Bringing GPU Support to Datashader

A RAPIDS Case Study

Jon Mease, Chief Scientist at Plotly

Acknowledgements

This work was overseen by Anaconda Inc
and was funded by NASA SBIR NNX17CG19C and the US Army Corps of Engineers

Overview

- Datashader Overview
- Obstacles to GPU Acceleration
- Enter RAPIDS
- Performance Results
- Demo

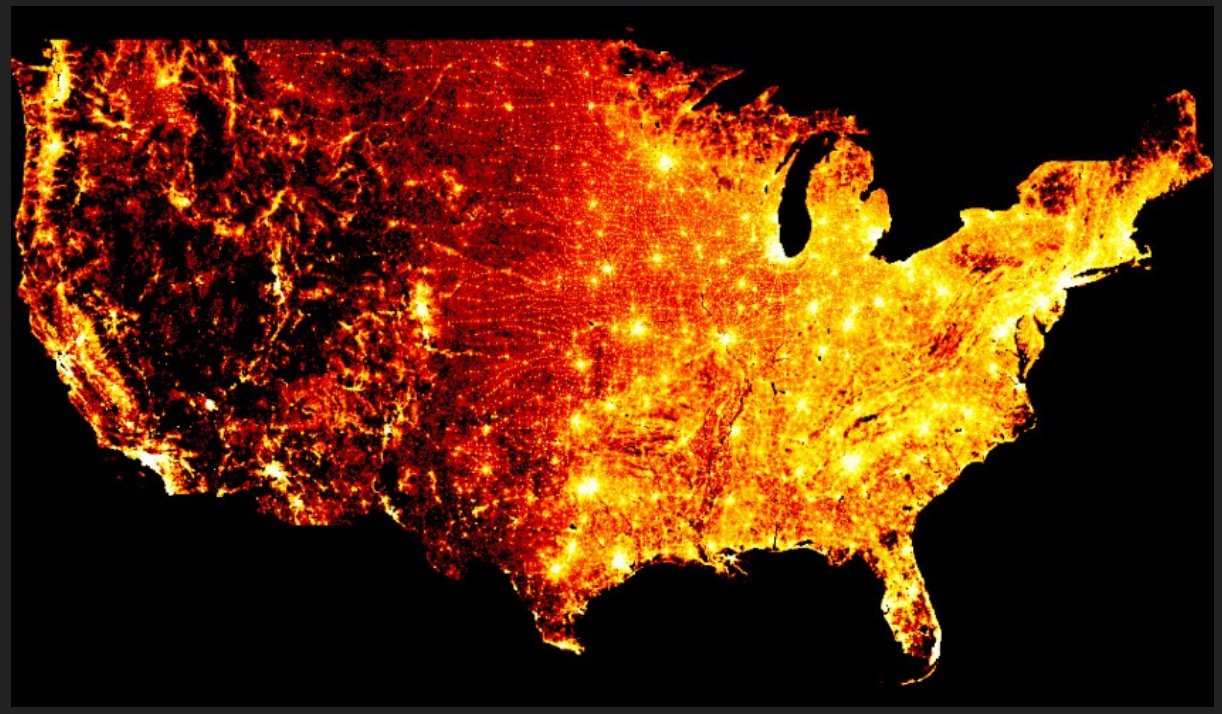
Datashader Overview

Datashader is a high-performance Python library for generating
a variety of principled visualizations of large datasets

Datashader Overview

Datashader is a high-performance Python library for generating a variety of principled visualizations of large datasets

- Data Visualizations
  - Points
  - Lines
  - Meshes
  - Networks



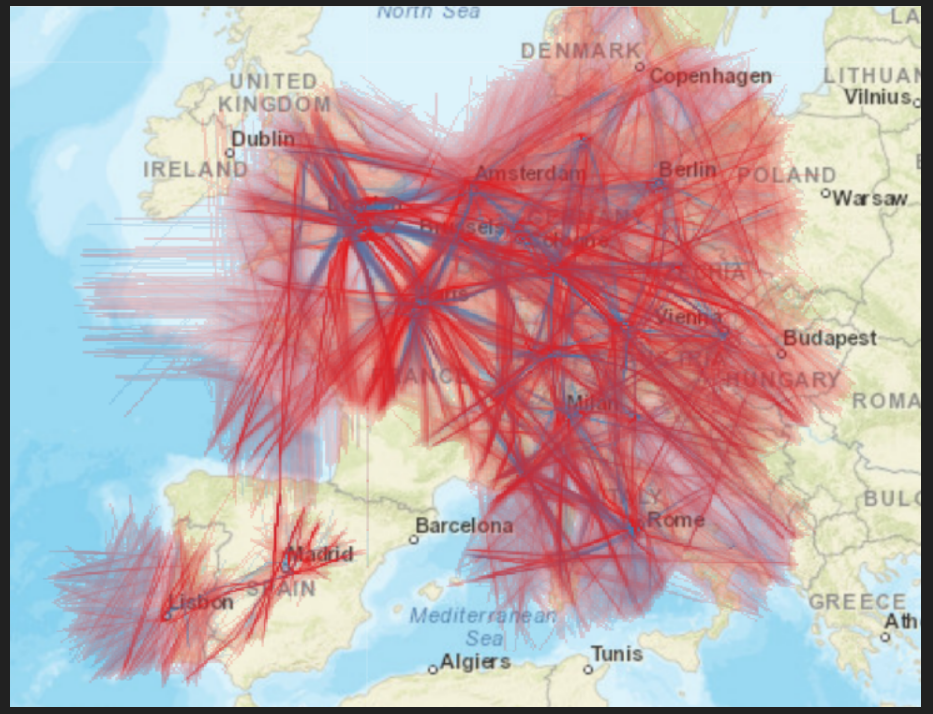
Visualization of 300 million data points
(one per person in the USA) from the 2010 census[1]

[1] examples.pyviz.org/census/census.html

Datashader Overview

Datashader is a high-performance Python library for generating a variety of principled visualizations of large datasets

- Data Visualizations
  - Points
  - Lines
  - Meshes
  - Networks



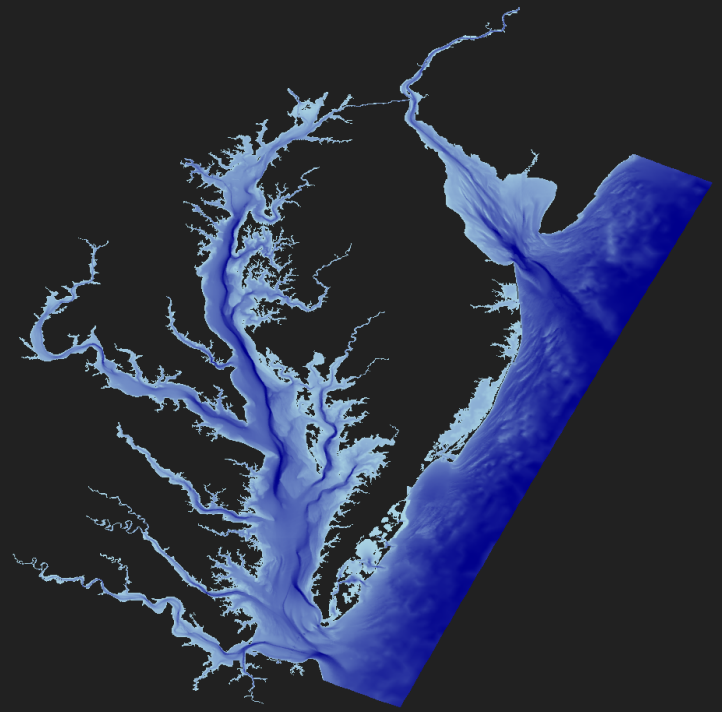
OpenSky flight dataset with over 10 million coordinates: Ascending (Blue) vs. descending flights (Red)[2]

[2] examples.pyviz.org/opensky/opensky.html

Datashader Overview

Datashader is a high-performance Python library for generating a variety of principled visualizations of large datasets

- Data Visualizations
  - Points
  - Lines
  - Meshes
  - Networks



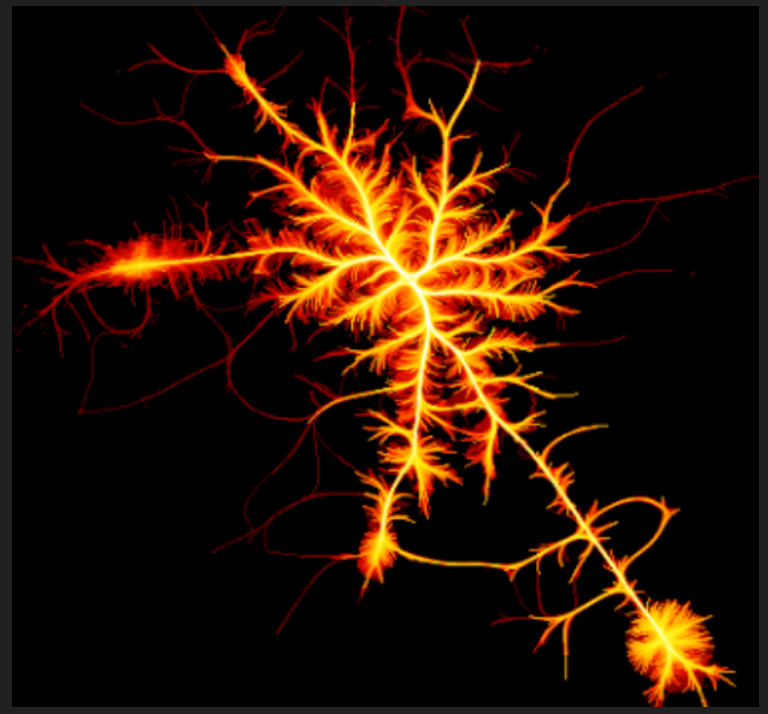
Depth of Chesapeake and Delaware Bays with over 1 million triangles[3]

[3] examples.pyviz.org/bay_trimesh/bay_trimesh.html

Datashader Overview

Datashader is a high-performance Python library for generating a variety of principled visualizations of large datasets

- Data Visualizations
  - Points
  - Lines
  - Meshes
  - Networks



Graph of research institutions linked by joint UK grants with over 600k edges[4]

[4] datashader.org/user_guide/Networks.html

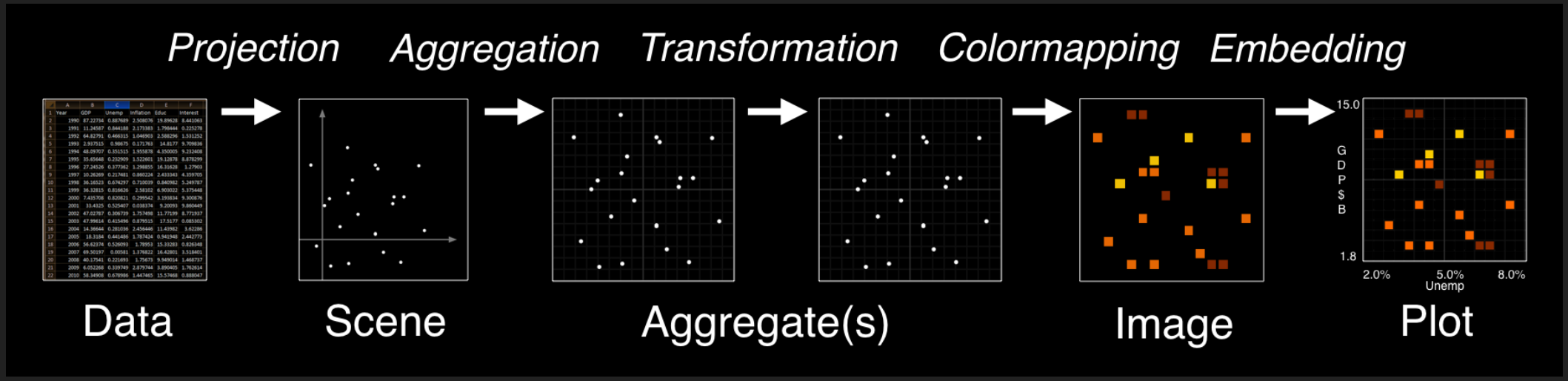
Datashader Overview

Datashader is a high-performance Python library for generating a variety of principled visualizations of large datasets

- High-Performance Python
  - Data Structures
    - **NumPy**
    - **pandas**
    - **xarray**
  - Custom algorithms
    - **Numba**

Datashader Overview

Datashader is a high-performance Python library for generating
a variety of principled visualizations of large datasets

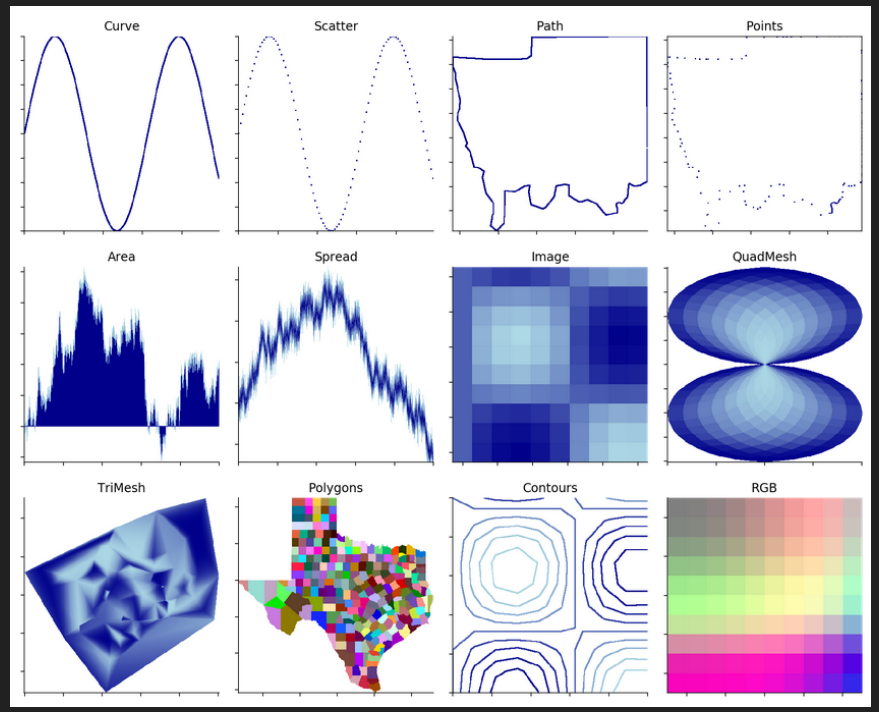


Datashader Pipeline[5]

[5] datashader.org/getting_started/Pipeline.html

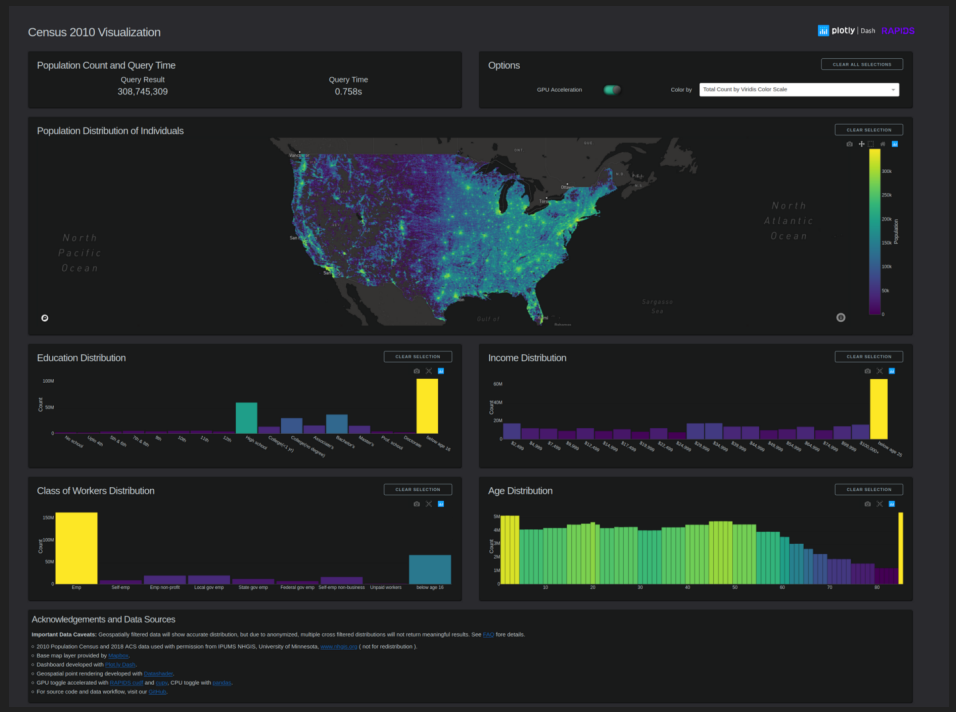
Datashader Integrations

HoloViews



Working with large data using datashader[6]

Dash



Making a Plotly Dash Census Viz Powered by RAPIDS[7]

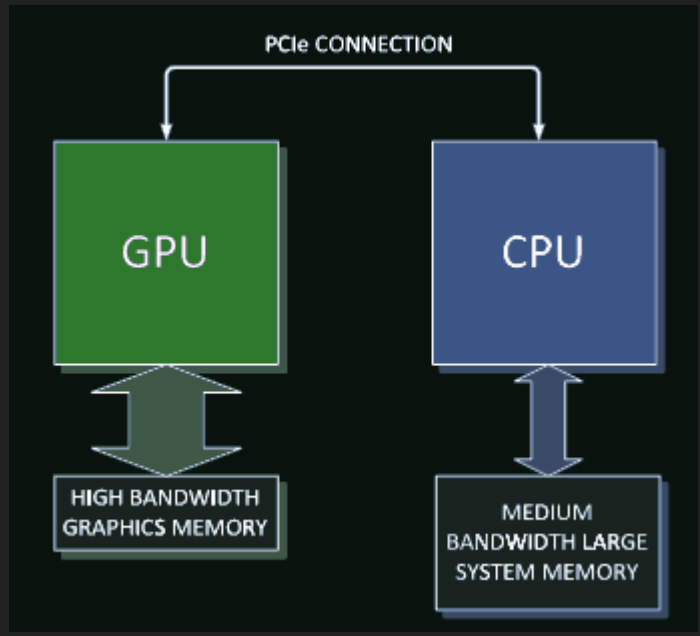
[6] holoviews.org/user_guide/Large_Data.html
[7] medium.com/rapids-ai/plotly-census-viz-dashboard-powered-by-rapids-1503b3506652

Obstacles to GPU Acceleration

Obstacles to GPU Acceleration

Data Transfer

- Inputs must be copied from CPU to GPU memory
- Outputs must be copied from GPU back to CPU memory
- Large transfers for a fast operation reduces GPU speedup



GPU and CPU memory connection over PCIe[8]

[8] devblogs.nvidia.com/nvlink-pascal-stacked-memory-feeding-appetite-big-data/

Obstacles to GPU Acceleration

Code Reuse

- Traditional GPU programming is done in C, C++, or Fortran
- Datashader's core algorithms are written in Python, pandas, NumPy, and Numba
- Duplicating algorithms across languages is not sustainable

C++ CUDA

```
__global__
void increment_by_one(int n, float *an_array)
{
    int index = threadIdx.x +
                blockIdx.x * blockDim.x;
    int stride = blockDim.x * gridDim.x;
    for (int i = index; i < n; i += stride)
        an_array[i] += 1;
}
```

Python

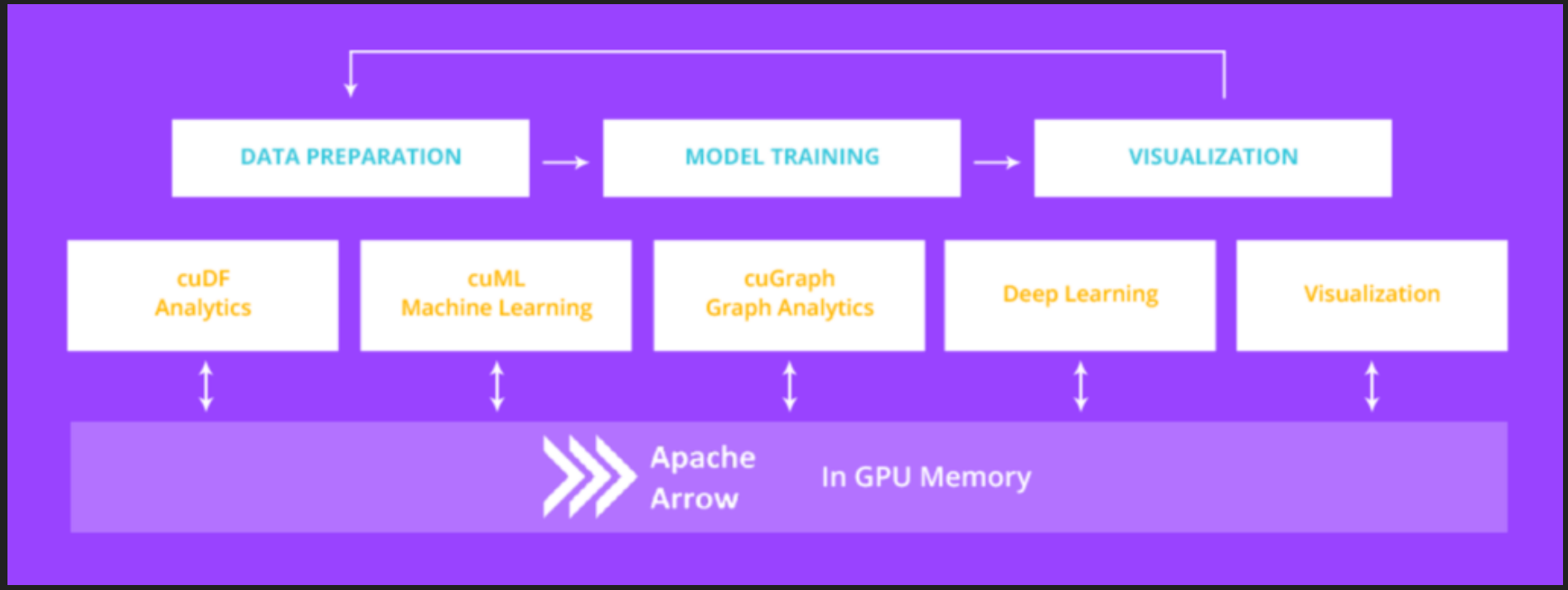
```
def increment_by_one(an_array):
    for i in range(an_array.size):
        an_array[i] += 1
```

Enter RAPIDS (* and friends)

- **cuDF** (Pandas style DataFrames)
- **cuML** (Scikit-Learn style Machine Learning)
- **cuGraph** (NetworkX style graph analysis)
- **CuPy*** (numpy style N-dimensional arrays)
- **Numba for CUDA*** (Python to CUDA Just-in-time compiler)

Avoid Data Transfer with cuDF and CuPy

The RAPIDS ecosystem solves the data transfer problem by enabling the execution of entire data-processing, analysis, and visualization pipelines on the GPU



RAPIDS architecture[9]

[9] devblogs.nvidia.com/gpu-accelerated-analytics-rapids/

Code Reuse with RAPIDS

The RAPIDS ecosystem makes it possible to implement
domain logic in Python, and share it across CPU and GPU code paths

Code Reuse with RAPIDS

DataFrame

- pandas and cuDF

N-dimensional Array

- numpy and CuPy

Custom Algorithms

- Numba and Numba for CUDA

Example: 1D Histogram

Shared Python domain function

```
def filter_sum(df, filter_col, filter_val, sum_col):
    filtered_df = df[df[filter_col] == filter_val]
    return filtered_df[sum_col].sum()
```

Execute on CPU with pandas

```
import pandas as pd
df = pd.read_csv("/path/to/receipts.csv")
result = filter_sum(df, 'category', 'food', 'total')
```

Execute on GPU with cuDF

```
import cudf
df = cudf.read_csv("/path/to/receipts.csv")
result = filter_sum(df, 'category', 'food', 'total')
```

Code Reuse with RAPIDS

DataFrame

- pandas and cuDF

N-dimensional Array

- numpy and CuPy

Custom Algorithms

- Numba and Numba for CUDA

Example: 1D Histogram

Shared Python domain function

```
def zscore_normalize(feature):
    shift = feature.mean(axis=0, keepdims=True)
    scale = feature.std(axis=0, keepdims=True)
    return (feature - shift) / scale
```

Execute on CPU with numpy

```
import numpy as np
feature = np.random.rand(50, 100)
normalized_feature = zscore_normalize(feature)
```

Execute on GPU with CuPy

```
import cupy
feature = cupy.random.rand(50, 100)
normalized_feature = zscore_normalize(feature)
```

Code Reuse with RAPIDS

DataFrame

- pandas and cuDF

N-dimensional Array

- numpy and CuPy

Custom Algorithms

- Numba and Numba for CUDA

Example: 1D Histogram

Shared element-wise domain function

```
@numba.jit
def increment_element_by_one(an_array, i):
    an_array[i] += 1
```

JIT-compile and execute on CPU

```
@numba.jit
def increment_by_one_cpu(an_array):
    for i in range(len(an_array)):
        increment_element_by_one(an_array, i)
```

JIT-compile and execute on GPU

```
@numba.cuda.jit
def increment_by_one_gpu(an_array):
    i = numba.cuda.grid(1)
    if i < an_array.size:
        increment_element_by_one(an_array, i)
```

Code Reuse with RAPIDS

DataFrame

- pandas and cuDF

N-dimensional Array

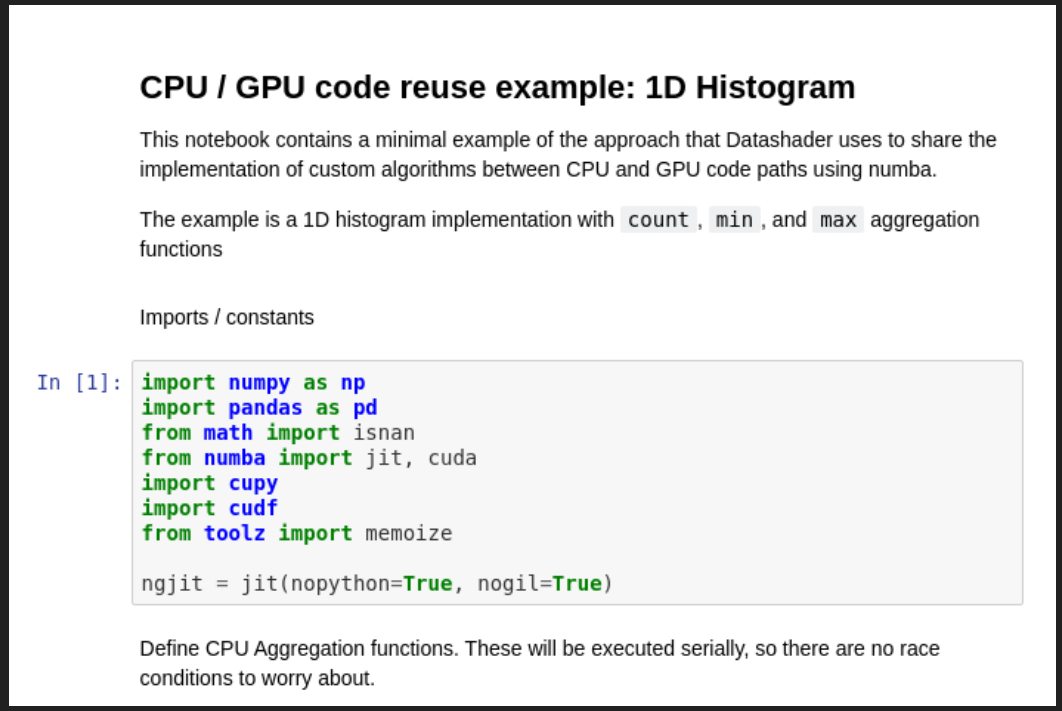
- numpy and CuPy

Custom Algorithms

- Numba and Numba for CUDA

Example: 1D Histogram

Full Example Notebook



CPU / GPU code reuse example: 1D Histogram

This notebook contains a minimal example of the approach that Datashader uses to share the implementation of custom algorithms between CPU and GPU code paths using numba.

The example is a 1D histogram implementation with `count`, `min`, and `max` aggregation functions

Imports / constants

In [1]:
```
import numpy as np
import pandas as pd
from math import isnan
from numba import jit, cuda
import cupy
import cudf
from toolz import memoize

ngjit = jit(nopython=True, nogil=True)
```

Define CPU Aggregation functions. These will be executed serially, so there are no race conditions to worry about.

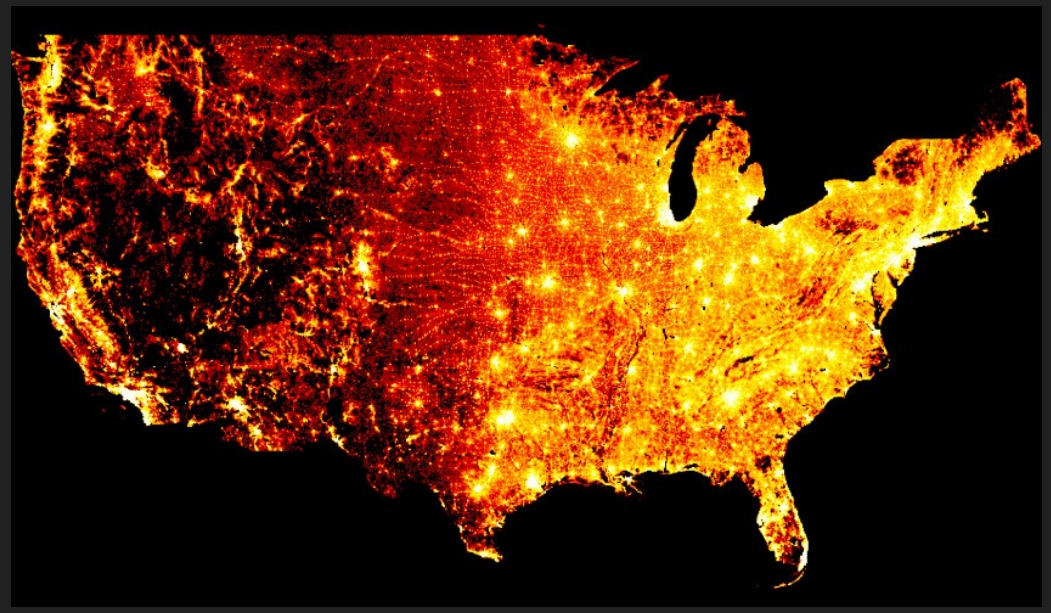
CPU/GPU code reuse example: 1D Histogram[10]

[10] anaconda.org/jonmmease/datashader-rapids-histogram-example

Datashader Performance Results

Points

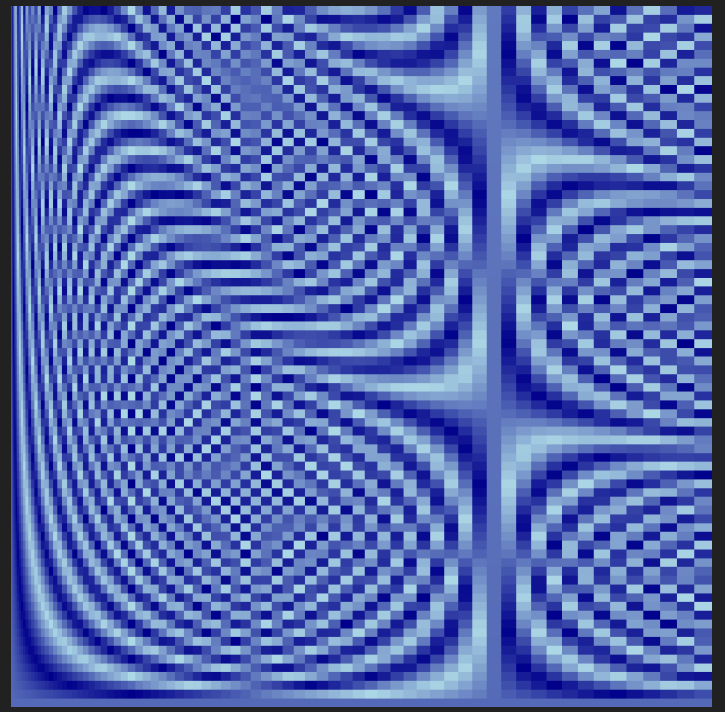
~10x speedups for Points / Lines rendering[11]



300 million data points
(one per person in the USA) from the 2010 census

QuadMesh

5x-120x speedup for QuadMesh rendering[12]

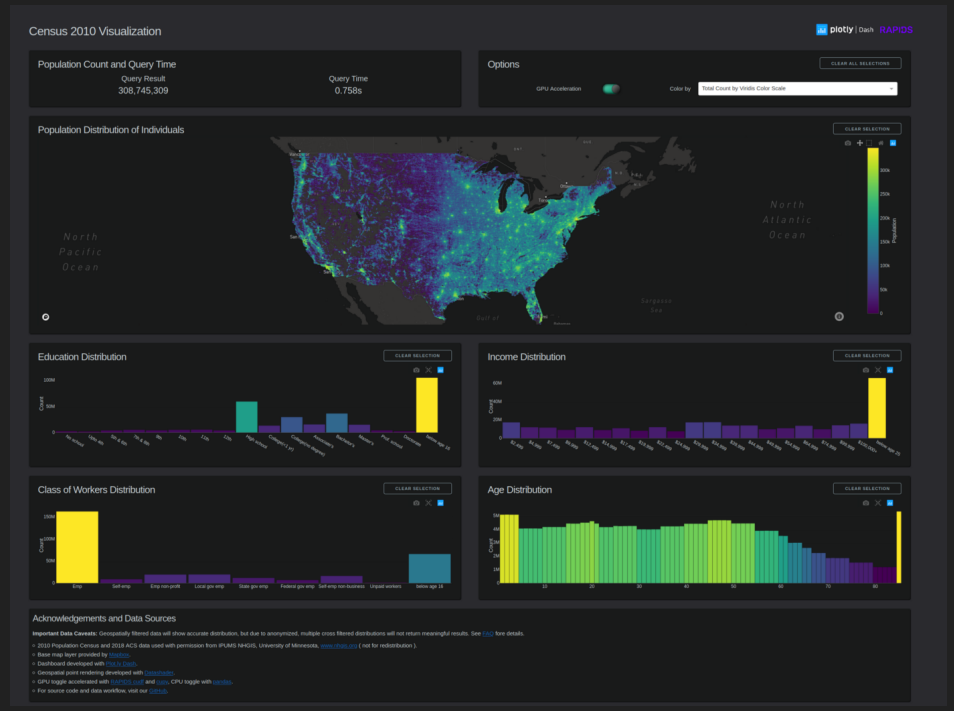


Synthetic QuadMesh pattern

[11] anaconda.org/jonmmease/datashader-rapids-census/notebook
[12] anaconda.org/jonmmease/datashader-rapids-quadmesh/notebook

Dash+RAPIDS+Datashader Example

Draft Screencast prepared by RAPIDS team



Blog post: medium.com/rapids-ai/plotly-census-viz-dashboard-powered-by-rapids-1503b3506652
Source code: github.com/rapidsai/plotly-dash-rapids-census-demo/

References

1. examples.pyviz.org/census/census.html
2. examples.pyviz.org/opensky/opensky.html
3. examples.pyviz.org/bay_trimesh/bay_trimesh.html
4. datashader.org/user_guide/Networks.html
5. datashader.org/getting_started/Pipeline.html
6. holoviews.org/user_guide/Large_Data.html
7. medium.com/rapids-ai/plotly-census-viz-dashboard-powered-by-rapids-1503b3506652
8. devblogs.nvidia.com/nvlink-pascal-stacked-memory-feeding-appetite-big-data/
9. devblogs.nvidia.com/gpu-accelerated-analytics-rapids/
10. anaconda.org/jonmmease/datashader-rapids-histogram-example
11. anaconda.org/jonmmease/datashader-rapids-census/notebook
12. anaconda.org/jonmmease/datashader-rapids-quadmesh/notebook