Digital Research Alliance of Canada

Accelerating Graph Analysis on GPUs

Jinhui Qin

SHARCNET | Compute Ontario | Digital Research Alliance of Canada

jhqin@sharcnet.ca

SHARCNET™

Compute • Calcul Ontario

What is a Graph?

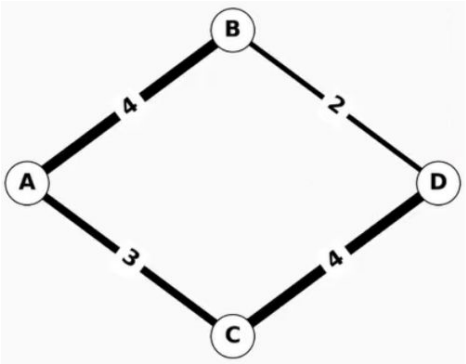
- A data structure to model relationships between data entities in a network
  - Nodes (Vertices) and edges
  - Directed, undirected, weighted, etc.

What is Graph Analytics?

- Graph Analytics helps understand complex relationships between linked entities
  - Centrality analysis
  - Community detection
  - Connectivity analysis
  - Path analysis
- Combining with ML/AI techniques, e.g GML, GNN, etc.
  - Graph classification
  - Node classification
  - Link prediction
  - etc.

NetworkX

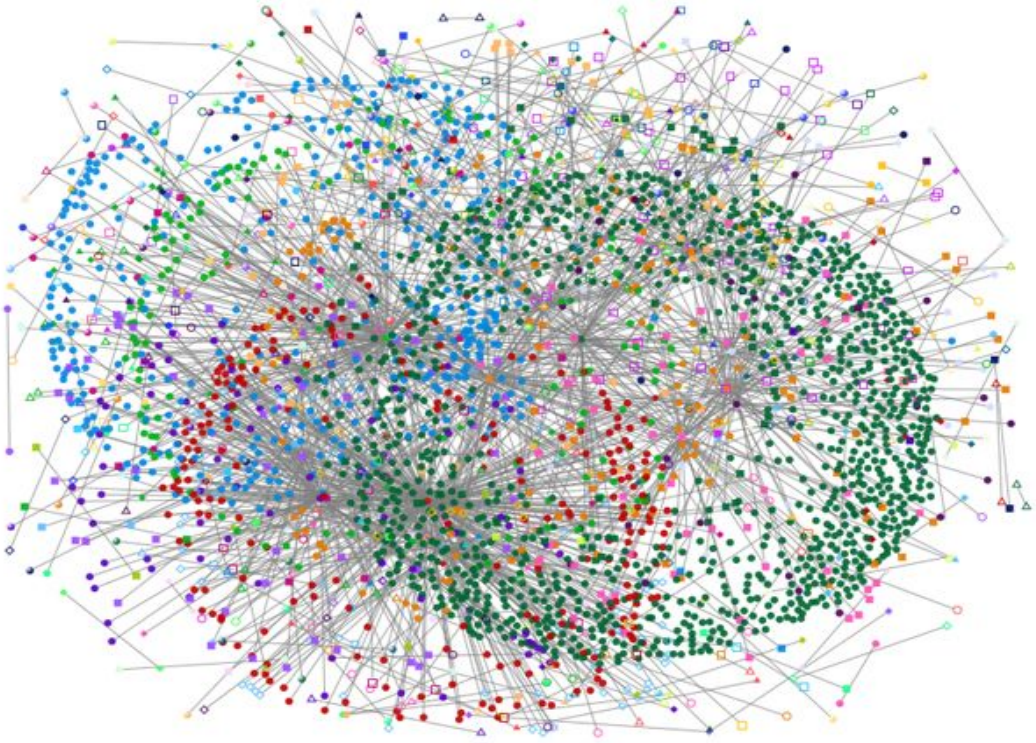
- A Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks



- The most popular python graph analytic library available
  - More than 40M PyPI downloads per month

```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edge("A", "B", weight=4)
>>> G.add_edge("B", "D", weight=2)
>>> G.add_edge("A", "C", weight=3)
>>> G.add_edge("C", "D", weight=4)
>>> nx.shortest_path(G, "A", "D",
weight="weight")
['A', 'B', 'D']
```

When dataset / graph sizes grow …



cuGraph

- Accelerating Graph analysis on GPUs
- With API similar to NetworkX
  - creating a graph
  - finding influencers
  - finding communities
  - exploring a graph
  - etc.
- Part of the RAPIDS suite
- Scalable
- Supported algorithms

```
import cugraph
import cudf
# load graph data
datafile = "./my-graph-data.csv"
gdf = cudf.read_csv(datafile,
           names=["src", "dst"],
           delimiter='\t',
           dtype=["int32","int32"])

# Creat a graph
G = cugraph.from_cudf_edgelist(gdf,
            source='src',
            destination='dst')

# call an algorithm
gdf_page = cugraph.pagerank(G)
```

CuGraph: NetworkX Compatibility

- Mimic NetworkX API
- Support NetworkX graph objects
- Suggestions
  - Replacing existing code with cuGraph as much as possible
  - Or simply update the calls to graph algorithms with cuGraph
- Differences in algorithms

```
import networkx as nx

# create a random graph
G = nx.barabasi_albert_graph(N,M)
... do some NetworkX stuff ...
...
# call nx algorithms
bc = nx.betweenness_centrality(G)
```

```
import networkx as nx
import cugraph as cnx

# create a random graph
G = nx.barabasi_albert_graph(N,M)
... do some NetworkX stuff ...
...
# call cugrpah algorithms
bc = cnx.betweenness_centrality(G)
```

nx-cugraph

- Using cuGraph as a backend to NetworkX on GPUs
  - Connects pylibcugraph and CuPy to NetworkX's API
    - pylibcugraph: a python wrapper around cuGraph low-level CUDA-based API
    - CuPy: a GPU-accelerated array library
- Setting an environment variable with zero code changes
  - Running on GPU when cuGraph and an algorithm is supported
  - Otherwise, falling back to CPU-based NetworkX
- Supported Algorithms

Enabling nx-cugraph backend

- Via an environment variable with zero code changes

```
$ NETWORKX_AUTOMATIC_BACKENDS=cugraph python my_networkx_script.py
```

- Via a keyword argument in function calls

```
import networkx as nx
...
nx.betweennees_centrality(G, k=1000, backend="cugraph")
'''
```

- Via a type-based dispatching

```
import networkx as nx
import nx_cugraph

G = nx.Graph()
...
nxcg_G = nx_cugraph.from_networkx(G)          # Graph type conversion
nx.betweenness_centrality(nxcg_G, k=1000)     # using cugraph backend
```

Work on the clusters

- Building an Apptainer container from a RAPIDS Docker container
- Both cuGraph and NetworkX are included in a RAPIDS container
  - Docs wiki: https://docs.alliancecan.ca/wiki/RAPIDS
- nx-cugraph needs to be added to a RAPIDS container
  - CUDA 11.2 or up, Python 3.9 or up, and NetworkX v.3.2 or up
  - or RAPIDS v. 23.10 or up

Adding nx-cugraph to a RAPIDS container

- Select a RAPIDS docker container from NVIDIA
- Build an Apptainer sandbox for RAPIDS
- Install nx-cugraph in the sandbox
- Convert the sandbox into an Apptainer image

```
$ apptainer build  --sandbox rapids-sandbox docker://<rapids-docker-image-tag>
$ sudo apptainer shell --writable rapids-sandbox
Apptainer> source /opt/conda/etc/profile.d/conda.sh
Apptainer> conda install -c rapidsai-nightly -c conda-forge -c nvidia nx-cugraph
Apptainer> exit
$ apptainer build rapids-nx-cugraph.sif rapids-sandbox
```

Note: Above steps need to be done on your own computer (https://apptainer.org/docs/user/latest/)
Submit a ticket (help@sharcnet.ca) for help if needed.

A RAPIDS-nx-cugraph container

- Based on a RAPIDS docker container from NVIDIA:
  - Docker image tag: nvcr.io/nvidia/rapidsai/notebooks:24.02-cuda11.8-py3.10
  - RAPIDS v.24.02 with a notebook server on Ubuntu 20.04
  - Working with CUDA 11.8 and Python 3.10
- Image file, rapidsas-24.02-nx-cugraph.sif, is available at:
  - https://staff.sharcnet.ca/jhqin/GIS-cuGraph/

Demo example

A large graph

- a citation graph of a U.S. patent dataset
- ~ 4 million nodes, and 16+ million edges
- Compute *betweenness centrality* with approximation
- Comparing the performance
  - NetworkX vs. cuGrapah vs. nx-cugraph

Betweenness-Centrality

A measure of the relative importance of a node or an edge in a graph

- counting the number of shortest paths that pass through a node (or an edge) vs total number of shortest paths for all node pairs

$$BC(v) = \sum_{s,t \in V} \frac{\sigma(s,t \mid v)}{\sigma(s,t)}$$

- Function in NetworkX

  nx.betweenness_centrality(*G*, *k*, …)

  - k, int, optional (default=None); k ≤ the total number of nodes;
  - higher k gives better approximation

Code examples: NetworkX vs cuGraph

```python
# nx-bc-demo.py
import sys
import time
import networkx as nx
import pandas

k = int(sys.argv[1])

# Reading dataset into Pandas DataFrame as an edgelist...
pandas_edgelist = pandas.read_csv( "cit-Patents.txt",
    skiprows=4, delimiter="\t", names=["src", "dst"],
    dtype={"src": "int32", "dst": "int32"})

# Creating Graph from Pandas DataFrame edgelist...
G = nx.from_pandas_edgelist(pandas_edgelist, source="src",
    target="dst",  create_using=nx.DiGraph)

# Calculating betweenness_centrality
st = time.time()
bc_result = nx.betweenness_centrality(G, k=k)
print(f"BC time with {k=} was: {(time.time() - st):.2f} s")
```

```python
# cg-bc-demo.py
import sys
import time
import cugraph as cg
import cudf

k = int(sys.argv[1])

# Reading dataset into CuDF DataFrame as an edgelist...
cudf_edgelist = cudf.read_csv( "cit-Patents.txt",
    skiprows=4, delimiter="\t", names=["src", "dst"],
    dtype={"src": "int32", "dst": "int32"})

# Creating Graph from cuDF DataFrame edgelist...
G = cg.from_cudf_edgelist(cudf_edgelist, source="src",
    destination ="dst", create_using=cg.Graph(directed=True))

# Calculating betweenness_centrality
st = time.time()
bc_result = cg.betweenness_centrality(G, k=k)
print(f"BC time with {k=} was: {(time.time() - st):.2f} s")
```

Submit a batch job

Demo directory contents:

```
[jhqin@gra-login1 Demo]$ tree
.
├── submit-job.sh                    # slurm job script
└── workdir                          # to be copied to a compute node
    ├── cg_bc_demo.py                # a demo cugraph python script
    ├── cit-Patents.txt              # a large graph dataset
    ├── nx_bc_demo.py                # a demo networkX python script
    ├── rapidsai-24-nx-cugraph.sif   # container image file
    └── run-job.sh                   # script to run in the container
```

Submit job:

```
[jhqin@gra-login1 Demo]$ sbatch submit-job.sh
```

Submit a batch job

Slurm script:

submit-job.sh

```
#!/bin/bash
#SBATCH --gres=gpu:1              # gpu request
#SBATCH --cpus-per-task=N         # number of CPU cores
#SBATCH --mem=M                   # host memory
#SBATCH --time=DD-HH:MM           # execution time
#SBATCH --account=def-user        # project account

module load StdEnv/2023 apptainer

src_dir=</full/path/to>/workdir
cp -r  $src_dir $SLURM_TMPDIR           # copy workdir to local disk on compute node

work_dir=$SLURM_TMPDIR/workdir
cd $work_dir
container=rapidsai-24-nx-cugraph.sif    # included in workdir

apptainer exec --nv $container  $work_dir/run-job.sh     # launching job execution
```

Submit a batch job

Script to run in the container

run-job.sh

```
#!/bin/bash
source /opt/conda/etc/profile.d/conda.sh
nvidia-smi

echo "=======networkX test========="
python nx_bc_demo.py 10
python nx_bc_demo.py 50

echo "=======cuGraph test========="
python cg_bc_demo.py  10
python cg_bc_demo.py  50
python cg_bc_demo.py  500

echo "=======nx-cugraph test========="
NETWORKX_AUTOMATIC_BACKENDS=cugraph python nx_bc_demo.py 10
NETWORKX_AUTOMATIC_BACKENDS=cugraph python nx_bc_demo.py 50
NETWORKX_AUTOMATIC_BACKENDS=cugraph python nx_bc_demo.py 500
```

Note:

run-job.sh should be *executable*.

```
[jhqin@gra-login1 Demo]$ chmod +x workdir/run-job.sh
```

Reference

- CuGraph documentation: https://docs.rapids.ai/api/cugraph/stable/
- CuGraph notebooks: https://github.com/rapidsai/cugraph/tree/main/notebooks
- RAPIDS 23.10 Release: https://medium.com/rapids-ai/rapids-23-10-release-075aa5a50570
- RAPIDS Docker Containers: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/rapidsai/containers/notebooks
- NVIDIA Technical Blog (Data Science): Accelerating NetworkX on NVIDIA GPUs for High Performance Graph Analytics, Nov. 2023 by Rick Ratzel
- NVIDIA Technical Blog (Data Science): Beginner's Guide to GPU Accelerated Graph Analytics in Python, Mar. 2021 by Tom Drabas
- Medium Blog: Introduction to Graph Analysis using cuGraph, Jul. 2023 by Don Acosta
- Medium Blog: Intro to Graph Analysis using cuGraph: Similarity Algorithms, Oct. 2023 by Don Acosta