# No-Code-Change GPU Acceleration for Your Pandas and NetworkX Workflows

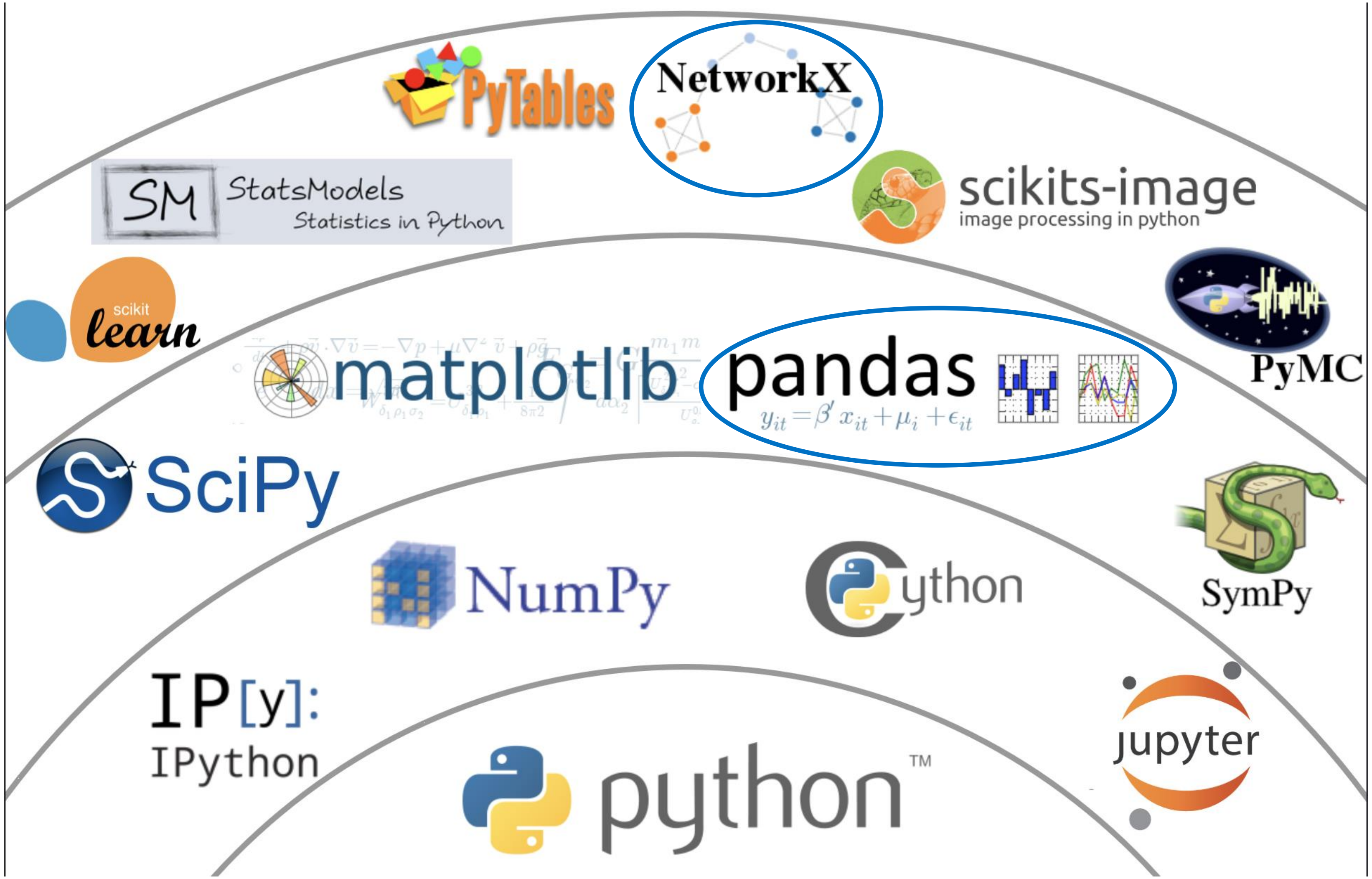
SciPy 2024
Wednesday, July 10, 4:00pm

Rick Ratzel – NVIDIA

Vyas Ramasubramani - NVIDIA

# The Scientific Python Stack

Circa 2015




Credit: Jake Vanderplas, 2015 SciPy Keynote

# A Typical Application Using DataFrames and Graph Analytics…

"Rank Wikipedia editors based on the 'importance' of their contributions"

## https://github.com/rlratzel/SciPy2024

```python
import os

import pandas as pd
import networkx as nx

nx.config.cache_converted_graphs = True  # This is the default in NX 3.4
revisions_df = pd.read_csv(
    "halved_revisions.csv",
    sep="\t",
    names=["title", "editor"],
    dtype="str",
)
nodedata_df = pd.read_csv(
    "full_data.csv",
    sep="\t",
    names=["nodeid", "title"],
    dtype={"nodeid": "int32", "title": "str"},
)
node_revisions_df = nodedata_df.merge(revisions_df, on="title")

edgelist_df = pd.read_csv(
    "full_graph.csv",
    sep=" ",
    names=["src", "dst"],
    dtype="int32",
)
G = nx.from_pandas_edgelist(edgelist_df, "src", "dst", create_using=nx.DiGraph)
nx_pr_vals = nx.pagerank(G)

pagerank_df = pd.DataFrame({"nodeid": nx_pr_vals.keys(), "pagerank": nx_pr_vals.values()})

final_df = node_revisions_df.merge(pagerank_df, on="nodeid").drop("nodeid", axis=1)
influence = final_df[["editor", "pagerank"]].groupby("editor").sum().reset_index()
most_influential_human = influence[~influence["editor"].str.lower().str.contains("bot")]
print(most_influential_human.sort_values(by="pagerank").tail(10))
```

- ~10GB CSV data
- ~21M nodes
- ~315M edges

```
(scipy_demo) dgx05% python demo.py

Read the Wikipedia revision history from halved_revisions.csv...
Done in: 0:01:08.949760

Read the Wikipedia page metadata from full_data.csv...
Done in: 0:00:17.499024

Connect page editors to the page ids...
Done in: 0:01:04.894338

Read the Wikipedia connectivity information from full_graph.csv...
Done in: 0:00:37.812829

Create a NetworkX graph from the connectivity info...
Done in: 0:14:10.917641

Run NetworkX pagerank...
Done in: 0:32:10.346657

Create a DataFrame containing PageRank values...
Done in: 0:00:10.625677

Merge the PageRank scores onto the per-page information...
Done in: 0:01:00.995132

Compute the most influential editors...
Done in: 0:01:19.192866

Show the most influential human editors...
                      editor  pagerank_sum
1121071      CommonsDelinker      0.068351
2679114      John of Reading      0.069264
1037544     Chris the speller      0.076814
5396511            Tom.Reding      0.080647
3406264     Materialscientist      0.081280
4489182              Rjwilmsi      0.082212
534689                 BD2412      0.086898
825038         BrownHairedGirl      0.089414
4459803        Rich Farmbrough      0.090024
4768550  Ser Amantio di Nicolao      0.096206
Done in: 0:00:07.298137
```

Total time:
52:08.53

*Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz*

# Many Other Options Exist, But None Are As Popular

**Python Data Processing Libraries**

dask
MODIN
vaex
Polars
RAPIDS
DuckDB
Xorbits
APACHE Spark

**Python Graph Libraries**

igraph
graph-tool
rustworkx
RAPIDS

- Pandas and NetworkX have well-known performance limitations
- Despite this, their usage generally outstrips alternatives due to:
  - Familiarity
  - Availability of tooling
  - Alternatives requiring changes to preexisting code
- While different API choices can lead to improved performance, it can slow or even permanently hinder adoption

# Even the Robots Know the Standards

Just ask ChatGPT…

How do I join two tables in Python?

Joining two tables in Python can be accomplished using the `pandas` library, which provides a powerful data manipulation and analysis toolset. Here is a step-by-step guide on how to join two tables (DataFrames) using `pandas`.

### Step 1: Install `pandas`

If you haven't already installed `pandas`, you can do so using pip:

…

### Step 4: Perform the Join

You can join two DataFrames using the `merge` function. Here are different types of joins you can perform:

#### Inner Join

An inner join returns only the rows that have matching values in both DataFrames.

python Copy code

```python
inner_join = pd.merge(df1, df2, on='ID', how='inner')
print("\nInner Join:")
print(inner_join)
```

How do I find the betweenness centrality values for each node in a graph?

To find the betweenness centrality values for each node in a graph, you can use the `networkx` library in Python. Betweenness centrality is a measure of the number of times a node acts as a bridge along the shortest path between two other nodes. Here's how you can calculate it:

### Step 1: Install `networkx`

If you haven't already installed `networkx`, you can do so using pip:

…

### Step 3: Calculate Betweenness Centrality

Use the `betweenness_centrality` function provided by `networkx` to calculate the betweenness centrality for each node:

python Copy code

```python
# Calculate betweenness centrality for all nodes
betweenness = nx.betweenness_centrality(G)

# Print the betweenness centrality for each node
print("\nBetweenness Centrality for each node:")
for node, centrality in betweenness.items():
    print(f"Node {node}: {centrality:.3f}")
```

# RAPIDS Accelerates Data Science End-to-End

## Data Loading & ETL

**RAPIDS cuDF**
Tabular Data Loading and Processing

**RAPIDS for Apache Spark**
Apache Spark Accelerator

**RAPIDS Dask**
Dask Accelerator

## Model Training & Analytics

**RAPIDS cuGraph**
Large Scale Graph Analytics

**RAPIDS cuML**
Machine Learning Toolkit

**Deep Learning**
cuGraph-powered GNNs

**RAPIDS for Apache Spark**
Apache Spark ML Accelerator

## Vector Search & Model Inference

**RAPIDS RAFT / cuVS**
Vector Search & ML

**Triton FIL**
Optimized tree model inference

## NVIDIA AI Enterprise

Development Tools | Cloud Native Management and Orchestration | Infrastructure Optimization

Cloud | Data Center | Edge | RTX Laptop

# cuDF: CUDA DataFrames

GPU DataFrame library with a Pandas-like API

## Pandas-like API on the GPU

**CPU**

pandas

```
>>> import pandas as pd
>>> df = pd.read_csv("filepath")
>>> df.groupby(“col”).mean()
>>> df.rolling(window=3).sum()
```

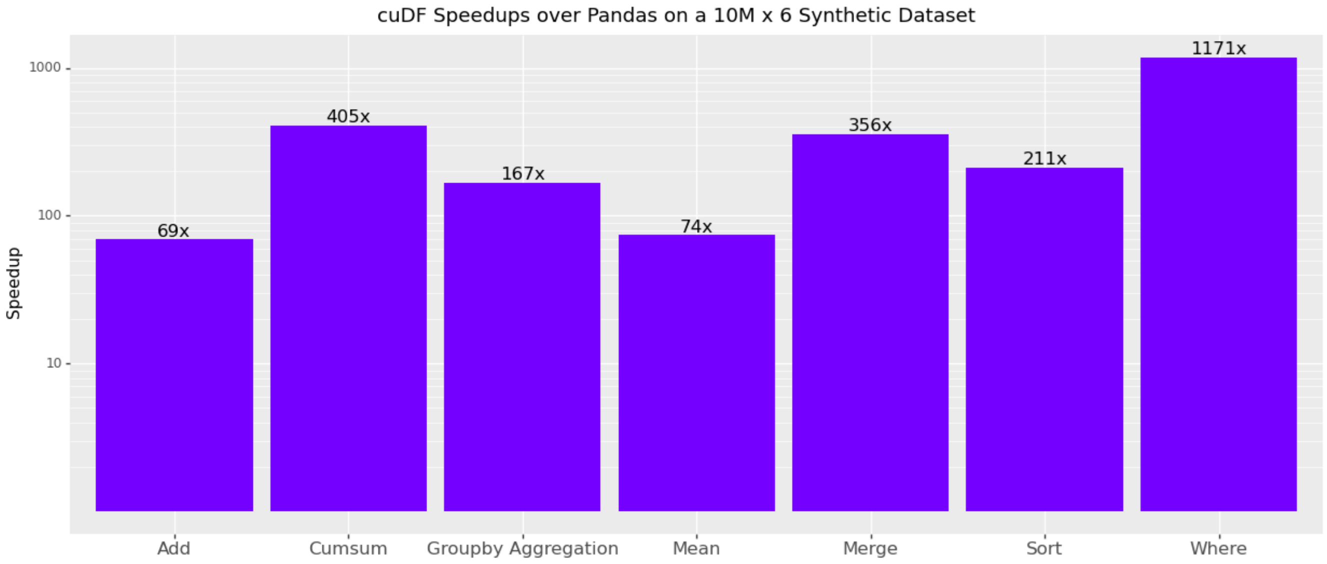
**GPU**

cuDF

```
>>> import cudf
>>> df = cudf.read_csv("filepath")
>>> df.groupby(“col”).mean()
>>> df.rolling(window=3).sum()
```

Average Speed-Ups: *10-100x*

## Best-in-Class Performance




Groupby | Strings and Regex | UDFs | Nested Types | Time Series

Indexing | Missing Data | CuPy Interoperability | Rolling Windows

NVIDIA A100 vs. AMD EPYC 7642 48-Core Processor
cuDF Python vs. Pandas

# cuGraph: GPU Accelerated Python Graph Analytics

GPU accelerated NetworkX-like python library

## NetworkX-like API on the GPU

**CPU**

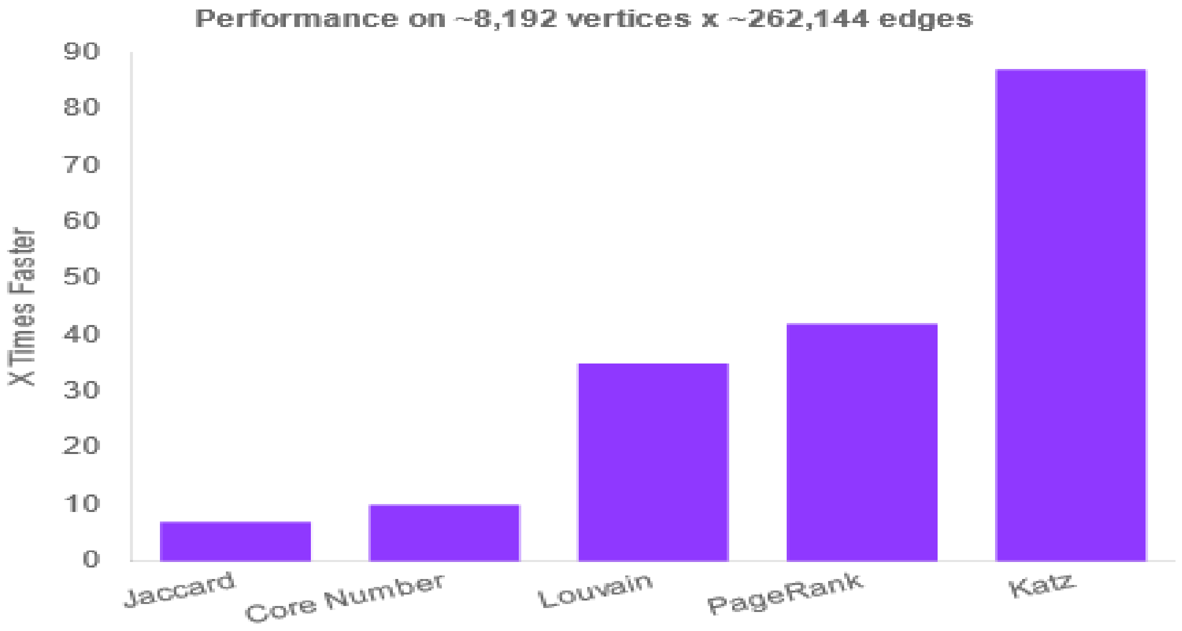
NetworkX

```
>>> import pandas as pd
>>> import networkx as nx
>>> df = pd.read_csv("filepath")
>>> G = nx.from_pandas_edgelist(df)
>>> nx.pagerank(G)
```

**GPU**

cuGraph

```
>>> import cudf
>>> import cugraph as cg
>>> df = cudf.read_csv("filepath")
>>> G = cg.from_cudf_edgelist(df)
>>> cg.pagerank(G)
```




| | | | | |
|---|---|---|---|---|
| Link Analysis | Community Detection | Graph Traversal | Sampling | Centrality |
| MultiGraphs | Connected Components | CuDF Interoperability | GNNs | |

NVIDIA A100 vs. AMD EPYC 7642 48-Core Processor
cuGraph Python vs. NetworkX

# Evolution of Accelerated Computing

Finding the right niche for every kind of user

Easier to Use

## *Zero Code Change: Acceleration Plugins* *(no-code change)*

cudf.pandas: Accelerated Pandas, nx-cugraph: Accelerated NetworkX, RAPIDS Spark Accelerator, Array-API backed Scikit-learn, …

## *Hybrid CPU/GPU libraries* *(minimal change)*

Pytorch, FAISS, Tensorflow, XGBoost, cuML-CPU, Dask, pySpark, …

## *GPU Python Libraries* *(GPU Python code)*

RAPIDS core libraries (RMM, cuDF, cuML, cuGraph, cuVS…), CuPy, Numba, OpenAI Triton, …

## *Python/CUDA libraries* *(Hybrid Python / CUDA code)*

CuPy RawKernels, Numba CUDA, Cython wrappers for CUDA, …

## *C++/CUDA high level* *(High-level C++/CUDA code)*

RAFT, CCCL (Thrust, CUB, libcucxx), cuBLAS, cuDNN, cuSolver, cuSPARSE, …

## *CUDA Toolkit* *(C++/CUDA code and kernels)*

Raw CUDA kernels

Higher Performance

# Accelerated Pandas

cudf.pandas: the zero-code-change GPU accelerator for Pandas built on cuDF

Coming soon: `cudf.polars`!

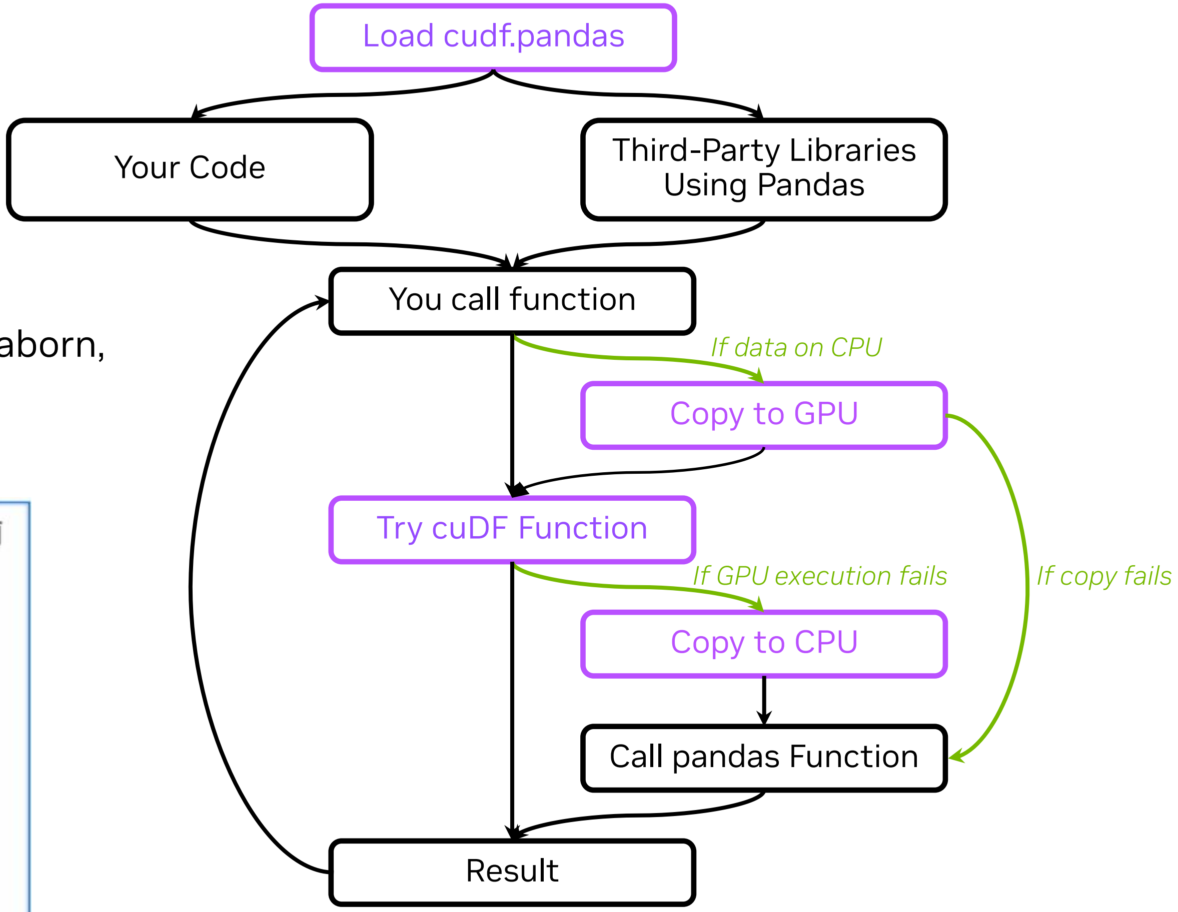
- Requires ***no changes*** to existing pandas code. Just
  - `%load_ext cudf.pandas`
  - `$ python -m cudf.pandas <script.py>`
- Accelerates workflows up to 150x using the GPU
- Compatible with code that uses third-party libraries
  - Integration tested with SciPy, scikit-learn, XGBoost, Matplotlib, seaborn, HoloViews, PyTorch, TensorFlow, …

[ ]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_parquet("data.parquet")
subset = data.index.indexer_between_time("09:30", "16:00")
data = data.iloc[subset]
results = data.groupby(pd.Grouper(freq="1D")).mean()

sns.lineplot(results)
plt.xticks(rotation=30)
```






NVIDIA

# Easy to Use, Built-in Profiler!

- Use the `%%cudf.pandas.profile` cell magic in Jupyter, or import it directly:

```
from cudf.pandas import Profiler

with Profiler() as p:
    # code goes here

p.print_per_function_stats()
```

- Shows which functions ran on the CPU and which ran on the GPU

```
%%cudf.pandas.profile

rng = pd.date_range("2023-01-01", "2023-02-01", freq="10ms")
data = pd.DataFrame(
    {
        "a": np.random.rand(len(rng)),
        "b": np.random.rand(len(rng))
    },
    index=rng
)
data = data.iloc[rng.indexer_between_time("09:30", "16:00")]
results = data.groupby(pd.Grouper(freq="1D")).mean()
results.head()
```

*Total time elapsed: 12.855 seconds*
*11 GPU function calls in 1.322 seconds*
*1 CPU function calls in 4.416 seconds*

*Stats*

| Function | GPU ncalls | GPU cumtime | GPU percall | CPU ncalls | CPU cumtime | CPU percall |
|---|---|---|---|---|---|---|
| date_range | 1 | 0.008 | 0.008 | 0 | 0.000 | 0.000 |
| DatetimeIndex.__len__ | 2 | 0.000 | 0.000 | 0 | 0.000 | 0.000 |
| DataFrame | 2 | 0.873 | 0.436 | 0 | 0.000 | 0.000 |
| DatetimeIndex.indexer_betwee… | 0 | 0.000 | 0.000 | 1 | 4.416 | 4.416 |
| _DataFrameIlocIndexer.__geti… | 1 | 0.127 | 0.127 | 0 | 0.000 | 0.000 |
| Grouper | 1 | 0.000 | 0.000 | 0 | 0.000 | 0.000 |
| DataFrame.groupby | 1 | 0.021 | 0.021 | 0 | 0.000 | 0.000 |
| DataFrameResampler.mean | 1 | 0.259 | 0.259 | 0 | 0.000 | 0.000 |
| DataFrame.head | 1 | 0.001 | 0.001 | 0 | 0.000 | 0.000 |
| DataFrame.__repr__ | 1 | 0.033 | 0.033 | 0 | 0.000 | 0.000 |

Not all pandas operations ran on the GPU. The following functions required CPU fallback:

- DatetimeIndex.indexer_between_time

To request GPU support for any of these functions, please file a Github issue here: https://github.com/rapidsai/cudf/issues/new/choose.

# Spoofing Pandas and Its Contents

## *Modules*

- What happens when you `import pd`?
- Normally, checks various built-ins, PYTHONPATH, etc
- But! We can insert a custom ***finder*** that runs first
- `cudf.pandas` implements a finder, the `ModuleAccelerator`, which returns proxy modules

```
In [1]: %load_ext cudf.pandas
In [2]: import pandas as pd
In [3]: pd
Out[3]: <module 'pandas'
(ModuleAccelerator(fast=cudf, slow=pandas))>

In [4]: import pandas.plotting as plt
In [5]: plt
Out[5]: <module 'pandas.plotting'
(ModuleAccelerator(fast=cudf, slow=pandas))>
```

https://docs.python.org/3/reference/import.html

## *Module Contents*

- What happens when you access attributes (free functions, classes, class methods)?
- `cudf.pandas` produces ***proxy objects***
- Proxies masquerade as pandas objects

```
In [6]: pd.DataFrame
Out[6]: pandas.core.frame.DataFrame
In [7]: type(pd.DataFrame())
Out[7]: pandas.core.frame.DataFrame
```

- But if you look more closely…

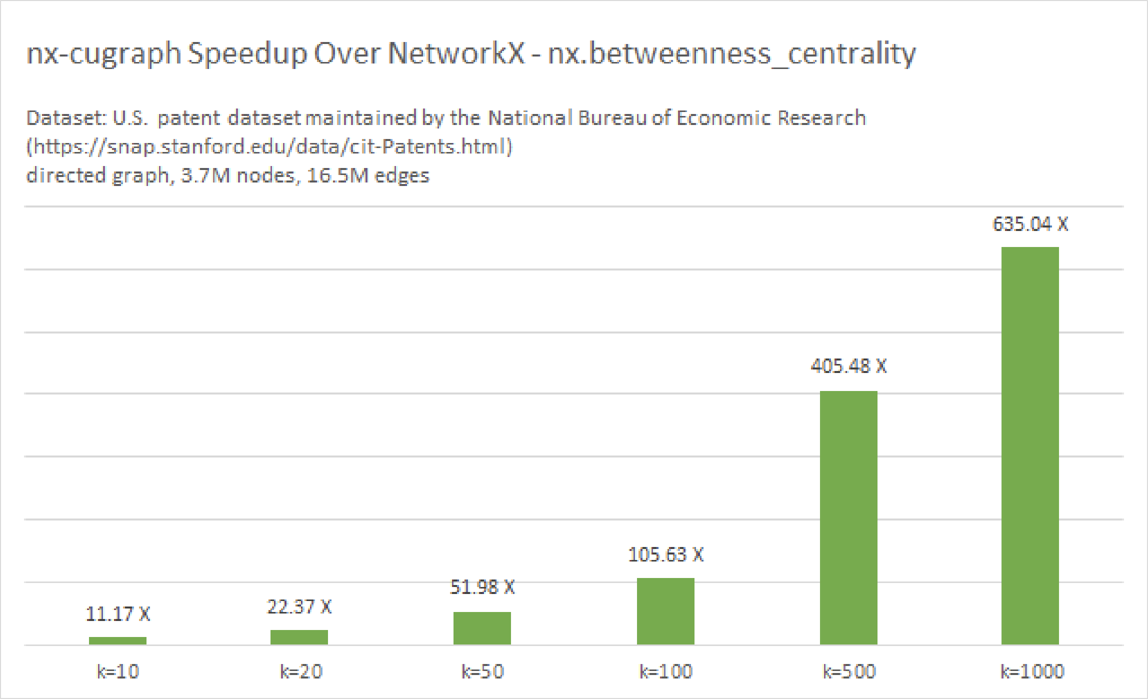
```
In [7]: type(pd.DataFrame)
Out[8]: cudf.pandas.fast_slow_proxy._FastSlowProxyMeta
```

- Fundamental limitations (numpy C API calls)
- Hacking import isn't ideal

# Accelerated NetworkX

## nx-cugraph: zero-code-change acceleration for NetworkX, powered by cuGraph

- Zero-code-change GPU-acceleration for NetworkX code
- Accelerates up to 600x depending on algorithm and graph size
- Support for 60 popular graph algorithms and growing
- Fallback to CPU for any unsupported algorithms




NetworkX 3.2. CPU: Intel(R) Xeon(R) Platinum 8480CL 2TB. GPU: NVIDIA H100 80GB

```
import pandas as pd
import networkx as nx

url = "https://data.rapids.ai/cugraph/datasets/cit-Patents.csv"
df = pd.read_csv(url, sep=" ", names=["src", "dst"], dtype="int32")
G = nx.from_pandas_edgelist(df, source="src", target="dst")

%time result = nx.betweenness_centrality(G, k=10)
```

```
user@machine:/# ipython bc_demo.ipy
CPU times: user 7min 38s, sys: 5.6 s, total: 7min 44s
Wall time: 7min 44s

user@machine:/# NETWORKX_BACKEND_PRIORITY=cugraph ipython bc_demo.ipy
CPU times: user 18.4 s, sys: 1.44 s, total: 19.9 s
Wall time: 20 s
```

# nx-cugraph

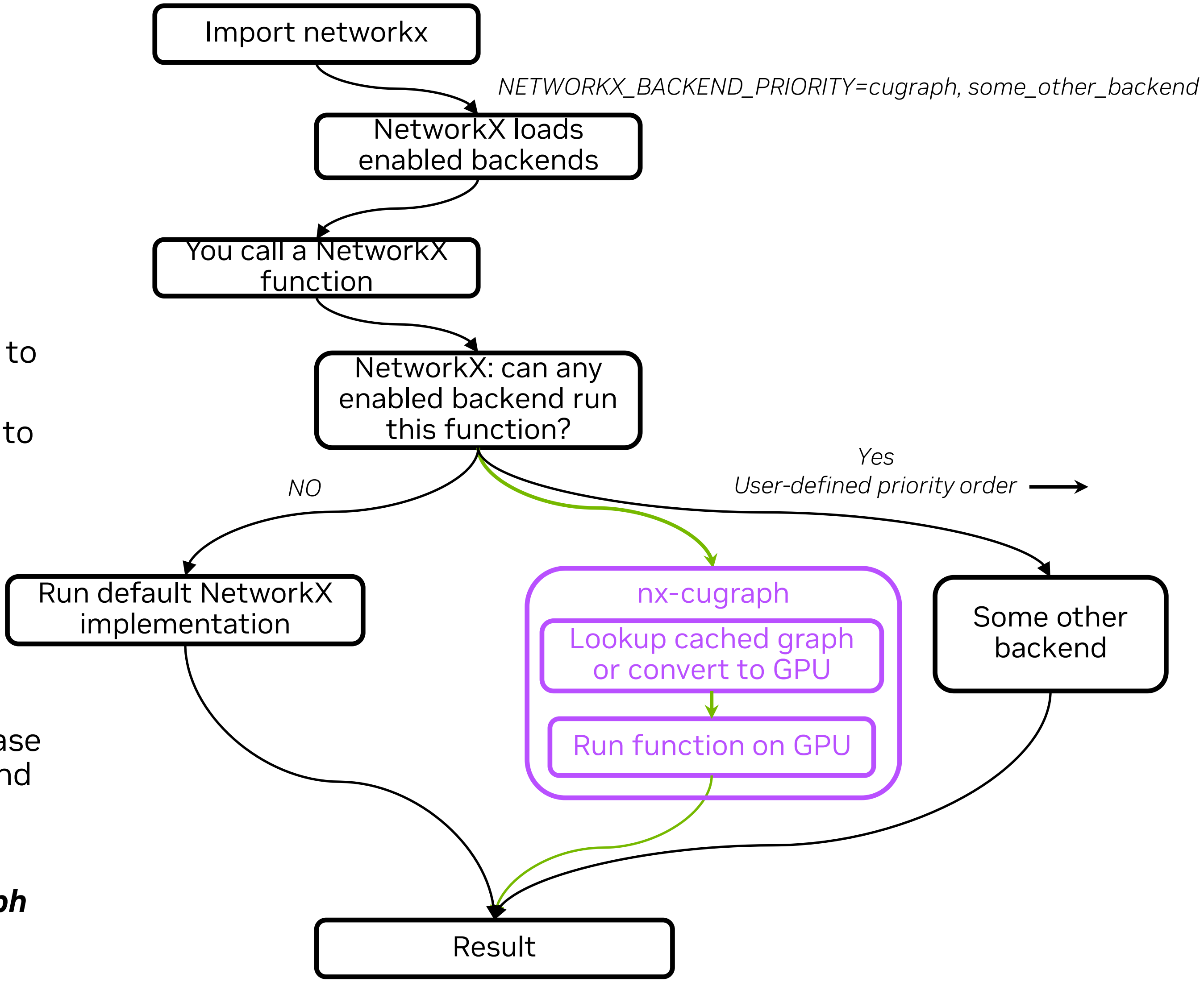
## A GPU-accelerated NetworkX backend

- nx-cugraph is a NetworkX ***backend***

- What's a NetworkX backend?
  - NetworkX added the ability to ***dispatch*** various function calls to third-party backends, starting in NetworkX 3.0
  - Backends provide an alternate implementation for NetworkX to call
    - Allows users to run implementations optimized for their environment without changing their code – the NetworkX "frontend" remains the same

- Multiple backends can be used together:
  - Ex. Access a graph in a remote graph database using a database backend, run algorithms on GPU using the nx-cugraph backend

  - Learn more about about NetworkX dispatching and other NetworkX backends at the poster session: ***Fast and Easy Graph Analytics with the NetworkX Ecosystem of Backends***



# A Typical Application, Revisited...

- No code changes! Simply enable both cudf.pandas and nx-cugraph:
  - Set environment variable `NETWORKX_BACKEND_PRIORITY=cugraph`
  - Run with the `cudf.pandas` module
  - Before:
    - `python demo.py`
  - After:
    - `NETWORKX_BACKEND_PRIORITY=cugraph python -m cudf.pandas demo.py`
- Total speedup: 52 min -> 22 min = 2.5x
  - Why not more?
- cudf.pandas:
  - CSV reads: ~20x speedup
  - Merges: ~130x speedup
  - Groupby-apply: ~200x speedup
  - Select+filter: ~3x speedup
- nx-cugraph:
  - Pagerank: ~10x speedup
  - Graph creation: 0.7x speedup (slower!)
- At 18 mins, graph creation dominates runtime!

*Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz*
*NVIDIA V100 GPU 32GB*

```
(scipy_demo) dgx05% NETWORKX_BACKEND_PRIORITY="cugraph" python -m
cudf.pandas demo.py

Read the Wikipedia revision history from halved_revisions.csv...
Done in: 0:00:04.479977

Read the Wikipedia page metadata from full_data.csv...
Done in: 0:00:00.452238

Connect page editors to the page ids...
Done in: 0:00:01.134292

Read the Wikipedia connectivity information from full_graph.csv...
Done in: 0:00:03.843927

Create a NetworkX graph from the connectivity info...
Done in: 0:18:21.004385

Run NetworkX pagerank...
Done in: 0:03:21.999969

Run again using the cached graph conversion...
… # cugraph prints an informative message about caching here
Done in: 0:00:10.242561

Create a DataFrame containing PageRank values...
Done in: 0:00:05.711783

Merge the PageRank scores onto the per-page information...
Done in: 0:00:00.301918

Compute the most influential editors...
Done in: 0:00:00.431072

Show the most influential human editors...
                         editor  pagerank_sum
1121010          CommonsDelinker      0.068355
2678970          John of Reading      0.069278
1037484         Chris the speller     0.076830
5396241                Tom.Reding     0.080654
3406082         Materialscientist     0.081262
4488962                  Rjwilmsi     0.082204
534633                     BD2412     0.086900
824982             BrownHairedGirl     0.089404
4459584            Rich Farmbrough     0.090030
4768324     Ser Amantio di Nicolao     0.096223
Done in: 0:00:02.381004
```

Total time:
**22:11.98**

unaccelerated:
52:08.53

**2.5X speedup**

# A Typical Application, Revisited…

## Why not even faster?

- cudf.pandas has a high memory footprint
  - Only using half the Wikipedia revision history
  - Processing ~10 Gb (4 + 1 + 5) of data
  - Reducing OOMs is ongoing work

- cudf.pandas → nx-cugraph handoff is not seamless
  - Graph creation is not accelerated because NetworkX acts like it's operating on a Pandas object
  - A python-based NetworkX Graph is created, not a GPU graph

- First algorithm call triggers conversion to that backend's graph type
  - For nx-cugraph: host->device data transfer
  - This conversion is cached
    - Subsequent algos reuse the GPU graph *(if the original was not modified)*
  - All other pagerank calls: **32:10 → 0:10 = ~190x speedup!**

```
(scipy_demo) dgx05% NETWORKX_BACKEND_PRIORITY="cugraph" python -m
cudf.pandas demo.py

Read the Wikipedia revision history from halved_revisions.csv...
Done in: 0:00:04.479977

Read the Wikipedia page metadata from full_data.csv...
Done in: 0:00:00.452238

Connect page editors to the page ids...
Done in: 0:00:01.134292

Read the Wikipedia connectivity information from full_graph.csv...
Done in: 0:00:03.843927

Create a NetworkX graph from the connectivity info...
Done in: 0:18:21.004385

Run NetworkX pagerank...
Done in: 0:03:21.999969

Run again using the cached graph conversion...
… # cugraph prints an informative message about caching here
Done in: 0:00:10.242561

Create a DataFrame containing PageRank values...
Done in: 0:00:05.711783

Merge the PageRank scores onto the per-page information...
Done in: 0:00:00.301918

Compute the most influential editors...
Done in: 0:00:00.431072

Show the most influential human editors...
                         editor  pagerank_sum
1121010        CommonsDelinker      0.068355
2678970        John of Reading      0.069278
1037484       Chris the speller      0.076830
5396241             Tom.Reding      0.080654
3406082      Materialscientist      0.081262
4488962               Rjwilmsi      0.082204
534633                  BD2412      0.086900
824982         BrownHairedGirl      0.089404
4459584         Rich Farmbrough      0.090030
4768324  Ser Amantio di Nicolao      0.096223
Done in: 0:00:02.381004
```

Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
NVIDIA V100 GPU 32GB

# A Low-Code Change Approach

If you are willing to make small changes, you can go even faster

```
import cudf
import networkx as nx

revisions_df = cudf.read_csv(
    "halved_revisions.csv",
    sep="\t",
    names=["title", "editor"],
    dtype="str",
)
nodedata_df = cudf.read_csv(
    "full_data.csv",
    sep="\t",
    names=["nodeid", "title"],
    dtype={"nodeid": "int32", "title": "str"},
)
node_revisions_df = nodedata_df.merge(revisions_df, on="title")

edgelist_df = cudf.read_csv(
    "full_graph.csv",
    sep=" ",
    names=["src", "dst"],
    dtype="int32",
)

# Create an nx_cugraph (not NetworkX) Graph compatible only with nx_cugraph algorithms
G = nx.from_pandas_edgelist(
    edgelist_df,
    source="src",
    target="dst",
    create_using=nx.DiGraph,
    backend="cugraph"
)
nx_pr_vals = nx.pagerank(G)

pagerank_df = cudf.DataFrame({"nodeid": nx_pr_vals.keys(), "pagerank": nx_pr_vals.values()})
final_df = node_revisions_df.merge(pagerank_df, on="nodeid").drop("nodeid", axis=1)
influence = final_df[["editor", "pagerank"]].groupby("editor").sum().reset_index()
most_influential_human = influence[~influence["editor"].str.lower().str.contains("bot")]
print(most_influential_human.sort_values(by="pagerank").tail(10))
```

```
Read the Wikipedia revision history from halved_revisions.csv...
Done in: 0:00:05.226320

Read the Wikipedia page metadata from full_data.csv...
Done in: 0:00:00.457513

Connect page editors to the page ids...
Done in: 0:00:00.299906

Read the Wikipedia connectivity information from full_graph.csv...
Done in: 0:00:03.864327

Create a NetworkX graph from the connectivity info...
Done in: 0:00:09.624684

Run NetworkX pagerank...
Done in: 0:00:06.452435

Create a DataFrame containing PageRank values...
Done in: 0:00:04.281305

Merge the PageRank scores onto the per-page information...
Done in: 0:00:00.190582

Compute the most influential editors...
Done in: 0:00:00.293617

Show the most influential human editors...
                      editor  pagerank
5767097        CommonsDelinker  0.070213
1536890        John of Reading  0.070812
5330010       Chris the speller  0.078867
3676022             Tom.Reding  0.081731
4843142       Materialscientist  0.083336
4298520               Rjwilmsi  0.084033
3835004                 BD2412  0.088707
1419682          BrownHairedGirl  0.091291
1303378         Rich Farmbrough  0.092024
2516853  Ser Amantio di Nicolao  0.098005
Done in: 0:00:01.393367
```

Total time:
**0:32.08**

unaccelerated:
52:08.53

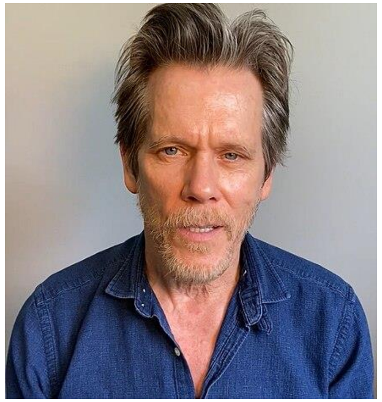
**98X speedup**

*Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz*
*NVIDIA V100 GPU 32GB*

# Further Exploration of the Data...

Six Degrees of SciPy!

```
shortest_paths = nx.shortest_path(G, source=scipy_nodeid)
```



```
Find the shortest path between the SciPy article and all articles...
Done in: 0:02:28.838333

Print the shortest paths...

Find the shortest path between SciPy and Orange juice...
SciPy
Python (programming language)
Blender (software)
Orange (fruit)
Orange juice

Find the shortest path between SciPy and Lake Leon (Florida)...
SciPy
Python (programming language)
Monty Python
Dave Chappelle
Xenia, Ohio
Dean Chenoweth
Tom Brown Park
Lake Leon (Florida)

Find the shortest path between SciPy and Kevin Bacon...
SciPy
Python (programming language)
Industrial Light & Magic
Ron Howard
Kevin Bacon
Done in: 0:00:00.360897
```

7:33 unaccelerated,
3X speedup

*Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz*
*NVIDIA V100 GPU 32GB*

NVIDIA

**Interested in talking to us?**

**Fill out this 2-minute-long survey**

## Relevant Resources

- cuDF Github - **https://github.com/rapidsai/cudf**
- cuGraph Github - **https://github.com/rapidsai/cugraph**
- RAPIDS website - **https://rapids.ai**
- Code from this talk: **https://github.com/rlratzel/SciPy2024**
- `pip install cudf-cu12 nx-cugraph-cu12 --extra-index-url https://pypi.nvidia.com`
- `conda install -c rapidsai -c conda-forge -c nvidia cudf nx-cugraph`

## Find us at SciPy

- BoF: Accelerated Python
- Poster Session: *Fast and Easy Graph Analytics with the NetworkX Ecosystem of Backends*

# Extras

# What about duck typing?

Check for behavior, not for inheritance

```python
# this function is duck-typing friendly, and will work
# for any DataFrame-like object:
def function_one(df: pd.DataFrame) -> pd.Series:
    return df.groupby("a").max()

# this function is *not* duck-typing friendly. It will
# raise for anything that's not a pandas DataFrame:
def function_two(df: pd.DataFrame) -> pd.DataFrame:
    if not isinstance(df, pd.DataFrame):
        raise TypeError("Not a pandas DataFrame!")
    return pd.concat([df, df])
```

# Under the hood

Deep import customization to hijack pandas imports

```python
# a custom importer for "pandas" that gives you a module
# containing our proxy DataFrame instead of the real pandas
class PandasImporter(importlib.abc.MetaPathFinder, importlib.abc.Loader):
    def find_spec(self, fullname, path, target=None):
        if fullname == "pandas":
            return importlib.machinery.ModuleSpec(
                fullname, self
            )
        return None

    def exec_module(self, module: ModuleType):
        module.DataFrame = DataFrame
        return module

sys.meta_path.insert(0, PandasImporter())
```

# Under the hood

Proxy objects that dispatch to cudf or pandas

```
class DataFrame:
    def __init__(self, *args, **kwargs):
        try:
            self._wrapped = cudf.DataFrame(*args, **kwargs)
        except Exception:
            self._wrapped = pd.DataFrame(*args, **kwargs)

    def count(self, *args, **kwargs):
        """Count the number of non-null elements along the given axis"""
        try:
            return self._wrapped.count(*args, **kwargs)
        except Exception:
            print("falling back to pandas!")
            result = self._wrapped.to_pandas().count(*args, **kwargs)
            return cudf.from_pandas(result)
```

# Under the hood

## Transparent fallback via proxied module

```
>>> df = DataFrame({'a': [1, 2, 3], 'b': [None, 3, 4]})
>>> df.count(axis=0)
a    3
b    2
dtype: int64
>>> df.count(axis=1)
falling back to pandas!
0    1
1    2
2    2
dtype: int64
```