Finch: A Datastructure-Driven Array Programming Language

Willow Ahrens, Teo Collins, Radha Patel, Changwan Hong, Saman Amarasinghe

March 11, 2024

Abstract

From FORTRAN to Numpy, arrays have revolutionized how we express computation. Arrays are the highest-performing datastructure with a long history of investment and innovation, from hardware support to compiler technology. However, arrays can only handle dense rectilinear integer grids. Real world arrays often contain underlying structure, such as sparsity, runs of repeated values, or symmetry. We describe a compiler, Finch, which adapts existing programs and interfaces to the structure and sparsity of the inputs. Finch enables programmers to capture complex, real-world data scenarios with the same productivity they expect from dense arrays. Our approach enables new loop optimizations across multiple domains, unifying techniques such as sparse tensors, databases, and lossless compression.

1 Introduction

- Fortran supported lots of control structures, no data structures, just dense arrays.

- We tried to emulate complex data using only dense arrays but with complicated control structures.

- Recently, people tried to build frameworks to support structured data, but gave up a lot of program side

- We're bringing these together, both need to work together.

1.1 Contributions

1. A complete set of level formats for expressing data patterns hierarchically in FiberTree-style decompositions. The first such set of formats to efficiently capture banded, triangular, run-length-encoded, or sparse-run-length-encoded datasets. The formats capture many use cases, from random updates to sequential construction.

2. The Finch array language, mirroring simple for-loops with imperative code blocks and if-conditions. The first array programming language for the above data formats to support multiple outputs, affine indexing, and imperfectly-nested loops.

3. Tensor lifecycles, a simple constraint on tensor reads and writes that elegantly restricts Finch programs to avoid complex data dependencies, and enables tensor polymorphism by providing implementers with well-defined functions to overload.

4. Wrapper Tensors which modify existing datastructures and recombine them to support new patterns, such as affine indexing, padding, transposition, and slicing.

5. Wrapper Levels which modify existing datastructures and enabling complex features such as atomic updates or contiguous versus separate allocation.

6. We define the first mappings from the existing pydata/sparse array api high-level operations to low level finch notation

7. ¡Performance Contributions¿

Feature / Tool	**Halide**	**Taco**	**Cora**	**Taichi**	**Finch**
Einsums and Contractions	✓	✓	✓	✓	✓
Parallelism	✓	✓	✓	✓	✓
Multiple LHS	✓		✓	✓	✓
Affine Indices	✓			✓	✓
Recurrence	✓				
If-Conditions and Masks	✓	✓		✓	✓
Scatter Gather	✓			✓	✓
Early Break		✓		✓	✓

Table 1: Feature support across various tools.

Feature / Tool	**Halide**	**Taco**	**Cora**	**Taichi**	**Finch**
Dense	✓	✓	✓	✓	✓
Padded	✓				✓
One Sparse		✓		✓	✓
Sparse		✓			✓
Run-length					✓
Symmetric					✓
Regular Sparse Blocks		✓			✓
Irregular Sparse Blocks					✓
Ragged			✓		✓

Table 2: Support for various data structures across tools.

2 Additional Topics

1. Concordant iteration
2. Dimensionalization
3. Bounds analysis/queries
4. Performance warnings

3 Evaluation

3.1 Data-Driven Performance Engineering

3.1.1 Sparse-Sparse Matrix Multiply

Examples that demonstrate performance engineering in a datastructure-driven model

3.1.2 SpMV

3.2 Programming over flexible data

3.2.1 RLE Blur

-Compare against the zip/unzip version

- Which Datasets? Check taco-rle or looplets. (perhaps consider if a neural network is the right call here)

3.2.2 Histogram

3.2.3 Graph Applications

3.2.4 High-level kernel fusion

Find an example where fusing the python interface gives a big speedup over non-fused kernels.

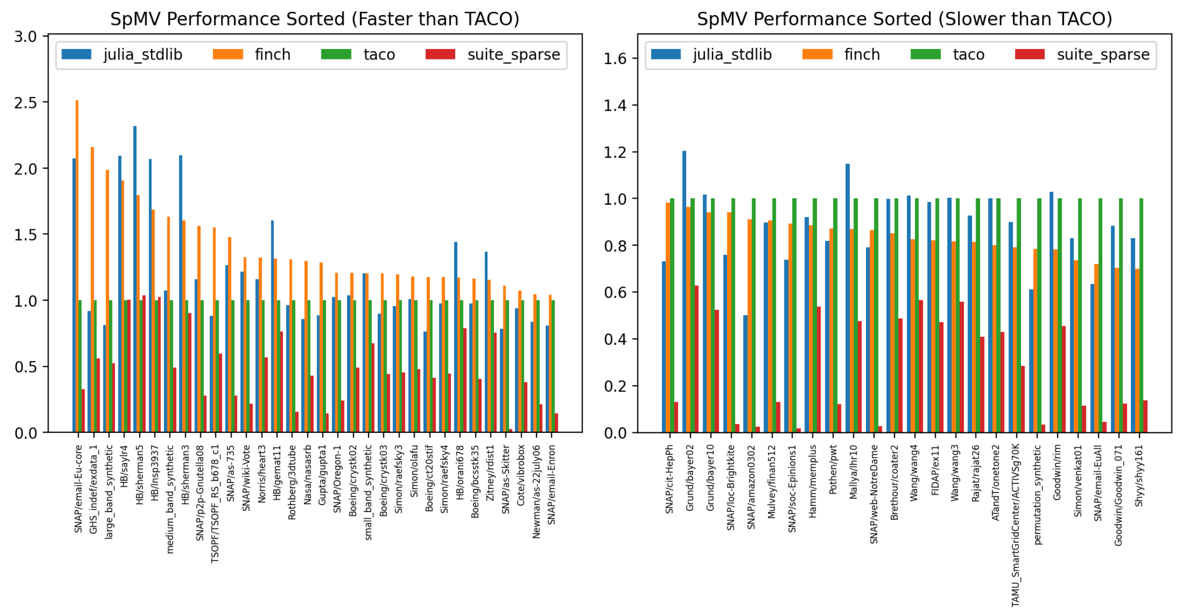
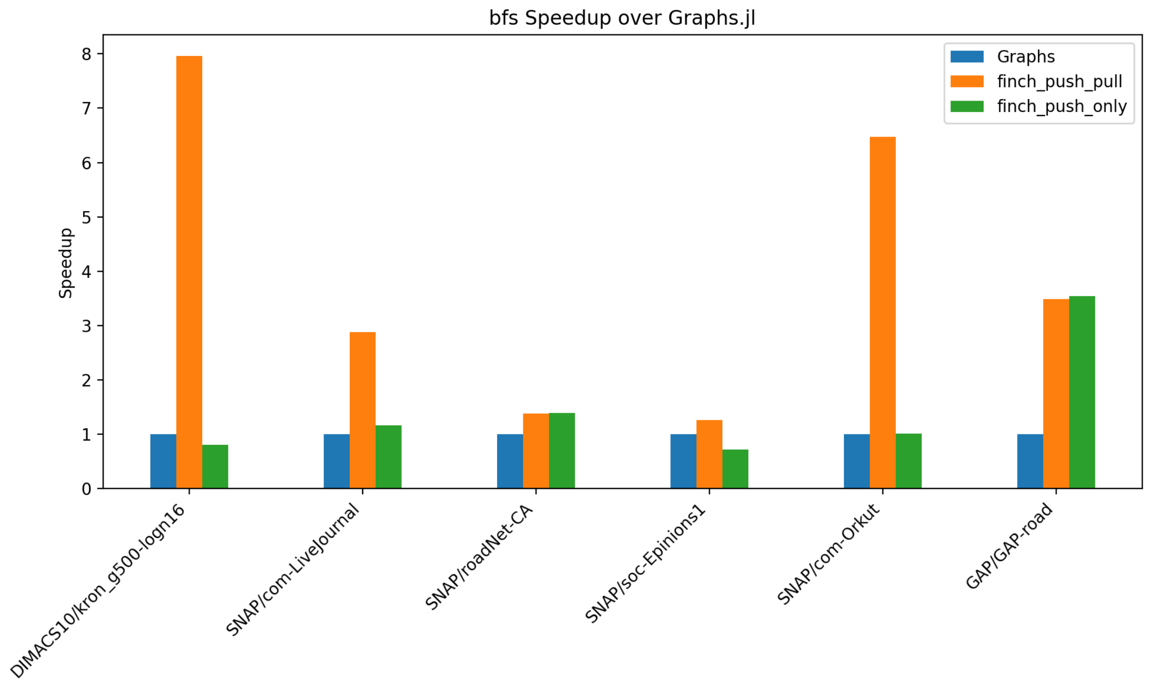


Figure 1: Performance of SpMV across various tools.

References



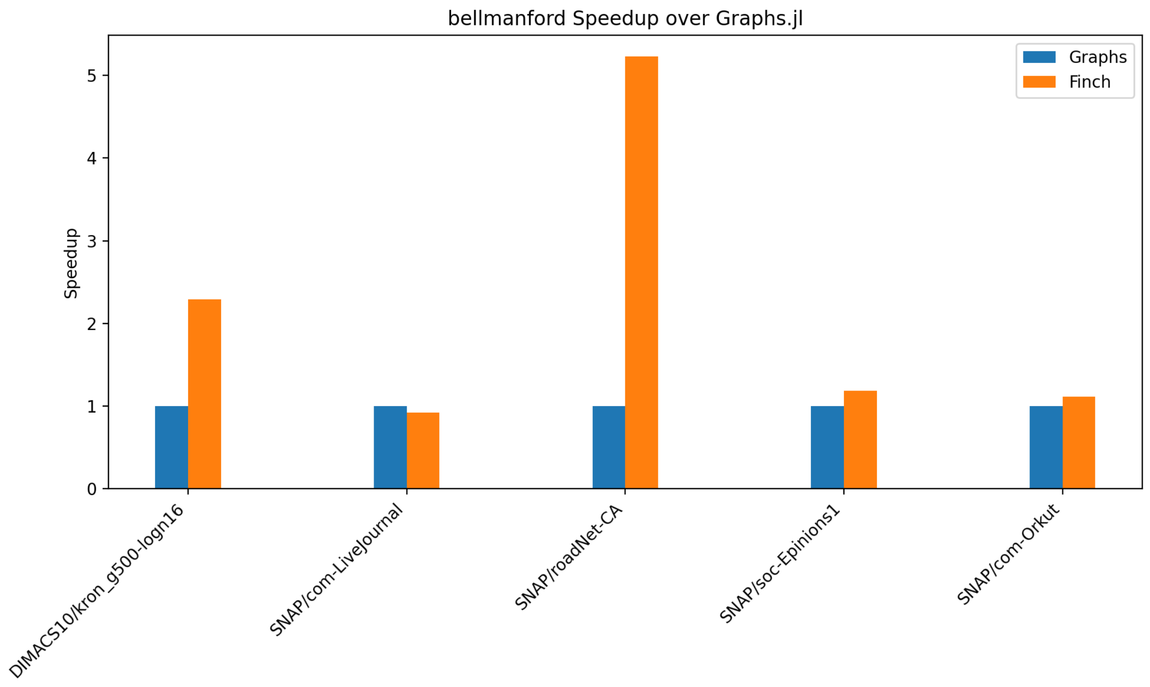


Figure 2: Performance of graph apps across various tools.