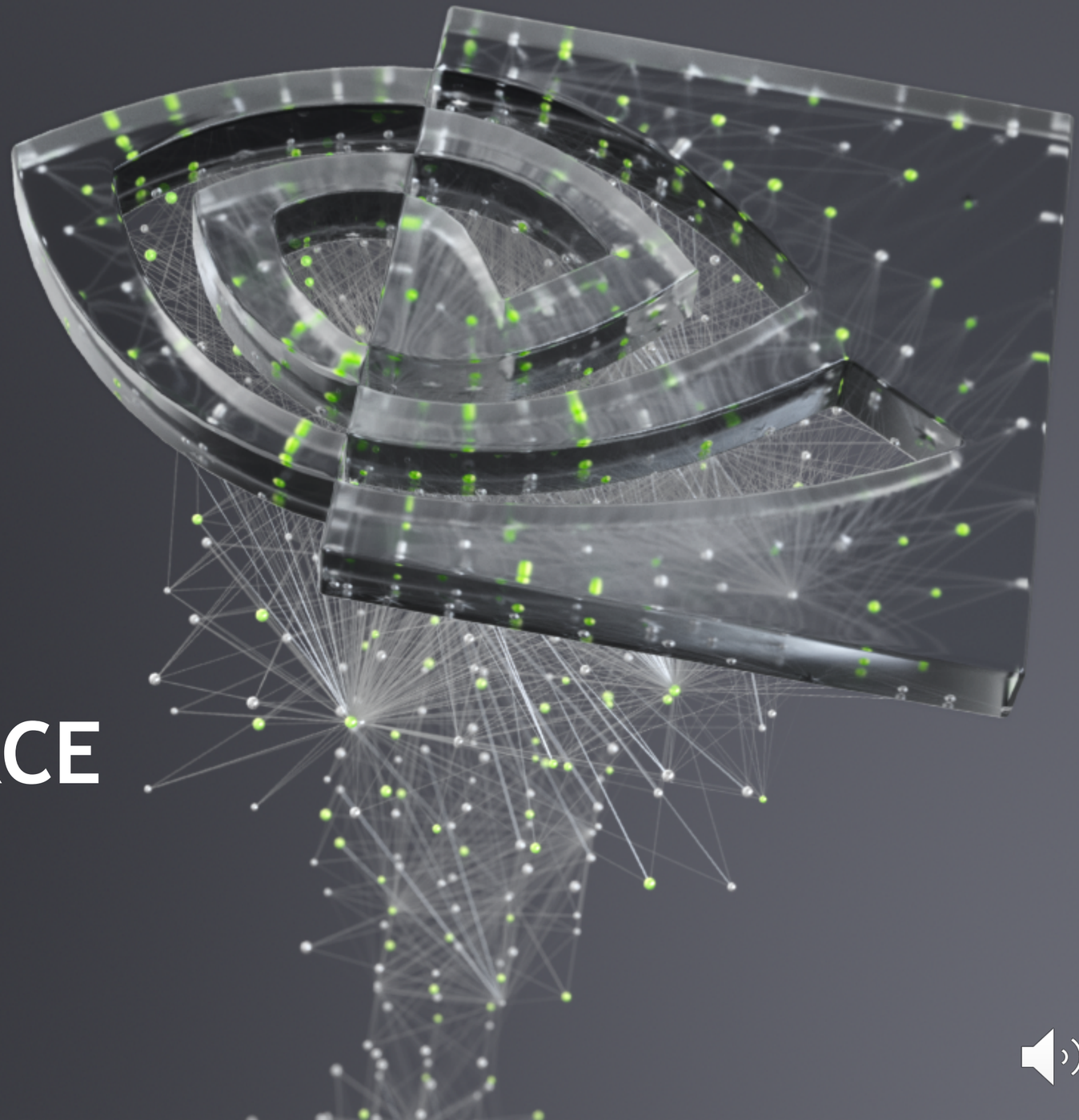


RAPIDS: OPEN-SOURCE GPU DATA SCIENCE

Keith Kraus

WHAT IS RAPIDS?

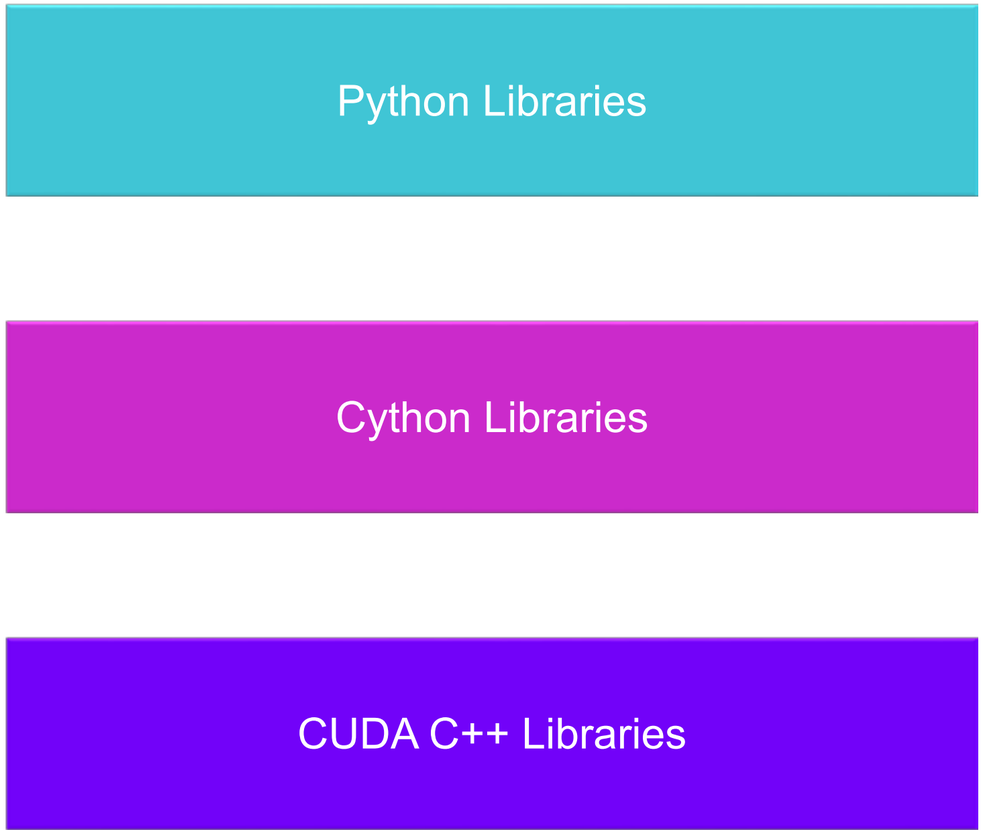
RAPIDS

A collection of open-source libraries for GPU-acceleration

Each library has a full-featured CUDA C++ library which contains the guts of the implementations as well as a higher-level modern C++ API.

On top of the C++ libraries, there's thin Cython libraries which contain wrappers around the C++ APIs to make them callable via Python with close to zero performance overhead.

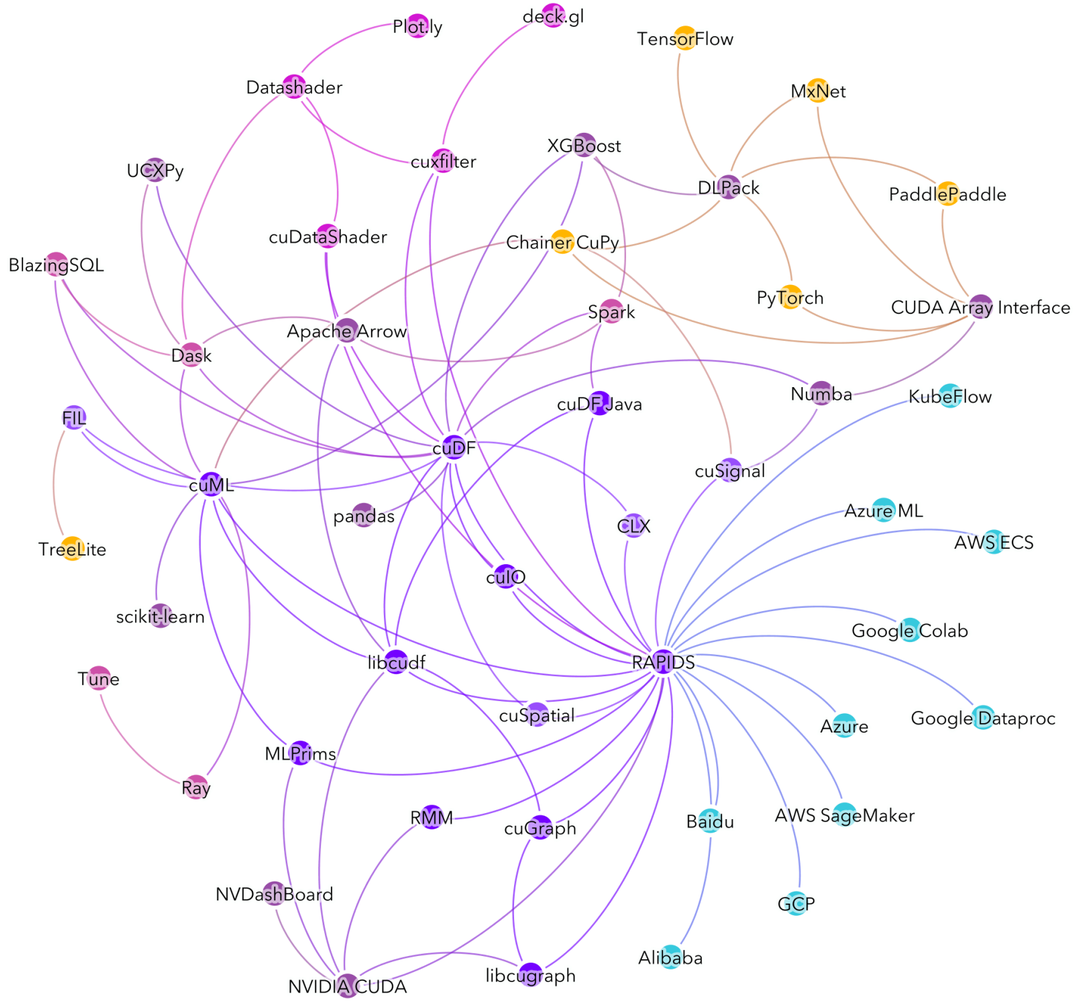
Finally, there's Python libraries built on top of the Cython libraries which mirror familiar PyData APIs to make building on top of and integrating the libraries as seamless as possible.



RAPIDS EVERYWHERE

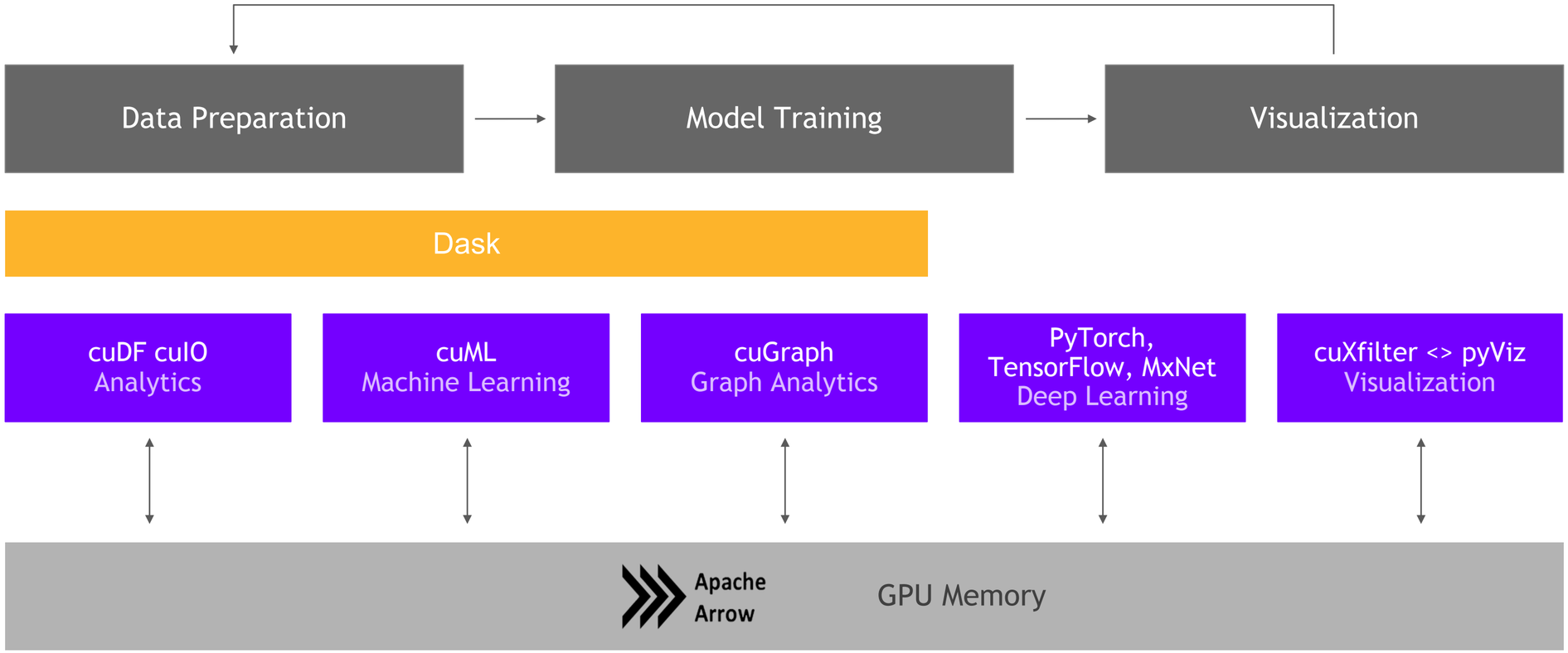
Our goal is to make RAPIDS as usable and performant as possible **wherever data science is done.**

We will continue to work with more open source projects to further democratize acceleration and efficiency in data science.



RAPIDS

Accelerating GPU Data Science End-to-End



RAPIDS BENCHMARKING: TPCX-BB

WHAT IS TPCX-BB®?

Comparing Big Data Platforms since the Cambrian Explosion of Big Data

TPC is the leader in benchmarking Data Analytics and Data Science Systems

TPCx-BB benchmark measures the performance of both hardware and software components by executing 30 frequently performed analytical queries in the context of retailers with physical and online store presence

Is the only TPC benchmark that starts from disk, does ETL (structured, semi-structured, and unstructured), and machine learning

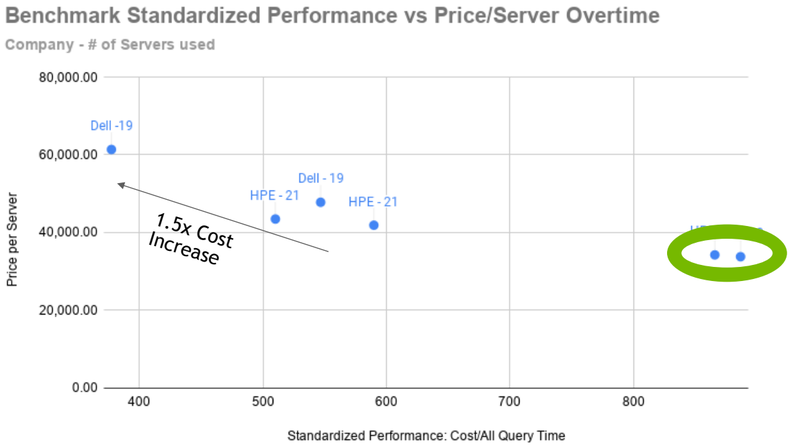
TPCX-BB

CPU Performance

Hadoop Processing, Reading from Disk

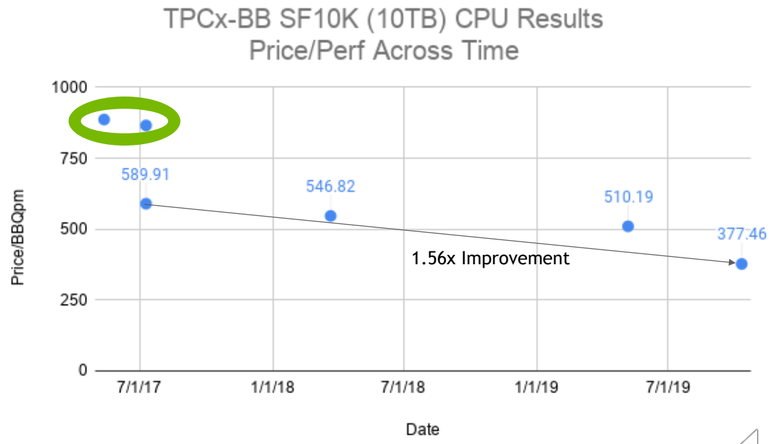
Spark In-Memory Processing

25-100x Improvement
Less code
Language flexible
Primarily In-Memory



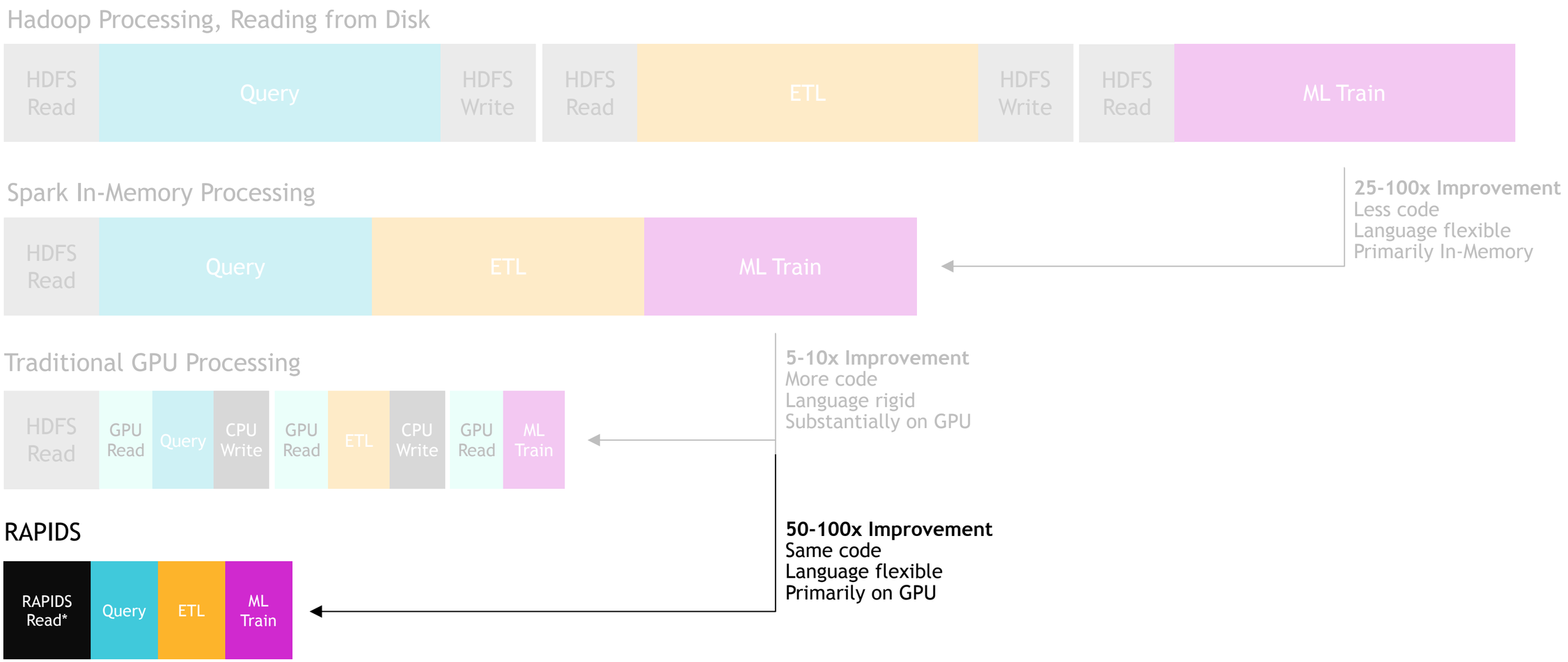
Current Leader, Dell: 19 servers @ $61K/server

Only ~1.5x speedup in last 2 years, driven primarily by scale up as opposed to scale out



TPCX-BB

GPU Performance



RAPIDS RUNNING TPCX-BB AT 1 TB AND 10 TB SFS

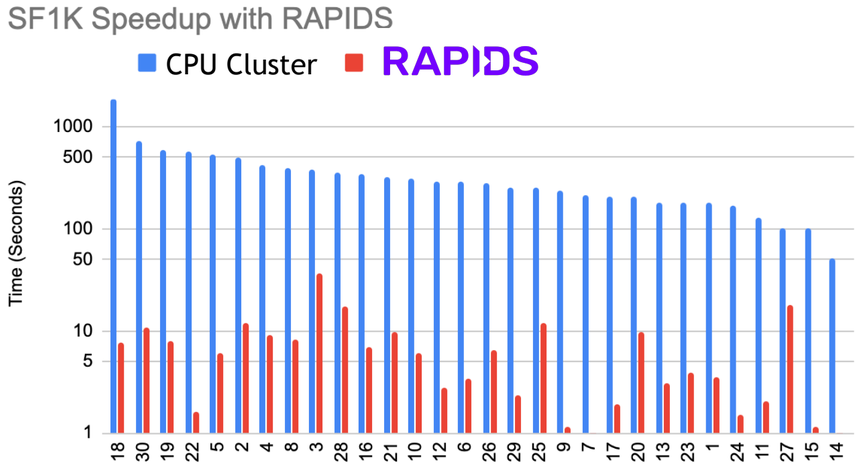
Up to 350x faster queries; Hours to Seconds!

Like other TPC benchmarks, TPCx-BB can be run at multiple “Scale Factors”:

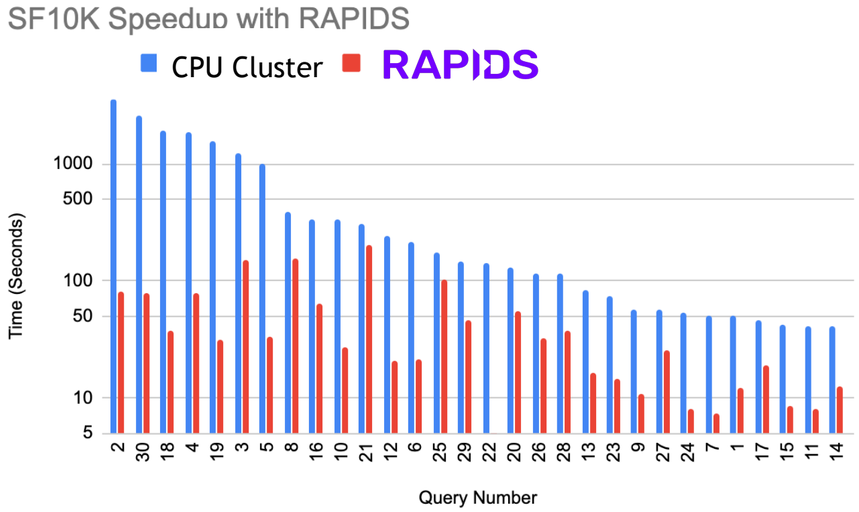
- SF1 - 1GB
- SF1K - 1 TB
- SF10K - 10 TB

We’ve been benchmarking RAPIDS implementations of the TPCx-BB queries at the SF1K (Single DGX-2) & SF10K (17x DGX-1) scales

Our results indicate that GPUs provide dramatic cost and time-savings for small scale *and* large-scale data analytics problems



Avg: 51x speed-up
>40x Normalized for Cost



Avg: 12x speed-up
>6x Normalized for Cost

https://github.com/rapidsai/tpcx-bb

QUERY SPOTLIGHT - UDFS AT SCALE ON GPUS

Query 3: What is viewed before a purchase?

Repartition web-clickstream table on user key

- Ensure all web activity records available within a single “chunk” (partition) of records that fit within the memory space of a single worker

Compute aggregate metrics on user’s sessions

Sort on event timestamp within user sessions

Run a user-defined-function with custom processing logic for classifying session behavior

- DataFrame APIs are great, but real business logic is complex, needing to support custom code
- RAPIDS uses Numba to compile simple Python expressions into GPU accelerated logic
- Run Python on GPUs!

```
@cuda.jit
def find_items_viewed_before_purchase_kernel(
    relevant_idx_col, user_col, timestamp_col, item_col, out_col, N
):
    """
    Find the past N items viewed after a relevant purchase was made,
    as defined by the configuration of this query.
    """
    i = cuda.grid(1)
    relevant_item = q03_purchased_item_IN

    if i < (relevant_idx_col.size):  # boundary guard
        # every relevant row gets N rows in the output, so we need to map the indexes
        # back into their position in the original array
        orig_idx = relevant_idx_col[i]
        current_user = user_col[orig_idx]

        # look at the previous N clicks (assume sorted descending)
        rows_to_check = N
        remaining_rows = user_col.size - orig_idx

        if remaining_rows <= rows_to_check:
            rows_to_check = remaining_rows - 1

        for k in range(1, rows_to_check + 1):
            if current_user != user_col[orig_idx + k]:
                out_col[i * N + k - 1] = 0

            # only checking relevant purchases via the relevant_idx_col
            elif (timestamp_col[orig_idx + k] <= timestamp_col[orig_idx]) & (
                timestamp_col[orig_idx + k]
                >= (timestamp_col[orig_idx] - q03_days_in_sec_before_purchase)
            ):
                out_col[i * N + k - 1] = item_col[orig_idx + k]
            else:
```

QUERY SPOTLIGHT - NATURAL LANGUAGE PROCESSING

Query 18 - are bad reviews correlated with bad sales?

Subset the data to a set of four months
- After joining tables containing store, store sales, data, and customer review data, split by row groups for better parallelism

For each store, regress date on the sum of net sales and retain the beta coefficient and select those stores with a negative slope
- Repartition this table to be one partition (it is small: only 192 rows at SF1000)

Make a list of all the unique store names
- RAPIDS has an extensive set of string functions, bringing string manipulation to the GPU

Find reviews that include any of the store names

For reviews that contain a store's name, return sentences containing a negative word and the negative word itself
- Break reviews into sentences
- Search sentences for words contained in a text file of negative words
- Return the store name, date of the review, sentence, and word for sentences where negative words appeared.
- NLP on GPU!

```
no_nulls["pr_review_content"] = no_nulls.pr_review_content.str.replace_multi(
    [". ", "? ", "! "], EOL_CHAR, regex=False
)
sentences = no_nulls.map_partitions(create_sentences_from_reviews)

# need the global position in the sentence tokenized df
sentences["x"] = 1
sentences["sentence_tokenized_global_pos"] = sentences.x.cumsum()
del sentences["x"]

# This file comes from the official TPCx-BB kit
# We extracted it from bigbenchqueriesmr.jar
with open("negativeSentiment.txt") as fh:
    negativeSentiment = list(map(str.strip, fh.readlines()))
    # dedupe for one extra record in the source file
    negativeSentiment = list(set(negativeSentiment))

word_df = sentences.map_partitions(
    create_words_from_sentences,
    global_position_column="sentence_tokenized_global_pos",
)
sent_df = cudf.DataFrame({"word": negativeSentiment})
sent_df["sentiment"] = "NEG"
sent_df = dask_cudf.from_cudf(sent_df, npartitions=1)

word_sentence_sentiment = word_df.merge(sent_df, how="inner", on="word")

word_sentence_sentiment["sentence_idx_global_pos"] = word_sentence_sentiment[
    "sentence_idx_global_pos"
].astype("int64")
sentences["sentence_tokenized_global_pos"] = sentences[
    "sentence_tokenized_global_pos"
].astype("int64")
```

NEXT STEPS

The Road to 1.0

GPU Direct Storage

- Currently, reading data from disk requires a mem-mapped read on the host and copy from host memory into GPU memory
- This adds latency and increases host memory requirements
- The upcoming CUDA cuFile API enables a direct to GPU memory data path between both local and remote NVME storage systems, which greatly increases throughput for IO heavy jobs

UCX InfiniBand support

- UCX and ucx-py have been integrated with Dask, allowing:
  - Utilization of high-speed networking hardware like NVLink and InfiniBand
  - Use of heuristics to determine fastest path networking from one worker to another
  - Even in the absence of accelerated networking hardware, Dask can now use lower level networking primitives instead of slower Python native TCP sockets
  - Current TPCx-BB results use UCX with NVLink, but not yet InfiniBand with RDMA; ongoing work in ucx-py will unlock node to node IB RDMA

Dask scheduler improvements

- RAPIDS team continues to contribute back to the Dask open-source project, and Dask will continue to be key to using RAPIDS at scale. As Dask improves, we expect our numbers to improve as well


UCX-PY

SCALE-OUT PROBLEMS

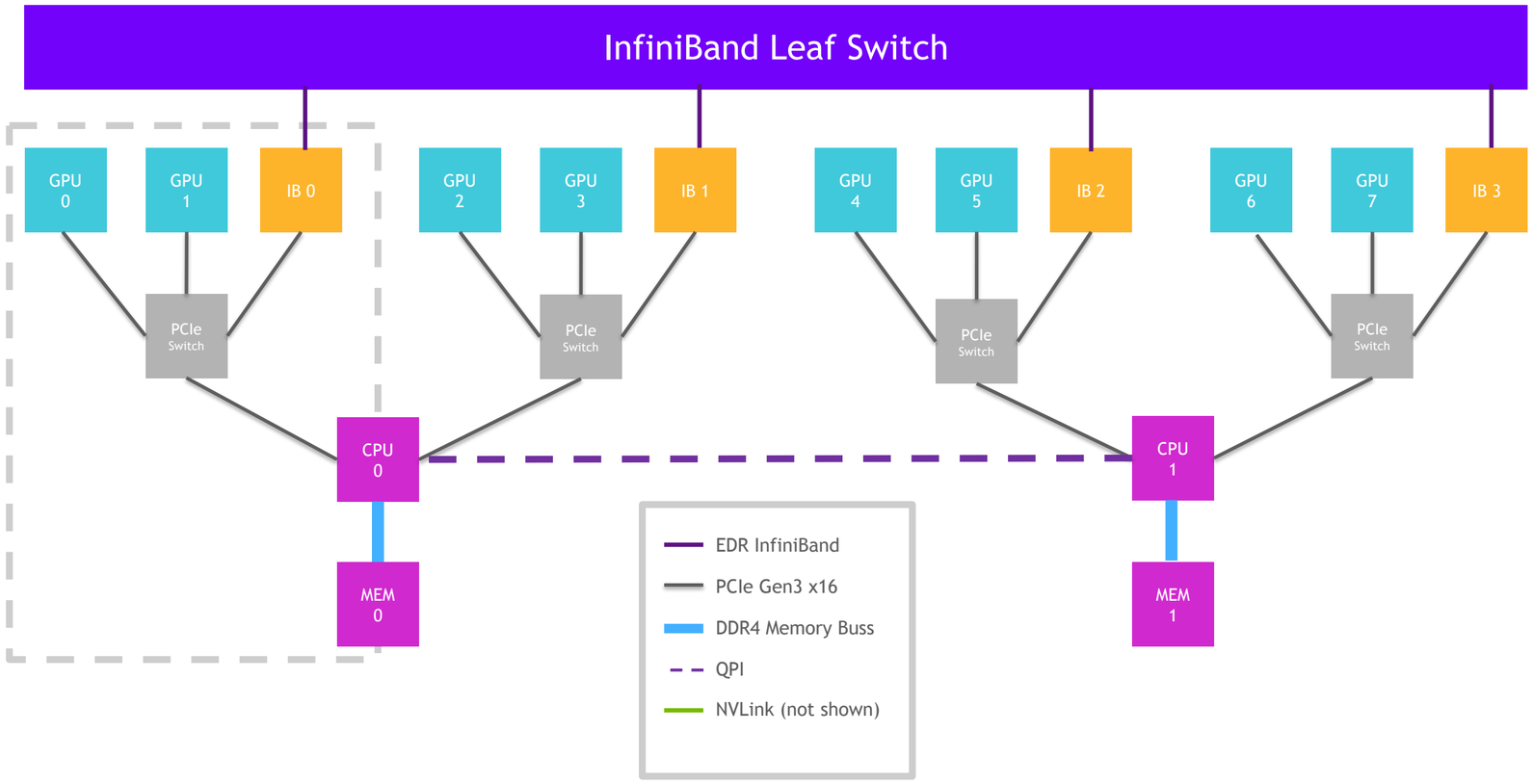
Scale-out bottlenecks occur at the networking layer

We need to efficiently pass data and minimize HtoD/DtoH transfers

Within a node, what is the best path to pass data between two GPUs?

Across multiple nodes, what is the best path to pass data between two GPUs?

Available Comms: TCP, NVLink, InfiniBand

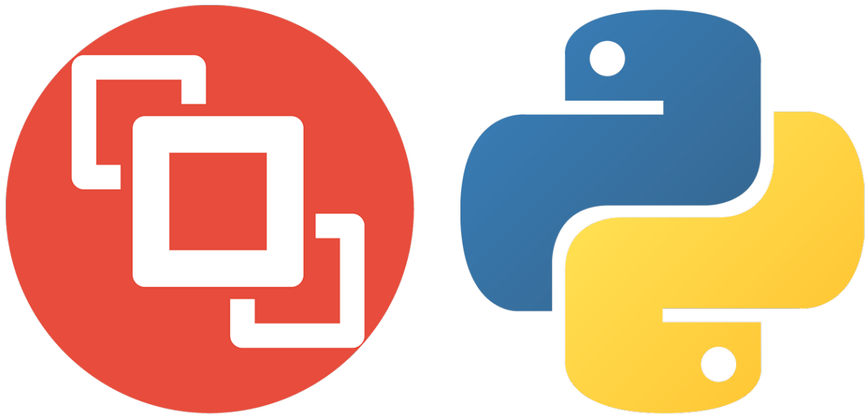


SCALE-OUT SOLUTION

OpenUCX is a low-latency high-bandwidth library with support for traditional and accelerated networking hardware

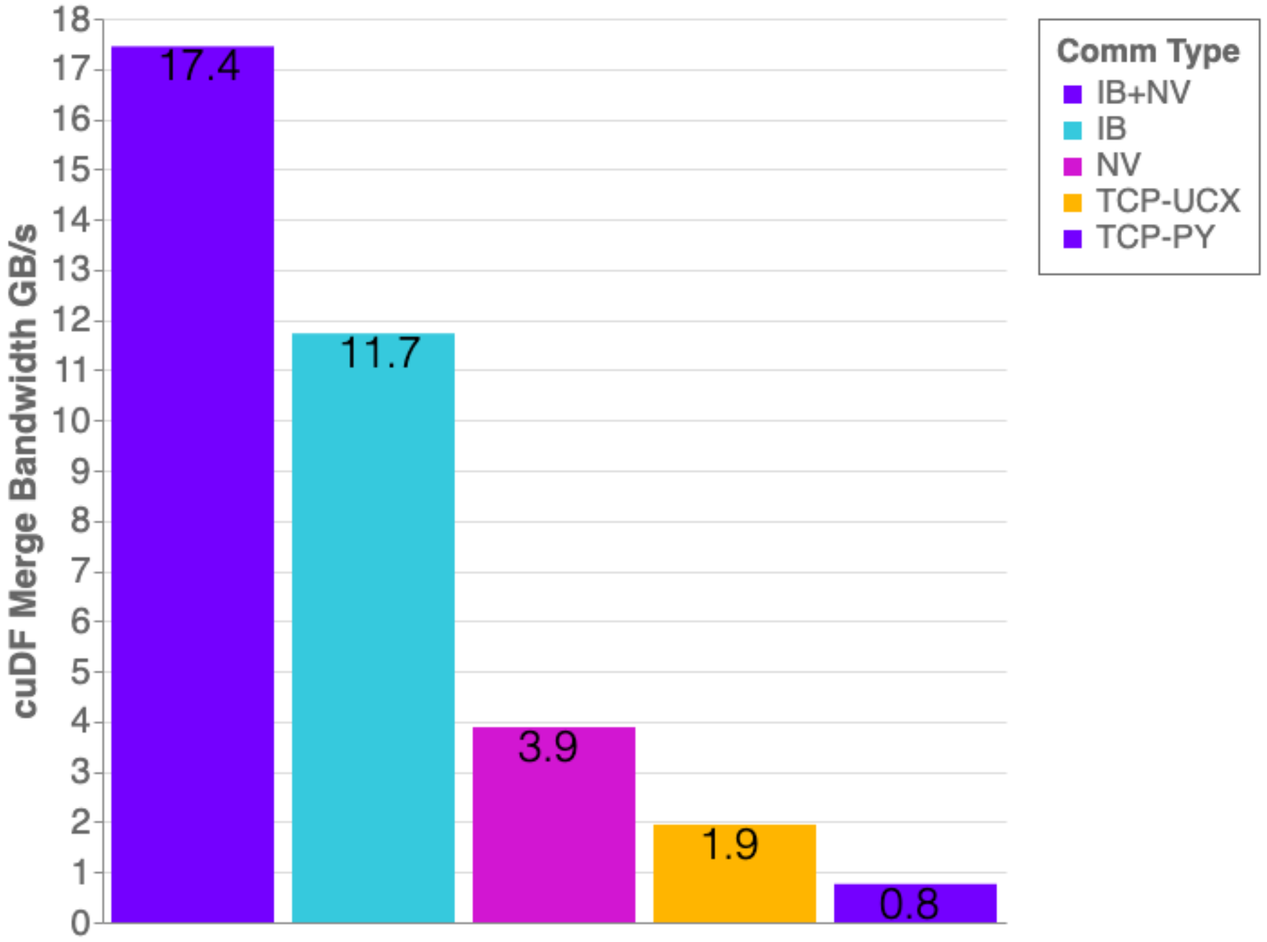
The recently created UCX-Py library allows Dask to expose accelerated networking to Distributed PyData

Adding Python layers has minimal impact on performance



1GB Messages	Theoretical	UCX-Py
NVLink	50 GB/s	46.5 GB/s
InfiniBand (Mellanox)	12.5GB/s	11.2 GB/s

MERGE BENCHMARK



Distributed Join/Merge with:

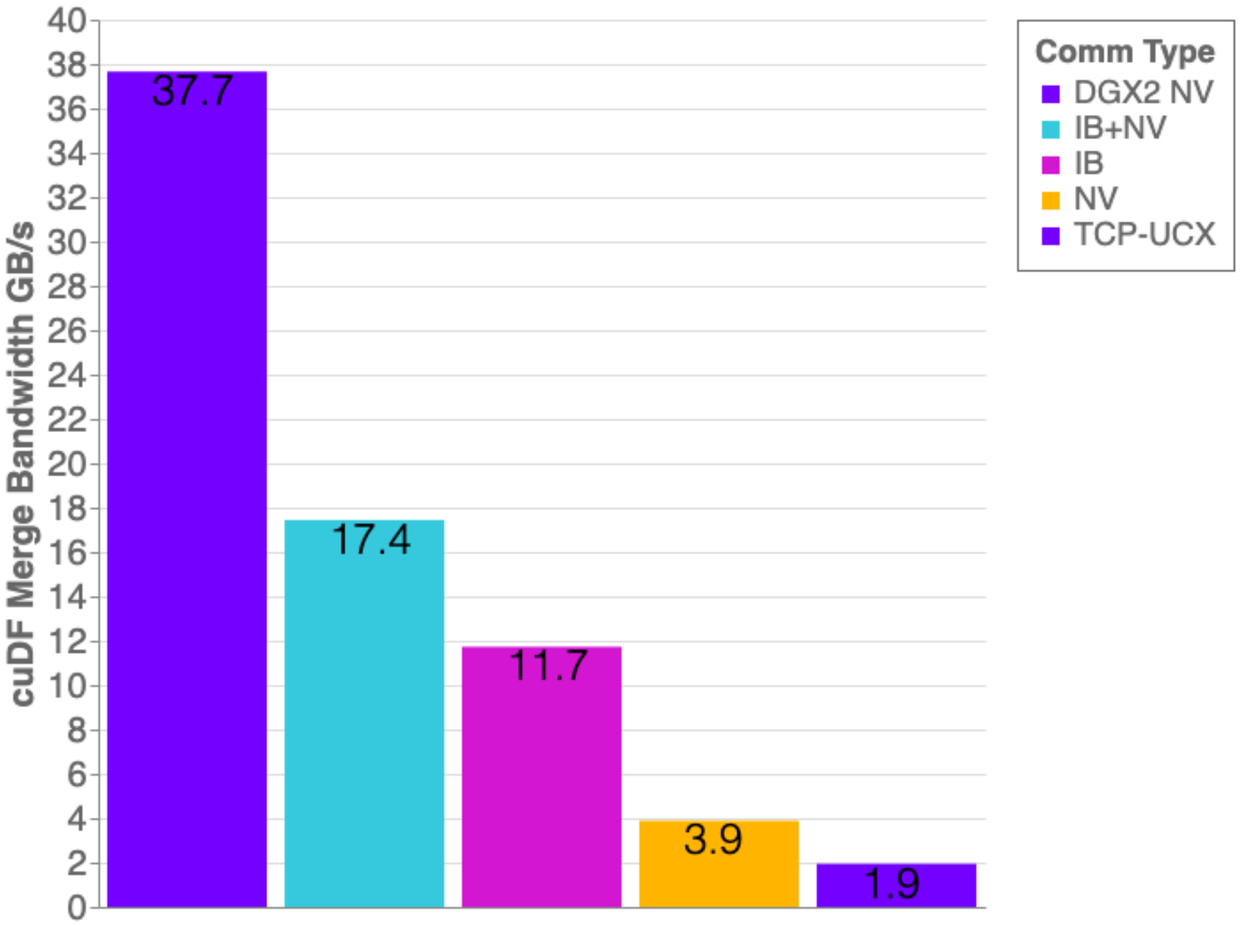
Two DataFrames of Random Data

2 Columns (Int64) per DataFrame

Balanced Partitions (30% Matching)

A DGX-1 (8GPUs Per Node) and Varied Networking Hardware

CHANGE THE NETWORK



Reduce Complexity and Increase Bandwidth with fully connected GPUs over NVLink

Use first 8 GPUs on DGX2 (fully connected GPUs with NVSwitch) for significantly better performance

DGX A100 has 2X bandwidth of a DGX2


CUDF

LIBCUDF CUDA/C++ LIBRARY

Table (dataframe) and column types and algorithms

CUDA kernels for sorting, join, groupby, reductions, partitioning, elementwise operations, etc.

Optimized GPU implementations for strings, timestamps, numeric types (more coming)

Primitives for scalable distributed ETL

```
std::unique_ptr<table>
gather(table_view const& input,
       column_view const& gather_map, …)
{
   // return a new table containing
   // rows from input indexed by
   // gather_map
}
```

CUDF PYTHON LIBRARY

In [2]:
```
#Read in the data. Notice how it decompresses as it reads the data into memory.
gdf = cudf.read_csv('/rapids/Data/black-friday.zip')
```

In [3]:
```
#Taking a look at the data. We use "to_pandas()" to get the pretty printing.
gdf.head().to_pandas()
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	F	0-17	10	A	2	0	3
1	1000001	P00248942	F	0-17	10	A	2	0	1
2	1000001	P00087842	F	0-17	10	A	2	0	12
3	1000001	P00085442	F	0-17	10	A	2	0	12
4	1000002	P00285442	M	55+	16	C	4+	0	8

In [6]:
```
#grabbing the first character of the years in city string to get rid of plus sign, and converting
 to int
gdf['city_years'] = gdf.Stay_In_Current_City_Years.str.get(0).stoi()
```

In [7]:
```
#Here we can see how we can control what the value of our dummies with the replace method and turn
strings to ints
gdf['City_Category'] = gdf.City_Category.str.replace('A', '1')
gdf['City_Category'] = gdf.City_Category.str.replace('B', '2')
gdf['City_Category'] = gdf.City_Category.str.replace('C', '3')
gdf['City_Category'] = gdf['City_Category'].str.stoi()
```

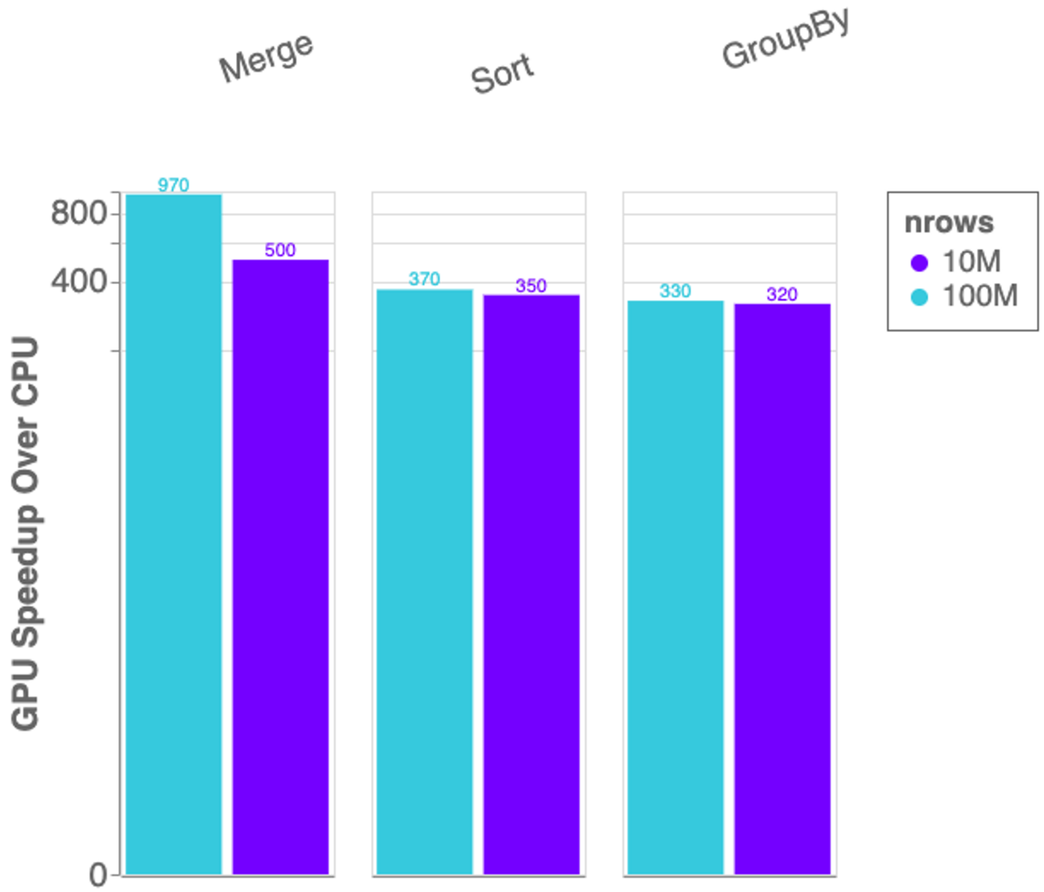
A Python library for manipulating GPU DataFrames following the Pandas API

Python interface to CUDA C++ library with additional functionality

Creating GPU DataFrames from Numpy arrays, Pandas DataFrames, PyArrow Tables, generic Python objects, and more

JIT compilation of User-Defined Functions (UDFs) using Numba

BENCHMARKS: SINGLE-GPU SPEEDUP VS. PANDAS



cuDF v0.13, Pandas 0.25.3

Running on NVIDIA DGX-1:

GPU: NVIDIA Tesla V100 32GB
CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz

Benchmark Setup:

RMM Pool Allocator Enabled

DataFrames: 2x int32 columns key columns, 3x int32 value columns

Merge: inner

GroupBy: count, sum, min, max calculated for each value column

CUDF STRING SUPPORT

Current v0.13 String Support

Regular Expressions

Element-wise operations

Split, Find, Extract, Cat, Typecasting, etc.

String GroupBys, Joins, Sorting, etc.

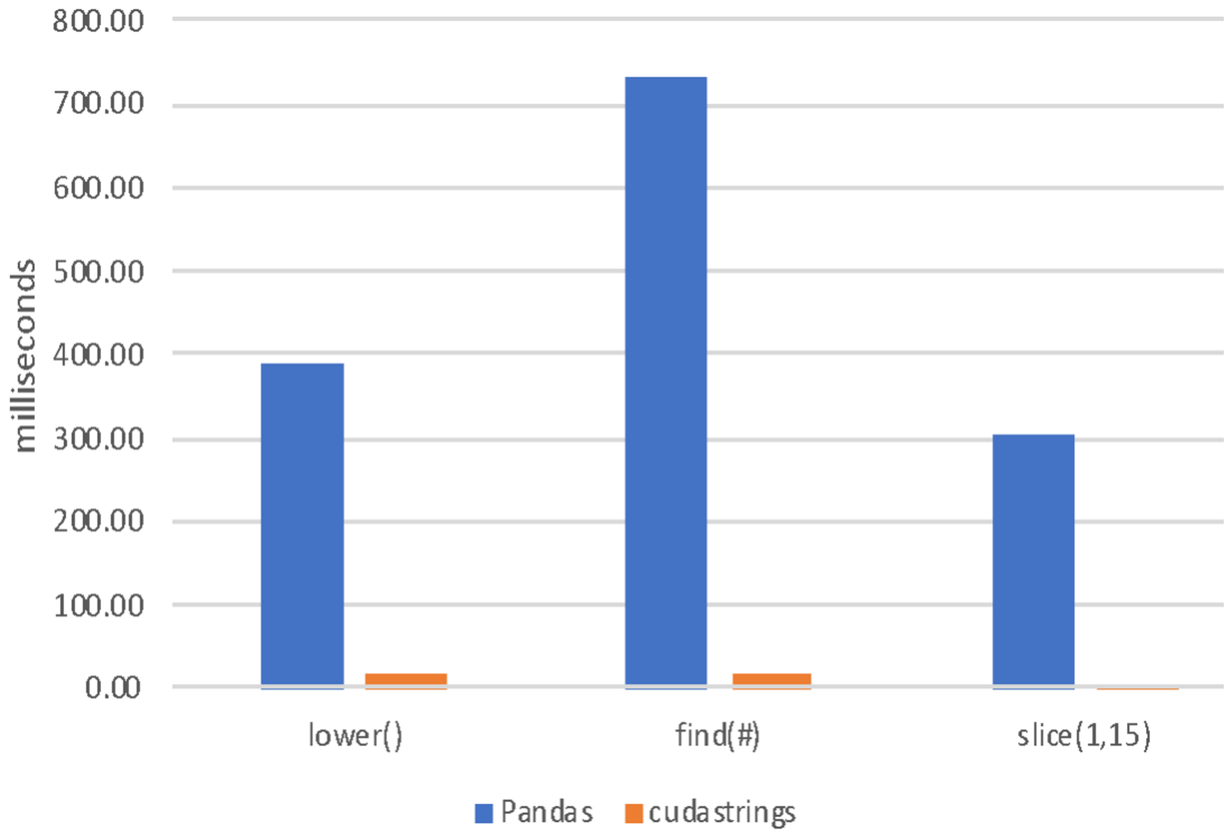
Categorical columns fully on GPU

Native String type in libcudf C++

Future v0.14+ String Support

Further performance optimization

JIT-compiled String UDFs



CUDF I/O FOR FASTER DATA LOADING

Full CUDA/C++ implementations with Python APIs that follow the Pandas API
Provide >10x speedup

CSV Reader - v0.2, CSV Writer v0.8
Parquet Reader – v0.7, Parquet Writer v0.12
ORC Reader – v0.7, ORC Writer v0.10
JSON Reader - v0.8
Avro Reader - v0.9
Kafka Source / Sink – v0.15

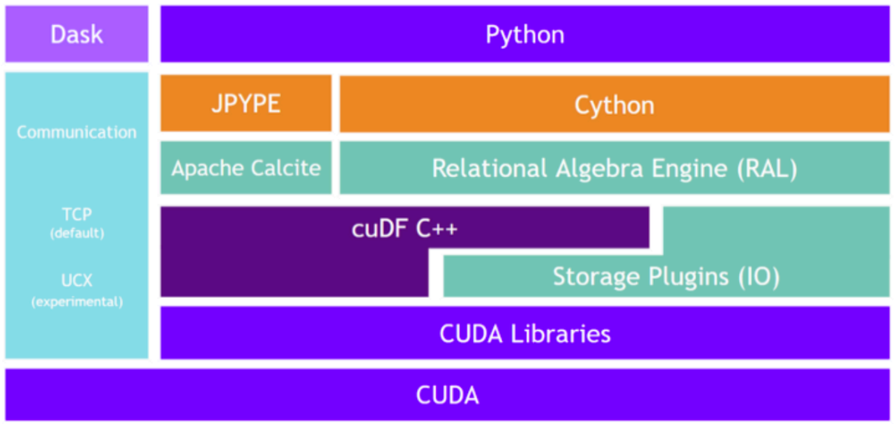
GPU Direct Storage integration in progress for bypassing PCIe bottlenecks!

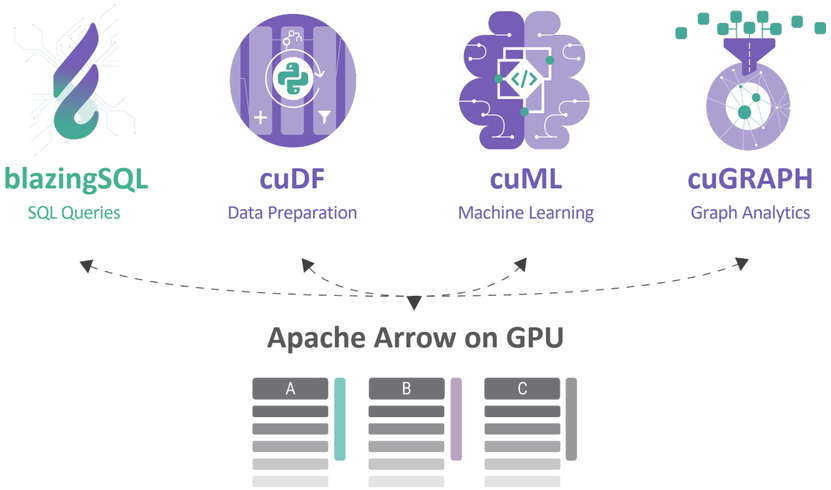
Key is GPU-accelerating both parsing and decompression wherever possible

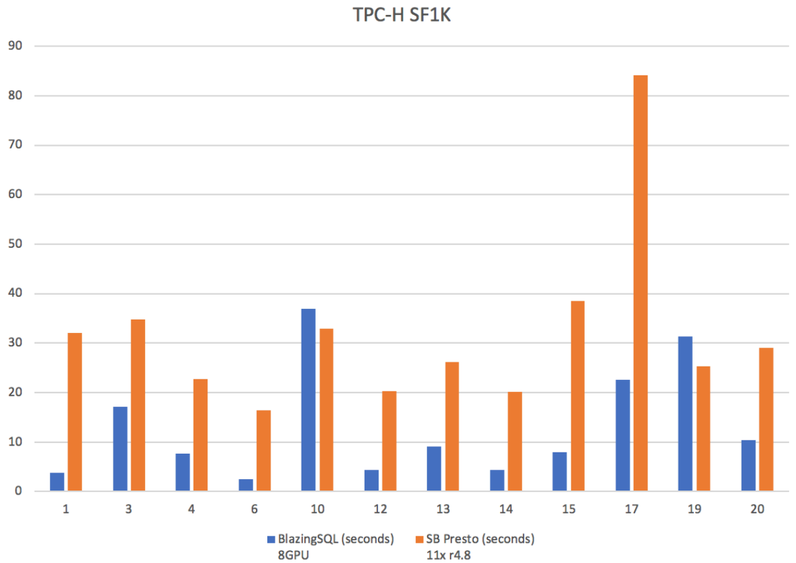
```
1]: import pandas, cudf

2]: %time len(pandas.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))

CPU times: user 25.9 s, sys: 3.26 s, total: 29.2 s
Wall time: 29.2 s
2]: 12748986

3]: %time len(cudf.read_csv('data/nyc/yellow_tripdata_2015-01.csv'))

CPU times: user 1.59 s, sys: 372 ms, total: 1.96 s
Wall time: 2.12 s
3]: 12748986

4]: !du -hs data/nyc/yellow_tripdata_2015-01.csv

1.9G    data/nyc/yellow_tripdata_2015-01.csv
```

BLAZINGSQL

SQL Open Source Engine built on RAPIDS







Contributes directly to libcudf and built fully on RAPIDS

Seamless integration with all of RAPIDS

2.4x Faster, 6.8x normalized for cost

https://blazingsql.com

COMING SOON IN CUDF 0.14+

Huge API documentation improvements

Legacy libcudf removal and NVStrings deprecation

CUDA 11.0 and Ampere support

Expanded data types:
- Unsigned integer
- Timedelta
- Dictionary (CUDA C++)
- Fixed-Point Decimal
- Nested: lists / structs

Optimized I/O for handling groups of small files and partitioned datasets

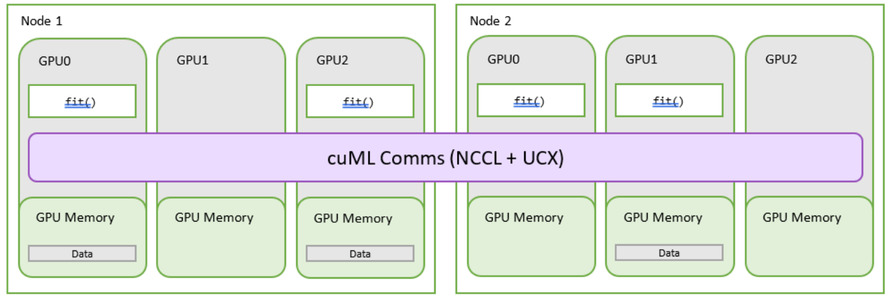
Support for CUDA streams and optimized asynchronous execution in libcudf

And more!

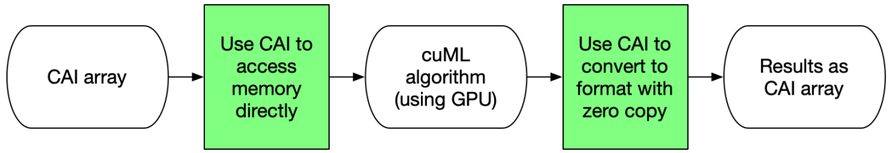

CUML

CUML – MAJOR THEMES

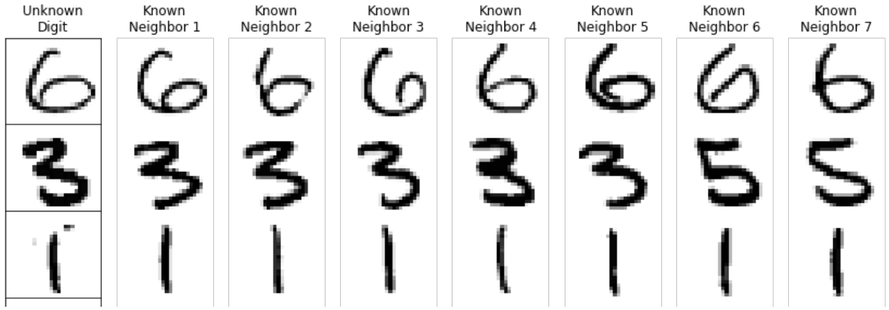
cuML went MNMG
"Multi-node, Multi-GPU" – With deep Dask integration and a high-performance comms backend built on NCCL and UCX, cuML can support huge data volumes on multiple GPUs.



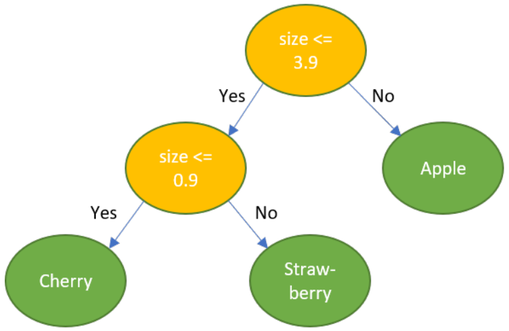
Interop and Integration
Support cuDF inputs? Sure. How about NumPy, cuPy, Pandas, and Numba too? No problem. With a new input/output module, cuML provides a customizable API to consume and output just about any array format.



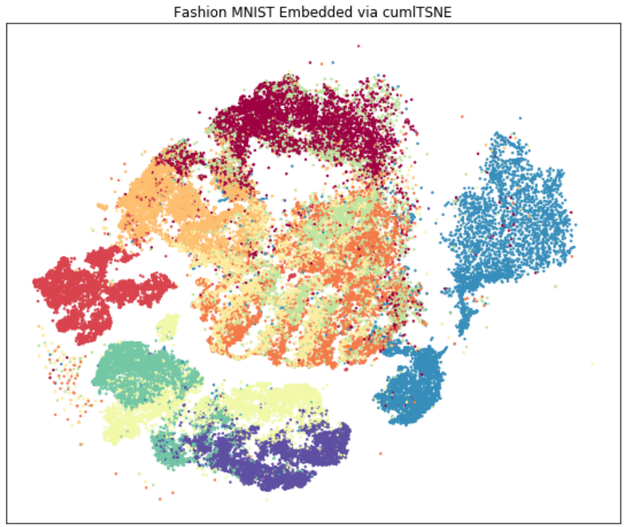
More Algorithms, and Not Just Algorithms
cuML added 10 major new algorithms and many smaller wrappers or variants. Support for preprocessing and utilities continues to grow and will improve further in 0.15.



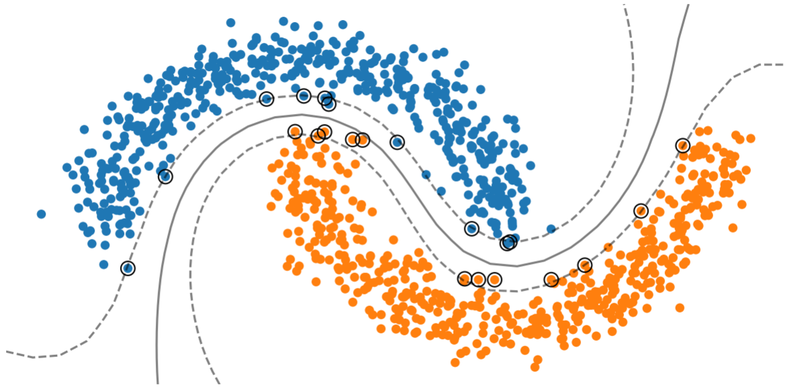
NEW ALGORITHMS IN CUML



Random Forests
(up to 45x faster than CPU)
+
Forest Inference
(35x faster than CPU)



T-SNE
(up to 200x faster than CPU)



Support Vector Machines
(up to 500x faster than CPU)

… and many more! Logistic Regression, Holt-Winters, ARIMA, Naïve Bayes, Mini-batch SGD Regression and Classification, and Random Projections.

Plus support for preprocessing and utilities, like train_test_split, GPU-accelerated metrics, label encoding, and more.

XGBOOST + RAPIDS: BETTER TOGETHER

XGBoost now builds on the CUDA array interface standard to provide zero-copy data import from cuDF, cuPY, Numba, PyTorch and more

Official Dask API makes it easy to scale to multiple nodes or multiple GPUs

Memory usage when importing GPU data decreased by 2/3 or more

New objectives support Learning to Rank on GPU

All RAPIDS changes are integrated upstream and provided to all XGBoost users – no special packages needed

ROAD TO 1.0

cuML: March 2020 – RAPIDS 0.14

cuML	Single-GPU	Multi-Node Multi-GPU
Gradient Boosted Decision Trees (GBDT)	■	■
Linear Regression	■	■
Logistic Regression	■	
Random Forest	■	■
K-Means	■	■
K-NN	■	■
DBSCAN	■	
UMAP	■	
Holt-Winters	■	
ARIMA	■	
t-SNE	■	
Principal Components	■	■
Singular Value Decomposition	■	■
SVM	■	

RAY JOINS THE RAPIDS ECOSYSTEM

Ray Tune is a scalable HPO library that allows the optimization to be performed in a distributed manner. It provides various search algorithms along with smarter ways to schedule them in order to arrive at the optimal solution quickly and efficiently.

Tune is built on Ray, a system for easily scaling applications from a laptop to a cluster.

Training large datasets with XGBoost can take hours and sometimes days on CPUs, as they rely on several hyperparameters that need to be tuned.

By keeping the whole workflow on the GPU, processing times can be reduced by over 30x with RAPIDS and Ray Tune.

Blog - https://nvda.ws/2WCsqou

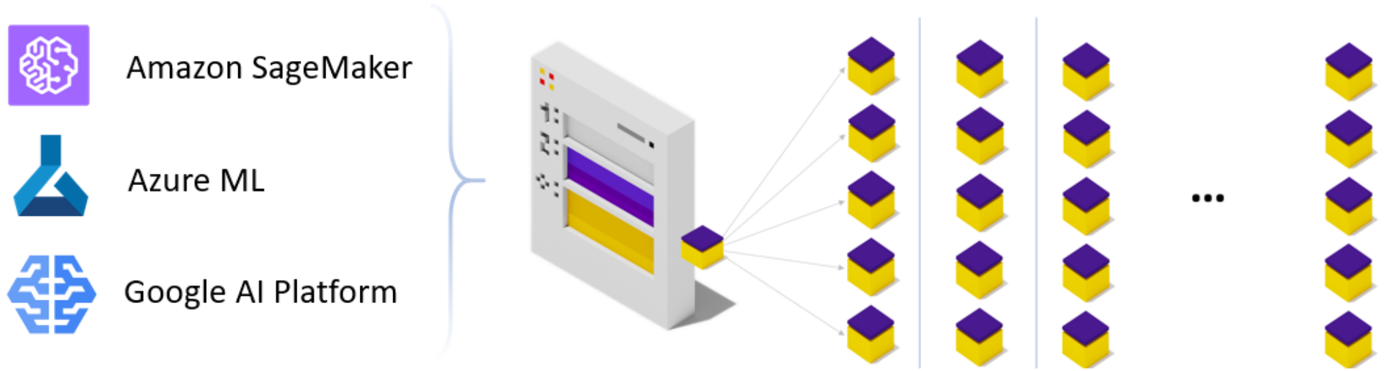
RAPIDS INTEGRATED INTO CLOUD ML FRAMEWORKS

Accelerated machine learning models in RAPIDS give you the flexibility to use hyperparameter optimization (HPO) experiments to explore all variants to find the most accurate possible model for your problem.

With GPU acceleration, RAPIDS models can train 40x faster than CPU equivalents, enabling more experimentation in less time.

The RAPIDS team works closely with major cloud providers and OSS solution providers to provide code samples to get started with HPO in minutes

https://rapids.ai/hpo



github.com/rapidsai/**cloud-ml-examples**

tune

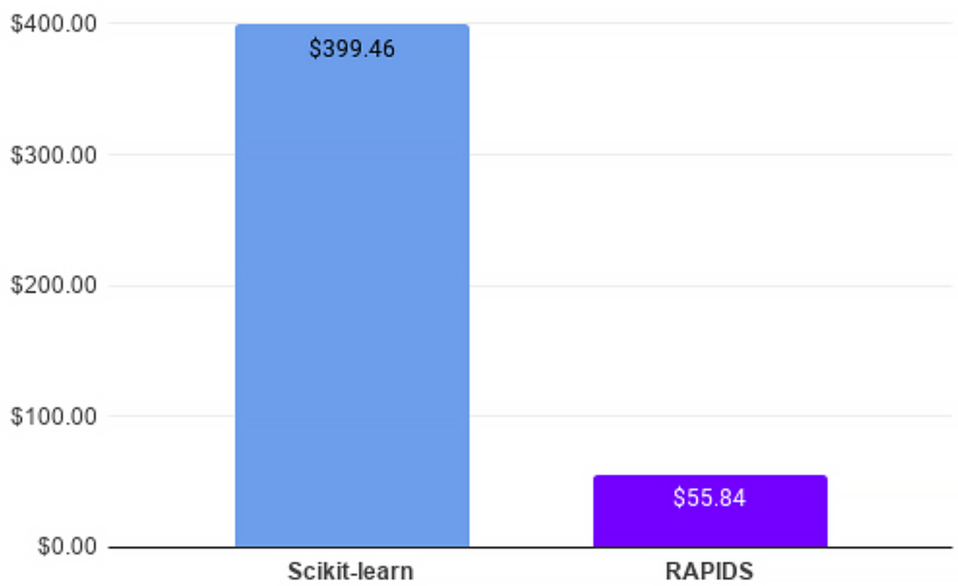
Ray/Tune

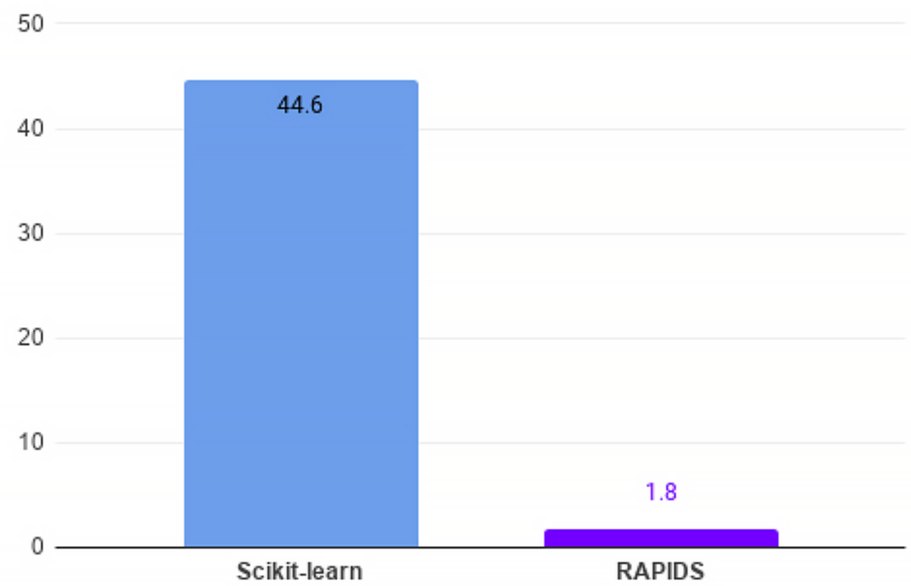
DASK

Dask ML

HPO USE CASE: 100-JOB RANDOM FOREST AIRLINE MODEL

Huge speedups translate into >7x TCO reduction





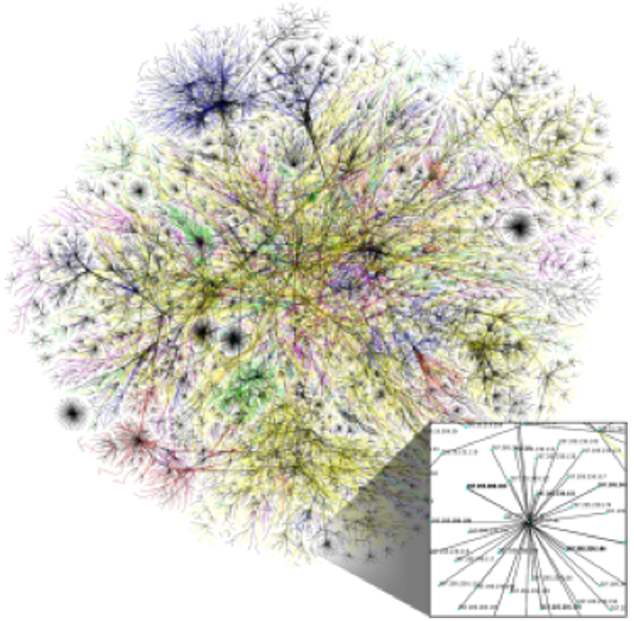
Based on sample Random Forest training code from cloud-ml-examples repository, running on Azure ML. 10 concurrent workers with 100 total runs, 100M rows, 5-fold cross-validation per run.

GPU nodes: 10x Standard_NC6s_v3, 1 V100 16G, vCPU 6 memory 112G, Xeon E5-2690 v4 (Broadwell) - $3.366/hour
CPU nodes: 10x Standard_DS5_v2, vCPU 16 memory 56G, Xeon E5-2673 v3 (Haswell) or v4 (Broadwell) - $1.017/hour"


CUGRAPH

CUGRAPH CURRENT FUNCTIONALITY

Community	Spectral Clustering - Balanced-Cut, Modularity Maximization Louvain Ensemble Clustering for Graphs Subgraph Extraction Kcore, K-Truss, and KCore Number Triangle Counting
Components	Weakly Connected Components Strongly Connected Components
Link Analysis	Page Rank (Multi-GPU) Personal Page Rank Katz
Link Prediction	Jaccard - Weighted and Unweighted Overlap Coefficient
Traversal	Single Source Shortest Path (SSSP) Breadth First Search (BFS)
Structure	COO-to-CSR (Multi-GPU) Transpose Graph and DiGraph Classes Renumbering and Un-renumbering



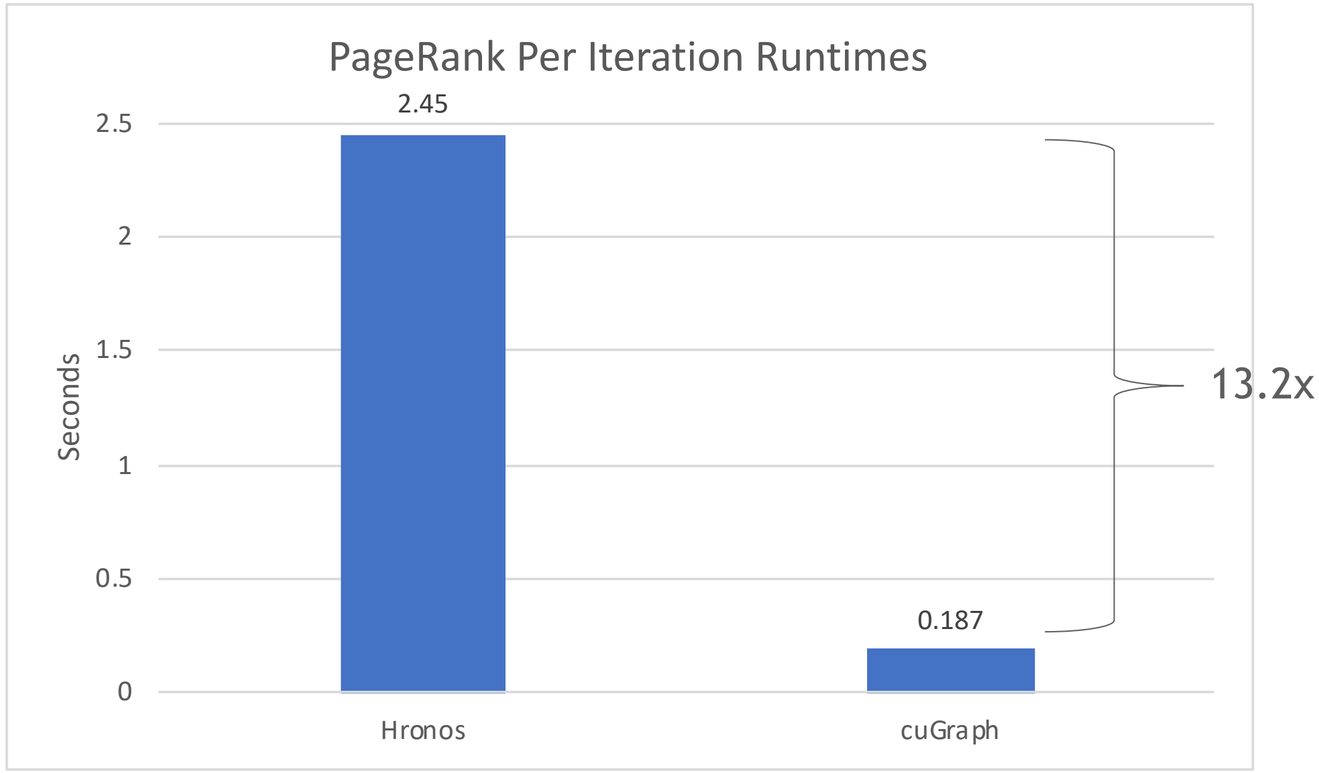
PAGERANK ON DGX A100 CLUSTER

Future Performance on 32x A100 GPUs vs Optimized CPU

Common Crawl 2012 Dataset	
Vertices	3,315,452,339
Edges	128,736,952,320
Per Iteration Time	0.187 seconds

Erdos-Renyi Synthetic Graph	
Size	4.7 TBs
Edges	256,000,000,000
Per Iteration Time	0.74 seconds

Comparing PageRank per iteration times against a recent paper by Stergious[1] using the Yahoo *Hronos* system (3,000 nodes[2])



1) Stergiou, S. (2020, April). Scaling PageRank to 100 Billion Pages. In Proceedings of The Web Conference 2020 (pp. 2761-2767).
2) Node configuration: 64GB RAM, 2x Intel Xeon E5-2620, 10Gbps Ethernet.

CUGRAPH ROADMAP

	0.15	0.16	0.17
Multi-GPU Multi-Node	PageRank Personal PageRank BFS	SSSP WCC SCC	Jaccard and Overlap Louvain K-Truss
Single GPU	Hungarian Edge Betweenness Centrality	Leiden Graph Edit Distance	Mutual Information Graph Coloring
Clean-up	Redo Louvain	Two-Hop Jaccard WCC	

STREAM PROCESSING WITH RAPIDS: CUSTREAMZ

GPU ACCELERATED STREAM PROCESSING WITH RAPIDS

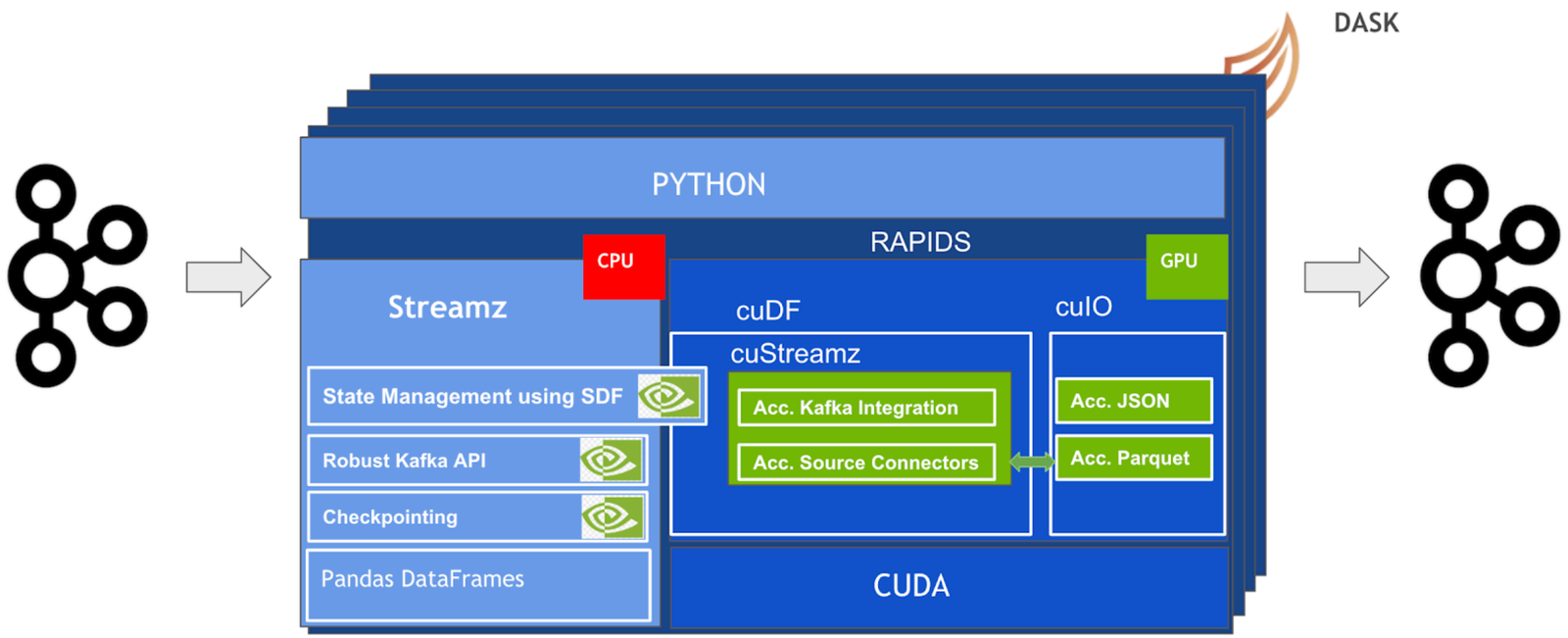
Structured telemetry events & unstructured logs growing at over 1000% year-over-year

Big Data streaming is becoming extremely complex to scale and operate at high efficiency

cuStreamz is a new RAPIDS library to accelerate stream processing on GPUs

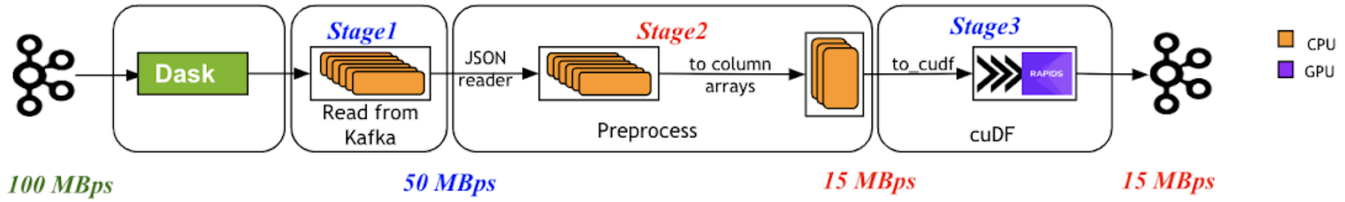
Leverages RAPIDS cuDF, Streamz, Kafka, Dask

40x stream processing performance increase



USED INTERNALLY AT NVIDIA

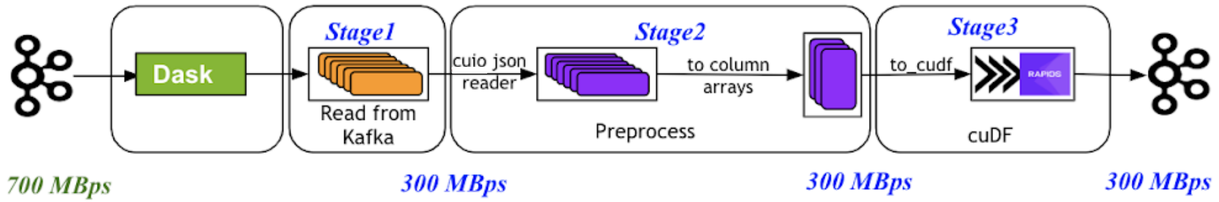
Phase I
GPU Util: 5%
g4dn.8xlarge
24 CPU Cores
($2.2/hr)



Kafka —> streamz Confluent Kafka —> **JSON.loads —> dataframe -> convert to cuDF** —> cuDF API() —> OUTPUT —> JSON —> Kafka

Phase 2

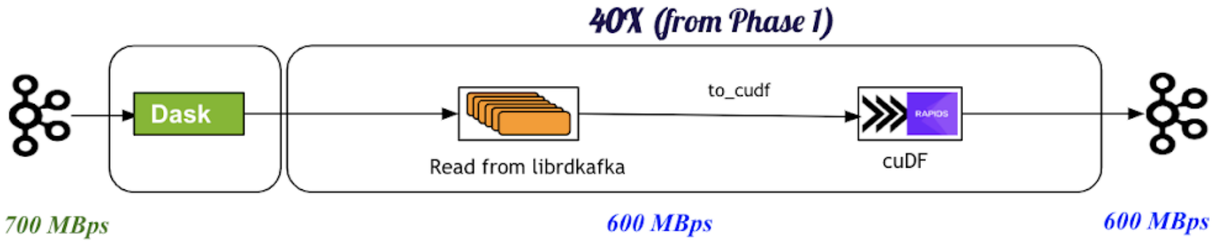
GPU Util: 95%
g4dn.8xlarge
24 CPU Cores
($2.2/hr)



Improvements
- cu-JSON Preprocessing on GPUs
- Snappy compression on Kafka
- Multiple Dask workers per GPU
- Con: Confluent Kafka GIL issue

Kafka —> streamz Confluent Kafka —> **GPU-acc cudf.read_json()** —> cuDF API() —> OUTPUT —> JSON —> Kafka

Phase 3
GPU Util: 95%
g4dn.4xlarge
8 CPU Cores
($1.2/hr)



Improvements
- Accelerated Kafka integration
- Better perf. with fewer CPU cores, processes, CUDA Context Memory

ACCELERATED KAFKA

First step for GPU streaming

- Foundation for feeding cudf with data in motion
- Provides the plumbing for consuming data from Kafka directly into cudf dataframes
    - Data is efficiently transferred from NIC to GPU memory with very little overhead
- 25X speedup vs using a Kafka consumer library like “python-confluent-kafka”
- Supports Kafka checkpointing for distributed consumption across multiple GPU(s)/Nodes(s)

```
import cudf

# Define the configurations for Kafka interaction
external_datasource_confs = {
    "metadata.broker.list": "localhost:9092",
    "group.id": "custreamz-test",
    "topic": "cudf_json_demo",
}

# Read JSON formatted messages from Kafka into a DF
# Messages read = last successful checkpoint to last message in topic
gdf = cudf.read_json(ex_ds_configs=external_datasource_confs)
```

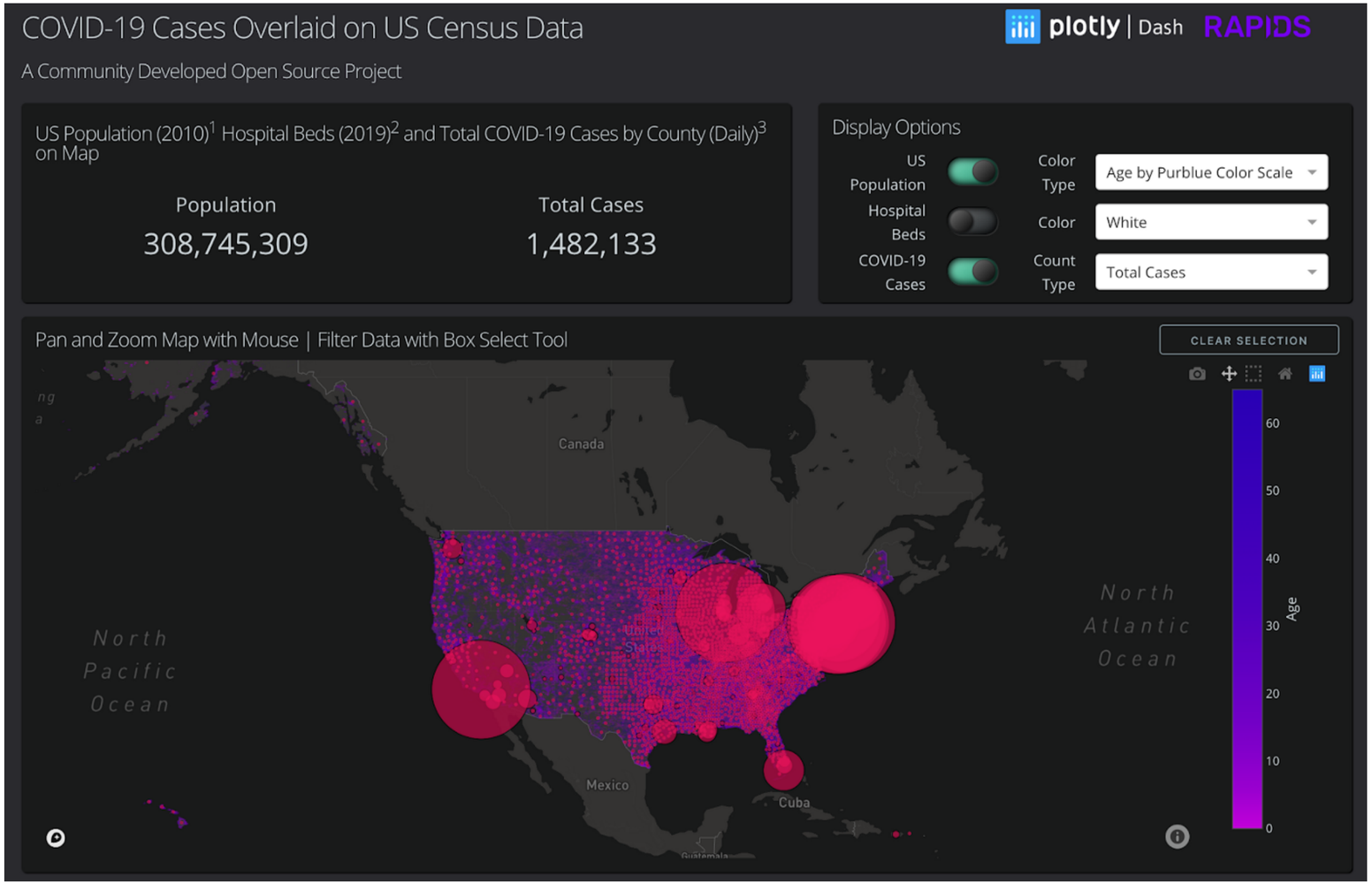
DATA VISUALIZATION

THE RAPIDS VIZ COMMUNITY

Plotly Dash
Integrating with Plotly's Dash -an open-source framework that enables developers to build interactive, data-rich analytical web applications in pure Python.
rapids.ai/plotly

Datashader
Native cuDF /dask_cuDF support and ongoing community development of more GPU accelerated features.
datashader.org

cuXfilter
Quickly create exploratory dashboards inline with your RAPIDS notebook workflow. Now with deck.gl charts and dask_cuDF support for 0.14.
github.com/rapidsai/cuxfilter



CYBER LOG ACCELERATOR

CLX

Cyber Log Accelerators

Built using RAPIDS and other GPU-accelerate compute platforms

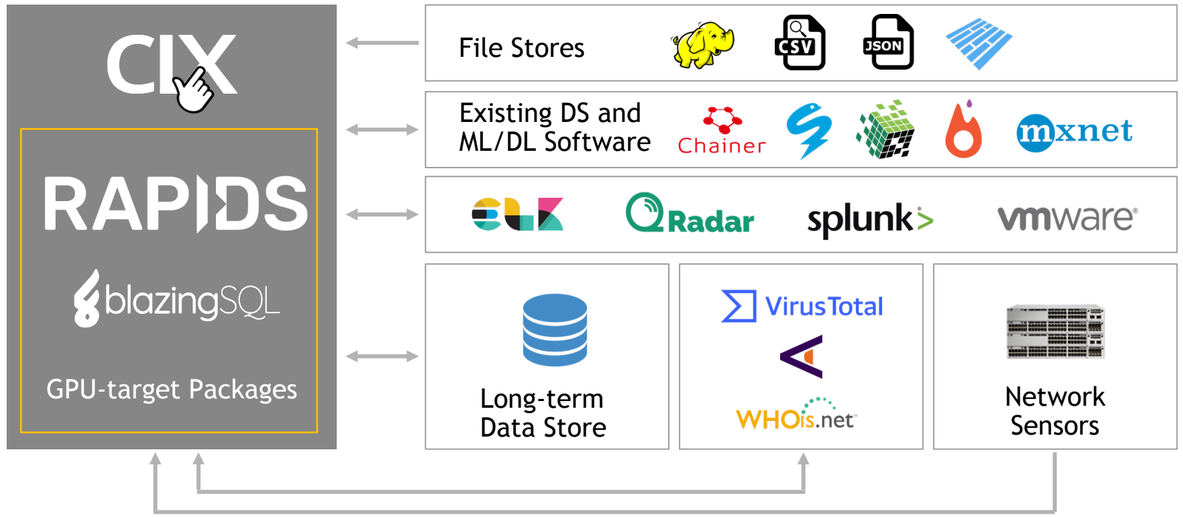
Targeted towards Senior SOC (Security Operations Center) Analysts, InfoSec Data Scientists, Threat Hunters, and Forensic Investigators

Notebooks geared towards info sec and cybersecurity data scientists and data engineers

SIEM integrations that enable easy data import/export and data access

Workflow and I/O **components** that enable users to instantiate new use cases while

Cyber-specific **primitives** that provide accelerated functions across cyber datatypes



CLX CONTAINS VARIOUS USE CASES AND CONNECTORS

Example Notebooks Demonstrate RAPIDS for Cybersecurity Applications

CLX	Type	Proof-of-Concept	Stable
DGA Detection	Use Case		
Network Mapping	Use Case		
Asset Classification	Use Case		
Phishing Detection	Use Case		
Security Alert Analysis	Use Case		
Splunk Integration	Integration		
CLX Query	Integration		
cyBERT	Log Parsing		
GPU Wordpiece Tokenizer	Pre-Processing		
Accelerated IPv4	Primitive		
Accelerated DNS	Primivite		

cyBERT

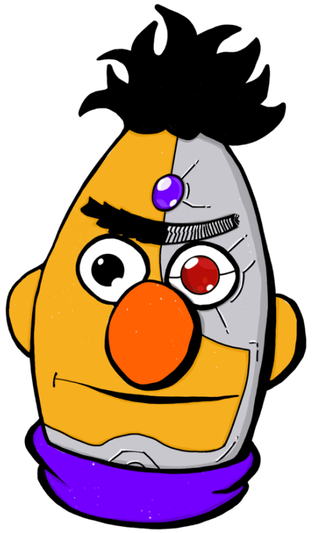
AI Log Parsing for Known, Unknown, and Degraded Cyber Logs

Provide a flexible method that does not use heuristics/regex to parse cybersecurity logs

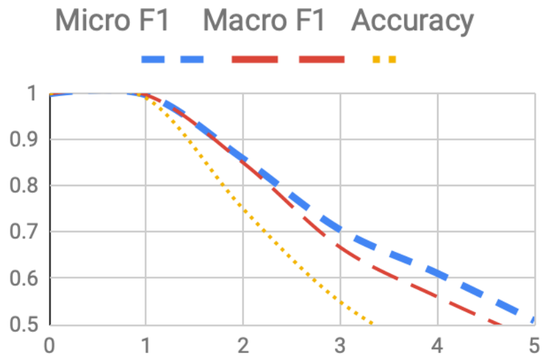
Built using RAPIDS and PyTorch, tested with a variety of language models (including BERT)

Parsing with **micro-F1 and macro-F1 > 0.999** across heterogeneous log types with a **validation loss of < 0.0048**

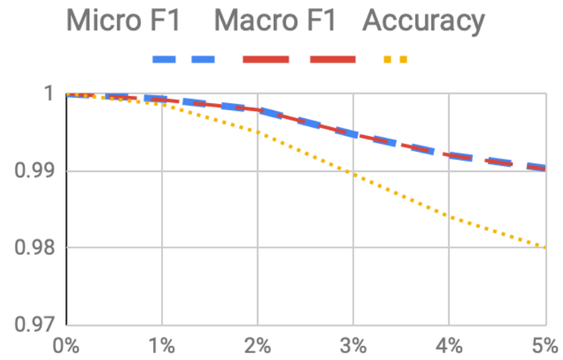
Second version **~160x (min) faster** than first version due to creation of the first all-GPU wordpiece tokenizer that supports non-truncation of logs/sentences



Log Parsing with Inserted Values



Log Parsing with Missing Values



GPU WORDPIECE TOKENIZER

Fully On-GPU Pre-Processing for BERT Training/Inference

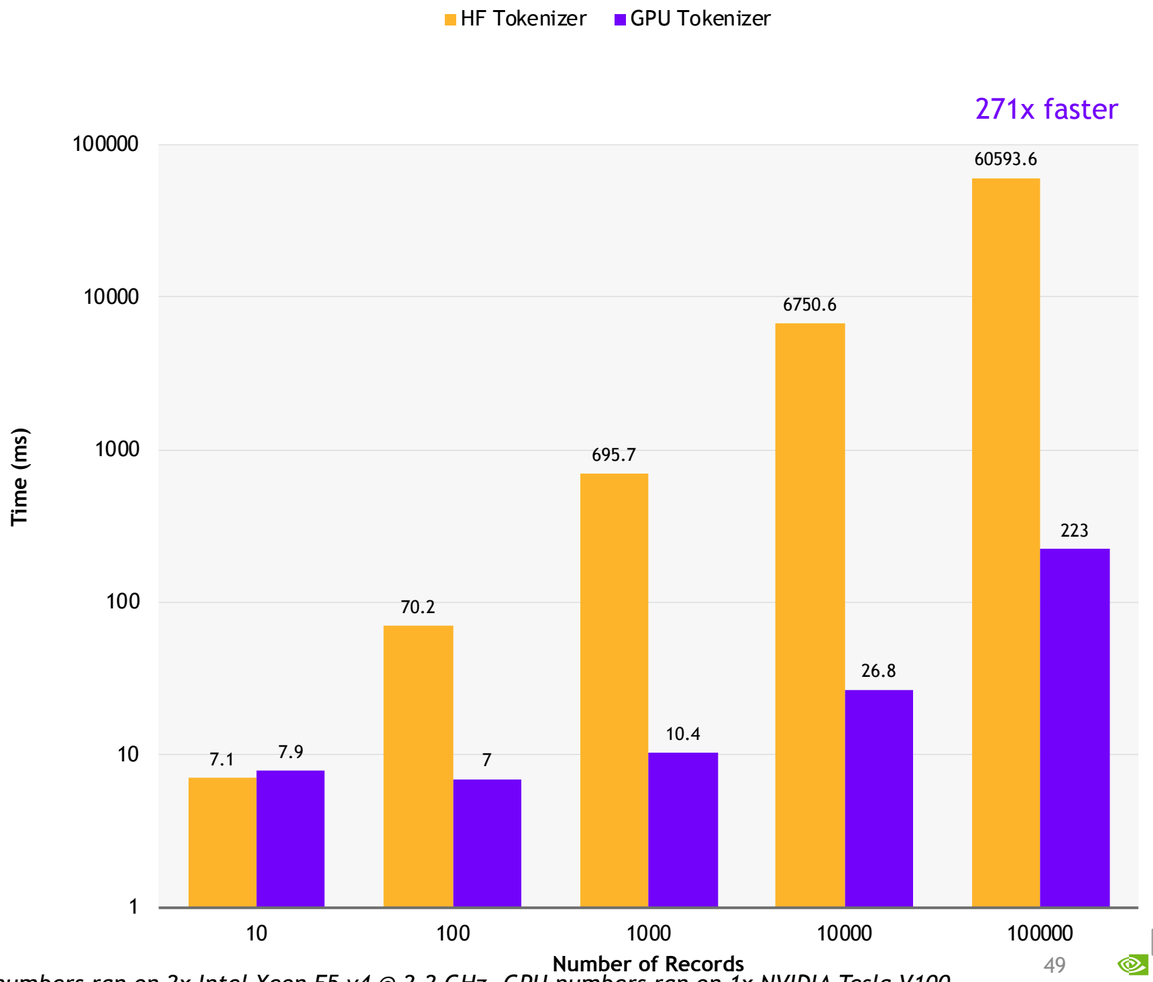
Only wordpiece tokenizer that supports non-truncation of logs/sentences

Returns encoded tensor, attention mask, and metadata to reform broken logs

Supports stride/overlap

Ready for immediate pipelining into PyTorch for inference

Up to 270x faster than production-ready Hugging Face wordpiece tokenizer (Python version)



CPU numbers ran on 2x Intel Xeon E5 v4 @ 2.2 GHz. GPU numbers ran on 1x NVIDIA Tesla V100


JOIN IN!

RAPIDS

Get Started

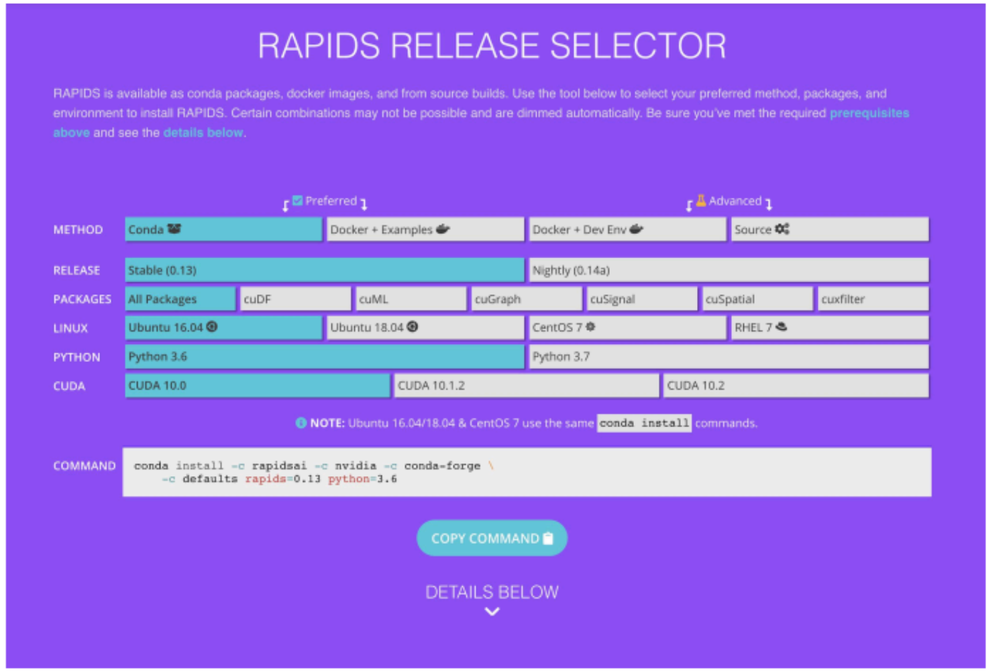
github.com/rapidsai

anaconda.org/rapidsai

NVIDIA GPU CLOUD

ngc.nvidia.com/rapidsai

hub.docker.com/rapidsai



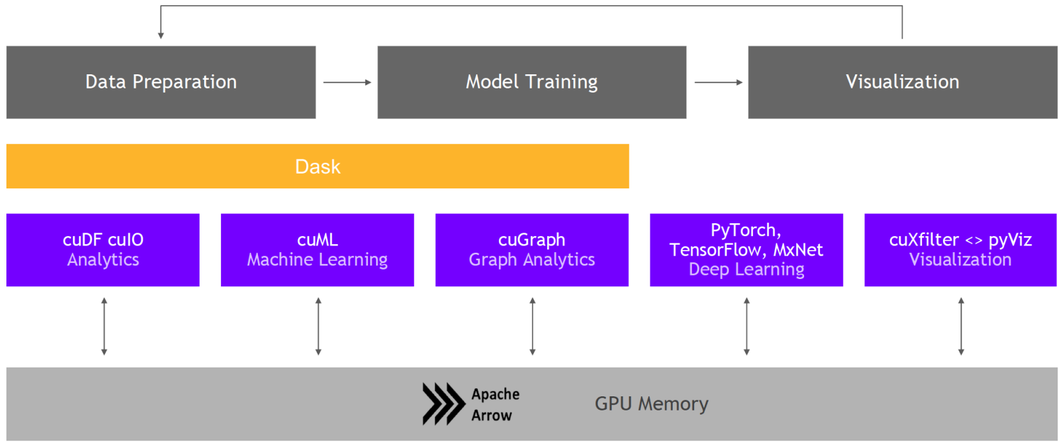
Amazon SageMaker

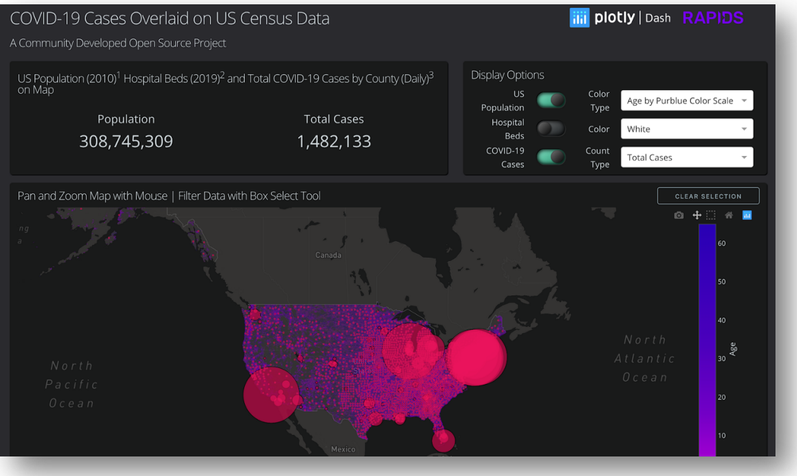
Azure

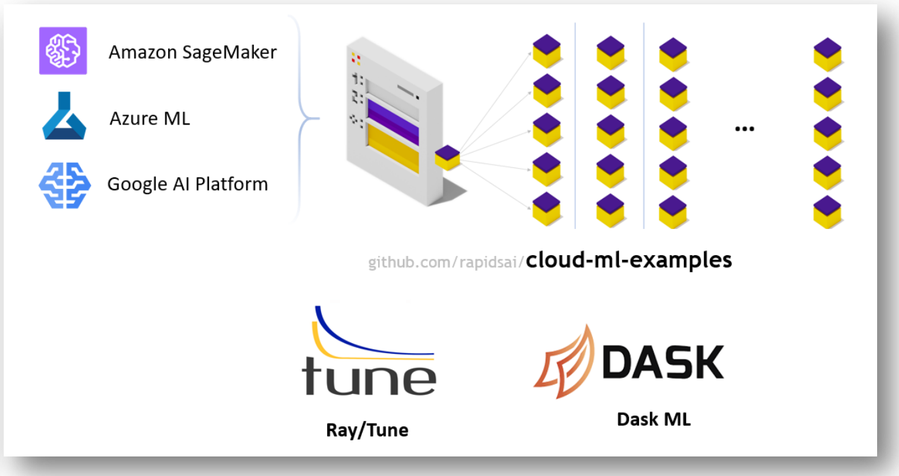
Google Cloud

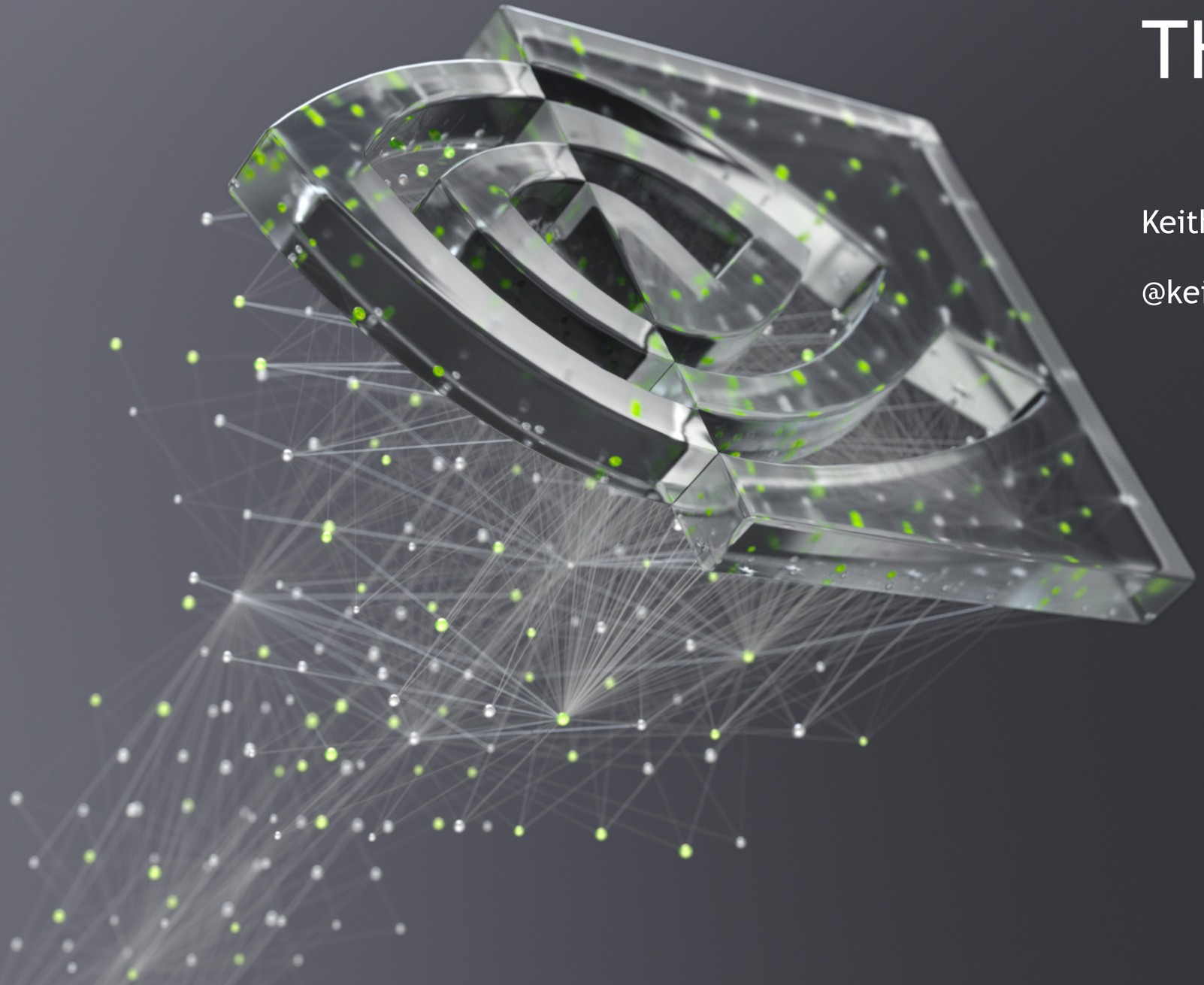
RAPIDS

Join the Community!









THANK YOU!

Keith Kraus

@keithjkraus

NVIDIA