Finch: A Datastructure-Driven Array Programming Language

WILLOW AHRENS, MIT CSAIL, USA
TEO COLLINS, MIT CSAIL, USA
RADHA PATEL, MIT CSAIL, USA
KYLE DEEDS, University of Washington, USA
CHANGWAN HONG, MIT CSAIL, USA
SAMAN AMARASINGHE, MIT CSAIL, USA

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.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

ACM Reference Format:
Willow Ahrens, Teo Collins, Radha Patel, Kyle Deeds, Changwan Hong, and Saman Amarasinghe. 2024. Finch: A Datastructure-Driven Array Programming Language. *J. ACM* 37, 4, Article 111 (August 2024), 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

1.1 Contributions

(1) A rich structured array programming language with for-loops and complex control flow constructs at the same level of productivity of dense arrays. To our knowledge, the Finch programming language is the first to support if-conditions, early breaks, and multiple left hand sides over structured data, as well as complex accesses such as affine indexing or scatter/gather.
(2) More complex array structures than ever before. A complete level-by-level structure-description language for expressing the structure of data hierarchically. The first such set of formats

Authors' addresses: Willow Ahrens, MIT CSAIL, Cambridge, Massachusetts, USA, willow@csail.mit.edu; Teo Collins, MIT CSAIL, Cambridge, Massachusetts, USA, teoc@mit.edu; Radha Patel, MIT CSAIL, Cambridge, Massachusetts, USA, rrpatel@mit.edu; Kyle Deeds, University of Washington, Seattle, Washington, USA, kdeeds@cs.washington.edu; Changwan Hong, MIT CSAIL, Cambridge, Massachusetts, USA, changwan@mit.edu; Saman Amarasinghe, MIT CSAIL, Cambridge, Massachusetts, USA, saman@csail.mit.edu.

0004-5411/2024/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

to efficiently capture banded, triangular, run-length-encoded, or sparse datasets, and any combination thereof.

(3) The Finch compiler specializes programs to data structures in a predictable, deterministic approach, making it easier to search the complex space of programs and datastructures to find an appropriate fit for a given application. The tensor interface makes it easy to extend Finch to new level formats. A unique tensor lifecycle model enables polymorphism by analyzing the appropriate stages to insert simple, overloadable, interface functions such as initialization or finalization.

(4) A high-level array programming language and fusion interface for operations such as map, broadcast, or reduce that can be compiled to efficient code using the previous loop-level abstractions.

(5) We evaluate the productivity of our language in several case studies, showing that Finch can be used to accelerate a wide range of applications, from classic operations such as spmv and spgemm, to more complex applications such as image processing and graph analytics. We also demonstrate how Finch can fuse high-level operations to achieve a significant speedup over non-fused kernels.

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 BACKGROUND

2.1 Looplets

2.2 FiberTrees

2.3 Concordant Iteration

2.4 Protocols

3 THE FINCH LANGUAGE

3.1 Syntax and Semantics

4 THE FINCH COMPILER

4.1 Dimensionalization

4.2 Concordization

4.3 Bounds Analysis

4.4 Performance Warnings

4.5 Wrapperization

4.6 Simplification and Algebraic Transformations

5 THE TENSOR INTERFACE

5.1 Tensor Lifecycle, Declare, Freeze, Thaw, Unfurl

5.2 Level Abstraction

(1) fibers
(2) assembly
(3) reassembly

5.3 Core Level Langage Primitives

(1) SparseList
(2) SparseDict
(3) ...

5.4 Wrapper Tensors

5.5 Scalars

5.5.1 Sparse Scalars.

5.5.2 Early Break Scalars.

6 THE FINCH HIGH-LEVEL API (NEEDS A NAME)

6.1 Finch Logic

6.2 Finch Interpreter

6.3 Lowering

6.3.1 Heuristic Optimization.

7 EVALUATION

7.1 Data-Driven Performance Engineering

7.1.1 Sparse-Sparse Matrix Multiply. Examples that demonstrate performance engineering in a datastructure-driven model

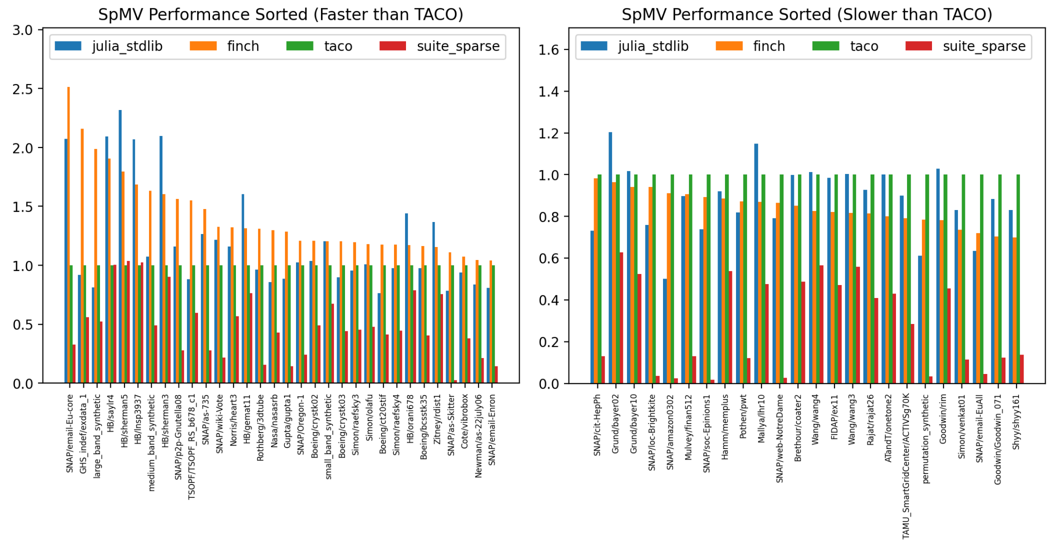


Fig. 1. Performance of SpMV across various tools.

7.1.2 *SpMV.*

7.2 Programming over flexible data

7.2.1 *Image Morphology.*

7.2.2 *Graph Analytics.*

7.2.3 *High-level kernel fusion.*
Find an example where fusing the python interface gives a big speedup over non-fused kernels.

ACKNOWLEDGMENTS

To Mateusz, Hameer, and Jaeyeon for their excellent programming contributions to the Finch codebase.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

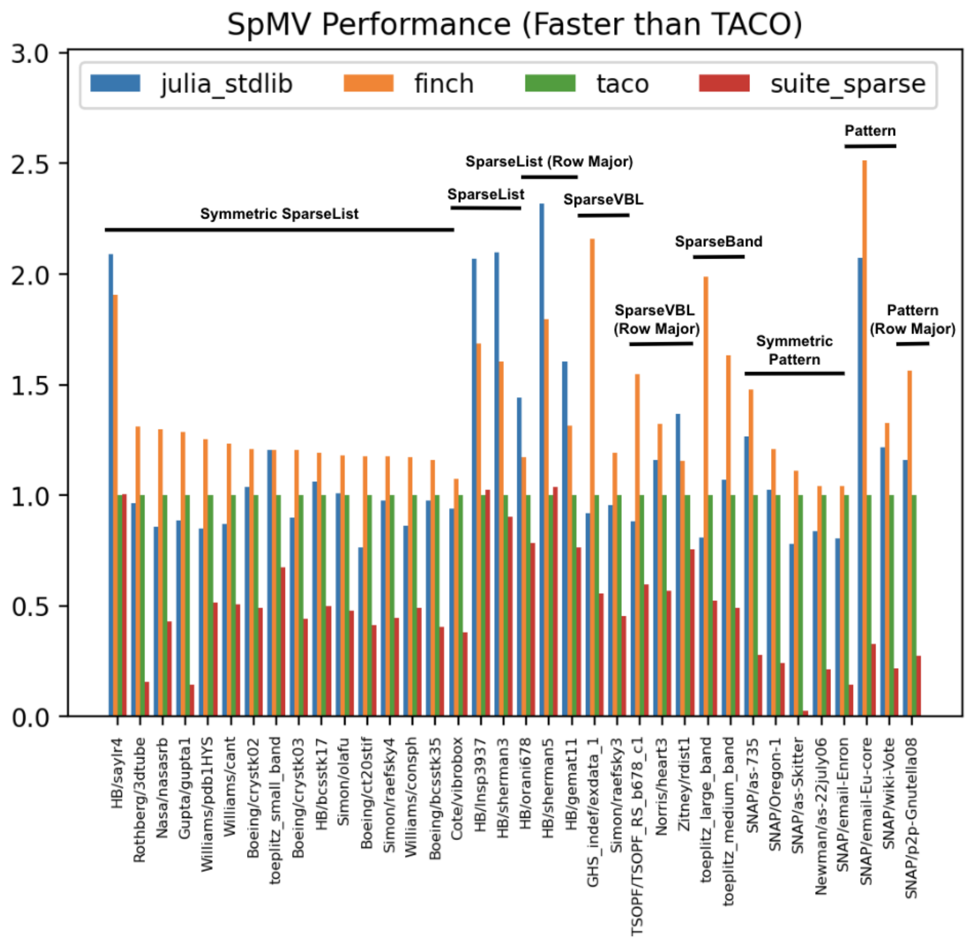


Fig. 2. Performance of SpMV by Finch format.

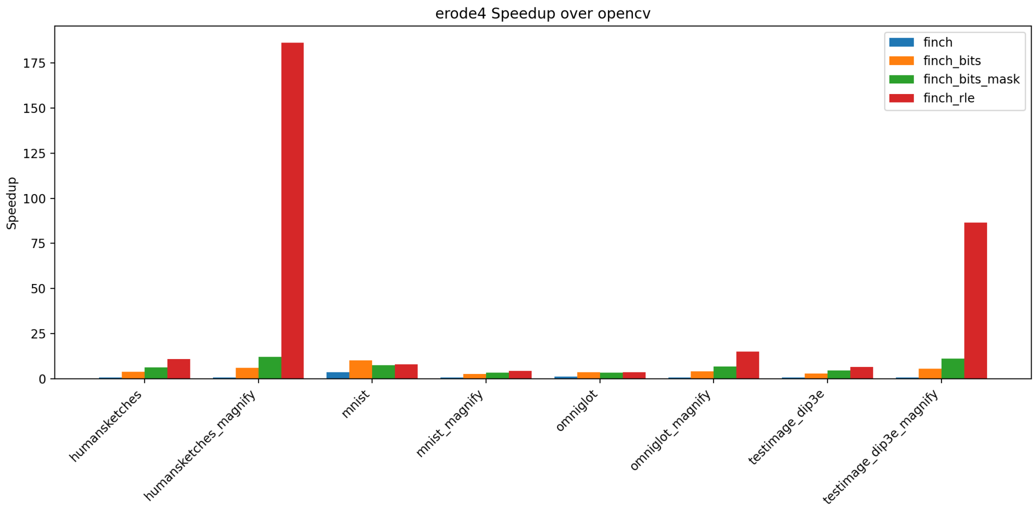


Fig. 3. Performance of Finch on erosion task (4 iterations).

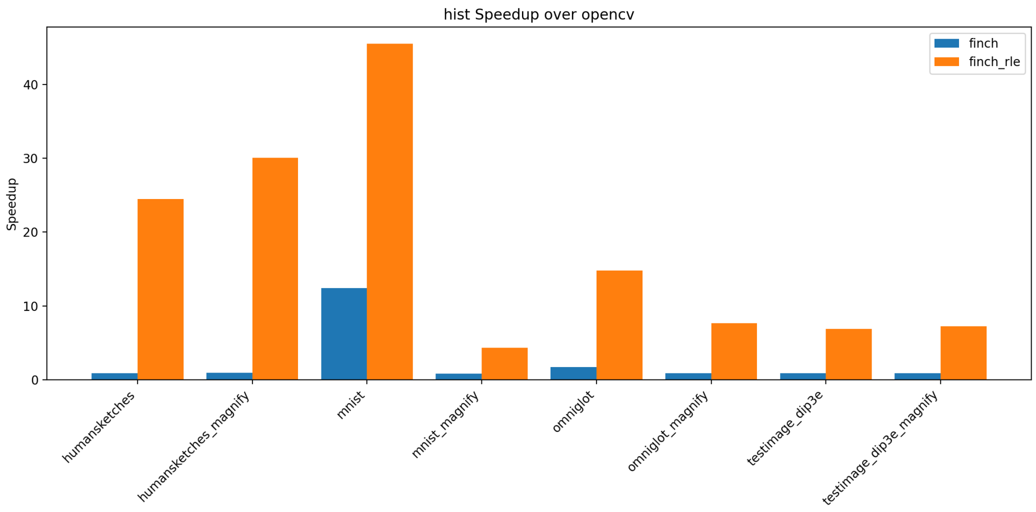


Fig. 4. Performance of Finch on masked histogram task.

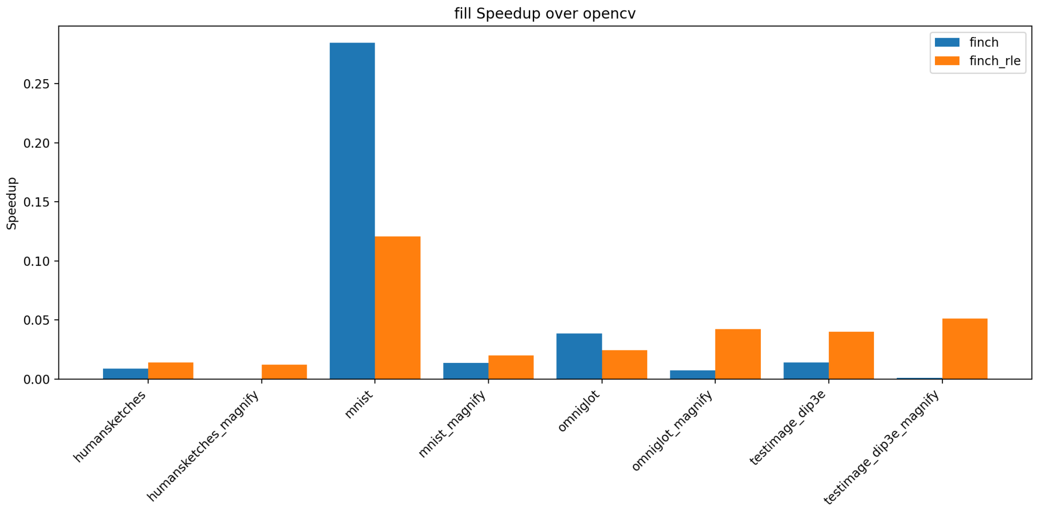
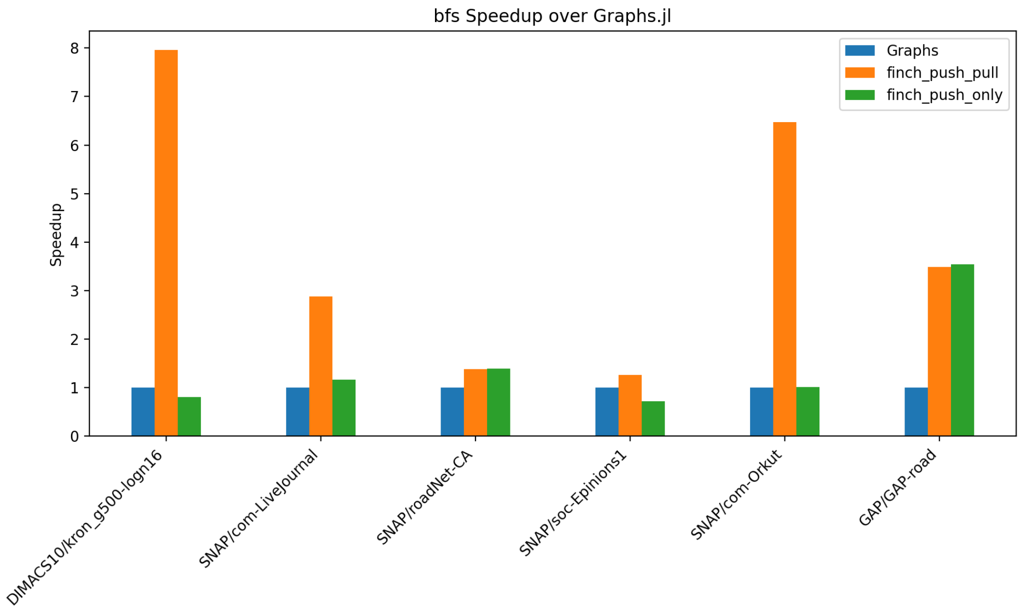


Fig. 5. Performance of Finch on flood fill task.



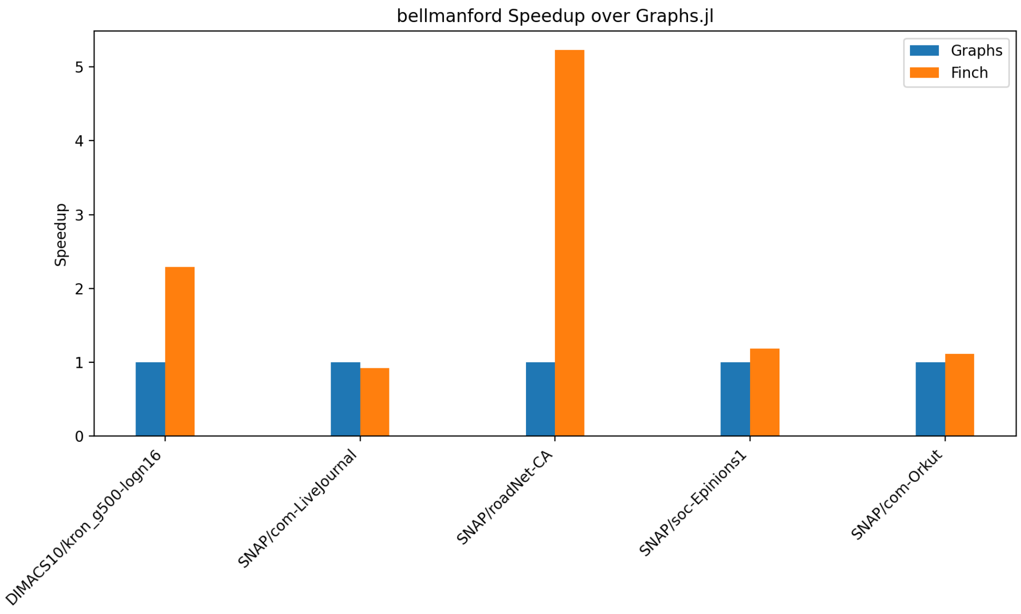


Fig. 6. Performance of graph apps across various tools.