Project Proposal for 15-418, Spring 2018:

A Parallel Sketching Library for GPUs

Eliot Robson erobson@andrew.cmu.edu
Taisuke Yasuda taisukey@andrew.cmu.edu

April 9, 2018

Abstract

In this course project, we will provide sequential and parallel implementations of various sketching algorithms and their applications. The sequential code will be written in C++ and the parallel code will be written for NVIDIA GPUs.

1 Background

Sketching algorithms are a class of randomized approximation algorithms designed to approximate large matrices with ones with less rows and/or columns. These algorithms are known to have provable approximation guarantees with high probability and can be applied to countless downstream applications for asymptotically faster approximation algorithms, including but most certainly not limited to linear regression, low rank approximations, and preconditioning. We refer the audience to a monograph by David Woodruff for an excellent overview of sketching algorithms [W+14], especially as applied to linear algebraic operations.

Some of the sketching algorithms we may implement are

- Gaussian sketch
- subsampled randomized Hadamard transform
- count sketch
- leverage score sampling sketch

and some of the applications we may implement are

- linear regression
- k rank approximation
- k means clustering
- principal component analysis

Note that the above is just a short list of some representative sketching algorithms and applications, and we could list out dozens more for each.

2 The Challenge

We expect the main challenge of the project to be the initial implementation of correct sequential versions of our algorithms, and the subsequent optimization on GPUs. Although the algorithms themselves are not incredibly complex, deciding on a framework to work in and setting everything up may be difficult in the first phase of the project. We expect to make use of existing code, as described in section 3, as well as libraries for linear algebraic operations. However, we still expect this to be one of the more challenging parts of the project. After the set up, the main objective and challenge of the project will be speeding up the code on GPUs. As described before, some parallelization strategies are known already. However, parallel programming on GPUs is an art that can be wildly different from parallelization strategies for distributed computation, and we expect this to take the majority of our time on this project.

3 Resources

3.1 Implementation Resources

Sketching algorithms for streaming problems have been implemented successfully, most notably (and the only one to our knowledge) by Yahoo in their Java sketching library, Data Sketches (`https://datasketches.github.io/`). Although this library supports parallel computation in the form of distributed computation, it has not yet been parallelized at the level of GPUs. Other attempts (`https://github.com/jaykar/matrix_sketching`) have been made by smaller organizations to implement these sketching algorithms on the GPU, to our knowledge, they have not been successful.

The two projects mentioned above give us an excellent platform from which we can kick off our project. Although Data Sketches does not allow us to check correctness since it is a randomized algorithm whose seed we cannot set, it gives us an excellent reference implementation for a sequential version of some of the algorithms we are interested in with respect to the speedup we may achieve. As far as we can tell, Data Sketches implements the heavy hitters algorithm, which we can compare to our implementation. The Data Sketches documentation mentions the frequent directions algorithm, which is equivalent to the low rank approximation problem, but we were unable to find the code for it at the time of this writing.

In addition to providing reference implementations, the website for Data Sketches also includes a discussion for the parallelization tricks used to support distributed computation, so we can base some of our strategies off of their reports.

The unsuccessful GPU code included a C++ implementation of some of the sketching algorithms we are interested in, which helps speed up the process of developing sequential versions of our code. The code included fragments of code for count sketch, Gaussian projections, linear regression, and rank k singular value decomposition. Some of these were incomplete and the library was missing some subroutines, as far as we could tell. This is not a professionally maintained project, and we may have to redo all of this; we will confirm as soon as our testing framework is set up.

3.2 Theoretical Resources

As a reference for our algorithms and applications, we expect to be mostly referring to the monograph [W+14], as well as course material from Algorithms for Big Data (Carnegie Mellon University Fall 2017 15-859, `http://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/teaching/15859-fall17/`) and Machine Learning for Large Datasets (Carnegie Mellon University Spring 2018 10-405, `http://www.cs.cmu.edu/afs/cs/user/dwoodruf/www/teaching/15859-fall17/`).

4 Goals and Deliverables

4.1 Core Achievements

As a baseline goal for the project, we plan to implement at least two sketching algorithms (e.g. count sketch and subsampled randomized Hadamard transform) and at least three applications of the sketching algorithms (e.g. linear regression, k rank approximation, and principal component analysis). This includes a full testing framework, fast sequential implementations, and optimized blazing fast GPU implementations. These algorithms were designed to speedup computations on extremely large data sets, so we might hope to achieve around $100\times$ speed up for sufficiently sized inputs.

4.1.1 Benchmarking Tests

We will be evaluating our performance on sketches by running our code on both synthetic and realistic test cases. We have four different sources of test cases in mind:

- randomly generated matrices (that we produce)
- small and large images downloaded from the internet
- matrices from the UF sparse matrix collection (`https://www.cise.ufl.edu/research/sparse/matrices/list_by_id.html`), used by experiments in the Clarkson and Woodruff count sketch paper [CW13]
- datasets from the UCI machine learning repository used by experiments in the Musco and Musco Nyström sampling paper [MM17] (`https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD`, `https://archive.ics.uci.edu/ml/datasets/Adult`, `https://archive.ics.uci.edu/ml/datasets/Covertype`

4.2 Extra Achievements

There are a few different directions in which we could extend our project. One direction is just to implement and optimize more sketching algorithms and applications. Another could be to do some analysis on the various sketching algorithms that we implement, comparing how the algorithms scale and which algorithms are best at which scales. Yet another direction could be to put in more software engineering effort and expose our CPU and GPU code to other frameworks, for example by providing a Python library (like NumPy and TensorFlow).

4.3 Poster Presentation

For the final poster presentation, we will likely be presenting demos of our code on images, as this makes for the most visually appealing application of the algorithms that we implement, as well as charts and graphs that show our speedups relative to our sequential implementations as well as existing sequential implementations. We could also compare our algorithms to exact algorithms for the tasks that we solve.

5 Platform Choice

We will be developing and testing on GPUs on the GHC machines.

6 Schedule

Below is our proposed schedule. The (∗) indicates a course deadline.

(∗)Proposal Checkpoint	April 4
Finish researching existing code	April 6
Decide on algorithms to implement	April 8
(∗)Proposal Due	April 9
Testing framework for all algorithms	April 15
Sequential code of all algorithms	April 18
(∗)Project Checkpoint I	April 18
Correct GPU code of all algorithms	April 24
Optimized I GPU code of all algorithms	April 27
(∗)Project Checkpoint II	April 27
Optimized II GPU code of all algorithms	May 1
Decide on extra achievements	May 2
Extra achievements	May 5
Project Report and Poster	May 7
(∗)Report Due	May 7
(∗)Poster Session	May 8

References

[CW13] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2013.

[MM17] Cameron Musco and Christopher Musco. Recursive sampling for the nystrom method. In *Advances in Neural Information Processing Systems*, pages 3836–3848, 2017.

[W+14] David P Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.