Sparse Matrix Support in LLVM

COMS 6156 Final Project Progress Report

Eric Feng and Chris Yoon (Group Name: "EricAndChris")

ef2648@columbia.edu cjy2129@columbia.edu

Overview

Sparse matrices are matrices where a significant number of its entries are zeros. They are ubiquitous in practical applications, such as machine learning, deep learning, computer graphics, networks, and graphs [1]. Treating sparse matrices like *dense matrices* (matrices whose elements are mostly nonzero), would waste memory and time: since operations on zeroes are often no-ops (e.g. $x+0=0$, $x\times 0=0$), they do not need to be stored explicitly in memory and sparsity-aware linear algebra operations can avoid wasting CPU/GPU cycles on no-op operations.

Sparsification of matrices is a program optimization technique that compresses sparse matrices by encoding zeros more compactly, and provides sparsity-aware linear algebra operations (such as transposition, multiplication, addition, and subtraction). Through sparsification, programs performing operations on large sparse matrices can significantly reduce both memory usage and runtime. Typically, such optimizations are provided via external (and often proprietary) accelerated linear algebra libraries, such as Intel MKL [2] and BLAST [3].

Our project aims to bring first-class support for sparse matrices in LLVM [4], a widely used compiler infrastructure. By doing so, we aim to enable languages with front ends that compile to LLVM IR, such as C (through Clang), FORTRAN, Julia, Rust, and Nim to natively support sparse matrix types as an extension of the language. Eventually, programmers who write code in such languages (specifically for machine learning, scientific computing, graphics, network/graph programs) will be able to enjoy the memory and performance gains from the languages adopting built-in sparse compilation in LLVM.

Research Questions

We formulate our research questions as below:

- **RQ1:** How can first-class support for sparse matrices be cleanly implemented inside the LLVM infrastructure?
- **RQ2:** Does first-class support for sparse matrices in LLVM ease the implementation of adopting sparse matrices in the front-end language?
- **RQ3:** Does our sparse matrix support bring noticeable memory and performance gains for sparse linear algebra programs?

Value to the User Community

We define our "user community" as the broader community of compiler and programming language developers, especially those who work on languages that make use of the LLVM com-

piler infrastructure. These developers and engineers may want to incorporate compiler features and optimizations provided by LLVM into their language, without having to build them from scratch. Of course, the general programming community will also benefit from having such optimizations provided in the language they use.

Typically, optimized linear algebra code (such as sparse matrices) is added to the compiler by linking external (and often proprietary) libraries, such as OpenBLAS or MKL. A compiler engineer has to manually link the library code into the compilation of the program, which may introduce unnecessary bloat and maintenance burden, as the library code gets updated. Moreover, the users of the language would also have to download the library code to enjoy those optimizations, which they may be reluctant to do so. Implementing such optimizations directly in LLVM, which we aim to do with sparse matrices, would allow compiler/language developers to readily adopt such features into their language without any reliance on external libraries.

To summarize the value propositions for bringing first-class support for sparse matrices in LLVM:

1. Languages that target LLVM can incorporate native sparse matrix algebra support
2. Users of such languages can then immediately enjoy the adoption of sparse compilation in the language
3. With the easy incorporation of sparse compilation, linear algebra code will be *more memory efficient* and *faster*

We emphasize that this is aligned with the recent trends and directions in the LLVM infrastructure. Namely:

- LLVM has recently implemented first-class (dense) matrix support; given the ubiquity of linear algebra programs, matrix and matrix operations are now considered essential first-class language features [5]
- The MLIR infrastructure already provides first-class support for sparse matrix/tensor algebra [6]; we aim to do the same for LLVM
- The Enzyme project [7] brings automatic differentiation to LLVM, which is typically implemented as a library/framework in a high-level language due to its complexity. This means, languages that hope to provide automatic differentiation (e.g. for machine learning support or scientific computing), do not have to implement their automatic differentiation library, and get such support for free if they compile to the LLVM IR
- Accelerated linear algebra, typically incorporated via external libraries, can be implemented directly in LLVM [8]. Programming languages that target LLVM can enjoy them without implementing them from scratch or relying on external libraries

Demo

We summarize our demo deliverables as follows:

- A brief overview of our `SparseMatrix`
- Identifying the sparse matrix optimizations in the generated code
- Memory usage and performance comparison by executing a real program with our optimization

Delivery

We summarize our project deliverables as follows:

- A specification and implementation of the `SparseMatrix` as an `IntrinsicType` in LLVM
- Implementation of Codegen for sparse matrix operations: matrix multiplication, matrix-vector multiplication, sparse-dense multiplication, addition, subtraction, and transpose in LLVM
- Memory and performance benchmarks on a sparse linear algebra test suite

Additional Information

Our implementation work will be done directly inside the LLVM codebase.

References

[1] SCOTT JENNIFER TUMA MIROSLAV. *Algorithms for sparse linear systems*. BIRKHAUSER VERLAG AG, 2023.

[2] Intel. Intel math kernel library. 2024. URL https://software.intel.com/en-us/mkl.

[3] Iain S. Duff, Michael A. Heroux, and Roldan Pozo. An overview of the sparse basic linear algebra subprograms: The new standard from the blas technical forum. *ACM Trans. Math. Softw.*, 28(2):239–267, jun 2002. ISSN 0098-3500. doi: 10.1145/567806.567810. URL https://doi.org/10.1145/567806.567810.

[4] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, CGO '04, page 75, USA, 2004. IEEE Computer Society. ISBN 0769521029.

[5] Adam Nemet. Llvm matrix support rfc. 2018. URL https://groups.google.com/g/llvm-dev/c/wsN3X4tBrxE/m/BZF5smRzDQAJ.

[6] Aart Bik, Penporn Koanantakool, Tatiana Shpeisman, Nicolas Vasilache, Bixia Zheng, and Fredrik Kjolstad. Compiler support for sparse tensor computations in mlir. *ACM Trans. Archit. Code Optim.*, 19(4), sep 2022. ISSN 1544-3566. doi: 10.1145/3544559. URL https://doi.org/10.1145/3544559.

[7] William Moses and Valentin Churavy. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12472–12485. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/9332c513ef44b682e9347822c2e457ac-Paper.pdf.

[8] Braedy Kuzma, Ivan Korostelev, João P. L. de Carvalho, José E. Moreira, Christopher Barton, Guido Araujo, and José Nelson Amaral. Fast matrix multiplication via compiler-only layered data reorganization and intrinsic lowering. *Software: Practice and Experience*, 53(9): 1793–1814, 2023. doi: https://doi.org/10.1002/spe.3214. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3214.