POSTER: Optimizing Sparse Computations Jointly

Kazem Cheshmi
Computer Science Department
University of Toronto
Toronto, Canada
kazem@cs.toronto.edu

Michelle Mills Strout
Computer Science Department
University of Arizona
Tucson, USA
mstrout@cs.arizona.edu

Maryam Mehri Dehnavi
Computer Science Department
University of Toronto
Toronto, Canada
mmehride@cs.toronto.edu

Abstract

This work proposes a framework called FuSy that analyzes the data dependence graphs (DAGs) of two sparse kernels and creates an efficient schedule to execute the kernels in combination. Sparse kernels are frequently used in scientific codes and in machine learning algorithms and very often they are used in combination. Iterative linear system solvers are an example where kernels such as sparse triangular solver (SpTRSV) and sparse matrix-vector multiplication (SpMV) are called consecutively in each iteration of the solver. Prior approaches typically optimize these sparse kernels independently leading to high synchronization overheads and low locality. We propose an approach that analyzes the DAGs of two sparse kernels and then creates a new order of execution that enables running the two kernels efficiently in parallel. To investigate the efficiency of our approach, we compare it with the state-of-the-art MKL library for two kernel combinations, SpTRSV-SpMV and SpMV-SpTRSV which are commonly used in iterative solvers. Experimental results show that our approach is on average 2.6× and 1.8× faster than the MKL library for a set of matrices from the Suitesparse matrix repository.

CCS Concepts: **• Computing methodologies → Shared memory algorithms**; **• Software and its engineering → Source code generation**.

Keywords: Sparse matrix code, loop fusion, loop-carried dependence

ACM Reference Format:
Kazem Cheshmi, Michelle Mills Strout, and Maryam Mehri Dehnavi. 2022. POSTER: Optimizing Sparse Computations Jointly. In *27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '22), February 12–16, 2022, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3503221.3508439

PPoPP '22, April 2–6, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9204-4/22/04.
https://doi.org/10.1145/3503221.3508439

1 Extended Abstract

The numerical methods [10] that are frequently used in real-world applications such as in scientific simulations and data analytic codes are often composed of a number of sparse matrix computations that execute inside an iteration of the numerical algorithm and between iterations. Because sparse kernels are often the most time-consuming operation in these applications, numerous library and compiler approaches have been proposed to optimize these kernels. However, prior work primarily optimizes sparse kernels in isolation thus when used to accelerate real-world simulations the realized speedups are sometimes not significant.

Numerous parallel sparse libraries [11, 16] and inspector-executor approaches that inspect memory access patterns at runtime such as [6, 8, 12, 15] optimize individual sparse matrix kernels. Kernels with a fully parallel outermost loop have sufficient parallelism and thus an efficient schedule is needed to create a balanced parallel implementation. Sparse kernels with partial parallelism, i.e. loop-carried dependencies, have irregular computation patterns that depend on the sparse matrix code and input data, thus runtime inspection is required to extract the computation patterns. In inspector-executor frameworks and libraries such as [2, 13], a data flow directed acyclic graph (DAG) is built to expose data dependencies. For example, the inspectors in [6, 15], use wavefront parallelism to create a parallel schedule for kernels with partial parallelism. First, the DAG is created, and is traversed in topological order to create a list of *wavefronts* that are iterations that can execute in parallel; this is known as *wavefront parallelism*.

Wavefront parallelism requires synchronization between wavefronts, and thus when applied to individual sparse kernels with loop-carried dependencies can be less efficient due to synchronization overheads. Also, for sparse kernels with non-uniform workloads, such as Cholesky [3], wavefront methods can lead to load imbalance.

DAG partitioning techniques such as DAGP [7] (used in [9]) typically create fewer wavefronts, thus reducing synchronization overheads, and group iterations that reuse data to improve data locality. DAGP adopts a multilevel approach [1] with coarsening and refinement for acyclic partitioning of DAG. These techniques are efficient for individual sparse computations, however, when applied to the joint DAG, they create some non-linear overheads or large exploration space and thus significantly increase analysis time. For example, when applied to the joint DAG of two sparse kernels such as sparse triangular solver (SpTRSV) and sparse matrix-vector multiplication (SpMV), DAGP becomes 5 times slower for selected set of matrices from Suitesparse [5], even though the joint DAG size increases two times.

We present an approach that creates an efficient schedule for when sparse kernels are used jointly. Our approach analyzes the data dependence graph of two sparse kernels to create a load-balanced parallel schedule of the combined code with good locality. Locality is improved by assigning dependent vertices to the same group and then executing that group of vertices via the same thread. Load balance is improved by assigning vertices throughout execution to create well-balanced tasks.

Results are collected on a Haswell multicore architecture with 12 cores of a Xeon E5-2680v3 processor and a 30MB L3 cache. We use matrices from the Suitesparse [5] repository to compare our approach with MKL [16]. The matrices are selected to be of different sizes and varying sparsity patterns from a small number of nonzero elements (1.4×10^5) to a large number of nonzero elements (1.1×10^8). We use MKL 2019.3.199 and call each kernel from the MKL library separately. The performance of both our approach and MKL are tested on two kernel combinations, SpTRSV-SpMV and SpMV-SpTRSV. Both combinations are used in iterative linear solver methods such as preconditioned GMRES [4] and Gauss-seidel. Here we focus on joint optimization of kernels within an iteration of the solver to ensure the stability of the solver. For the combinations of SpTRSV-SpMV and SpMV-SpTRSV, we improved over the MKL library by an average speedup of 2.6× and 1.8× consecutively.

This work focuses primarily on optimizing the combination of SpTRSV and SpMV using the proposed approach. As future work, the DAGs of other sparse kernels and their combination should be studied to create an efficient schedule for joining these kernels. Also, a combination of some sparse kernels might not be cost-efficient, and hence new models should be investigated for the efficiency of joint execution. We also plan to apply the proposed approach to real-world benchmarks to demonstrate the effect of sparse kernel combinations.

Acknowledgments

This work was supported in part by NSERC Discovery Grants (RGPIN-06516, DGECR00303), the Canada Research Chairs program, and U.S. NSF awards NSF CCF-1814888, NSF CCF-1657175; used the Extreme Science and Engineering Discovery Environment (XSEDE) [14] which is supported by NSF grant number ACI-1548562; and was enabled in part by Compute Canada and Scinet[1]. We also like to thank George Huang for his involvement in the prototyping of the approach.

[1]www.computecanada.ca

References

[1] Thang Nguyen Bui and Curt Jones. 1993. *A heuristic for reducing fill-in in sparse matrix factorization.* Technical Report. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA.

[2] Kazem Cheshmi, Shoaib Kamil, Michelle Mills Strout, and Maryam Mehri Dehnavi. 2017. Sympiler: transforming sparse matrix codes by decoupling symbolic analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–13.

[3] Kazem Cheshmi, Shoaib Kamil, Michelle Mills Strout, and Maryam Mehri Dehnavi. 2018. ParSy: inspection and transformation of sparse matrix computations for parallelism. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, 779–793.

[4] Kazem Cheshmi, Danny M Kaufman, Shoaib Kamil, and Maryam Mehri Dehnavi. 2020. NASOQ: numerically accurate sparsity-oriented QP solver. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 96–1.

[5] Timothy A Davis and Yifan Hu. 2011. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (2011), 1.

[6] R Govindarajan and Jayvant Anantpur. 2013. Runtime dependence computation and execution of loops on heterogeneous systems. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).* IEEE Computer Society, 1–10.

[7] Julien Herrmann, M. Yusuf Özkaya, Bora Uçar, Kamer Kaya, and Ümit V. Çatalyürek. 2019. Multilevel Algorithms for Acyclic Partitioning of Directed Acyclic Graphs. *SIAM Journal on Scientific Computing (SISC)* 41, 4 (2019), A2117–A2145. https://doi.org/10.1137/18M1176865

[8] Mahdi Soltan Mohammadi, Tomofumi Yuki, Kazem Cheshmi, Eddie C Davis, Mary Hall, Maryam Mehri Dehnavi, Payal Nandy, Catherine Olschanowsky, Anand Venkat, and Michelle Mills Strout. 2019. Sparse computation data dependence simplification for efficient compiler-generated inspectors. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation.* 594–609.

[9] M. Yusuf Özkaya, Anne Benoit, Bora Uçar, Julien Herrmann, and Ümit V. Çatalyürek. 2019. A scalable clustering-based task scheduler for homogeneous processors using DAG partitioning. In *33rd IEEE International Parallel and Distributed Processing Symposium.*

[10] Yousef Saad. 2003. *Iterative methods for sparse linear systems.* SIAM.

[11] Yousef Saad and Andrei V Malevsky. 1995. P-Sparslib: a portable library of distributed memory sparse iterative solvers. In *Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, St. Petersburg.* Citeseer.

[12] Michelle Mills Strout, Larry Carter, Jeanne Ferrante, Jonathan Freeman, and Barbara Kreaseck. 2002. Combining performance aspects of irregular gauss-seidel via sparse tiling. In *International Workshop on Languages and Compilers for Parallel Computing.* Springer, 90–110.

[13] Michelle Mills Strout, Mary Hall, and Catherine Olschanowsky. 2018. The sparse polyhedral framework: Composing compiler-generated inspector-executor code. *Proc. IEEE* 106, 11 (2018), 1921–1934.

[14] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 5 (Sept.-Oct. 2014), 62–74. https://doi.org/10.1109/MCSE.2014.80

[15] Anand Venkat, Mahdi Soltan Mohammadi, Jongsoo Park, Hongbo Rong, Rajkishore Barik, Michelle Mills Strout, and Mary Hall. 2016. Automating wavefront parallelization for sparse matrix computations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE Press, 41.

[16] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™.* Springer, 167–188.