BLAS-on-flash: an alternative for training large ML models?

Suhas Jayaram Subramanya
Microsoft Research India
t-sujs@microsoft.com

Srajan Garg
IIT Bombay
srajan.garg@gmail.com

Harsha Vardhan Simhadri
Microsoft Research India
harshasi@microsoft.com

ABSTRACT

Many ML training tasks admit learning algorithms that can be composed with linear algebra. On large datasets, the working set of these algorithms overflows the memory. For such scenarios, we propose a library that supports BLAS and sparseBLAS subroutines on large matrices resident on inexpensive non-volatile memory. We demonstrate that such libraries can achieve near in-memory performance and be used for fast implementations of complex algorithms such as eigen-solvers. We believe that this approach could be a cost-effective alternative to expensive big-data compute systems.

1 INTRODUCTION

Data analysis pipelines, such as those that arise in scientific computing as well as ranking and relevance, rely on learning from datasets that are 100s of GB to a few TB in size. Many algorithms for the learning tasks involved in these pipelines, such as topic modeling [4], matrix factorizations [22], spectral clustering [21], extreme multi-label learning [32], are memory limited as opposed to being limited by the compute. That is, on large datasets, a training algorithm that requires a few hours of compute on a multi-core workstation would run out of DRAM for its working set.

This forces users to move the training algorithm to big-data platforms such as Spark [42] or Parameter Servers [25, 38], incurring three costs: (1) the cost of porting code to a distributed framework, (2) cost of purchasing and maintaining a cluster nodes or non-availability in certain environments, and (3) inefficiencies of the platform in using the hardware. Training on these platforms can require dozens of nodes for moderate speedups over single threaded code for non-trivial algorithms [16, 26]. This could be due to the platform overheads as well as the mismatch between the structure of the algorithm and the platform's programming model [7, 13, 40], resulting in low processor utilization.

Several light-weight frameworks on single node/workstations demonstrate that this inefficiency is unnecessary for many classes of algorithms that admit multi-threaded implementations that are order(s) of magnitude more efficient [12, 23, 34, 35]. In similar spirit, it is widely observed that many machine learning problems admit models with training algorithms that are essentially compositions of linear algebra operations on sparse and dense matrices. High performance training code for these algorithms is typically written as a main thread consisting of glue code that invokes linear algebra calls through standard APIs such as BLAS [8] and sparseBLAS [15]. High performance implementations for these standard APIs are provided by hardware vendors [19, 20, 28, 29].

Linear algebra kernels offer plenty of locality, so much so that bandwidth required for supporting multiprocessor systems can be provided by a PCIe or SATA bus [3, 39]. Further, recent developments in hardware and software eco-system position non-volatile memory as an inexpensive alternative to DRAM [2, 11, 14, 33]. Hardware technology and interfaces for non-volatile memories have increasingly lower end-to-end latency (few μs) [18] and higher bandwidth: from 4-8 GT/s in PCIe3.0 to 16GT/s in PCIe4.0 [31] and 32GT/s in PCIe5.0. Hardware manufactures are packaging non-volatile memory with processing units, e.g. Radeon PRO SSG [1].

These observations point to a cost-effective solution for scaling linear algebra based algorithms to large datasets in many scenarios. Use inexpensive PCIe-connected SSDs to store large matrices corresponding to the data and the model, and exploit the locality of linear algebra to develop a libraries of routines that can operate on these matrices with a limited amount of DRAM. By conforming to standard APIs, the library could be a replacement for code that would have linked to Intel MKL or OpenBLAS [41].

We present preliminary empirical evidence that this approach can be practical, easy, and fast by developing a library which provides near in-memory speeds on NVM-resident data for subroutines on dense matrices and sparse matrices. These can be easily used to write equally fast implementations for algorithms such as k-means clustering. To illustrate that this approach is not limited to simple kernels, we built a general purpose eigen-solver which is critical to dimensionality reduction and spectral methods. Specifically, we provide a implementation of block Krylov-Schur [43] algorithm which achieves near in-memory speed as compared to the IRAM algorithm [37] in ARPACK [24]. On a large bag-of-words data set (~100GB), our implementation, running on a multi-core workstation with a small DRAM, outperforms Spark MLlib's `computeSVD` [27] deployed on hundreds of workers.

This suggests that for complicated numerical routines, our approach is capable of running fast on a large datasets while providing significant benefits in hardware efficiency as compared to general-purpose big-data systems. Further, we envision our library being useful in the following scenarios: (1) Environments without multi-node support for MPI, Spark etc., (2) Laptops and workstations or VMs in cloud with limited RAM but equipped with large non-volatile memories, (3) Batch mode periodic retraining of large scale models in production data analysis pipelines, (4) Extending the capabilities of legacy single-node ML training code.

2 IMPLEMENTATION DETAILS

Our library implements pipelined external memory parallel algorithms by composing existing math libraries on in-memory blocks with prefetching via a standard Linux asynchronous I/O syscall, `io_submit`. The I/O layer uses NVMe block drivers to access the SSD. I/O queues are packed with many asynchonous requests to extract maximum bandwidth. Intel MKL is used for in-memory computation, but could be easily replaced with other vendor libraries. The size of matrices that the library can handle is limited by the size of the SSD.

Prefetch block sizes for BLAS level 3 routines (e.g.`gemm`) as well as SparseBLAS level 2 and 3 routines on Compressed Sparse Row/Column matrices such (e.g. `csrgemv, csrmm, cscmm`) are tuned to the smallest size that provides sufficient locality for the computation to not be bottleneck by I/O. A maximum of 32GB RAM is used for the runs reported here. We use the fact that BLAS and sparseBLAS computations can be tiled so that they write the output to disk just once [5, 10], thus saving on write bandwidth. We also implemented some utility functions such as `transpose` and `sort` [9] for format conversions (e.g. `csrcsc`).

Our library can be linked to native code with a few modification. We require that large blocks of memory be allocated with the library's allocator rather than the standard allocator.

```
float *mat = (float *)malloc(len); // Replace with
flash_ptr<float> mat = flash::malloc<float>(big_len);
```

The `flash_ptr<T>` type supports pointer arithmetic and can be cast and used as a normal pointer through memory mapping where necessary (e.g. for functionality not supported by the library). A call to a matrix operation invoked with a `flash_ptr` type rather than a normal type is linked to our library. We allow the user to compile code with a flag that disables the library by treating `flash_ptr<Type>` as a normal pointer of type `Type*`. The next section presents performance of a kernel we linked to this library: Lloyd's EM iteration for k-means, written entirely in linear algebra.

Using these subroutines, we built a general-purpose eigensolver, as opposed to sketching based approaches that approximate the spectrum for large data sets [17, 30]. Most eigensolvers, including ARPACK, are iterative and use repeated matrix-vector products for constructing Krylov subspaces. On large sparse matrices, such as those that arise in bag-of-words representation, repeated matrix-vector products on out-of-memory matrix is the rate-limiting step. We address this by using block methods that grow the Krylov subspace many columns at a time using matrix-matrix products, thus reducing the number of iterations to completion. We find that block Krylov-Schur [43] method reduces the number of iterations to convergence for datasets such as bag-of-words where extremal eigenvalues are not pathologically clustered.

3 EXPERIMENTAL SETUP AND RESULTS

We compare the performance of our library's subroutines with the corresponding in-memory version on two Linux machines: (1) a 28 core bare-metal Sandbox with dual Xeon(R) E5-2690 v4 CPUs and 3.2TB Samsung PM1725a SSD on PCIe3.0x8 bus which provides 3GB/s sustained read and 0.5GB/s sustained write, (2) a 32 core Azure VM with dual Xeon(R) E5-2698Bv3 CPUs and a virtual SSD that supports 40k IOps.

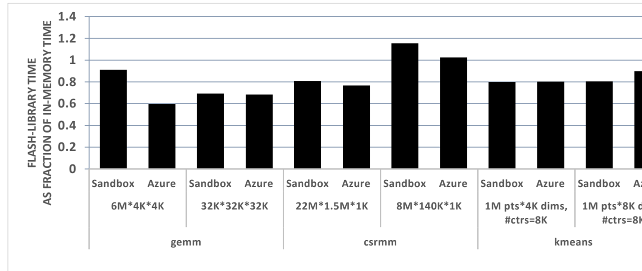

Figure 1: Fraction of in-memory performance achieved by flash-based algorithms.

We measure the time taken to completion for some subroutines and problem sizes, and report the relative slowdown of the flash-based subroutines compared to the corresponding in-memory version in the Figure above. The time for flash-based subroutines is measured from the beginning of the first read from SSD to the last write to SSD. The two instances of `csrmm` reported in figure are performed on matrices with sparsity 10^{-4} and 10^{-3} respectively. We notice that the peformance ratio is just under one in most cases except one – `csrmm` on Sandbox – where our library outperforms MKL. We suspect that this instance is poorly tuned in MKL, and our tiling was better. Not surprisingly, the bare-metal sandbox with a high-end SSD narrowly outperforms the Azure VM.

We compare the time required by the block Krylov-Schur solver with that of Spark MLlib's computeSVD for finding the top 500 singular values of a large sparse matrix (100GB) corresponding to a text data-set represented as bag of words (tolerance: 10^{-4}). Both algorithms require the multiplication of the given matrix A (or $A^T A$ in the case of non-symmetric matrices) with a vector. For large instances, we store the matrix in the SSD while Spark distributes it across workers. The Spark job is deployed through yarn to workers with 1 core and 8GB memory each on a cluster with Xeon(R) E5-2450L CPUs and 10Gb Ethernet. Across runs, our library is faster than Spark job with 128 cores, and we do not see any benefit from more Spark workers.

Platform	Cores/Workers	Time
Sandbox	28	52min
Spark	128	250min
Spark	256	405min
Spark	512	425min

Table 1: Time to compute 500 singular values of bag of words data (22M docs, 1.5M vocabulary, 6B nnzs).

4 DISCUSSION AND FUTURE WORK

Preliminary results suggests that libraries that utilize fast non-volatile memories could provide an alternative to big-data systems for training large machine learning models, and could offer more hardware efficiency. Our library can also support GPU and other PCIe storage devices like Optane. We are linking our library with large scale applications: SVD-based topic models [4, 36] and extreme multi-label learning tasks for datasets with about 100M points and 50M labels [6, 32].

ACKNOWLEDGMENTS

The authors would like to thank Anirudh Badam and Muthian Sivathanu for their useful comments and advice.

REFERENCES

[1] AMD. 2018. Radeon™ Pro SSG. (2018). https://pro.radeon.com/en/product/pro-series/radeon-pro-ssg/

[2] Joy Arulraj and Andrew Pavlo. 2017. How to Build a Non-Volatile Memory Database Management System. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 1753–1758. https://doi.org/10.1145/3035918.3054780

[3] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. 2011. Minimizing Communication in Numerical Linear Algebra. *SIAM J. Matrix Anal. Appl.* 32, 3 (2011), 866–901. https://doi.org/10.1137/090769156 arXiv:https://doi.org/10.1137/090769156

[4] Trapit Bansal, C. Bhattacharyya, and Ravindran Kannan. 2014. A Provable SVD-based Algorithm for Learning Topics in Dominant Admixture Corpus. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 1997–2005. http://dl.acm.org/citation.cfm?id=2969033.2969050

[5] Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. 2016. Parallel Algorithms for Asymmetric Read-Write Costs. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '16)*. ACM, New York, NY, USA, 145–156. https://doi.org/10.1145/2935764.2935767

[6] Kush Bhatia, Kunal Dahiya, Himanshu Jain, Yashoteja Prabhu, and Manik Varma. [n. d.]. The Extreme Classification Repository: Multi-label Datasets and Code. ([n. d.]). http://manikvarma.org/downloads/XC/XMLRepository.html

[7] Mikhail Bilenko, Tom Finley, Shon Katzenberger, Sebastian Kochman, Dhruv Mahajan, Shravan Narayanamurthy, Julia Wang, Shizhen Wang, and Markus Weimer. 2016. Salmon: Towards Production-Grade, Platform-Independent Distributed ML. In *The ML Systems Workshop at ICML*.

[8] L. Susan Blackford, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, Michael Heroux, Linda Kaufman, Andrew Lumsdaine, Antoine Petiet, Roldan Pozo, Karin Remington, and R. Clint Whaley. 2002. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Softw.* 28, 2 (June 2002), 135–151. https://doi.org/10.1145/567806.567807

[9] Guy E. Blelloch, Phillip B. Gibbons, and Harsha Vardhan Simhadri. 2010. Low Depth Cache-oblivious Algorithms. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '10)*. ACM, New York, NY, USA, 189–199. https://doi.org/10.1145/1810479.1810519

[10] Erin Carson, James Demmel, Laura Grigori, Nicholas Knight, Penporn Koanantakool, Oded Schwartz, and Harsha Vardhan Simhadri. 2016. Write-Avoiding Algorithms. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 648–658. https://doi.org/10.1109/IPDPS.2016.114

[11] Dask Development Team. 2016. *Dask: Library for dynamic task scheduling*. http://dask.pydata.org

[12] Laxman Dhulipala, Guy Blelloch, and Julian Shun. 2017. Julienne: A Framework for Parallel Graph Algorithms Using Work-efficient Bucketing. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '17)*. ACM, New York, NY, USA, 293–304. https://doi.org/10.1145/3087556.3087580

[13] David Dinh, Harsha Vardhan Simhadri, and Yuan Tang. 2016. Extending the Nested Parallel Model to the Nested Dataflow Model with Provably Efficient Schedulers. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '16)*. ACM, New York, NY, USA, 49–60. https://doi.org/10.1145/2935764.2935797

[14] Aleksandar Dragojevic, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory, In 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014). https://www.microsoft.com/en-us/research/publication/farm-fast-remote-memory/

[15] Iain S. Duff, Michael A. Heroux, and Roldan Pozo. 2002. An Overview of the Sparse Basic Linear Algebra Subprograms: The New Standard from the BLAS Technical Forum. *ACM Trans. Math. Softw.* 28, 2 (June 2002), 239–267. https://doi.org/10.1145/567806.567810

[16] A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani, P. Sharma, J. Yang, J. Demmel, J. Harrell, V. Krishnamurthy, M. W. Mahoney, and Prabhat. 2016. Matrix Factorization at Scale: a Comparison of Scientific Data Analytics in Spark and C+MPI Using Three Case Studies. *ArXiv e-prints* (July 2016). arXiv:cs.DC/1607.01335

[17] N. Halko, P. G. Martinsson, and J. A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288. https://doi.org/10.1137/090771806 arXiv:https://doi.org/10.1137/090771806

[18] Intel®. 2017. Optane™ Memory. (2017). https://www.intel.com/content/www/us/en/architecture-and-technology/optane-memory.html

[19] Intel®. 2018. Math Kernel Library Sparse BLAS level 2 and 3 routines. (2018). https://software.intel.com/en-us/mkl

[20] Intel®. 2018. Math Kernel Library Sparse BLAS level 2 and 3 routines. (2018). https://software.intel.com/en-us/mkl-developer-reference-c-sparse-blas-level-2-and-level-3-routines

[21] Ravi Kannan, Santosh Vempala, and Adrian Vetta. 2004. On Clusterings: Good, Bad and Spectral. *J. ACM* 51, 3 (May 2004), 497–515. https://doi.org/10.1145/990308.990313

[22] Abhishek Kumar, Vikas Sindhwani, and Prabhanjan Kambadur. 2013. Fast Conical Hull Algorithms for Near-separable Non-negative Matrix Factorization. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28 (ICML'13)*. JMLR.org, I–231–I–239. http://dl.acm.org/citation.cfm?id=3042817.3042845

[23] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-scale Graph Computation on Just a PC. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI'12)*. USENIX Association, Berkeley, CA, USA, 31–46. http://dl.acm.org/citation.cfm?id=2387880.2387884

[24] Rich Lehoucq, Kristi Maschhoff, Danny Sorensen, and Chao Yang. [n. d.]. ARPACK Software. ([n. d.]).

[25] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 583–598. http://dl.acm.org/citation.cfm?id=2685048.2685095

[26] Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! But at What Cost?. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems (HOTOS'15)*. USENIX Association, Berkeley, CA, USA, 14–14. http://dl.acm.org/citation.cfm?id=2831090.2831104

[27] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. 2016. MLlib: Machine Learning in Apache Spark. *J. Mach. Learn. Res.* 17, 1 (Jan. 2016), 1235–1241. http://dl.acm.org/citation.cfm?id=2946645.2946679

[28] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable Parallel Programming with CUDA. *Queue* 6, 2 (March 2008), 40–53. https://doi.org/10.1145/1365490.1365500

[29] NVIDIA. 2017. cuSPARSE library. (2017). http://docs.nvidia.com/cuda/cusparse/index.html

[30] Daisuke Okanohara. [n. d.]. redsvd: C++ library for solving several matrix decompositions. ([n. d.]). https://code.google.com/archive/p/redsvd/

[31] PCI-SIG. 2017. PCI Express Base Specification Revision 4.0, Version 1.0. (October 2017). https://members.pcisig.com/wg/PCI-SIG/document/10912?downloadRevision=active

[32] Yashoteja Prabhu and Manik Varma. 2014. FastXML: A Fast, Accurate and Stable Tree-classifier for Extreme Multi-label Learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 263–272. https://doi.org/10.1145/2623330.2623651

[33] ScaleMP™. 2018. vSMP Foundation Flash Expansion. (2018). http://www.scalemp.com/products/flx/

[34] Julian Shun and Guy E. Blelloch. 2013. Ligra: A Lightweight Graph Processing Framework for Shared Memory. In *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '13)*. ACM, New York, NY, USA, 135–146. https://doi.org/10.1145/2442516.2442530
[35] Julian Shun, Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W. Mahoney. 2016. Parallel Local Graph Clustering. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 1041–1052. https://doi.org/10.14778/2994509.2994522
[36] Harsha Vardhan Simhadri. 2017. SVD and Importance sampling-based algorithms for large scale topic modeling. https://github.com/Microsoft/ISLE. (2017).
[37] D. C. Sorensen. 1992. Implicit Application of Polynomial Filters in a k-Step Arnoldi Method. *SIAM J. Matrix Anal. Appl.* 13, 1 (1992), 357–385. https://doi.org/10.1137/0613025 arXiv:https://doi.org/10.1137/0613025
[38] DMTK Team. [n. d.]. Multiverso: Parameter Server for Distributed Machine Learning. ([n. d.]). https://github.com/Microsoft/Multiverso
[39] Jeffrey Scott Vitter. 2001. External Memory Algorithms and Data Structures: Dealing with Massive Data. *ACM Comput. Surv.* 33, 2 (June 2001), 209–271. https://doi.org/10.1145/384192.384193
[40] Markus Weimer, Yingda Chen, Byung-Gon Chun, Tyson Condie, Carlo Curino, Chris Douglas, Yunseong Lee, Tony Majestro, Dahlia Malkhi, Sergiy Matusevych, et al. 2015. REEF: Retainable Evaluator Execution Framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1343–1355.
[41] Zhang Xianyi. 2017. OpenBLAS. (2017). http://www.openblas.net/
[42] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1863103.1863113
[43] Yunkai Zhou and Yousef Saad. 2008. Block Krylov–Schur method for large symmetric eigenvalue problems. *Numerical Algorithms* 47, 4 (01 Apr 2008), 341–359. https://doi.org/10.1007/s11075-008-9192-9