I. MOTIVATION

The libSkylark\(^1\) library provides sketching-based matrix computations for machine learning suitable for general statistical data analysis and optimization applications.

II. RANDOMNESS

Our requirements are random access to multiple potentially non-overlapping statistically independent stream of random numbers. In a parallel environment we require the random access to automatically adapt to varying process grid configurations.

Unfortunately most conventional pseudorandom number generators (PRNGs) \([8]\) are inherently sequential. This means that generating only a subset of the pseudorandom stream (e.g. a striped view) in parallel would go through the unnecessary computation of all intermediate samples to be discarded. This is a severe performance penalty, dramatically shrinking the envelope for scalability.

A. Counter based random access pseudo-random streams are the solution infrastructure

We will turn to counter-based PRNGs (CBRNGs for short), functions of the form: \(x_n = b_k(n)\). Here \(n\) is a counter, \(k\) is a key, both integers, and \(b_k()\) is a member of a family of bijections. CBRNGs are inherently parallelizable: one can implement either the multistream approach (each stream with different keys \(k\)) and the substream approach (fixing \(k\) and associating different streams with non-overlapping, continuous ranges of counter ranges \(n_i...n_j\) ranges). Both multistream and substream approaches support random access patterns. Since \(n\) can be arbitrary, within the limits imposed by counter ranges, random access is readily established. Random123 \([10]\) provides a high performance implementation of CBRNGs.

B. From random access pseudo-random streams to matrices of random samples in parallel

Given counter and key, a single invocation of the CBRNG yields a pair of 64-bit numbers that we can regard as a uniformly random 128-bit unsigned integer. For our sketching matrices we need to generate entries that are samples drawn from various statistical distributions. We can hide all the details by using a MicroURNG (Random123) lightweight URNG that can be templated on the CBRNG of our choice.

Each sketching matrix gets a slice of the global stream that is uniquely identified by a seed\(^2\). We introduce the “one stream for all processes” approach, a random access stream where every single process is granted the capability to compute any matrix entry. This is massively parallel and distribution independent.

\(^1\)http://xdata-skylark.github.io/libskylark/

\(^2\)libSkylark context
III. HIGH-PERFORMANCE SKETCHING

The implementation of sketching operations requires at its core a robust, efficient and scalable mechanism for the distributed generation of samples to populate a sketch object and scalable high performance matrix-matrix multiplication. For dense sketches we use Elemental’s distributed dense matrices [9] and this implies the use of the distribution schemes this library provides. For sparse sketches we use the doubly compressed sparse column (DCSC) storage format [3] provided by CombBLAS. To avoid realizing the sketching matrix, we implemented our own GEMM sketching operators. For dense sketches this follow a similar strategy as discussed in [5], [11], [13].

We experiment our parallel framework through the computation of sketches on BlueGene/Q™. In most cases the better performance can be attribute to the much lower cost of generating random samples and avoiding to communication involved with the matrix of random samples. In addition to these savings, we require less random samples for FJLT [1] (b) as in the JLT [7] (a) case. As sparse sketches [4] (c) typically are small objects, we are only marginally better than straightforward sparse matrix matrix multiplication.

![Figure 3](image1.png)

Fig. 3. Timings strong scaling runs for elemental (blue) and libSkylark (orange) on BG/Q using 4 OpenMP threads per core. Squares denote the total runtime, circles the random sample generation time and triangles the distributed gemm. (a) JLT on tall-and-thin dense matrices, (b) FJLT on tall-and-thin dense matrices.

IV. APPLICATIONS

On top of the sketching layer, a suite of NLA and ML primitives are implemented, i.e. least-squares solver (based on the Blendenpik [2]).

A. Word Embedding for Natural Language Processing

Word embeddings map words to vectors so that the similarity between words and word combinations is captured by correlation between vectors. It is a very useful tool for Natural Language Processing.

We used libSkylark’s randomized SVD (based on [6]) to compute word embeddings. Specifically, we used gensim.corpora.WikiCorpus for parsing the Wikipedia dump and hyperwords for converting to Positive Pointwise Mutual Information (PPMI) matrix. For word $w$ and context $c$, $PMI(w,c)$ is defined as the log ratio between $w$ and $c$’s joint probability and the product of their marginal probabilities. We compute the embedding by obtaining a PPMI (zeroing negative entries in the PMI matrix) and subsequently applying our randomized SVD. Note that PPMI is a special case ($k=1$) of SPPMI (Shifted PPMI): $SPPMI(w,c) = \max(PMI(w,c) - \log(k), 0)$.

In Table I we report the performance of our embeddings against GloVe\(^3\), using similar methodology and suite of test datasets as in related papers from Omer Levy.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>GloVe</th>
<th>randsvd</th>
</tr>
</thead>
<tbody>
<tr>
<td>RADINSKY_MTURK</td>
<td>0.599986</td>
<td>0.644235</td>
</tr>
<tr>
<td>WS353 (353/353 pairs)</td>
<td>0.492414</td>
<td>0.571149</td>
</tr>
<tr>
<td>WS353_RELATEDNESS</td>
<td>0.489905</td>
<td>0.494797</td>
</tr>
<tr>
<td>LƯONG_RARE</td>
<td>0.331318</td>
<td>0.368968</td>
</tr>
<tr>
<td>WS353_SIMILARITY</td>
<td>0.543087</td>
<td>0.651343</td>
</tr>
<tr>
<td>MÉN (3000/3000 pairs)</td>
<td>0.664732</td>
<td>0.672004</td>
</tr>
</tbody>
</table>

B. Phone Classification for Speech Recognition

Phone classification is used as part of a speech recognition pipeline. We used libSkylark kernel-based classifier [12] to train a classifier using the TIMIT dataset. The solver exhibits good scaling on BlueGene/Q™ and on a commodity cluster, and state-of-the-art classification quality (comparable to DNN’s), as shown in Figure 4.

![Figure 4](image2.png)

Fig. 4. Strong and weak scaling on TIMIT.

V. ACKNOWLEDGMENTS

We acknowledge the support from XDATA program of the Defense Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323.

We would also like to thank Jack Poulson for his helpful discussions on Elemental.

IBM, the IBM logo, ibm.com, and BlueGene/Q are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

\(^3\)http://www-nlp.stanford.edu/data/glove.6B.100d.txt.gz
REFERENCES


