nglpy-cuda Documentation

Dan Maljovec

Dec 24, 2020

Contents:

CHAPTER 1

nglpy-cuda

A reimplementation of the Neighborhood Graph Library (NGL) developed by Carlos Correa and Peter Lindstrom that supports pruning a graph on the GPU. Developed as a replacement for nglpy where a CUDA-compatible GPU is available.

Given a set of arbitrarily arranged points in any dimension, this library is able to construct several different types of neighborhood graphs mainly focusing on empty region graph algorithms such as the beta skeleton family of graphs.

Consider using an optimized approximate nearest neighbor library (see ann-benchmarks for an updated list of algorithms and their relative performance) to construct the initial graph to be pruned, otherwise this library will rely on the exact k-nearest algorithm provided by scikit-learn.

1.1 Prerequisites

Nvidia CUDA Toolkit (TODO: determine minimum version number) - tested on 9.1.

Otherwise, all other python requirements can be installed via pip:

```
pip install -r requirements.txt
```

1.2 Installation

There is an experimental package available on pip, however the prerequisite libraries are not specified correctly, so be sure you have numpy, scipy, sklearn, and faiss installed (subject to change).

```
pip install nglpy_cuda
```

1.3 Build

1.3.1 Building the Python package

For now, don't install this yet, but set it up in development mode:

```
python setup.py develop
```

Run the test suite to verify it is able to make the CUDA calls without erroring:

```
python setup.py test
```

From here you should be ready to use the library. Only proceed below if you run into some install issues and want to try to at least build the shared library that you can use in C/C++ applications.

1.3.2 Building and Testing the CUDA Library Separately

Until I get this packaged appropriately, use the following command to compile the CUDA code:

```
nvcc src/ngl_cuda.cu -I include/ --compiler-options "-fPIC" --shared -o libnglcu.so
```

The CUDA API can then be tested with a small C++ example (TODO: provide small data file in repo for testing this next line):

```
g++ -L. -I include/ src/test.cpp -lnglcu -o test
./test -i <input file> -d <# of dimensions> -c <# of points> -n <neighbor edge file> -
↪k <k neighbors to prune> -b <beta parameter> -p <shape descriptor> -s
↪<discretization steps> -r <positive integer means use the relaxed version>
```

1.4 Usage

The Python interface exposes the a Graph object that can be be iterated over its edges which produces a tuple where the first two values are the integer indices and the third value is the distance between the two points:

```
import numpy as np
import nglpy_cuda as ngl

X = np.random.uniform(size=(10, 2))
graph = ngl.Graph(X, relaxed=False)

for edge in graph:
    print(edge)
```

[~ Dependencies scanned by PyUp.io ~]

CHAPTER 2

Installation

2.1 Stable release

To install nglpy-cuda, run this command in your terminal:

```
$ pip install nglpy_cuda
```

This is the preferred method to install nglpy-cuda, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for nglpy-cuda can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/maljovec/nglpy_cuda
```

Or download the tarball:

```
$ curl  -OL https://github.com/maljovec/nglpy_cuda/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use nglpy-cuda in a project:

```
import nglpy_cuda
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/maljovec/nglpy_cuda/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

nglpy-cuda could always use more documentation, whether as part of the official nglpy-cuda docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/maljovec/nglpy_cuda/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *nglpy_cuda* for local development.

1. Fork the *nglpy_cuda* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/nglpy_cuda.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv nglpy_cuda
$ cd nglpy_cuda/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 nglpy_cuda tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/maljovec/nglpy_cuda/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_nglpy_cuda
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- Daniel Patrick Maljovec <maljovec002@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.2.0 (2018-10-03)

- Addition of probabilistic graphs
- Minor changes to API structure, now you must call build after initializing an object

6.2 0.1.0 (2018-07-15)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search