Cache Friendly Shuffles for Machine Learning

May 8, 2016

1 Overview

2 Least Squares

3 Word Embeddings

3.1 Introduction

In the word embeddings problem, given context counts $X_{w,w'}$ we want to find word vectors $v_w \in \mathbb{R}^k$ that minimizes the loss:

$$min_{v,C} \sum_{w,w'} X_{w,w'}(log(X_{w,w'}) - \|v_w + v_{w'}\|^2 - C)^2$$

3.2 Experiment Details

We ran our experiments on the Edison compute nodes which feature two twelve core 2.4 GHz processors. However, we used only up to twelve cores/threads to avoid effects of NUMA. Word vectors were length 100 double arrays.

We used the first 10^9 bytes of English Wikipedia from http://mattmahoney.net/dc/textdata as corpus data. After running the text preprocessing script supplied by the link, we computed co-occurrence counts of pairs of distinct words to create the parameter dependence graph. This graph was then fed into gpmetis, computing a min-k-cut partitioning to create a cache-friendly ordering of the datapoints. k was set such that each block of k datapoints would reference just enough word vectors to fit into the L1-cache.

Hogwild was then run on the permuted co-occurrence graph generated by gpmetis, maintaining the same ordering throughout execution. Although we experimented with both data sharding and no-data sharding, only results from data sharding are presented. To test hogwild without a cache-friendly shuffle, we randomly shuffled the datapoints before execution.

We also ran the experiments on subsets of the corpus, repeating the procedure on the first %10, %25, %50 and %75 of the corpus data. In the full corpus data, there were $200,000$ word vectors, and $30,000,000$ datapoints.

3.3 Results

We achieve between %40 – %50 speedup over hogwild without a cache friendly permutation, measuring runtime to a fixed number of epochs. (Image for speedup with full 80 mb)

Furthermore, the speedup is maintained on different subsets and sizes of the data. (Image for time with different portions of data)

Additionally, convergence of loss is not adversely affected. (Image time loss graph)

3.4 Analysis

A %40 – %50 runtime gain over regular hogwild is a result of keeping at least one length 100 double array in the L1-cache between stochastic gradient calls. In a non-cache-friendly

permutation, each of the two vectors visited by a datapoint is typically not in the cache, incurring two vectors worth of cache misses per datapoint. After running min-k-cut on the parameter dependence graph, we found that each block of k datapoints references around k distinct vectors. Thus, in a cache-friendly permutation, one of the vectors referenced by a datapoint is already in the L1-cache from the previous stochast gradient call. So a cache-friendly permutation incurs only one vectors worth of cache misses per datapoint, naturally leading to a %40 – %50 reduction in runtime.

4 Conclusion

5 Future