libSkylark: A Framework for High-Performance Matrix Sketching for Statistical Computing

G. Kollias, H. Avron
IBM Research – Watson, USA
Email: {gkollias, haimav}@us.ibm.com

Y. Ineichen, C. Bekas, A. Curioni
IBM Research – Zurich, Switzerland
Email: {yin, bek}@zurich.ibm.com

V. Sindhwani
Google
Email: sindhwani@google.com

K. Clarkson
IBM Research – Almaden, USA
Email: klclarks@us.ibm.com

I. Motivation

The libSkylark[1] library provides sketching-based matrix computations for machine learning suitable for general statistical data analysis and optimization applications.

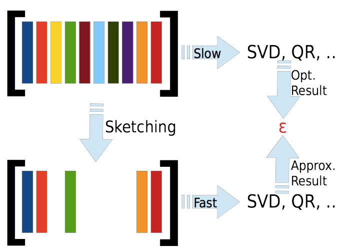


Fig. 1. Sketching is a way to compress matrices that preserves key matrix properties; it can be used to speed up many matrix computations.

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 strided 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 different keys k) and the substream approach (fixing k and associating different streams with non-overlapping, continuous ranges of counter ranges $n_i \dots n_j$ 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.

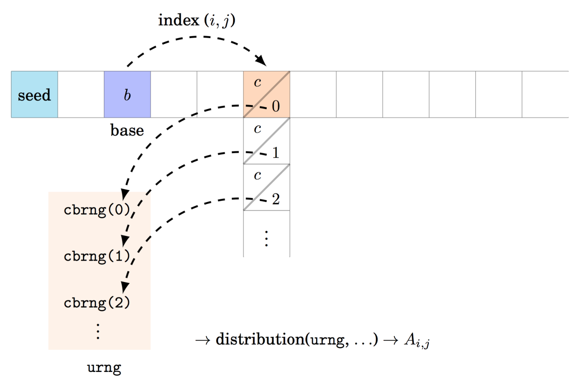


Fig. 2. Behind the scenes: Random123 stream provides two `uint64_t` values r_0 and r_1 given a base b and some index i.

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.

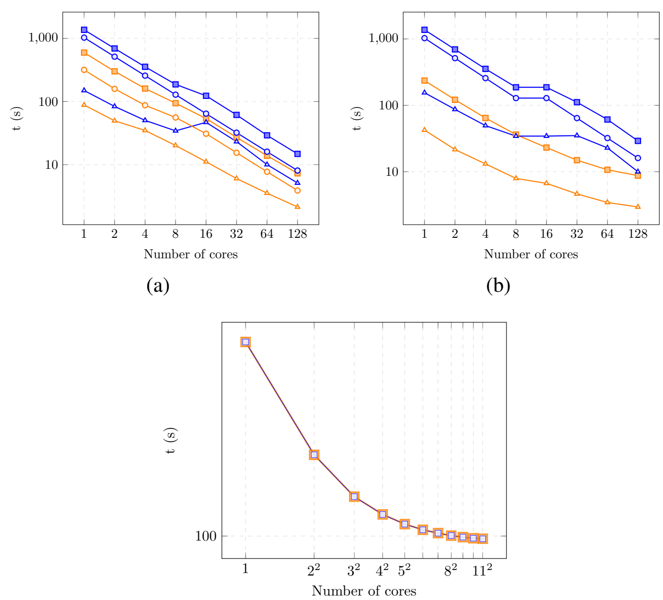
[1] http://xdata-skylark.github.io/libskylark/

[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$^{\text{TM}}$. 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 sketchs [4] (c) typically are small objects, we are only marginally better than straight forward sparse matrix matrix multiplication.



(a) (b) (c)

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.

TABLE I. Spearman rho scores

Benchmark	GloVe	randsvd
RADINSKY_MTURK (287/287 pairs)	0.599986	0.634235
WS353 (353/353 pairs)	0.492414	0.571149
WS353_RELATEDNESS (252/252 pairs)	0.489905	0.494797
LUONG_RARE (1782/2034 pairs)	0.331318	0.368968
WS353_SIMILARITY (203/203 pairs)	0.543087	0.651343
MEN (3000/3000 pairs)	0.664732	0.672004

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.

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$^{\text{TM}}$ and on a commodity cluster, and state-of-the art classification quality (comparable to DNN's), as shown in Figure 4.

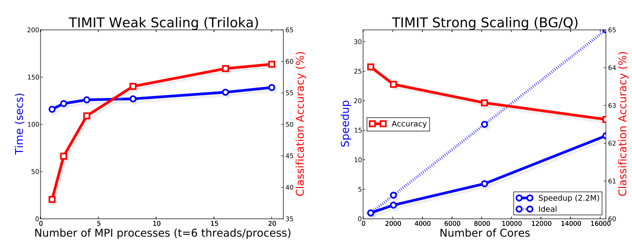


Fig. 4. Strong and weak scaling on TIMIT.

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.

[3]http://www-nlp.stanford.edu/data/glove.6B.100d.txt.gz

References

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 557–563. ACM, 2006.

[2] H. Avron, P. Maymounkov, and S. Toledo. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 32(3):1217–1236, 2010.

[3] A. Buluc and J. R. Gilbert. On the representation and multiplication of hypersparse matrices. In *Proc. 22nd IEEE International Symposium on Parallel and Distributed Processing (22nd IPDPS'08)*, pages 1–11, Miami, Florida, USA, Apr. 2008. IEEE Computer Society (Los Alamitos, CA).

[4] K. L. Clarkson and D. 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.

[5] J. Gunnels, C. Lin, G. Morrow, and R. van de Geijn. A flexible class of parallel matrix multiplication algorithms. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998*, pages 110–116, Los Alamitos, CA, USA, Mar 1998. IEEE Computer Society.

[6] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[7] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[8] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[9] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software (TOMS)*, 39(2):13, 2013.

[10] J. K. Salmon, M. A. Moraes, R. O. Dror, and D. E. Shaw. Parallel random numbers: As easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 16:1–16:12, New York, NY, USA, 2011. ACM.

[11] M. D. Schatz, J. Poulson, and R. A. van de Geijn. Scalable universal matrix multiplication algorithms: 2d and 3d variations on a theme. Submitted to ACM Transactions on Mathematical Software, 2012.

[12] V. Sindhwani and H. Avron. High-performance kernel machines with implicit distributed optimization and randomization. In *JSM Proceedings, Tradeoffs in Big Data Modeling - Section on Statistical Computing*, 2014. To appear in Technometrics.

[13] R. A. Van De Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 9(4):255–274, 1997.