75,000,000,000 Streaming Inserts/Second Using Hierarchical Hypersparse GraphBLAS Matrices

Jeremy Kepner[1,2,3], Tim Davis[4], Chansup Byun[1],
William Arcand[1], David Bestor[1], William Bergeron[1], Vijay Gadepally[1,2], Matthew Hubbell[1],
Michael Houle[1], Michael Jones[1], Anna Klein[1], Peter Michaleas[1], Lauren Milechin[5],
Julie Mullen[1], Andrew Prout[1], Antonio Rosa[1], Siddharth Samsi[1], Charles Yee[1], Albert Reuther[1]
[1]MIT Lincoln Laboratory Supercomputing Center, [2]MIT Computer Science & AI Laboratory,
[3]MIT Mathematics Department, [4]Texas A&M, [5]MIT Department of Earth, Atmospheric and Planetary Sciences

***Abstract*—The SuiteSparse GraphBLAS C-library implements high performance hypersparse matrices with bindings to a variety of languages (Python, Julia, and Matlab/Octave). GraphBLAS provides a lightweight in-memory database implementation of hypersparse matrices that are ideal for analyzing many types of network data, while providing rigorous mathematical guarantees, such as linearity. Streaming updates of hypersparse matrices put enormous pressure on the memory hierarchy. This work benchmarks an implementation of hierarchical hypersparse matrices that reduces memory pressure and dramatically increases the update rate into a hypersparse matrices. The parameters of hierarchical hypersparse matrices rely on controlling the number of entries in each level in the hierarchy before an update is cascaded. The parameters are easily tunable to achieve optimal performance for a variety of applications. Hierarchical hypersparse matrices achieve over 1,000,000 updates per second in a single instance. Scaling to 31,000 instances of hierarchical hypersparse matrices arrays on 1,100 server nodes on the MIT SuperCloud achieved a sustained update rate of 75,000,000,000 updates per second. This capability allows the MIT SuperCloud to analyze extremely large streaming network data sets.**

I. Introduction

The global Internet is expected to exceed 100 terabytes per second (TB/s) by the year 2022 creating significant performance challenges for the monitoring necessary to improve, maintain, and protect the Internet, particularly with the rising social influence of adversarial botnets encompassing a significant fraction of Internet traffic [1]–[3]. Origin-destination traffic matrix databases are fundamental network analysis tool for a wide range of networks, enabling the observation of temporal fluctuations of network supernodes, computing background models, and inferring the presence of unobserved traffic [4]–[8]. Rapidly constructing these traffic matrix databases is a significant productivity, scalability, representation, and performance challenge [9]–[16].

Our team has developed a high-productivity scalable platform—the MIT SuperCloud—for providing scientists and engineers the tools they need to analyze large-scale dynamic data [17]–[19]. The MIT SuperCloud provides interactive analysis capabilities accessible from high level programming environments (Python, Julia, Matlab/Octave) that scale to thousands of processing nodes. Traffic matrices can be manipulated on the MIT SuperCloud using distributed databases (SciDB and Apache Accumulo), D4M associative arrays [17], [20], and now the SuiteSparse GraphBLAS hypersparse matrix library [21]–[23].

For IP network traffic data, the IP address space requires a hypersparse matrix (#entries $<<$ #rows and #columns) that is either $2^{32}\times2^{32}$ for IPv4 or $2^{64}\times2^{64}$ for IPv6. Our prior work represented traffic matrices using D4M associative arrays using sorted lists of strings to describe the row and column labels of an underlying standard sparse matrix [24]. D4M associative arrays provide maximum flexibility to represent the row and columns with arbitrary strings and are extremely useful during the feature discovery stage of algorithm development. For IP traffic matrices, the row and column labels can be constrained to integers allowing additional performance to be achieved using a hypersparse matrix library such as the SuiteSparse GraphBLAS. In either case, the memory hierarchy presents a significant performance bottleneck as doing lots of updates to slow memory is prohibitive. This work benchmarks an implementation of hierarchical hypersparse matrices that reduces memory pressure and dramatically increases the update rate into a hypersparse matrices.

II. Hierarchical Hypersparse Matrices

The SuiteSparse GraphBLAS library is an OpenMP accelerated C implementation of the GraphBLAS.org sparse matrix standard. Python, Julia, and Matlab/Octave bindings allow the performance benefits of the SuiteSparse GraphBLAS C library to be realized in these highly productive programming environments. Streaming updates to a large hypersparse matrix can be be accelerated with a hierarchical implementation optimized to the memory hierarchy (see Fig. 1). Rapid updates are performed on the smallest hypersparse matrices in the fastest memory. The strong mathematical properties of the GraphBLAS allow a hierarchical implementation of hypersparse matrices to be implemented via simple addition. All creation and organization of hypersparse row and column indices are handled naturally by the GraphBLAS mathematics.

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001 and National Science Foundation CCF-1533644. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering or the National Science Foundation.

978-1-7281-7445-7/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPSW50202.2020.00046

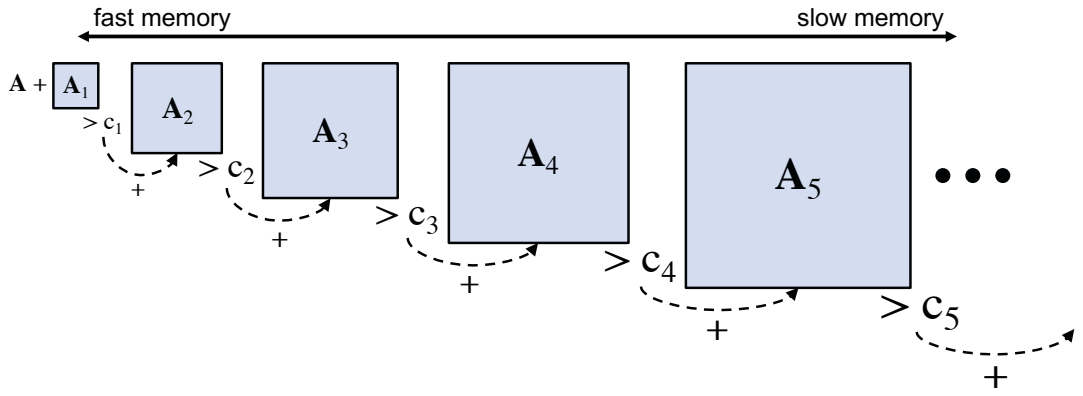

Fig. 1. Hierarchical hypersparse matrices store increasing numbers of nonzero entries in each layer (adapted from [24]). If the number of nonzero (stored) entries in layer $\mathbf{A}_i$ surpasses the nonzero threshold count c_i then $\mathbf{A}_i$ is added to $\mathbf{A}_{i+1}$ and $\mathbf{A}_i$ is cleared. Hierarchical hypersparse matrices ensure that the majority of updates are performed in fast memory.

If the number of nonzero (nnz) entries exceeds the threshold c_i, then $\mathbf{A}_i$ is added to $\mathbf{A}_{i+1}$ and $\mathbf{A}_i$ is cleared. The overall usage is as follows

- Initialize N-level hierarchical hypersparse matrix with cuts c_i
- Update by adding data $\mathbf{A}$ to lowest layer

$$\mathbf{A}_1 = \mathbf{A}_1 + \mathbf{A}$$

- If $\mathrm{nnz}(\mathbf{A}_1) > c_1$, then

$$\mathbf{A}_2 = \mathbf{A}_2 + \mathbf{A}_1$$

  and reset $\mathbf{A}_1$ to an empty hypersparse matrix of appropriate dimensions.

The above steps are repeated until $\mathrm{nnz}(\mathbf{A}_i) \leq c_i$ or $i = N$. To complete all pending updates for analysis, all the layers are added together

$$\mathbf{A} = \sum_{i=1}^{N} \mathbf{A}_i$$

Hierarchical hypersparse matrices dramatically reduce the number of updates to slow memory. Upon query, all layers in the hierarchy are summed into the hypersparse matrix. The cut values c_i can be selected so as to optimize the performance with respect to particular applications. The majority of the complex updating is performed by using the existing GraphBLAS addition operation. The corresponding Matlab/Octave GraphBLAS code for performing the update is direct translation of the above mathematics as follows

```
function Ai = HierAdd(Ai,A,c);
  Ai{1} = Ai{1} + A;
  for i=1:length(c)
    if (GrB.entries(Ai{i}) > c(i))
      Ai{i+1} = Ai{i+1} + Ai{i};
      Ai{i} = Ai{length(c)+2};
    end
  end
```

The goal of hierarchical arrays are to manage the memory footprint of each level, so the GraphBLAS `GrB.entries()` command returns the number of entries in the GraphBLAS hypersparse matrix, which may include some materialized zero values. In addition, `GrB.entries()` executes much faster than the number of nonzeros command `nnz()`. The last hypersparse matrix stored in the hierarchical array is empty and is used to reinitialize layers whose entries have been cascaded to a subsequent layer.

III. Performance Optimization

The performance of a hierarchical GraphBLAS for any particular problem is determined by the number of layers N and the cut values c_i. The parameters are tuned to achieve optimal performance for a given problem. Examples of differents sets of cut values with different c_1 and different ratios between cut values are shown in Figure 2. These sets of cut values allow exploration of the update performance of closely spaced cuts versus widely spaced cuts. Figure 3 shows the single node performance using different numbers of processes and threads on a simulated Graph500.org R-Mat power-law network data. The data set contains 100,000,000 connections that are inserted in groups of 100,000.

IV. Scalability Results

The scalability of the hierarchical hypersparse matrices are tested using a power-law graph of 100,000,000 entries divided up into 1,000 sets of 100,000 entries. These data were then simultaneously loaded and updated using a varying number of processes on varying number of nodes on the MIT SuperCloud up to 1,100 servers consisting of mixture of 64-core Intel Xeon 7210 servers, 28-core Intel Xeon 2683v3 servers, and 32-core AMD 6274 servers for a total of 34,000 processor cores. This experiment mimics thousands of processors, each creating many different graphs of 100,000,000 edges each. In a real analysis application, each process would also compute various

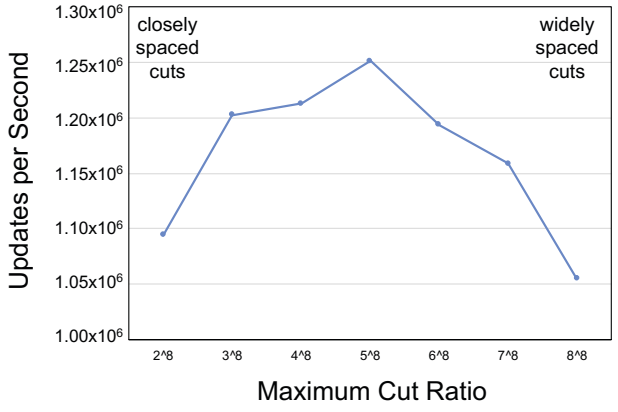

Fig. 2. Single thread update rate on a 64-core Intel Xeon 7210 server for different cut ratio sets, ranging from $\{2^2, \ldots, 2^8\}$ to $\{8^2, \ldots, 8^8\}$. For these data, performance is optimal with ratio spacings in the 3 to 6 range. Actual cut values are determine by multiplying the cut ratios by a base value (in this case 2^{17}).

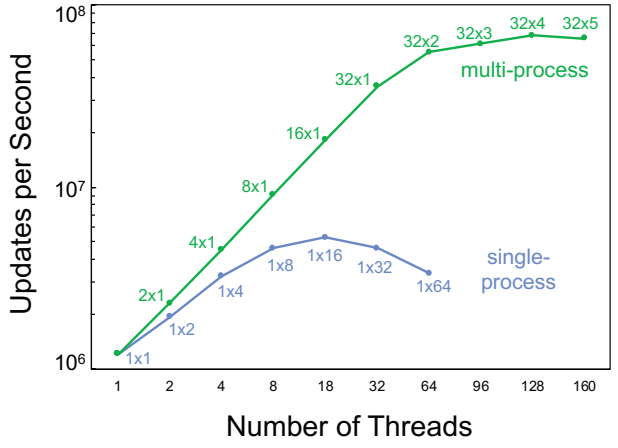

Fig. 3. Single node performance with different configurations of hierarchical GraphBLAS processes and threads on a 64-core Intel Xeon 7210 server. Each data point is labeled: (#processes)x(#threads/process). Single process achieves maximum performance with 16 threads. Maximum overall performance is achieved with 32 processes each with four threads.

network statistics on each of the streams as they are updated. The update rate as a function of number of server nodes is shown on Fig. 4. The achieved update rate of 75,000,000,000 updates per second is significantly larger than the rate in prior published results. This capability allows the MIT SuperCloud to analyze extremely large streaming network data sets.

V. Conclusion

The GraphBLAS implementation of provides a lightweight in-memory database ideal for analyzing hypersparse network data. Streaming data into hypersparse matrices puts enormous pressure on a memory hierarchy. GraphBLAS hierarchical matrices reduce memory pressure and increase update performance. The linearity properties of sparse matrices allow

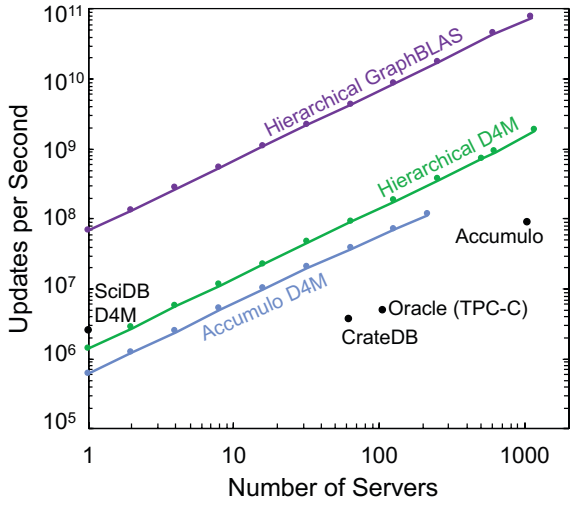

Fig. 4. Update rate as a function of number of servers for hierarchical GrapBLAS hypersparse matrices and other previous published work: Hierarchical D4M [19], Accumulo D4M [25], SciDB D4M [26], Accumulo [27], Oracle TPC-C benchmark, and CrateDB [28]

a hierarchical sparse matrix to be implemented using simple addition operations. The performance of hierarchical GraphBLAS comes from controlling the number entries at each level and can be tuned for any particular application. Hierarchical sparse matrices achieve over 1,000,000 updates per second in a thread instance and are significantly faster than non-hierarchical sparse matrices. Scaling to 34,000 instances on 1,100 server nodes on the MIT SuperCloud achieved a sustained update rate of 75,000,000,000 updates per second.

Acknowledgement

The authors wish to acknowledge the following individuals for their contributions and support: Bob Bond, Alan Edelman, Laz Gordon, Charles Leiserson, Dave Martinez, Mimi McClure, Victor Roytburd, Michael Wright.

References

[1] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, vol. 1, 2018.

[2] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," *Journal of Economic Perspectives*, vol. 31, no. 2, pp. 211–36, 2017.

[3] "https://www.neosit.com/files/neos_distil_bad_bot_report_2018.pdf."

[4] A. Soule, A. Nucci, R. Cruz, E. Leonardi, and N. Taft, "How to identify and estimate the largest traffic matrix elements in a dynamic environment," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, pp. 73–84, ACM, 2004.

[5] Y. Zhang, M. Roughan, C. Lund, and D. L. Donoho, "Estimating point-to-point and point-to-multipoint traffic matrices: an information-theoretic approach," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 5, pp. 947–960, 2005.

[6] V. Bharti, P. Kankar, L. Setia, G. Gürsun, A. Lakhina, and M. Crovella, "Inferring invisible traffic," in *Proceedings of the 6th International Conference*, p. 22, ACM, 2010.

[7] P. Tune, M. Roughan, H. Haddadi, and O. Bonaventure, "Internet traffic matrices: A primer," *Recent Advances in Networking*, vol. 1, pp. 1–56, 2013.

[8] J. Kepner, K. Cho, K. Claffy, V. Gadepally, P. Michaleas, and L. Milechin, "Hypersparse neural network analysis of large-scale internet traffic," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–11, Sep. 2019.

[9] V. G. Castellana, M. Minutoli, S. Bhatt, K. Agarwal, A. Bleeker, J. Feo, D. Chavarría-Miranda, and D. Haglin, "High-performance data analytics beyond the relational and graph data models with gems," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1029–1038, IEEE, 2017.

[10] F. Busato, O. Green, N. Bombieri, and D. A. Bader, "Hornet: An efficient data structure for dynamic sparse graphs and matrices on gpus," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–7, IEEE, 2018.

[11] A. Yaar, S. Rajamanickam, M. Wolf, J. Berry, and . V. atalyrek, "Fast triangle counting using cilk," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–7, Sep. 2018.

[12] Y. Hu, H. Liu, and H. H. Huang, "High-performance triangle counting on gpus," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–5, Sep. 2018.

[13] M. Bisson and M. Fatica, "Update on static graph challenge on gpu," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–8, Sep. 2018.

[14] R. Pearce and G. Sanders, "K-truss decomposition for scale-free graphs at scale in distributed memory," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–6, Sep. 2018.

[15] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, D. Staheli, and J. Kepner, "Static graph challenge: Subgraph isomorphism," in *High Performance Extreme Computing Conference (HPEC)*, IEEE, 2017.

[16] E. Kao, V. Gadepally, M. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song, *et al.*, "Streaming graph challenge: Stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–12, IEEE, 2017.

[17] J. Kepner, W. Arcand, W. Bergeron, N. Bliss, R. Bond, C. Byun, G. Condon, K. Gregson, M. Hubbell, J. Kurz, *et al.*, "Dynamic distributed dimensional data model (d4m) database and computation system," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5349–5352, IEEE, 2012.

[18] V. Gadepally, J. Kepner, L. Milechin, W. Arcand, D. Bestor, B. Bergeron, C. Byun, M. Hubbell, M. Houle, M. Jones, *et al.*, "Hyperscaling internet graph analysis with d4m on the mit supercloud," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–6, IEEE, 2018.

[19] A. Reuther, J. Kepner, C. Byun, S. Samsi, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Klein, L. Milechin, J. Mullen, A. Prout, A. Rosa, C. Yee, and P. Michaleas, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–6, Sep. 2018.

[20] J. Kepner and H. Jananthan, *Mathematics of Big Data*. MIT Press, 2018.

[21] J. Kepner, P. Aaltonen, D. Bader, A. Bulu, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. Mattson, and J. Moreira, "Mathematical foundations of the graphblas," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9, Sep. 2016.

[22] T. A. Davis, "Algorithm 1000: Suitesparse:graphblas: Graph algorithms in the language of sparse linear algebra," *ACM Trans. Math. Softw.*, vol. 45, pp. 44:1–44:25, Dec. 2019.

[23] T. A. Davis, "Graph algorithms via SuiteSparse: GraphBLAS: triangle counting and k-truss," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pp. 1–6, Sep. 2018.

[24] J. Kepner, V. Gadepally, L. Milechin, S. Samsi, W. Arcand, D. Bestor, W. Bergeron, C. Byun, M. Hubbell, M. Houle, M. Jones, A. Klein, P. Michaleas, J. Mullen, A. Prout, A. Rosa, C. Yee, and A. Reuther, "Streaming 1.9 billion hypersparse network updates per second with d4m," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Sep. 2019.

[25] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, A. Reuther, A. Rosa, and C. Yee, "Achieving 100,000,000 database inserts per second using accumulo and d4m," in *High Performance Extreme Computing Conference (HPEC)*, IEEE, 2014.

[26] S. Samsi, L. Brattain, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, M. Jones, A. Klein, *et al.*, "Benchmarking scidb data import on hpc systems," in *High Performance Extreme Computing Conference (HPEC)*, pp. 1–5, IEEE, 2016.

[27] R. Sen, A. Farris, and P. Guerra, "Benchmarking apache accumulo bigdata distributed table store using its continuous test suite," in *Big Data (BigData Congress), 2013 IEEE International Congress on*, pp. 334–341, IEEE, 2013.

[28] CrateDB, "Big Bite: Ingesting Performance of Large Clusters." https://crate.io/a/big-cluster-insights-ingesting/, 2016. [Online; accessed 01-December-2018].