Progress Report on the Early-Career Research Project: Energy-Efficient Parallel Graph and Data Mining Algorithms

Principal Investigator: Dr. Aydın Buluç
Staff Scientist
Lawrence Berkeley National Laboratory
One Cyclotron Road, MS 59R4104
Berkeley, CA 94720
abuluc@lbl.gov

Program Manager: Steven Lee
Reporting Period: 10/16-9/17

Data are fundamental sources of insight for experimental and computational sciences. The Department of Energy acknowledges the challenges posed by fast-growing scientific data sets and more complex data. The graph abstraction provides a natural way to represent relationships among complex fast-growing scientific data sets. On future exascale systems, power consumption is of primary concern yet existing graph algorithms consume too much energy per useful operation due to their high communication costs, lack of locality, and inability to exploit hierarchy. This project explores methods to increase the energy efficiency of parallel graph algorithms and data mining tasks. A new family of algorithms will be developed to drastically reduce the energy footprint and running time of the graph and sparse matrix computations that form the basis of various data mining techniques. This project will also exploit the well-known duality between graph and sparse matrices to develop communication-avoiding graph algorithms that consume significantly less power. This project is relevant to DOE mission-critical science including bioinformatics and genomics with particular emphasis on plant genomics that can result in better biofuels through efficient genetic mapping, climate science where recent graph-based methods show increased accuracy in hurricane predictions, and combustion science where graph search techniques are used to analyze extreme-scale simulation data.

1 Summary

The early-career research project has made important progress during FY 2017. Specifically, our accomplishments include:

- Release of Version 1 of the GraphBLAS standard, as described in Section 3.1.
- A work-efficient sparse-matrix-sparse-vector multiplication algorithm, a key kernel used in graph algorithms and machine learning (detailed in Section 3.2).
- Scalable parallelization of the reverse Cuthill-Mckee (RCM) sparse-matrix ordering algorithm using linear-algebraic primitives (Section 3.3).
- Development of the High-Performance Markov Cluster Algorithm, in collaboration with ECP funded projects ExaBiome and ExaGraph (detailed in Section 3.4).

- New parallel algorithms for Deep Neural Network training on KNL and GPU clusters (detailed in Section 3.5).

DOE relevance of each research accomplishment is mentioned within the corresponding section. Our next year plans can be found in Section 4. In FY17 alone, our project resulted in five peer-reviewed publications [3, 7, 14, 10, 4]. In addition, we have one paper under review [5].

The report concludes with software artifacts, presentations, community service, and references. In particular, Dr. Buluç was the Vice Chair for the Applications Track in the technical program committee of the SC conference (i.e. The International Conference for High Performance Computing, Networking, Storage and Analysis). He also served as the PC co-chair of the IEEE Graph Algorithm Building Blocks workshop. This fiscal year, Dr. Buluç has been appointed as an Adjunct Assistant Professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley, and Dr. Azad has been appointed as a career-track Research Scientist at LBNL.

2 Personnel

The project is lead by Aydın Buluç, a staff scientist, who spent about 50% of his time on this project. In addition, Ariful Azad, a research scientist, spent a significant amount of this time (40%) on this project. Carl Yang, a GSRA from UC Davis, was partially funded to assist with developing GraphBLAS kernels on GPUs.

3 Progress and Accomplishments

3.1 The GraphBLAS C API

The GraphBLAS is an effort to define standard building blocks for graph algorithms in the language of linear algebra [2, 12]. The PI Buluç co-leads a multi-institutional group of researchers that are working on defining the right primitives and providing initial reference implementations. Thanks to these efforts, the number of systems that support graph algorithms in the language of matrices has increased steadily [13, 9, 11]. Our own library, Combinatorial BLAS (CombBLAS) [6, 1], remains the gold standard among those efforts due to its pioneering status.

We have successfully ran the third peer-reviewed Graph Algorithms Building Blocks (GABB) workshop at IPDPS'17, where the attendance was 30-40 people at any given time. Dr. Buluç co-chaired the program committee in 2017. There were 12 submissions that were peer-reviewed and 9 of them got accepted to be part of the proceedings. The call for papers for next version is out at `http://www.graphanalysis.org/workshop2018.html`.

Dr. Buluç is a member of the five person team (along with Carl Yang who is also partially funded by this project) that works hands on to develop a C language application programming interface (API). The current C API specification [8] is provisionally final and it has been released publicly. We have received numerous feedback from the implementers and the community at-large since the initial release of the API. This allowed us to improve the spec in various ways. The design rationale of the spec is published in a shorter paper [7].

A very important novel GraphBLAS concept that we would like to highlight in this report is a mask. A mask is either a one- or a two-dimensional construct. Masks are similar to vectors and matrices, except that they have structure (indices) but no values. Masks are used to control which values from an operation are written to the output object. This way, they allow efficient execution of graph algorithms without materializing unnecessary temporary objects. We envision to be broadly applicable to the growing area of sparse machine learning algorithms.

This report summarizes some of the mathematical concepts of GraphBLAS below. Consider a graph represented as an n-by-n adjacency matrix $\mathbf{A}$, where A_{ij} is the weight of the edge from vertex i to vertex j, and a second k-by-n matrix $\mathbf{B}$ representing a subset (of size k) of the vertices in the graph, such that B_{ji} is 1 if the jth element of the subset is vertex i (and all other elements of $\mathbf{B}$ are 0). The traditional matrix product $\mathbf{B} \times \mathbf{A}$ over real arithmetic of these two matrices returns the cost based on the edge weights of reaching the set of vertices adjacent to the vertices in $\mathbf{B}$. This fundamental operation can be used to construct a wide range of graph algorithms.

We extend the range of graph operations by keeping the basic pattern of a matrix-matrix multiplication, but varying the operators and the interpretation of the values in the matrices (the *domain*). By carefully choosing operators and the domain, we control the relation between matrix operations familiar in linear algebra and graph operations, thereby enabling composable graph algorithms.

This generalized matrix multiplication is performed on an algebraic semiring. A semiring is an algebraic structure over a domain D with two binary operators $\oplus$ and $\otimes$. The *addition* operator, $\oplus$, is a commutative monoid with an identity element $\mathbf{0}$ (not necessarily the number 0) while the *multiplication* operator, $\otimes$, is a commutative monoid with an identity element $\mathbf{1}$ (not necessarily the number 1). The additive identity is also an annihilator for the multiplication operator ($\otimes$), and multiplication distributes over addition. The most common semirings used in the graph algorithms community are shown in Table 1.

Table 1: Common semirings used with graph algorithms.

Semiring	operators		domain	$\mathbf{0}$	$\mathbf{1}$
	$\oplus$	$\otimes$			
Standard arithmetic	$+$	$\times$	$\mathbb{R}$	0	1
max-plus algebras	max	$+$	$\{-\infty \cup \mathbb{R}\}$	$-\infty$	0
min-max algebras	min	max	$\infty \cup \mathbb{R}_{\geq 0}$	∞	0
Galois fields (*e.g.*, GF2)	xor	and	$\{0, 1\}$	0	1
Power set algebras	$\cup$	$\cap$	$\mathcal{P}(\mathbb{Z})$	$\varnothing$	U

It is often convenient to change the semiring applied to a matrix. This means we must represent the matrix and the semiring separately, and the two come together only when an operation is performed. Mathematically, the ability to change semirings when moving from one GraphBLAS operation to the next impacts the meaning of the *implied zero* in a sparse representation of the matrix. This element in real arithmetic is the number zero (0), which is the identity of the addition operator and the annihilator of the multiplication operator. As the semiring changes, this implied zero changes to the identity of the addition operator

and the annihilator of the multiplication operator for the new semiring. Nothing changes in the stored matrix, but the implied values within the sparse matrix change with respect to a particular operation.

This feature has significant impact on the definitions of GraphBLAS operations. Consider matrix multiplication over the domain $\mathbb{S}$ with semiring operators $\oplus$ and $\otimes$:

$$\mathbf{C} = \mathbf{A}\oplus.\otimes\mathbf{B} = \mathbf{AB}.$$

Using index notation familiar in linear algebra

$$\mathbf{C}(i,j) = \bigoplus_{k=1}^{l} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

for matrices with dimensions

$$\mathbf{A} : \mathbb{S}^{m\times l} \qquad \mathbf{B} : \mathbb{S}^{l\times n} \qquad \mathbf{C} : \mathbb{S}^{m\times n}$$

The summation notation only works, however, if we redefine the implied zero of the sparse matrices as we change the semiring (to the corresponding additive identity). Depending on the domains associated with the matrix elements and the operations, this can lead to awkward definitions of the operations involving the implied zeros. A cleaner approach based on set notation avoids this problem. For example, we can define the previous matrix multiplication as

$$\mathbf{C}(i,j) = \bigoplus_{k\in\mathbf{ind}(\mathbf{A}(i,:))\cap\mathbf{ind}(\mathbf{B}(:,j))} (\mathbf{A}(i,k) \otimes \mathbf{B}(k,j)),$$

where $\mathbf{ind}(\mathbf{A}(i,:))$ is the set of the column indices of the elements that are stored in row i of matrix $\mathbf{A}$, and $\mathbf{ind}(\mathbf{B}(:,j))$ is the set of the row indices of the elements that are stored in column j of matrix $\mathbf{B}$.

In other words, the binary operation $\otimes$ is applied to the elements in the intersection of the two sets $\mathbf{ind}(\mathbf{A}(i,:))$ and $\mathbf{ind}(\mathbf{B}(:,j))$, and the results of this operation are accumulated using the $\oplus$ operator. These notations are equivalent. By defining pairwise operations over set intersections, however, we avoid needing to define how the semiring's additive identity interacts with the matrix's implied zeros.

3.2 A Parallel Algorithm for Sparse-Matrix Sparse-Vector Product

Sparse matrix-sparse vector multiplication (SpMSpV) is a key GraphBLAS kernel, which enables efficient parallelization of various graph algorithms such as breadth-first search, maximal independent set, bipartite graph matching, and reverse Cuthill-McKee algorithm. It also is used as a kernel in important sparse machine learning techniques such as support vector machine (SVM) training and logistic regression.

We designed and developed a work-efficient multithreaded algorithm for SpMSpV where the matrix, the input vector, and the output vector are all sparse [3]. As thread counts increase, existing multithreaded SpMSpV algorithms can spend more time accessing the sparse matrix data structure than doing arithmetic. Our shared-memory parallel SpMSpV

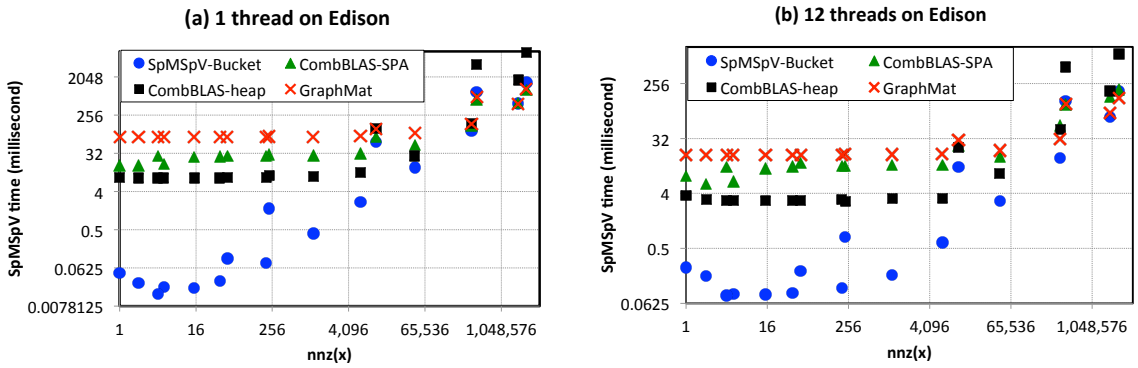

Figure 1: Runtime of four SpMSpV algorithms when the adjacency matrix of `ljournal-2008` is multiplied by sparse vectors with different number of nonzero entries using (a) 1 thread and (b) 12 threads on Edison. The sparse vectors represent frontiers in a BFS starting from the first vertex of `ljournal-2008`.

algorithm is work efficient in the sense that its total work is proportional to the number of arithmetic operations required. The key insight is to avoid each thread individually scan the list of matrix columns.

Our algorithm is simple to implement and operates on existing column-based sparse matrix formats. It performs well on diverse matrices and vectors with heterogeneous sparsity patterns. An example performance result that show the superiority of our algorithm across different vector densities is shown in Figure 1. A high-performance implementation of the algorithm attains up to 15x speedup on a 24-core Intel Ivy Bridge processor and up to 49x speedup on a 64-core Intel KNL manycore processor. In contrast to implementations of existing algorithms, the performance of our algorithm is sustained on a variety of different input types include matrices representing scale-free and high-diameter graphs.

3.3 Reverse Cuthill-McKee ordering in Distributed Memory

Ordering vertices of a graph is key to minimize fill-in and data structure size in sparse direct solvers, maximize locality in iterative solvers, and improve performance in graph algorithms. Except for naturally parallelizable ordering methods such as nested dissection, many important ordering methods have not been efficiently mapped to distributed-memory architectures. We developed the first-ever distributed-memory implementation of the reverse Cuthill-McKee (RCM) algorithm for reducing the profile of a sparse matrix [4]. Our parallelization uses a two-dimensional sparse matrix decomposition. We achieve high performance by decomposing the problem into a small number of primitives and utilizing optimized implementations of these primitives. Our implementation attains up to 38x speedup on matrices from various applications on 1024 cores of a Cray XC30 supercomputer and shows strong scaling up to 4096 cores for larger matrices.

3.4 High-Performance Markov Clustering

HipMCL is a high-performance parallel algorithm for large-scale network clustering. HipMCL parallelizes popular Markov Cluster (MCL) algorithm that has been shown to be one of the most successful and widely used algorithms for network clustering. It is based on random walks and was initially designed to detect families in protein-protein interaction networks. Despite MCL's efficiency and multi-threading support, scalability remains a bottleneck as it fails to process networks of several hundred million nodes and billion edges in an affordable running time. HipMCL overcomes all of these challenges by developing massively-parallel algorithms for all components of MCL. HipMCL can be x1000 times faster than the original MCL without any information loss. It can easily cluster a network of 75 million nodes with 68 billion edges in 2.4 hours using 2000 nodes of Cori supercomputer at NERSC. HipMCL is developed in C++ language and uses standard OpenMP and MPI libraries for shared- and distributed-memory parallelization. The paper on HipMCL has been under revision [5].

3.5 Parallel Training of Deep Neural Networks

Deep Neural Networks (DNNs) significantly improved the accuracy of regression and classification problems that often arise in sciences. Neural Network training is computationally expensive, sometimes prohibitively so. For example, training ImageNet dataset on one Nvidia K20 GPU needs 21 days. Large scale parallelism and heavy reliance on hardware accelerators are often the only tools to address the computational complexity. However, these accelerators have limited on-chip memory compared with CPUs.

We performed an in-depth study of scaling DNN training on clusters of accelerators. We used both self-host Intel Knights Landing (KNL) clusters and multi-GPU clusters as our target platforms. From the algorithm aspect, we focused on Elastic Averaging Stochastic Gradient Descent (EASGD), due to its demonstrated convergence properties, to design algorithms for HPC clusters.

We redesigned four efficient algorithms for HPC systems to improve EASGD's poor scaling on clusters. Async EASGD, Async MEASGD, and Hogwild EASGD are faster than existing counterpart methods (Async SGD, Async MSGD, and Hogwild SGD) in all comparisons. In addition to the algorithmic improvements, we use some system-algorithm codesign techniques to scale up the algorithms. By reducing the percentage of communication from 87% to 14%, our Sync EASGD achieves 5.3× speedup over original EASGD on the same platform. We get 91.5% weak scaling efficiency on 4253 KNL cores, which is higher than the state-of-the-art implementation.

Besides the algorithmic refinements, the system-algorithm codesign techniques are important for scaling up deep neural networks. The techniques we introduce include: (1) using single-layer layout and communication to optimize the network latency and memory access, (2) using multiple copies of weights to speedup the gradient descent, and (3) partitioning the KNL chip based on data/weight size and reducing communication on multi-GPU systems. By reducing the communication percent from 87% to 14%, our Sync EASGD achieves 5.3× speedup over original EASGD on the same platform. Using ImageNet dataset to train GoogleNet on 2176 KNL cores, the weak scaling efficiency of Intel Caffe is 87% while our implementation is 92%. Using ImageNet to train VGG on 2176 KNL cores, the weak scaling efficiency of Intel Caffe is 62% while our implementation is 78.5%.

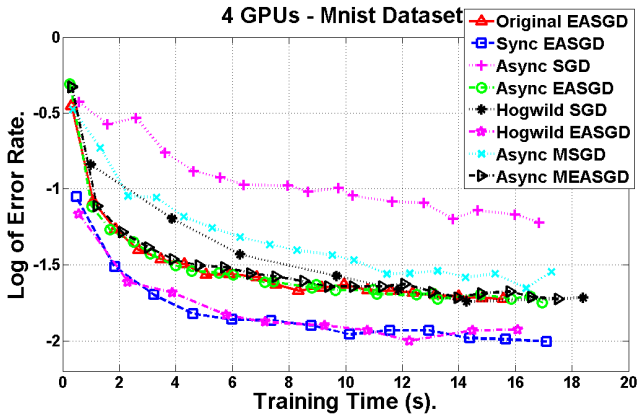

Figure 2: To visualize the comparisons, we use error rate (1.0 − accuracy) as the algorithm benchmark. Then we use log_{10} scale of error rate to make the comparisons more clear. Among these methods, Original EASGD, Hogwild SGD, Async SGD, and Async MSGD are the existing methods. The rest of them are our methods. Each point on the figure is a single run. For example, Sync EASGD has 13 points in the figure. It means we run 13 mutually independent Sync EASGD cases with different numbers of iterations. It also means longer time or more iterations will help us to get a higher accuracy, even with different initiations. The experiments are conducted on 4 Tesla M100 GPUs that are connected with a 96-lane, 6-way PCIe switch.

An overall accuracy-vs-time comparison of several algorithms tested and redesigned are shown in Figure 2. Among them, Original SGD, Hogwild SGD, Async SGD, and Async MSGD are the existing methods. We also observe that Sync EASGD or Hogwild EASGD is the fastest method among them. Sync EASGD and Hogwild EASGD are essentially tied for fastest. Sync EAGSD incorporates a number of optimizations that we describe in more detail in our paper [14].

4 Plans for the next fiscal year

Approximate-weight Perfect Matching on Bipartite Graphs: Direct solvers for sparse linear systems, such as SuperLU and STRUMPACK, can not afford to perform expensive dynamic pivoting during runtime. Instead, they pre-permute the matrix to have a diagonal with non-zero values. Once the nonzero diagonal constraint is satisfied, the solver would also like the sum or the product of the absolute values on the diagonal to have as high value as possible. In graph-theoretical terms, this problem translates into finding an approximate-weight perfect matching on a bipartite graph. Ideally, maximum-weight perfect matching is expected to work better with sparse linear solvers. However, no practical parallel algorithm exists for maximum-weight perfect matching problem, forcing distributed-memory solvers rely on sequential matching libraries. We are currently developing a distributed-memory $2/3 - \epsilon$ approximation algorithm for maximum-weight perfect matching problem based on the sequential algorithm by Pettie and Sanders. We will extensively study the impact the matching quality on the runtime and numerical stability of

sparse linear solvers such as SuperLU and STRUMPACK.

Parallel Connected Components on Distributed Memory: The distributed-memory parallel MCL (HipMCL) algorithm performs a series of sparse matrix manipulations iteratively. After the iterations converge, the clusters are extracted by finding the weakly-connected components of the final converged matrix. This is equivalent to finding connected components on the symmetrized version of the final converged matrix. This operation needs to be performed in distributed memory as the data is already distributed. Using linear-algebraic primitives of Combinatorial BLAS, we will develop an efficient implementation of the Awerbuch-Shiloach algorithm (to be named LACC). This algorithm is chosen for its simplicity, performance guarantees, and suitability to Combinatorial BLAS. The algorithm and its performance results will be written as a tech report that will be submitted to publication.

Sparse training of Deep Neural Networks: The common practice in deep learning training is to use dense matrix-matrix operations for high performance, mostly due to fully connected neural network layers. It is recognized that sparsity in internal connection weights, inputs, and/or outputs can be exploited for increased performance, accuracy (i.e. minimize generalization error), and even interpretability. Deep Learning training achieves sparsity using masks. The GraphBLAS effort explicitly targets these masked basic linear algebra subroutines. We will inherit expertise from the GraphBLAS effort and our long-running research programs in developing highly-scalable sparse matrix kernels.

Stochastic gradient descent (SGD) is the algorithm of choice for training NNs. Data parallelism alone, however, is not sufficient in scaling SGD to 1000s of nodes efficiently. Due to its noisy gradients, small-batch SGD avoids getting trapped in sharp minima. Increasing the training batch size hurts this natural ability of SGD. Therefore, we will also exploit model parallelism where the NN is partitioned (as opposed to replicated) on multiple nodes. The crux of our approach is its synergy with exploiting network sparsity. A fully connected NNs, while trivial to partition equally, would incur too much communication. A pruned network is more amenable for partitioning with a low communication cost. Since a single NN pruning step is known to lose accuracy, retraining and pruning alternates in phases. Hence, we might need to repartition occasionally, preferably using a cheap partitioner.

Software artifacts

Combinatorial BLAS: Both Dr. Buluç and Dr. Azad contributed significantly to the development of the Combinatorial BLAS (CombBLAS) library [6, 1] along both the functionality and the performance axes. We released significantly improved version 1.6 in October 2017. The new features include complete in-node multithreading support, fully parallel text I/O for both vectors and matrices, extended support for the distributed CSC data structure, and our novel Reverse Cuthill-McKee (RCM) implementation.

HP-CONCORD: HP-CONCORD (High-Performance CONCORD) is a highly-scalable distributed-memory implementation of the CONCORD-ISTA algorithm for sparse inverse covariance matrix estimation. It is implemented in C++ with MPI and OpenMP. The main bottleneck of HP-CONCORD is iterative sparse-dense matrix-matrix multiplication which is handled by the SpDM^3 library that is distributed with HP-CONCORD. Code is available at `https://bitbucket.org/penpornk/spdm3-hpconcord`.

Selected Talks

- " Faster Parallel Graph BLAS Kernels and New Graph Algorithms in Matrix Algebra", Aydın Buluç, *invited talk* at EECS Department of UC Berkeley (October 2017) and Google Research (November 2017)
- "Parallel Algorithms across the GraphBLAS Stack", Aydın Buluç, *invited talk* at the ACS HPC and Data Analytics Workshop, June 2017.
- " A work-efficient parallel sparse matrix-sparse vector multiplication algorithm", Ariful Azad, *conference talk* at the IEEE International Symposium on Parallel and Distributed Processing, May 2017
- "The reverse Cuthill-McKee algorithm in distributed-memory", Aydın Buluç, *conference talk* at the IEEE International Symposium on Parallel and Distributed Processing, May 2017

Community Service

Aydın Buluç:

- *Associate Editor*: ACM Transactions on Parallel Computing
- *Program Committee Leadership*:
  - *Vice-chair, Applications (2017)*: ACM/IEEE Intl. Conf. for HPC, Networking, Storage and Analysis (SC)
  - *Co-chair (2017)*: IEEE Graph Algorithms Building Blocks (GABB) workshop at IPDPS
- *Program Committee*:
  - IEEE International Parallel & Distributed Processing Symp. (IPDPS), 2017
  - ACM Conf. on Bioinformatics, Comp. Biology, & Health Infor. (BCB), 2017
  - IEEE Cluster, 2017
  - Workshop on Irregular Applications: Architectures and Algorithms (IA^3), 2017
  - PLDI ARRAY Workshop on Libraries, Languages & Compilers for Prog., 2017
- *Steering Committee*: Graph Algorithms Building Blocks (GABB), IPDPS workshop, 2017.

Ariful Azad:

- *Program Committee*:
- ACM/IEEE Intl. Conf. for HPC, Networking, Storage and Analysis (SC), 2017
- IEEE International Workshop on High Performance Comp. Biology (HiCOMB), 2017
- IEEE Cluster, 2017

References

[1] Combinatorial BLAS Library, version 1.6, 2017. `http://graphblas.org/~aydin/CombBLAS/html/`.

[2] The graphblas forum, 2017. `http://graphblas.org/`.

[3] A. Azad and A. Buluç. A work-efficient parallel sparse matrix-sparse vector multiplication algorithm. In *Proceedings of the IPDPS*, 2017.

[4] A. Azad, M. Jacquelin, A. Buluç, and E. G. Ng. The reverse Cuthill-McKee algorithm in distributed-memory. In *Proceedings of the IPDPS*, 2017.

[5] A. Azad, G. A. Pavlopoulos, C. A. Ouzounis, N. C. Kyrpides, and A. Buluç. HipMCL: A high-performance parallel implementation of the markov cluster algorithm for large-scale networks. *Under Review*, 2017.

[6] A. Buluç and J. R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *The International Journal of High Performance Computing Applications*, 25(4):496 – 509, 2011.

[7] A. Buluç, T. Mattson, S. McMillan, J. Moreira, and C. Yang. Design of the GraphBLAS API for C. In *IEEE Workshop on Graph Algorithm Building Blocks, IPDPSW*, 2017.

[8] A. Buluç, T. Mattson, S. McMillan, J. Moreira, and C. Yang. The GraphBLAS C API Specification, version 1.0.0. Technical report, The GraphBLAS Signatures Subgroup, May 2017. http://graphblas.org/ aydin/GraphBLAS_API_C.pdf.

[9] K. Ekanadham, W. P. Horn, M. Kumar, J. Jann, J. Moreira, P. Pattnaik, M. Serrano, G. Tanase, and H. Yu. Graph Programming Interface (GPI): A linear algebra programming model for large scale graph computations. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 72–81, New York, NY, USA, 2016. ACM.

[10] M. Ellis, E. Georganas, R. Egan, S. Hofmeyr, A. Buluç, B. Cook, L. Oliker, and K. Yelick. Performance characterization of de novo genome assembly on leading parallel systems. In *Europar - International European Conference on Parallel and Distributed Computing*, 2017.

[11] V. Gadepally, J. Bolewski, D. Hook, D. Hutchison, B. Miller, and J. Kepner. Graphulo: Linear algebra graph kernels for nosql databases. In *International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, pages 822–830. IEEE, 2015.

[12] J. Kepner, P. Aaltonen, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, J. Moreira, J. Owens, C. Yang, M. Zalewski, and T. Mattson. Mathematical foundations of the GraphBLAS. In *IEEE High Performance Extreme Computing (HPEC)*, 2016.

[13] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey. Graphmat: High performance graph analytics made productive. *Proceedings of the VLDB Endowment*, 8(11):1214–1225, 2015.

[14] Y. You, A. Buluç, and J. Demmel. Scaling deep learning on GPU and Knights Landing clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'17)*, 2017.