Final Project Report for 15-418, Spring 2018:

A Parallel Sketching Library for GPUs

Eliot Robson Taisuke Yasuda

erobson@andrew.cmu.edu taisukey@andrew.cmu.edu

May 7, 2018

Abstract

In this course project, we implemented sketching algorithms and applications on CPU in C++ and in CUDA on the GPU.

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.

1.1 Leverage Score Sampling

The algorithm for fast approximation of leverage scores is given by

- Compute a count sketch SA of the input matrix A
- Compute a factorization $SA = QR^{-1}$ so that Q is orthonormal
- Compute R, the inverse of R^{-1}
- Compute a Gaussian sketch RG of R
- Compute $A(RG)$
- Compute the row norms times a scalar factor

and the sketch is just sampling proportionally to these scores.

2 Approach

2.1 Leverage Score Sampling

At first glance, the algorithm seems to suffer from a lot of dependencies, as the algorithm involves a lot of matrix operations both from the left and right. Another level of complexity is added by the fact that CUBLAS, the CUDA basic linear algebra subroutines library, is written to execute everything in column major order. We carefully describe from an implementation point of view how to milk out all the independent components to this algorithm.

2.1.1 Sampling Gaussians

First note that we can start the expensive sampling of the Gaussian matrix G at the very beginning in parallel and hide the latency using the time it takes to do the expensive linear algebraic computations that forms the core of the algorithm. In addition, although we may simply sample the matrix as a $d \cdot O(\log n)$ matrix, we may access the columns in any order that is friendly to the cache as long it is consistent with using a permutation of the Gaussian matrix, since the entries are just IID random variables. We will come back to this later, so keep this in mind.

2.1.2 QR Decomposition

We describe our strategy for computing R^{-1} efficiently. The most important observation is that we can factor the matrix SA using a QR factorization algorithm, which gives us an orthonormal matrix Q as desired as well as the matrix R^{-1} that is *upper triangular*. Although this was likely an obvious observation for researchers in the numeric linear algebra community, it took us a while to make use of this fact since we learned about this algorithm in a theory class, which only emphasized the fact that Q was orthonormal (the more important fact when proving correctness). We make use of the CUSOLVER routine `geqrf` to compute this QR factorization. Note that when computing R, we just want to solve for the matrix X such that

$$R^{-1}X = I.$$

This is efficient when R^{-1} is upper triangular, since the computation is now just back substitution to solve for a single column. Since CUBLAS has an efficient implementation of solving triangular systems of equations, `trsm`, we use this.

2.1.3 Load-Balancing the Computation of R

However, our profits from doing the QR factorization go beyond just this. Observe that when solving the upper triangular system $R^{-1}x = e_i$ in order to find the ith column of R, we immediately find that $x_j = 0$ for all $i < j \leq n$ (this is used to show that the inverse of an upper triangular matrix is also upper triangular). Thus, the computation of the ith column of R only depends on the first i columns of R^{-1}. Fortunately, CUBLAS operates in column major order, so we can efficiently copy out the first i columns, as these are contiguous in memory. This also helps us do some load balancing: we know that the last columns of R take longest to compute, while the first columns are instantaneous. Thus, the best way to assign the work is as follows: if we have T threads computing R, then we should divide the columns of R into $2T$ blocks $B_1, B_2, \ldots, B_{2T}$, and assign blocks B_t and B_{2T-t+1} to thread $t \in [T]$.

2.1.4 Split Counters and G Layout Tricks for Computation of RG

At this time, recall that the columns of R corresponding to column blocks B_t and B_{2T-t+1} reside on thread t. The next step is to compute RG. This is the step in which we play with the interpretation of the matrix G. Note that each of the $O(\log n)$ columns of G specifies a linear combination of the columns of R. Then, we can precompute the partial sum of this linear combination, just for the columns of B_t and B_{2T-t+1} that reside on thread t, and put them together at the end by linearity, similar to how split counters can be used to accumulate partial counts. Then, to do this, we only see the rows of G corresponding to B_t and B_{2T-t+1}, and furthermore, we can take them to be contiguous in memory, and even only resident on thread t since these rows are only used by thread t. Thus, we can minimize communication between threads this way.

2.1.5 Computation of Approximated Leverage Scores

In the final step for computing the approximated leverage scores, we must combine RG into one matrix since the row norms of ARG must be computed. However, once RG is computed, A can be split up into row blocks and the ith leverage score estimate

$$\ell_i := \frac{\beta}{d} \left\| (e_i^\top A)(RG) \right\|^2$$

can be computed independently. In addition, we will need the CDF for sampling later, for which we can precompute the partial sums as in split counters.

2.1.6 Sampling and Applying the Sketch

The last step is to sample according to the distribution and apply the sketch, but these are easily parallelized.

3 Results

4 Division of Work

Equal work was performed by both project members.

References

[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.