15-745 F25 Project Proposal

Sun A Cho and Yufei Shi

October 23, 2025

1 Project Title

Exploring Automatic Parallelization in Tensor Algebra Compiler

2 Group Info

Sun A Cho (sunac@andrew.cmu.edu) and Yufei Shi (yshi2@andrew.cmu.edu)

3 URL for Project Web Page

https://cmu-15745-f25.github.io/project-website/

4 Project Description

4.1 Background

The Tensor Algebra Compiler (taco) [3] is a compiler-based library that automatically generates efficient kernels for tensor algebra on both dense and sparse tensors. It allows programmers to write high-level tensor expressions in index notation form without manually implementing optimized code for each possible combination of tensor type and storage format. The compiling and optimizing techniques implemented in taco enables it to determine how to traverse and combine tensor elements and generate optimized kernels that are competitive with hand-optimized kernels.

taco supports parallel execution of the generated kernels through OpenMP: it can automatically insert OpenMP directives for parallel loops, and it allows programmers to specify parallelized index variables and race strategies. However, many opportunities for automatic optimizations remain untapped. For example, taco does not automatically detect when a reduction can safely be executed in parallel, nor does it attempt to unroll loops.

4.2 Motivation

Modern tensor computations, particularly in machine learning and graph workloads, often involve high-dimensional and / or sparse data. Efficient execution of such workloads requires exploiting multiple forms of parallelism and optimizing memory accesses. While taco provides mechanisms for parallelization and programmer-specified strategies, we think that the compiler could leverage additional static information to further optimize performance:

Automatic instruction-level parallelization of tensor kernels. This includes detecting reductions that can safely be executed in parallel.

Loop unrolling with prefetching. This includes automatic loop unrolling and cache-aware prefetching.

For this project, we hope to explore and investigate these optimization opportunities, with goals scoped to what can be achieved within the course timeline:

75% Goal:

1. Commutativity-aware analysis: We will extend the use of OpenMP directives to explore the potential performance gain from executing the reducible operations commutatively, thus achieving a better parallelism even in reduction loops.
2. Loop unrolling analysis: Creating an analysis that automatically determines an optimal unroll factor while also potentially exposing the exact prefetching schedule based on the unroll factor.

100% Goal:

1. Commutative-aware analysis
2. Cache-aware loop unrolling
3. Automatic enhanced parallelism code generation: We will be using the analyses above (in the 75% goal) and re-generate the C++ code that will include all the pragmas and primitives to expose more parallelism in the computation.

125% Goal:

1. Commutative-aware analysis
2. Cache-aware automatic unrolling
3. Automatic enhanced parallelism code generation
4. Software pipelining: We will use software pipelining to even further extract instruction-level parallelism in the computations. This will require us to more in-depth, new analysis such as dependence analysis, evaluating the optimal loop initiation interval and completing a modulo register allocation.

5 Logistics

5.1 Plan of Attack and Schedule

Week 1 (Oct 20): Both: Literature review.
Both: Familiarizing ourselves with the taco code base.
Both: Building, running, and testing taco with OpenMP.

Week 2 (Oct 27): Both: Keep learning taco's code base and find where the OpenMP pragmas are determined and inserted in taco's compilation pipeline.
Both: Initial evaluations of taco's benchmarks and identify opportunities of improvements.

Week 3 (Nov 3): Sun A: Hand-analyze the taco IR to find the potential parallelism in a accumulation sum or computation (understand what situation parallel reduction makes sense) and brainstorm a cost function).
Yufei: Analyze the effect of loop unrolling on ILP and register allocations and brainstorm a cost function.

Week 4 (Nov 10): Sun A: Finalize the cost function and start writing a pass that will use the said cost function to determine whether to parallelize the reduction.
Yufei: Finalize the cost function and start writing a pass that will use the said cost function to determine the unrolling factor.

Week 5 (Nov 17): Both: Finalize the cost functions.
Both: Finalize the implementation of the analysis passes.
Both: Complete an initial evaluation on one benchmark application.

Week 6 (Nov 24): Both (Mon-Tue only): Run experiments and evaluate results.
Thanksgiving so nothing else is planned. We could use this as buffer time to work on things if we are sufficiently desperate.

Week 7 (Dec 1): Both: Finish writing the report.
Both: Working on the poster.

5.2 Milestone

By November 20th, we hope to have completed the analyses needed to automatically parallelize and loop-unroll the kernels, and have completed an initial evaluation on one benchmark application.

5.3 Literature Search

We reviewed several key literature related to tensor algebra compilation and sparse tensor computation frameworks.

Kjølstad et al., The Tensor Algebra Compiler [3].
The foundational paper that introduced taco.

Kjølstad, Sparse Tensor Algebra Compilation [4].
Kjølstad's dissertation which generalizes the theory behind taco and includes more details on optimizing transformations implemented in taco for improving parallelism and locality.

Kjølstad et al., Tensor Algebra Compilation with Workspaces [2].
This work introduces the concept of workspaces—temporary dense buffers used to optimize sparse tensor operations by reducing indirect accesses.

Yadav et al., DISTAL: The Distributed Tensor Algebra Compiler [7].
This paper introduces DISTAL which generalizes tensor algebraic computations to distributed and heterogeneous environments. DISTAL is implemented by extending taco and add scheduling languages for mapping computation across different hardware.

Bik et al., Compiler Support for Sparse Tensor Computations in MLIR [1].
This paper describes the integration of sparse tensor compilation into the MLIR framework [6]. This approach separates sparsity concerns from algorithmic logic, letting the compiler handle low-level implementation details.

5.4 Resources Needed

We anticipate only using the taco compiler and the LLVM compiler infrastructure for this project. The taco compiler is open-source and available at `https://github.com/tensor-compiler/taco`, and the LLVM toolchain (including its bundled OpenMP library) is open-source and available at `https://releases.llvm.org` [5]. We will be primarily developing this project on macOS with arm CPUs, which is available to all members of the group. If we have the opportunity to also target GPUs, we will be using the GHC clusters, which have x86 CPUs and Nvidia RTX GPUs, available through CMU computing services.

5.5 Getting Started

We have studied some literature related to this project, and were able to build taco with OpenMP enabled. However, we have not yet able to run the tests successfully due to taco's default `gtest` framework not handling commands correctly on macOS. We expect to solve this by the end of this week.

References

[1] 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), September 2022.

[2] Fredrik Kjolstad, Willow Ahrens, Shoaib Kamil, and Saman Amarasinghe. Tensor algebra compilation with workspaces. In *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 180–192, 2019.

[3] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. The tensor algebra compiler. *Proc. ACM Program. Lang.*, 1(OOPSLA), October 2017.

[4] Fredrik Berg Kjølstad. *Sparse tensor algebra compilation.* PhD thesis, Massachusetts Institute of Technology, 2020.

[5] C. Lattner and V. Adve. Llvm: a compilation framework for lifelong program analysis transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86, 2004.

[6] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, 2021.

[7] Rohan Yadav, Alex Aiken, and Fredrik Kjolstad. Distal: the distributed tensor algebra compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, page 286–300, New York, NY, USA, 2022. Association for Computing Machinery.