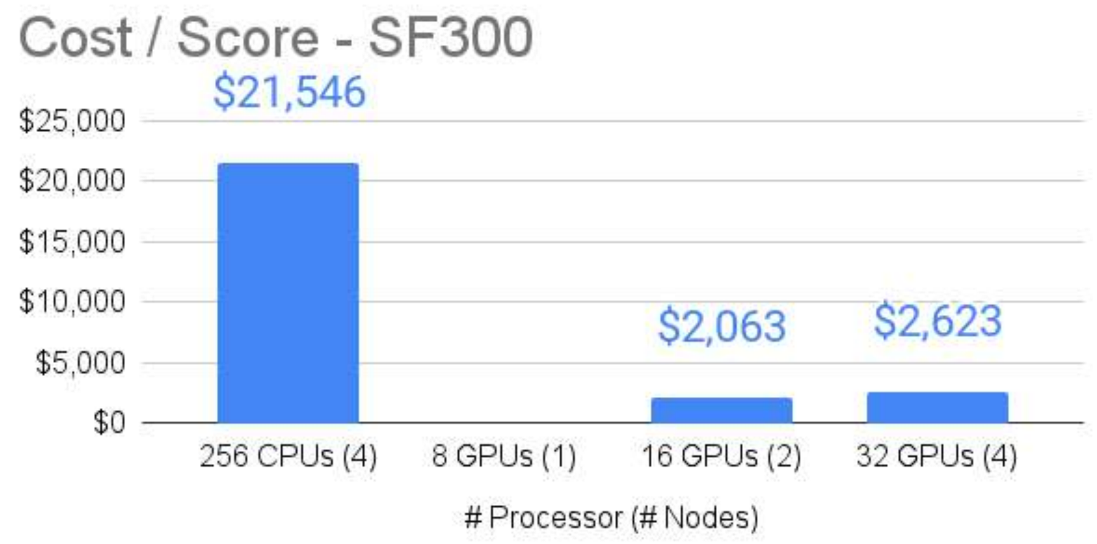
$$Score = \frac{SF \cdot (\#\, use\ cases)}{geomean(Perf_{training}, Perf_{serving})}$$

Cost	CPU nodes	GPU nodes
SF30	\$ 44,248	\$ 270,000 / node
SF300	\$ 309,091	

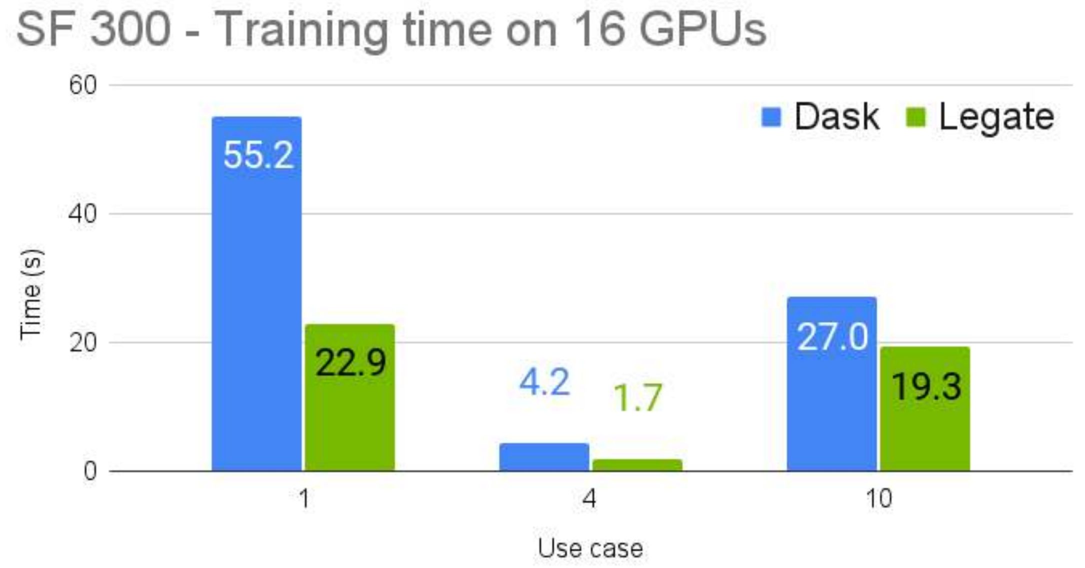








Legate versions are ~10X more cost effective than the CPU counterparts



Legate is1.4-2.5X faster than Dask

NVIDIA

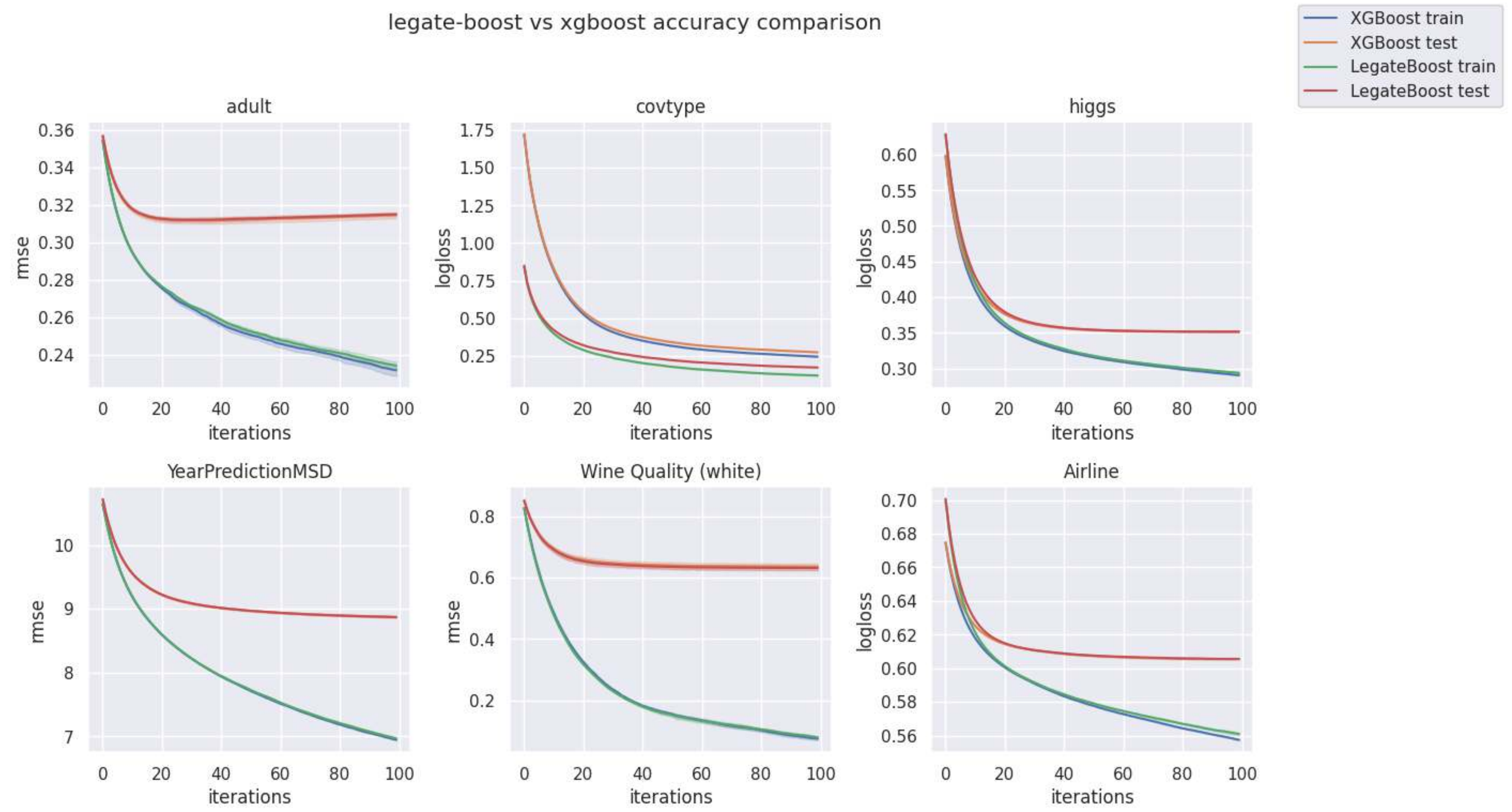
* Disclaimer: the Legate numbers aren’t official TPCx-AI results and do not abide by the official rules of TPCx-AI

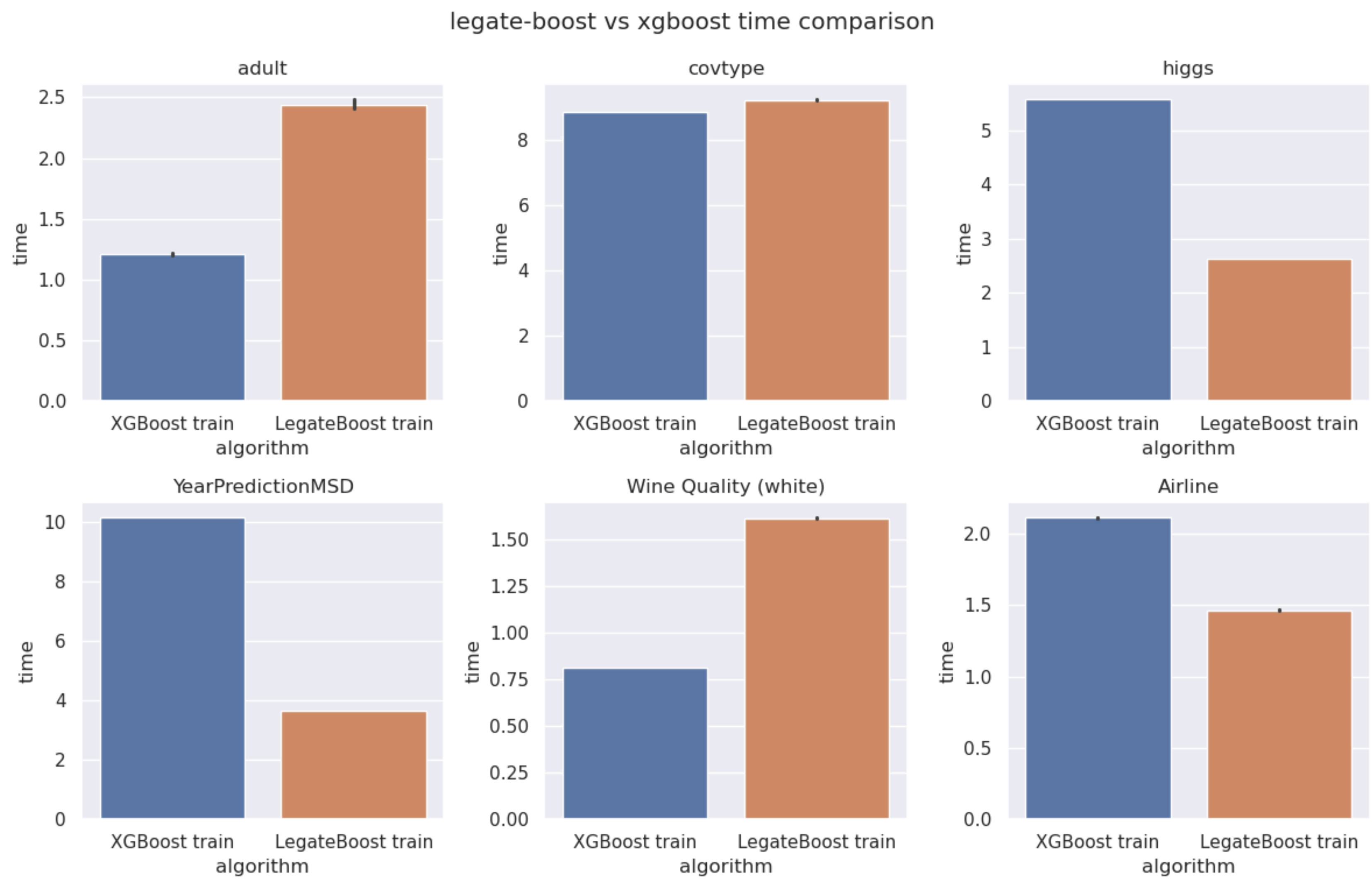
Legate Boost: Gradient Boosting on Legate

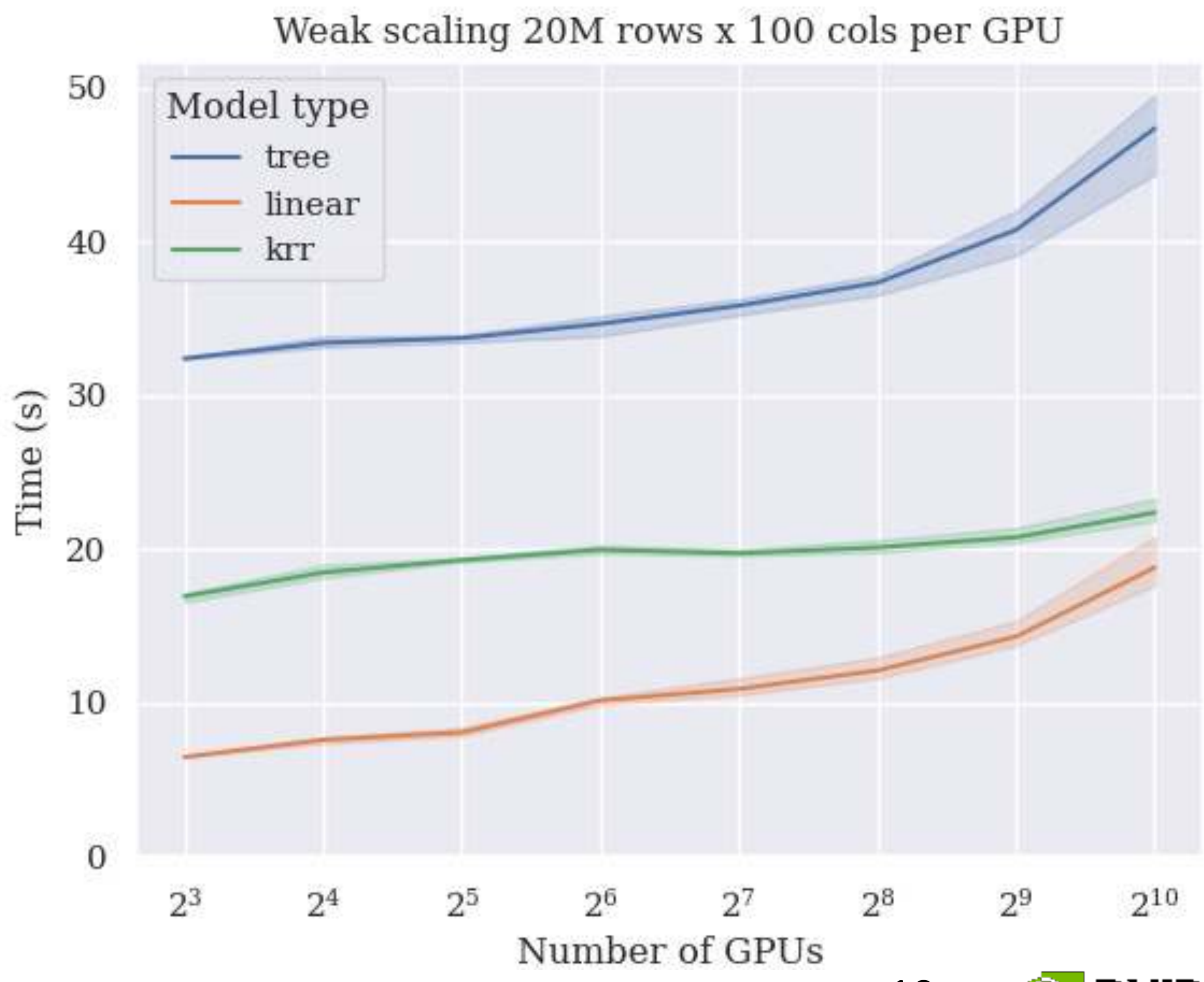
Composability for the win!

- Gradient boosting library built on Legate
- Composability boosts development productivity
  - **Majority** of the library components are **written in cuPyNumeric**
  - Only several core kernels are written as **custom Legate tasks that are seamlessly composed with cuPyNumeric** components
  - Library users can **implement custom objective functions in cuPyNumeric**

Competitive performance compared to the SOTA







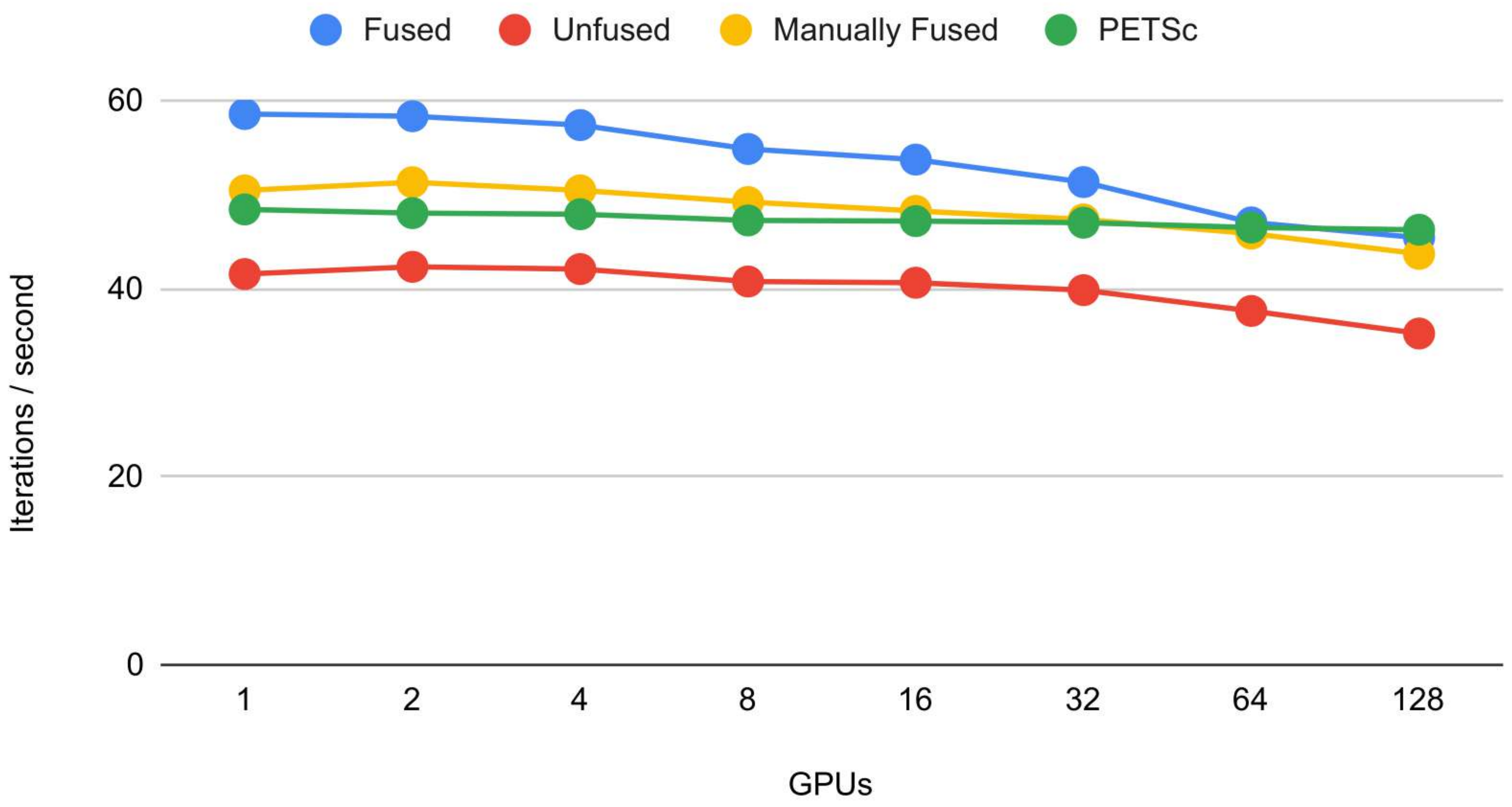
Quotes from Rory Mitchell, the author of Legate Boost (emphasis mine):

*“**Existing software packages** such as XGBoost [3] or LightGBM [5] **require tens of thousands of lines of carefully tuned C++ code to achieve** high levels of parallel performance and implement **state of the art features**. **Legate Boost's** implementation within the legate parallel programming framework **is dramatically simpler, more extensible and more maintainable, yet providing comparable performance** compared to existing libraries.”*

Fusion Across Library Boundaries*

Cross-library optimization enabled by common foundation

Conjugate Gradient Throughput on DGX-A100 cluster



Removes

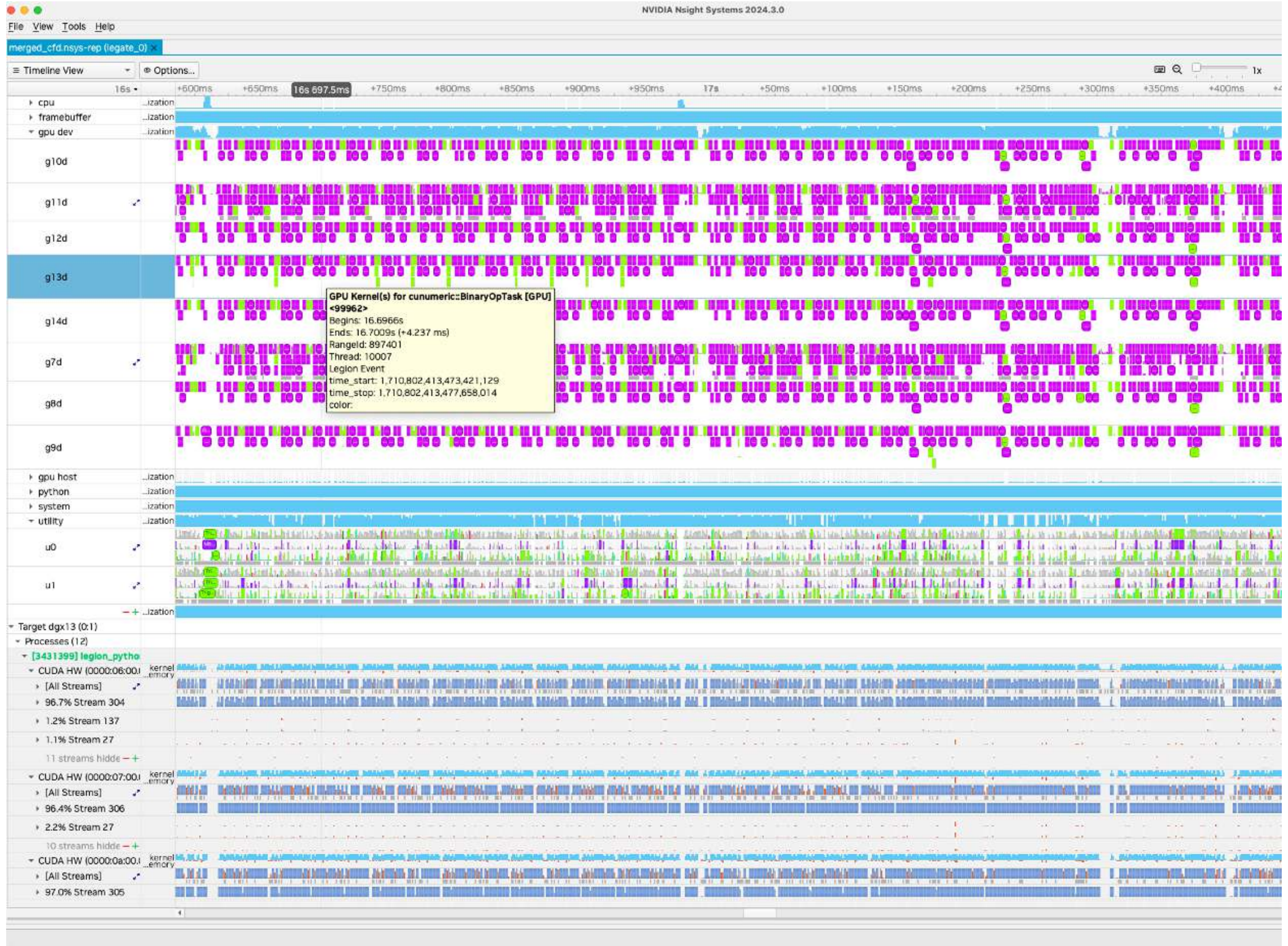
- Task launch overhead (1 launch instead of N)
- Extra temporary allocations
- Sync/data movement between kernels

* Rohan Yadav, Shiv Sundram, Wonchan Lee, Michael Garland, Michael Bauer, Alex Aiken, Fredrik Kjolstad, *Composing Distributed Computations Through Task and Kernel Fusion*. ASPLOS 2025 (to appear)

Legate and CUDA Activities, All in the Same Profile

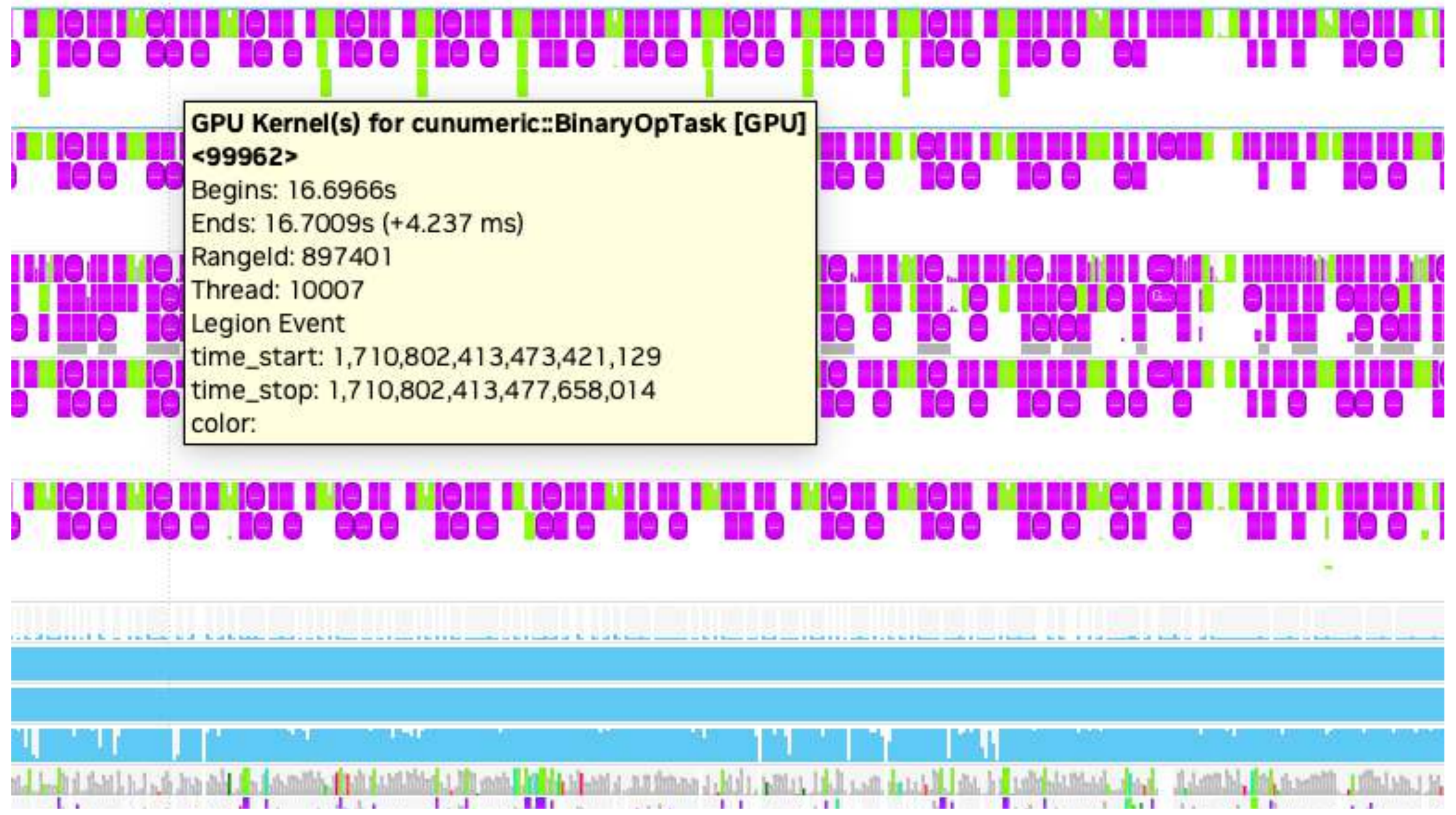
Nsight systems integration for Legate applications

- Performance debugging often requires a cross-cutting investigation across multiple abstraction layers
  - For Legate applications, we need both Legate-level (logical) views and CUDA-level (physical) views to get the complete picture
- Nsight systems supports rendering of profiling data from both sides, providing a holistic view to the execution trace



Logical view
Legate tasks, Legate stores, runtime activities

Physical view
GPU (kernels), CPU, OS, network



Semantic information from Legate applications are used to match kernels with originating tasks

Join Us!

- Try Legate and cuPyNumeric today:

```
conda install -c conda-forge -c legate cupynumeric
```

- For any questions or comments, please contact the Legate team on legate@nvidia.com or the discussion page on GitHub

- Resources
  - Legate – Get Started
  - NVIDIA Legate Core official documentation
  - Legate presentation at GTC’24
  - Effortlessly Scale NumPy from Laptops to Supercomputers with NVIDIA cuPyNumeric

NVIDIA