Sparse Matrices Correctness 2019 Paper

ACM Reference Format:
. 2019. Sparse Matrices Correctness 2019 Paper. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Sparse matrix data formats can be used to compress large matrices with a small number of non-zero elements into a more efficient representation. Scientific and engineering software that make use of sparse matrix formats are often implemented in low-level imperative languages such as C++ and Fortran. The optimized nature of these software often means that the structural organization of sparse matrix formats and mathematical computations are heavily intertwined. Additionally, the myriad data formats and complexities involved in tuning the operations on these formats to achieve efficient implementations means that the development of software that utilizes sparse matrices is a tedious and often error-prone task.

To mitigate the introduction of errors into code during development, techniques such as unit testing and test-driven development are often employed. These methods aim to reveal bugs by testing the individual components of a piece of software during development. While useful, these methods are both tedious, requiring test cases to be written manually, and incomplete, meaning they are incapable of checking every scenario or combination of scenarios.

In the context of sparse matrices, the use of object-oriented libraries such as PETSc [1] and Eigen [2] may help reduce the frequency of bugs. Libraries such as these provide data abstractions targeted towards specific classes of solvers, allowing developers to assemble sparse matrices without having to address the structural complexities that a sparse matrix format may present. However, the utility of these libraries may be limited due to solver limitations or other performance restrictions.

Often, the solvers provided by popular linear algebra libraries used in scientific computing require assembled sparse matrices as input, leaving the entire assembly process to the developer. Considering the problem domains in which they are used, this can quickly become a complex task. For example, sparse matrices are often used to represent the meshes used in the finite element and finite difference methods. In order to represent physical properites, these meshes can contain rich datasets, the values of which may depend on the complex spatial relationships of the mesh components. As a result, the matrix assembly process can quickly become intertwined with the mathematics required to determine the precise values that will be placed in the matrix. Furthermore, as performance improvements are made by, for example, exploiting parallelism in modern hardware, the assembly process and embedded calculations can become even more complex.

Alternatively, there is a body of research that takes the approach of designing and building compilers capable of automatically making decisions about sparse matrix formats and operations. Some compilers [3, 4] allow developers to work with dense matrices in code, generating a sparse matrix program at compile time. Others [5] allow the user to provide the compiler with an abstract description of a sparse format, from which the compiler can make automatic optimizations in code that accesses the sparse data.

We describe an approach that allows developers to reason about the inherent complexities of sparse matrix formats and operations and to determine invariants that can be used to verify implementations. Elements of this approach include declarative modeling and automatic, push-button analysis using the Alloy Analyzer [6], a lightweight bounded model checking tool. Characteristics of sparse matrices, with their numerous representations and ...hard-to-get-right-implementation-details... are approached using abstraction based methods [7], including data abstraction [8] and predicate abstraction [9], data and functional refinement [10], and other techniques, manually, as part of a modeling process...

The benefits of this approach lie in its generality. By using Alloy to perform the modeling and analysis, the modeler may choose the programming language that best fits their needs when transitioning from model to implementation. Can reason about existing software. Can reason about design of new sparse matrix libraries. Can reason about compiler design.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

REFERENCES

[1] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, VictorEijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang, "PETSc Web page," https://www.mcs.anl.gov/petsc, 2019. [Online]. Available: https://www.mcs.anl.gov/petsc

[2] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[3] A. Bik and H. Wijshoff, "Advanced compiler optimizations for sparse computations," *Journal of Parallel and Distributed Computing*, vol. 31, no. 1, pp. 14 – 24, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731585711410

[4] A. J. C. Bik and H. A. G. Wijshoff, "Automatic data structure selection and transformation for sparse matrix computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 2, pp. 109–126, Feb 1996.

[5] V. Kotlyar, K. Pingali, and P. Stodghill, "A relational approach to the compilation of sparse matrix programs," in *Euro-Par'97 Parallel Processing*, C. Lengauer, M. Griebl, and S. Gorlatch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 318–327.

[6] D. Jackson, *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2012.

[7] E. M. Clarke, O. Grumberg, and D. E. Long, "Model checking and abstraction," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 5, pp. 1512–1542, 1994.

[8] J. Dingel and T. Filkorn, "Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving," in *Computer Aided Verification: 7th International Conference, CAV '95 Liège, Belgium, July 3–5, 1995 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 54–69. [Online]. Available: https://doi.org/10.1007/3-540-60045-0_40

[9] S. Graf and H. Saïdi, "Construction of abstract state graphs with PVS," in *International Conference on Computer Aided Verification*. Springer, 1997, pp. 72–83.

[10] J. Woodcock, *Using Z : specification, refinement, and proof*. London New York: Prentice Hall, 1996.

[11] G. Arnold, J. Hlzl, A. Sinan Kksal, R. Bodik, and M. Sagiv, "Specifying and verifying sparse matrix codes," *ACM SIGPLAN Notices*, vol. 45, pp. 249–260, 09 2010.

[12] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[13] J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Z. Bai, Ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[14] Y. Saad, "Sparskit: a basic tool kit for sparse matrix computations - version 2," 1994.

[15] A. Milicevic, J. P. Near, E. Kang, and D. Jackson, "Alloy*: A general-purpose higher-order relational constraint solver," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, May 2015, pp. 609–619.

[16] T. Dyer, "Article github repository," 2019. [Online]. Available: https://github.com/atdyer/alloy-lib