Overview

Sparse tensors are a core abstraction underlying computational science workflows. Sparse data contains a large portion of zero or "fill" values, and occurs naturally in critical national security applications such as graph algorithms, mesh simulations, and data analytics. Efficient sparse algorithms require complex performance tradeoffs between different data structures, execution strategies, and architectural mappings. State of the art sparse compiler frameworks such as TACO, SparseTIR, and Finch have demonstrated that compilers are an effective means of achieving peak performance, but are mostly serial and too complicated for domain scientists to use. We propose new sparse compiler abstractions for the kind of architectures encountered in leadership high performance supercomputers, as well as automatic optimization techniques to raise the level of abstraction, achieving performance portability and accelerating the pace of scientific discovery.

Sparse Compilers for High-Performance Architectures: Sparse frameworks must be able to generate implementations that can effectively utilize modern CPUs, GPUs, and cluster computing environments. The Finch tensor compiler has demonstrated impressive flexibility and performance in single-threaded settings, supporting loops, statements, if conditions, breaks, etc, over a wide variety of sparse formats, including run-length-encoding, symmetry, triangles, padding, or blocks, improving performance over previous approaches by up to 20×. We plan to augment Finch with new parallel capabilities. First, we will implement concurrent sparse datastructures, enabling concurrent updates to Finch's existing feature-rich format description language. Next, we will build load balancing schedulers, using the compiler to implement efficient sparsity-aware work-stealing to tackle the irregularity inherent in sparse workloads. Additionally, we plan to investigate GPU-specific memory allocation approaches to tackle dynamic sparse outputs. Finally, we will build a distributed sparse tensor framework capable of optimizing communication costs using graph partitioning techniques, which have previously gone unconsidered in compiler approaches.

Optimizing Sparse Computation Automatically Current frameworks require users to manually navigate a minefield of low-level decisions such as tiling, loop nesting, and format selection. To truly provide an accessible high-level interface, systems must optimize implementation details automatically. We have already built the first ever cost-based sparse program optimizer, Galley, automatically optimizing the fusion strategy, loop order, and data structures. However, fundamental challenges remain. Current optimizers rely on simplistic heuristics to predict costs, failing to capture sparsity patterns common to scientific datasets (e.g., block-diagonal, banded, triangular, or power-law distributions). To address this, we will adapt database techniques, such as sketching, sampling, and histogramming, to optimize scientific workloads. Additionally, sparse optimizers struggle to handle high-dimensional tensor network settings such as quantum computing or computational chemistry, where the cost of optimization and compilation can become a significant bottleneck. To handle these domains, we need to adapt advanced network scheduling algorithms to account for sparsity and use reshaping to limit the dimensionality of intermediate tensors.

Intellectual Merit This work would push forward the state of the art in sparse array programming by developing compiler techniques to make a comprehensive optimization landscape supporting all relevant programs and optimizations. No existing sparse tensor compilers support productive APIs in the entirety without sacrificing the full performance of modern HPC architectures. Our exploration of new sparsity estimators will be the first study of this problem beyond the restricted setting of sparse matrix multiplication.

Broader Impacts Sparse array programming is a fundamental computing paradigm across scientific and industrial disciplines. Performant implementations of productive interfaces will remove the cognitive overhead of program optimization from end users, freeing them to focus on the science and making the technology more accessible to a broader audience. This work will be incorporated into the PyData/Sparse project which sees over 750k downloads per month, immediately improving the computational efficiency of a wide range of important applications.

Keywords: sparse array programming; relational algebra; query optimization; high-performance