Building backends using `entry_points` and NetworkX's parallel backend

Aditi Juneja

Building backends using `entry_points` and NetworkX's parallel backend

Aditi Juneja

What are backends?

Def. : The part of a software system that is not usually visible or accessible to a user of that system

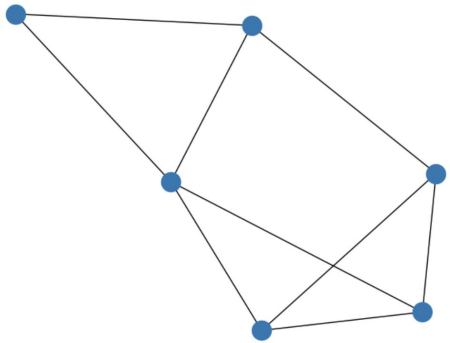
What are backends?

Def. : The part of a software system that is not usually visible or accessible to a user of that system

Basically, something in the back

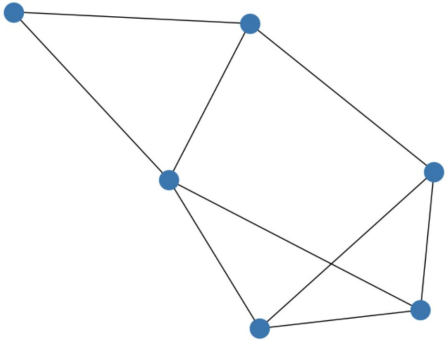
About NetworkX

- A pure Python library with around 600 graph algorithms
- Create a graph object using `nx.Graph()` and play around with it!
- Apply different algorithms and know your graph better :)



About NetworkX

- A pure Python library with around 600 graph algorithms
- Create a graph object using `nx.Graph()` and play around with it!
- Apply different algorithms and know your graph better :)
- But, with very large graphs....



Solution?

Solution?

- Large number of issues/PRs proposing adding parallel implementations
    - Adds inconsistency
    - Adds additional dependencies
- Graph libraries in different languages - C, Rust…
- Graph libraries supporting GPUs (cu-graphs)
- Using a different graph object (GraphBLAS)

Solution:

Plug-in a backend and dispatch(redirect) the function call to the alternate implementation in the backend!

Solution:

Plug-in a backend and dispatch(redirect) the function call to the alternate implementation in the backend!

HOW?

```python
@py_random_state(5)
@nx._dispatchable(edge_attrs="weight")
def betweenness_centrality(
    G, k=None, normalized=True, weight=None, endpoints=False, seed=None
):
    r"""Compute the shortest-path betweenness centrality for nodes.

    Betweenness centrality of a node $v$ is the sum of the
    fraction of all-pairs shortest paths that pass through $v$
```

Inside NetworkX

```python

@py_random_state(5)
def betweenness_centrality(
    G,
    k=None,
    normalized=True,
    weight=None,
    endpoints=False,
    seed=None,
    get_chunks="chunks",
):
    """The parallel computation is implemented by dividing the nodes into chunks and
    computing betweenness centrality for each chunk concurrently.
```

Inside nx-parallel backend

```python
@py_random_state(5)
@nx._dispatchable(edge_attrs="weight")
def betweenness_centrality(
    G, k=None, normalized=True, weight=None, endpoints=False, seed=None
):
    r"""Compute the shortest-path betweenness centrality for nodes.

    Betweenness centrality of a node $v$ is the sum of the
    fraction of all-pairs shortest paths that pass through $v$
```

Inside NetworkX

```python

@py_random_state(5)
def betweenness_centrality(
    G,
    k=None,
    normalized=True,
    weight=None,
    endpoints=False,
    seed=None,
    get_chunks="chunks",
):
    """The parallel computation is implemented by dividing the nodes into chunks and
    computing betweenness centrality for each chunk concurrently.
```

Inside nx-parallel backend

```python
@py_random_state(5)
@nx._dispatchable(edge_attrs="weight")
def betweenness_centrality(
    G, k=None, normalized=True, weight=None, endpoints=False, seed=None
):
    r"""Compute the shortest-path betweenness centrality for nodes.

    Betweenness centrality of a node $v$ is the sum of the
    fraction of all-pairs shortest paths that pass through $v$
```

Inside NetworkX

```python

@py_random_state(5)
def betweenness_centrality(
    G,
    k=None,
    normalized=True,
    weight=None,
    endpoints=False,
    seed=None,
    get_chunks="chunks",
):
    """The parallel computation is implemented by dividing the nodes into chunks and
    computing betweenness centrality for each chunk concurrently.
```

Inside nx-parallel backend

From A User's perspective...

1. Type-based

```
H = nxp.ParallelGraph(G)
nx.betweenness_centrality(H)
```

2. Using a `backend` kwarg

```
nx.betweenness_centrality(G, backend="parallel")
```

3. Environment variable

```
export NETWORKX_AUTOMATIC_BACKENDS=parallel && python nx_code.py
```

4. As a `config` parameter(WIP)

```
with nx.config(backend="parallel"):
   nx.betweenness_centrality(G)
```

Plug-in? - Python `entry_points`

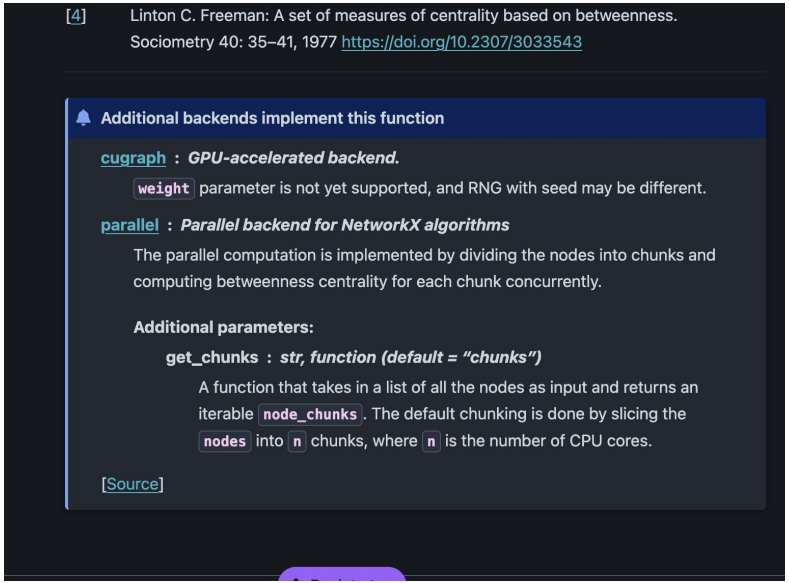
An entry point is defined by three main properties:

- **Group**: Indicates what type of object the entry point provides.
- **Name**: Identifies the entry point within its group.
- **Object Reference**: Points to a Python object, either in the form `importable.module` or `importable.module:object.attr`. This is used to look up the actual object at runtime.

Ref. https://packaging.python.org/en/latest/specifications/entry-points/

NetworkX's `entry_points`

1. `networkx.backends`
   a. Backend interface
2. `networkx.backend_info`
   a. Additional docs
   b. Default configs




```
[project.entry-points."networkx.backends"]
parallel = "nx_parallel.interface:BackendInterface"

[project.entry-points."networkx.backend_info"]
parallel = "_nx_parallel:get_info"
```

Parallel backend (nx-parallel)

- Uses `joblib.Parallel`
- “Chunking” and its limitations?
- Config

Ref. https://joblib.readthedocs.io/en/latest/parallel.html

Backend Testing

```
PYTHONPATH=. \
NETWORKX_TEST_BACKEND=parallel \
NETWORKX_FALLBACK_TO_NX=True \
    pytest --pyargs networkx "$@"
```

How can “you” use all this?

1. If you are a pure Python Library owners/maintainers…..
2. If you want to distribute your graph analysis implementations but UI limitations…..
3. If you are a potential open-source contributors…..

To Know More:

1. nx-parallel GitHub : https://github.com/networkx/nx-parallel
2. NetworkX's backend and config docs : https://networkx.org/documentation/latest/reference/backends.html
3. Python entry_points : https://packaging.python.org/en/latest/specifications/entry-points/
4. Embarrassingly parallel and joblib.Parallel : https://joblib.readthedocs.io/en/latest/parallel.html
5. Some blog (by me) documenting work on nx-parallel : https://github.com/Schefflera-Arboricola/blogs/tree/main/networkx/GSoC24
6. NetworkX's Tutorial : https://networkx.org/documentation/latest/tutorial.html
7. NetworkX's backend dispatching code : https://networkx.org/documentation/latest/tutorial.html
8. Some more interesting networkx backends : nx-cugraphs, python-graphblas, arangodb, visualisation backend, etc.

About Me

- **Who I am?**
  - Core developer at NetworkX
  - GSoC’24 Contributor at NumFOCUS(NetworkX)
- **Who I was?**
  - Independent Contractor at NumFOCUS(NetworkX)

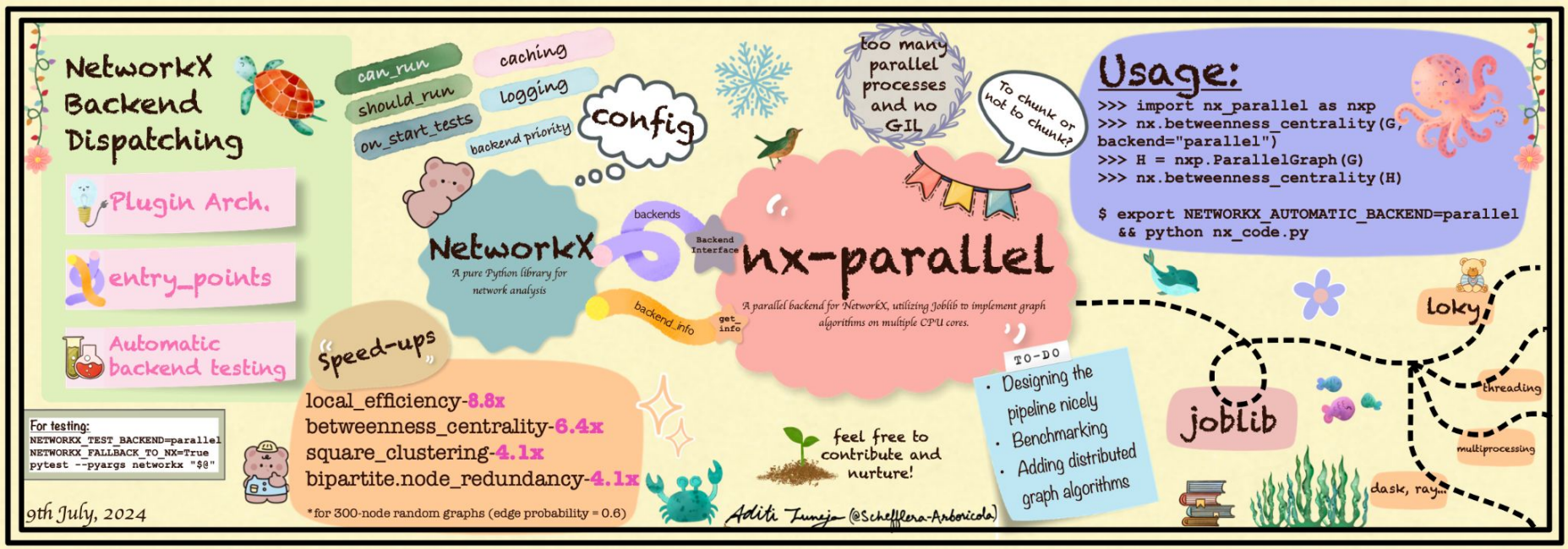
My Work

What I am doing?

- Working on adding config and setting the pipeline currently and documenting config's usage
- Improving benchmarking in nx-parallel
- Automating updation of things like `get_info`'s output, etc. within pre-commit hooks

What I have done?

- Maintaining nx-parallel
    - added 15 new parallel graph algorithms with a maximum of 8x speed-up and minimum of 2x
    - Improved old infrastructure and performance of already existing algorithms
    - setup the ASV benchmarking infrastructure
    - experimented with "chunking" and included the "get_chunks" utility
    - maintaining nx-parallel's conda feedstock
- Documented most of the networkx's backend dispatching portion
- Participating in general discussions, PR reviews and activities in networkx and networkx backends communities


NetworkX Backend Dispatching
Plugin Arch.
entry_points
Automatic backend testing
For testing:
NETWORKX_TEST_BACKEND=parallel
NETWORKX_FALLBACK_TO_NX=True
pytest --pyargs networkx "$@"
9th July, 2024
can_run
should_run
on_start_tests
caching
logging
backend priority
config
NetworkX
A pure Python library for network analysis
backends
Backend Interface
backend_info
get_info
too many parallel processes and no GIL
To chunk or not to chunk?
nx-parallel
A parallel backend for NetworkX, utilizing Joblib to implement graph algorithms on multiple CPU cores.
Usage:
>>> import nx_parallel as nxp
>>> nx.betweenness_centrality(G, backend="parallel")
>>> H = nxp.ParallelGraph(G)
>>> nx.betweenness_centrality(H)
$ export NETWORKX_AUTOMATIC_BACKEND=parallel && python nx_code.py
Speed-ups
local_efficiency-8.8x
betweenness_centrality-6.4x
square_clustering-4.1x
bipartite.node_redundancy-4.1x
*for 300-node random graphs (edge probability = 0.6)
feel free to contribute and nurture!
TO-DO
Designing the pipeline nicely
Benchmarking
Adding distributed graph algorithms
joblib
loky
threading
multiprocessing
dask, ray...
Aditi Juneja (@Schefflera-Arboricola)

Thank you :)

How to make it fast?

-

What do you understand by “Backends”?