Symmetry-Exploiting Compiler Optimizations for MLIR Tensor Programs

GAURAV ARYA and APURVA GANDHI

1 Project Web Page

https://github.com/gaurav-arya/compilers-symmetry-final-project

2 Project Description

Many tensor and linear algebra workloads in machine learning and general scientific computing—such as attention matrices, covariance computations, or physical simulation kernels—exhibit computation with symmetric structure. But without explicit user annotation, modern compiler stacks may treat such tensors as dense and unstructured, resulting in redundant computation and memory usage. In this project, we propose to implement compiler analyses in MLIR [1] for automatically detecting and exploiting symmetry in tensor computations. Specifically, we aim to build a compiler analysis and transformation pass on MLIR that:

(1) Detects symmetric outputs from tensor computations.
(2) Propagates symmetry metadata through the IR.
(3) Applies symmetry-exploiting code transformations.

By detecting symmetry at compile time, we can perform optimizations that significantly reduce memory usage and redundant computation of tensor programs.

Example. Figure 1 presents an example of a simple tensor program that would benefit from our proposed compiler pass[1]. This program is an example of a computation that *creates* symmetry: in the pseudocode (Fig. 1a), observe that even though the input matrix A may not be symmetric, the intermediate tensor B is guaranteed to be symmetric. Annotating this symmetry in the IR can enable downstream optimizations: in particular, we can replace the assignment $D \leftarrow C^T$ with a copy, as the transpose of a symmetric matrix is itself.

Research Questions. We consider the following research question in this project:

(RQ1) Can symmetry detection be effectively performed at the MLIR level, and to what extent can this symmetry be exploited for performance speedups on linear algebra workloads?

(RQ2) Can we effectively compose symmetry-exploiting optimizations with other numeric compiler transformations, such as algebraic simplifications or even automatic differentiation (AD)?

Our main research question will be RQ1. Previous works on symmetry-optimizing compiler optimizations, such as SySTeC [4], have demonstrated the potential for significant speedups by exploiting symmetry in tensor programs. By working in the StableHLO dialect of MLIR, we have access to high-level linear algebra structure (tensors, matrix multiplications, transposition, etc.), as can be seen in, e.g., Fig. 1b. Thus in this project we expect to be able to give at least a partially affirmative answer to RQ1, while also investigating limitations (e.g. IR expressivity) that may prevent further speedups.

As a stretch goal, we hope to also investigate RQ2. An advantage of working in MLIR is that we should be able to compose our new analyses with a rich set of existing compiler transformations

[1]MLIR is taken from https://github.com/EnzymeAD/Enzyme-JAX/issues/1481

Authors' Contact Information: Gaurav Arya, gauravar@cs.cmu.edu; Apurva Gandhi, apurvag@cs.cmu.edu.

```
1: function MAIN(A) // 4x4 matrix, corresponds to %arg0
2:     B ← LowerTriangle(A) + LowerTriangle(A)^T // corresponds to %3
3:     C ← B * B // corresponds to %4
4:     D ← C^T // corresponds to %5
5:     E ← D + C // corresponds to %6
6:     return E
```

(a) Pseudocode

```
func.func @main(%arg0: tensor<4x4xf32> {enzymexla.memory_effects = []})
-> tensor<4x4xf32> attributes {enzymexla.memory_effects = []} {
    %c = stablehlo.constant dense<[[true, false, false, false], [true, true, false, false],
                        [true, true, true, false], [true, true, true, true]]> : tensor<4x4xi1>
    %cst = stablehlo.constant dense<0.000000e+00> : tensor<4x4xf32>
    %c_0 = stablehlo.constant dense<[[false, true, true, true], [false, false, true, true],
                        [false, false, false, true], [false, false, false, false]]> : tensor<4x4xi1>
    %0 = stablehlo.transpose %arg0, dims = [1, 0] : (tensor<4x4xf32>) -> tensor<4x4xf32>
    %1 = stablehlo.select %c_0, %0, %cst : tensor<4x4xi1>, tensor<4x4xf32>
    %2 = stablehlo.select %c, %arg0, %cst : tensor<4x4xi1>, tensor<4x4xf32>
    %3 = stablehlo.add %2, %1 : tensor<4x4xf32>
    %4 = stablehlo.dot_general %3, %3, contracting_dims = [1] x [0], precision = [DEFAULT, DEFAULT]
        : (tensor<4x4xf32>, tensor<4x4xf32>) -> tensor<4x4xf32>
    %5 = stablehlo.transpose %4, dims = [1, 0] : (tensor<4x4xf32>) -> tensor<4x4xf32>
    %6 = stablehlo.add %5, %4 : tensor<4x4xf32>
    return %6 : tensor<4x4xf32>
}
```

(b) StableHLO MLIR

Fig. 1. An example of code that presents opportunities for symmetry-exploiting optimizations.

for tensor programs. One interesting such compiler transformation is *automatic differentiation*. To elaborate, for our project, we would be working in the codebase of the Enzyme AD [2, 3, 5] tool for MLIR/LLVM. The Enzyme team believes that symmetry-exploiting optimizations could be particularly beneficial when composed with AD, and it would be interesting to investigate whether our symmetry-exploiting optimizations can be used to improve the efficiency of AD.

Goals.

- **75% Goal:** Select an appropriate representation of symmetry for StableHLO tensors; successfully detect and annotate symmetry generated in at least two tensor operations (e.g. $A + A^T$ and $A * A^T$). and implement and benchmark at least one symmetry-exploiting optimization pass (e.g. A^T can be replaced with A if A is symmetric).
- **100% Goal:** Everything in 75% goal; implement and benchmark at least one additional symmetry-exploiting optimization pass; benchmark performance on at least three different programs, including at least one that uses tensors of dimension greater than 2; and demonstrate benefits of composing symmetry-exploiting optimizations with at least one other MLIR compiler pass.
- **125% Goal:** Everything in 100% goal; demonstrate benefits of composing symmetry-exploiting optimizations with AD; and develop and benchmark compiler analyses for exploiting other forms of structure (sparsity, upper/lower triangularity, etc.) in addition to symmetry.

3 Logistics

3.1 Plan of Attack and Schedule

- **Oct 23 - 29:** Get familiar with relevant parts of MLIR and StableHLO IR, including understanding the IR, how to manipulate it and how to add symmetry metadata to it.
- **Oct 30 - Nov 5:** Enumerate symmetry generation patterns to detect in our pass. Write a pass to detect at least one symmetry-generating pattern.
- **Nov 6 - 12:** Write a pass to leverage the annotated symmetry metadata to perform at least one symmetry-exploiting optimization.
- **Nov 13 - 19:** Enrich analysis pass to detect additional symmetry-generating patterns. Write milestone report.
- **Nov 20 - 26:** Implement at least one more symmetry-exploiting optimization pass and benchmark these. Based on progress, consider doing the stretch task of benchmarking how our optimizations help when composed with Enzyme's AD or other MLIR algebraic simplification passes.
- **Nov 27 - Dec 3:** Based on progress, consider stretch task of detecting and exploiting another form of structure (e.g., sparsity, upper/lower triangularity, etc.). Write final report and prepare poster.

We plan to divide work 50/50 by pair-coding for most/all tasks.

3.2 Milestone

Our project milestone will be to: (1) Detect and annotate symmetry generated in at least one tensor operation (e.g., $A + A^T$). (2) Leverage this annotation to perform at least one symmetry-exploiting optimization.

3.3 Literature Search

We have listed some relevant papers in the References below. SySTeC [4] is a recent paper that tries to exploit symmetry and sparsity, but works with lower-level IR that represents computations as loop-nests with indexed computation. In our project, we plan to explore using MLIR and work with the StableHLO dialect, which we hypothesize will: (1) make it easier to reason about symmetry since this IR operates at the tensor level and (2) allow us to easily integrate with real-world projects like Enzyme AD [2, 3, 5].

3.4 Resources Needed

Our understanding is that we will be able to use CPUs to effectively benchmark our optimizations —our local machines should be sufficient for this. Although it is not necessary, we could potentially also benchmark on GPUs, and have access to a cluster for doing so.

3.5 Getting Started

We have had an initial talk with the Enzyme team to familiarize ourselves with the codebase (https://github.com/EnzymeAD/Enzyme-JAX). The following PRs / issues / code can help us get started:

- https://github.com/EnzymeAD/Enzyme-JAX/pull/1479 (adding a symmetric matrix multiply operation to StableHLO)
- https://github.com/EnzymeAD/Enzyme-JAX/issues/1481 and https://github.com/EnzymeAD/Enzyme-JAX/issues/1489 (example IR that opportunities optimizations for symmetry-exploiting optimizations)

- https://github.com/EnzymeAD/Enzyme-JAX/blob/a7c38bce984c3adedafb8e03282c0e39640ab6f9/src/enzyme_ad/jax/Utils.h#L557 (existing analysis for proving a value is not NaN)
- https://github.com/EnzymeAD/Enzyme-JAX/pull/1482 (example of annotating IR with analysis results)

References

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

[2] William S. Moses and Valentin Churavy. 2020. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 1046, 14 pages.

[3] William S. Moses, Valentin Churavy, Ludger Paehler, Jan Hückelheim, Sri Hari Krishna Narayanan, Michel Schanen, and Johannes Doerfert. 2021. Reverse-mode automatic differentiation and optimization of GPU kernels via enzyme. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 61, 16 pages. doi:10.1145/3458817.3476165

[4] Radha Patel, Willow Ahrens, and Saman Amarasinghe. 2025. SySTeC: A Symmetric Sparse Tensor Compiler. In *Proceedings of the 23rd ACM/IEEE International Symposium on Code Generation and Optimization* (Las Vegas, NV, USA) *(CGO '25)*. Association for Computing Machinery, New York, NY, USA, 47–62. doi:10.1145/3696443.3708919

[5] Mai Jacob Peng, William S. Moses, Oleksandr Zinenko, and Christophe Dubach. 2025. Sound and Modular Activity Analysis for Automatic Differentiation in MLIR. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 347 (Oct. 2025), 28 pages. doi:10.1145/3763125