Agentic MLIR: LLM-Planned Transform IR with Verified Correctness

Robert Luo Ŧ, Jinman Zhao Ŧ, Seyed Aryan Vahabpour ŧ

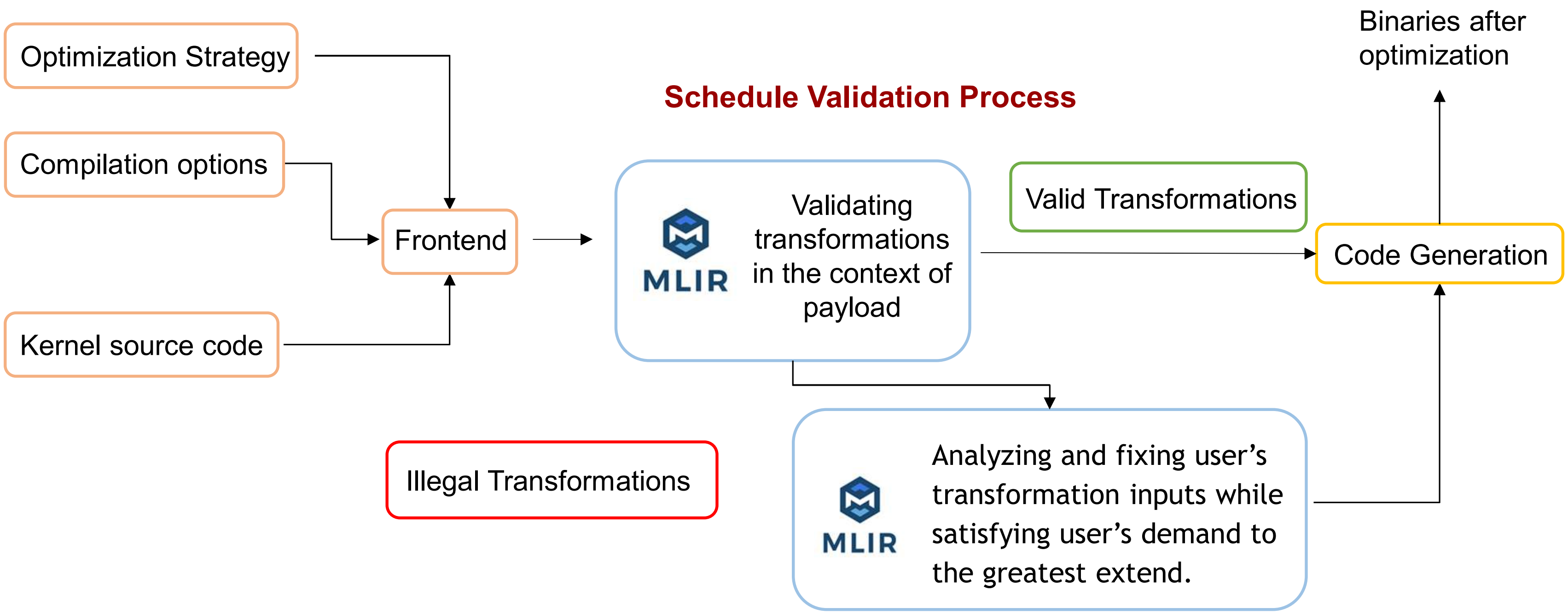
Xingyu Yue ł , Kai-Ting Amy Wang*ł

Ŧ University of Toronto
Toronto, Ontario, Canada
yulang.luo@mail.utoronto.ca
jzhao@cs.toronto.edu

ŧ University of Waterloo
Waterloo, Ontario, Canada
avahabpo@uwaterloo.ca

ł Huawei Canada Research Center
Markham, Ontario, Canada
xingyu.yue@huawei.com
kai.ting.wang@huawei.com

Background: A Transform Driven Polyhedral Compiler



* J.Zhao, S.A.Vahabpour, X.Yue, K.-T.A.Wang and T.S.Abdelrahman, "PolyMorphous: An MLIR-Based Polyhedral Compiler with Loop Transformation Primitives," 2025, IEEE International Parallel and Distributed Processing Symposium (IPDPS)

PyDSL

```
def mvt_schedule(targ: AnyOp):
    fuse_target1 = match(targ, "fuse_target1")
    fuse_target2 = match(targ, "fuse_target2")
    fuse_res = fuse(fuse_target2, fuse_target1, 2)
    tile_res = tile(fuse_res, [32, 32], 4)
    reorder(get_loop(tile_res, 2), get_loop(tile_res, 3))
    parallel(get_loop(tile_res, 0))


@compile(locals(), transform_seq=mvt_schedule, dump_mlir=False,
          auto_build=True, target_class=Poly)
def mvt(n: Index, x1: MemrefF321D, x2: MemrefF321D,
        y1: MemrefF321D, y2: MemrefF321D, A: MemrefF322D) -> None:
    """@tag("fuse_target1")"""
    for i in arange(n):
        for j in arange(n):
            x1[i] = x1[i] + A[i, j] * y1[j]
    """@tag("fuse_target2")"""
    for i in arange(n):
        for j in arange(n):
            x2[i] = x2[i] + A[i, j] * y2[j]
```

Matching target loop

Fusing the loop

Tiling the fused the loop

Interchanging the loop i and j

Parallelizing the outermost loop

Enable polyhedral analysis

Tagging target loop

Optimization Strategy

Capable of composing many transform

Separating the Schedule From the Source Code

Compilation options

Kernel source code

https://github.com/Huawei-CPLLab/PyDSL

Schedule Validation Process (Quick Review)

- Let R depends on S and $\theta_S(x_S)$ represents the time operator $S(x_S)$ is going to run (similarly for R)
  - The scheduling function θ has the following format: $\theta_S(x_S) = C_S T_{S_C} x_S + t_S$
  - Matrix C allows us to correct the illegal transformations using skew or shifts if possible.
- The set of equations $\Delta_{\{R,S\}} = \theta_R(x_R) - \theta_S(x_S) \geq 0$ must hold for all the instances of S and R with dependencies.
- If any of these dependencies fails, we need to correct the schedule by shifts or skews.
- If these inequalities hold for all the dependencies, the transformation is legal, no need for correction.
- Affine form of Farkas Lemma:

$$T_{R,\bullet}\vec{x}_R + t_R - (T_{S,\bullet}\vec{x}_S + t_S) - \delta = \lambda_0 + \vec{\lambda}^T \left(D \begin{pmatrix} \vec{x}_S \\ \vec{x}_R \end{pmatrix} + \vec{d} \right)$$

- To be able to to find the correct value for shifts and skews, we solve the system of equations to find matrix C shown above.
- We use the Presburger-Simplex solver available in MLIR to solve for the unknown variables.

Leveraging Affine Analysis

```
func.func private @chunking(%arg0: index, %arg1: index, %arg2: memref<?xi32>, %arg3:
memref<?xi32>) {
    affine.for %arg4 = 1 to %arg0 {
      %0 = arith.index_cast %arg4 : index to i32
      affine.store %0, %arg2[%arg4] : memref<?xi32>
      affine.for %arg5 = 1 to %arg1 {                   [1]
        %1 = affine.load %arg3[%arg5] : memref<?xi32>
        %2 = affine.load %arg2[%arg4] : memref<?xi32>
        %3 = arith.addi %1, %2 : i32                    [2]
        affine.store %3, %arg3[%arg5] : memref<?xi32>
      }
    }
    return
}
```

Source code for a simple test case

```
for (Operation *writeOp : it->second) {
      …
      if (readMap.find(arg) != readMap.end()) {
        for (Operation *readOp : readMap[arg]) {
          …
          for (unsigned i = 1; i <= commonLoops + 1; i++) {
            …
            DependenceResult result = checkMemrefAccessDependence(
                writeAccess, readAccess, i, &readAfterWrite, nullptr, false);
            if (hasDependence(result))
              recordDependenceEdge(dependenceEdges, writeOp, readOp,
                                   readAfterWrite, numSym, arg, i - 1,
                                   EdgeType::RAW);
            result = checkMemrefAccessDependence(
                readAccess, writeAccess, i, &writeAfterRead, nullptr, false);
            if (hasDependence(result))
              recordDependenceEdge(dependenceEdges, readOp, writeOp,
                                   writeAfterRead, numSym, arg, i - 1,
                                   EdgeType::WAR);
          }
        }
      }
```

Source code for building Dependence Polyhedrons

```
%1 = affine.load %arg3[%arg5] : memref<?xi32>
affine.store %3, %arg3[%arg5] : memref<?xi32>
depth: 2
parallelDepth: -1
Domain: 0, Range: 4, Symbols: 2, Locals: 0
( ) -> ( Id<0xaaaae6c232d0> Id<0xaaaae6c23fd0> Id<0xaaaae6c232d0>
Id<0xaaaae6c23fd0> ) : [ Id<0xaaaae6c1db00> Id<0xaaaae6c1e1f0> ]11
constraints
(Value  Value Value Value Value Value const)
  0 -1  0  1  0  0  0  = 0
 -1  0  1  0  0  0  0  = 0
  0 -1  0  1  0  0  0  = 0
  1  0  0  0  0  0 -1 >= 0
 -1  0  0  0  1  0 -1 >= 0
  0  1  0  0  0  0 -1 >= 0
  0 -1  0  0  0  1 -1 >= 0
  0  0  1  0  0  0 -1 >= 0
  0  0 -1  0  1  0 -1 >= 0
  0  0  0  1  0  0 -1 >= 0
  0  0  0 -1  0  1 -1 >= 0

numSymbolVars: 2
numDimVars: 4
numLocalVars: 0
```

→ Dependence Polyhedron

[1] WAR dependency at level 2

```
%1 = affine.load %arg3[%arg5] : memref<?xi32>
affine.store %3, %arg3[%arg5] : memref<?xi32>
depth: 0
Domain: 0, Range: 4, Symbols: 2, Locals: 0
( ) -> ( Id<0xaaaae6c232d0> Id<0xaaaae6c23fd0> Id<0xaaaae6c232d0>
Id<0xaaaae6c23fd0> ) : [ Id<0xaaaae6c1db00> Id<0xaaaae6c1e1f0> ]10
constraints
(Value  Value Value Value Value Value const)
  0 -1  0  1  0  0  0  = 0
  1  0  0  0  0  0 -1 >= 0
 -1  0  0  0  1  0 -1 >= 0
  0  1  0  0  0  0 -1 >= 0
  0 -1  0  0  0  1 -1 >= 0
  0  0  1  0  0  0 -1 >= 0
  0  0 -1  0  1  0 -1 >= 0
  0  0  0  1  0  0 -1 >= 0
  0  0  0 -1  0  1 -1 >= 0
 -1  0  1  0  0  0 -1 >= 0

numSymbolVars: 2
numDimVars: 4
numLocalVars: 0
```

→ Dependence Polyhedron

[2] WAR dependency at level 0

Leveraging the Simplex Solver

```
SmallVector<DynamicAPInt, 8> mlir::affine::correctIter(…) {

  IntMatrix farkasRHS(0, 0);
  IntMatrix farkasLHS(0, numStmt * (maxDim + maxSym + 1));

  SmallVector<int64_t, 8> StrongsatisfyPositions;
  DenseMap<int, bool> ShouldUpdateOpOrder;
  for (dependenceEdge e : dependenceEdges) {
    if (!e.isEmpty) {
      …
      computeFarkasRHS(&(e.dependenceConstraints), farkasRHS, e.TRs,
                    schedules[e.src], schedules[e.dst], maxDim,
maxSym,
                     numLocal);
      computeFarkasLHS(schedules[e.src], schedules[e.dst],
farkasLHS, e.src, e.dst, scheduleOrder, maxDim, maxSym, numLocal,
false);
       …
    }
  LexSimplex simplex(farkasLHS.getNumColumns() +
farkasRHS.getNumColumns());
  IntMatrix simplexEq(farkasLHS.getNumRows(),
farkasLHS.getNumColumns() + farkasRHS.getNumColumns() + 1);
  …
  auto res = simplex.findIntegerLexMin();
 …
}
```

Our source code for correcting the transformation using the simplex method and Farkas Lemma

Farkas LHS

```
0 -1 1 -1 1 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 -1 1 -1 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 -1 -1 -1 -1 -1 -1 -1 -1 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 1 -1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 -1 1 0 0 1 -1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 0 0 0 0 1 -1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 -1 0 0 1 -1 0 0 0 0 0 0 1 -1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 -1 -1 -1 -1 -1 -1 -1 -1
```

Farkas RHS

```
-1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 -1 0 0 0 0 1 0 0
0 0 0 -1 0 0 0 0 1 0
0 0 0 0 -1 0 0 0 0 1
0 0 0 0 0 0 -1 0 0 0
0 0 0 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 -1 0 0 0
0 0 0 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 -1 0 0 0
0 0 0 0 0 -1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

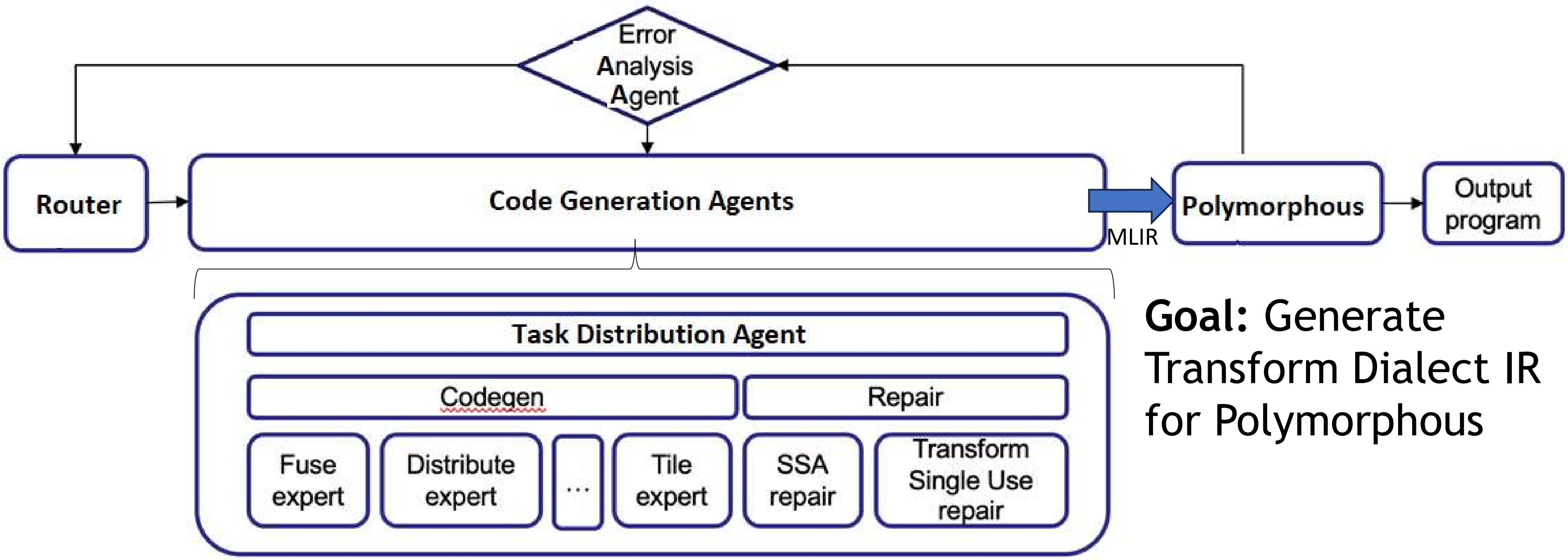
Farkas Matrices for the WAR dependency at depth 2

Output of Presburger Simplex Solver

```
Found a solution!
1 0 0 0 0 1 0 1 0 0 2 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
```

Agentic MLIR: LLM-Planned Transform IR Generation

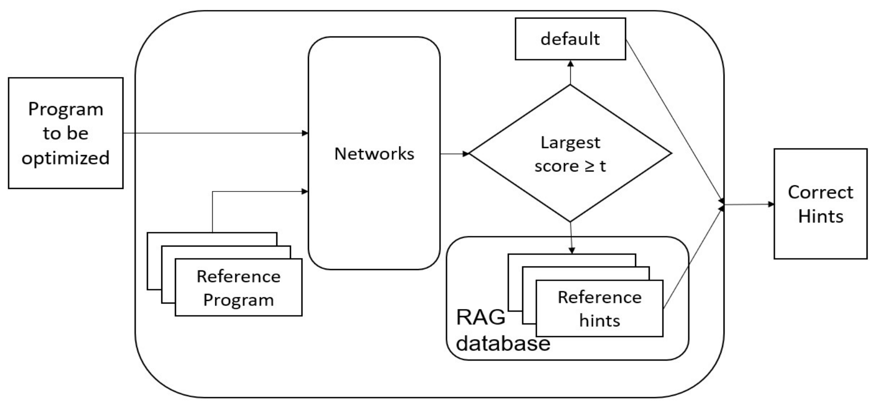
High-level System Diagram



Goal: Generate Transform Dialect IR for Polymorphous

Router: RAG Database

1. Identify the similarity between the input kernel and the human expert kernels that are in the database.
2. Provide appropriate hints to other agents for how to optimize the input kernels.
3. Router is trained using AI generated data and AI generated ground truth.
4. Retrieval Augmented Generation: router is a RAG database. The Code Generation Agents uses hints supplied by the Router in order to optimize the kernels.

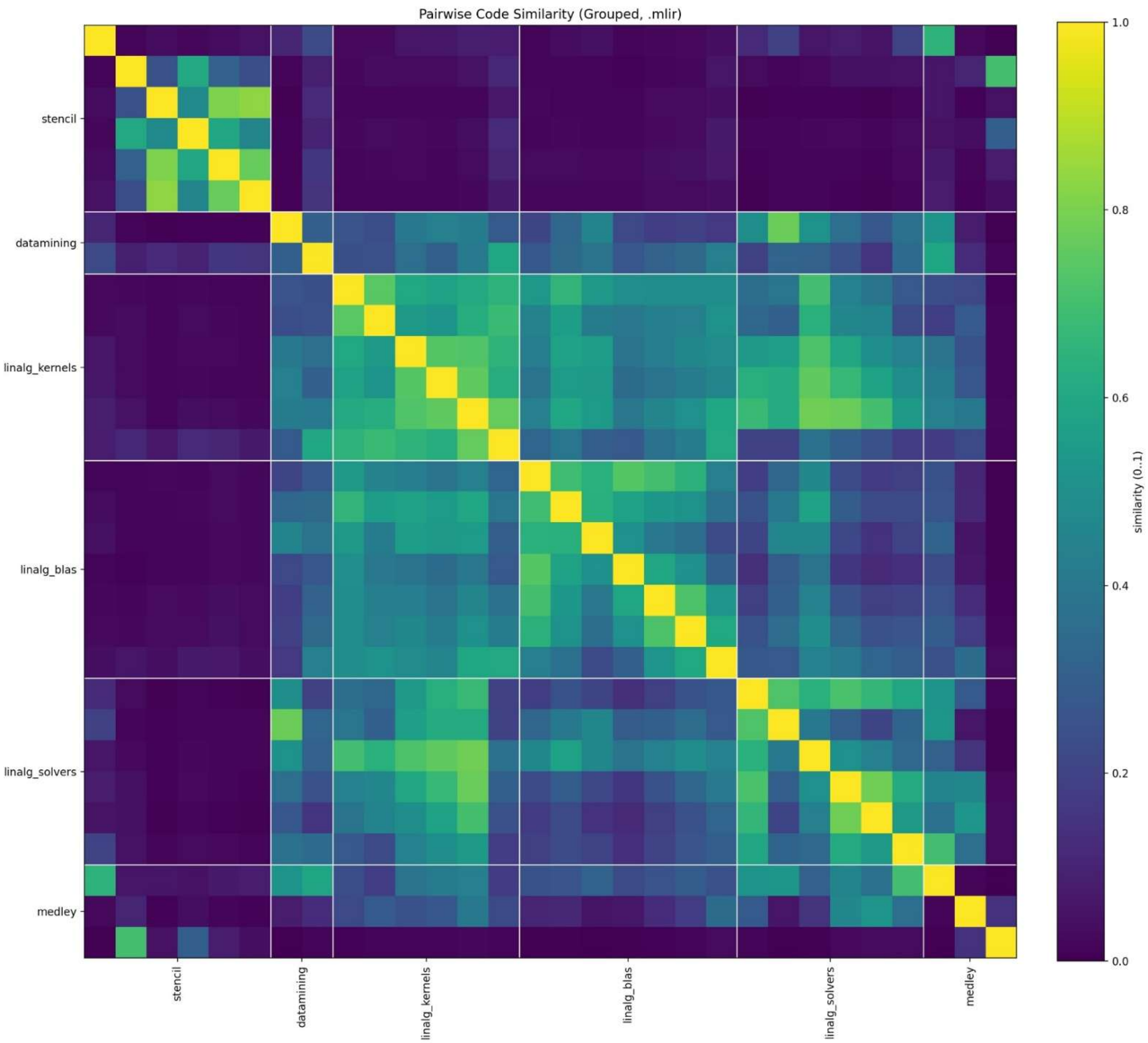


Router Output

Router produces a human understandable strategy for how to optimize an input kernel

```
D1-2-2, D2-1-3 = Distribute L0-1-3 (S0 / S1)
T1-1-6, T2-2-5, T3-3-4, T4-4-3, T5-5-2, T6-6-1 = Tile D2-1-3
Reorder T2-2-5, T3-3-4          // (i_t, j_t, k_t, ...) → (i_t,
k_t, j_t, ...)
Reorder T4-4-3, T5-5-2          // (…, ii, jj, kk) → (…, ii, kk,
jj)
Parallelize T1-1-6              // i_t parallel

D3-2-2, D4-1-3 = Distribute L3-1-3 (S2 / S3)
T7-1-6, T8-2-5, T9-3-4, T10-4-3, T11-5-2, T12-6-1 = Tile D4-1-3
Reorder T8-2-5, T9-3-4
Reorder T10-4-3, T11-5-2
Parallelize T7-1-6
```



Code Generation Agents Output

```
transform.sequence failures(propagate) {
  ^bb1(%arg1: !transform.any_op):
    // ========================
    // Kernel 1: root = loop_2 (deepest 3-deep nest)
    // Sets: 0 = init (statement_0), 1 = compute (statements 1..6)
    // ========================
    %k1_root = transform.structured.match attributes {loop_2} in %arg1 : (!transform.any_op) -> !transform.any_op
    // Distribute into (init, compute) nests under the same outer scope.
    %k1_init, %k1_comp = transform.validator.distribute %k1_root : (!transform.any_op) -> (!transform.any_op, !transform.any_op)

    // Tile only the compute nest (3-D tile: [32,32,32])
    %k1_bi, %k1_bj, %k1_bk, %k1_ei, %k1_ej, %k1_ek = transform.validator.tile %k1_comp { tile_sizes = [32, 32, 32] } : (!transform.any_op) -> (!transform.any_op,
!transform.any_op, !transform.any_op, !transform.any_op, !transform.any_op, !transform.any_op)

    // Reorder blocks to (i_t, k_t, j_t) and elements to (ii, kk, jj)
    %k1_bk_after, %k1_bj_after = transform.validator.reorder %k1_bj and %k1_bk : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
    %k1_ek_after, %k1_ej_after = transform.validator.reorder %k1_ei and %k1_ej : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)

    // Parallelize tile loops that are provably independent (i_t and j_t)
    %k1_par_i = transform.validator.parallel %k1_bi : (!transform.any_op) -> !transform.any_op


    // ========================
    // Kernel 2: root = loop_5
    // Sets: 0 = init (statement_7), 1 = compute (statements 8..13)
    // ========================
    %k2_root = transform.structured.match attributes {loop_5} in %arg1 : (!transform.any_op) -> !transform.any_op
    %k2_init, %k2_comp = transform.validator.distribute %k2_root : (!transform.any_op) -> (!transform.any_op, !transform.any_op)

    %k2_bi, %k2_bj, %k2_bk, %k2_ei, %k2_ej, %k2_ek = transform.validator.tile %k2_comp { tile_sizes = [32, 32, 32] } : (!transform.any_op) -> (!transform.any_op,
!transform.any_op, !transform.any_op, !transform.any_op, !transform.any_op, !transform.any_op)

    %k2_bk_after, %k2_bj_after = transform.validator.reorder %k2_bj and %k2_bk : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
    %k2_ek_after, %k2_ej_after = transform.validator.reorder %k2_ei and %k2_ej : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)

    %k2_par_i = transform.validator.parallel %k2_bi : (!transform.any_op) -> !transform.any_op


    // ========================
    // Kernel 3: root = loop_8
    // Sets: 0 = init (statement_14), 1 = compute (statements 15..20)
    // ========================
    %k3_root = transform.structured.match attributes {loop_8} in %arg1 : (!transform.any_op) -> !transform.any_op
    %k3_init, %k3_comp = transform.validator.distribute %k3_root : (!transform.any_op) -> (!transform.any_op, !transform.any_op)

    %k3_bi, %k3_bj, %k3_bk, %k3_ei, %k3_ej, %k3_ek = transform.validator.tile %k3_comp { tile_sizes = [32, 32, 32] } : (!transform.any_op) -> (!transform.any_op,
!transform.any_op, !transform.any_op, !transform.any_op, !transform.any_op, !transform.any_op)

    %k3_bk_after, %k3_bj_after = transform.validator.reorder %k3_bj and %k3_bk : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)
    %k3_ek_after, %k3_ej_after = transform.validator.reorder %k3_ei and %k3_ej : (!transform.any_op, !transform.any_op) -> (!transform.any_op, !transform.any_op)

    %k3_par_i = transform.validator.parallel %k3_bi : (!transform.any_op) -> !transform.any_op
  }
```

Experimental Results

- **Codegen Correctness:**
  - 30/30 produced executable that produced correct outputs.
- **Proposal Quality:**
  - 27/30 had runtime better or equal to baseline (no transformation).
  - 14/30 had runtime better or equal to expert written schedules.
- **Example: symm.mlir x1.8 of human expert in runtime**

Tested on Kunpeng 920-3226 2.6 GHz 32-core CPUs and the C program is compiled with gcc-13 –O3 (13.1.0), and ran with OMP_NUM_THREADS=16.
LLM used is DeepSeek-V3-0324.

Polybench	# Name	Best GCC/Exp	Best GCC/LLM	Expert/LLM
datamining	correlation	349.7	8.2	0.0
datamining	covariance	365.2	214.5	0.6
kernels	2mm	79.2	61.5	0.8
kernels	3mm	149.5	78.3	0.5
kernels	atax	7.8	3.0	0.4
kernels	bicg	18.3	7.6	0.4
kernels	doitgen	1.2	1.2	1.0
kernels	mvt	83.7	86.8	1.0
stencils	adi	15.6	15.6	1.0
stencils	fdtd-2d	12.6	12.6	1.0
stencils	heat-3d	19.7	19.7	1.0
stencils	jacobi-1d	2.7	0.3	0.1
stencils	jacobi-2d	16.1	16.1	1.0
stencils	seidal-2d	15.5	15.5	1.0
blas	gemm	283.4	6.5	0.0
blas	gemver	53.1	46.6	0.9
blas	gesummv	26.6	2.1	0.1
blas	symm	5.8	10.6	1.8
blas	syr2k	87.2	87.2	1.0
blas	syrk	30.2	26.1	0.9
blas	trmm	744.0	15.8	0.0
solver	cholesky	10.6	10.3	1.0
solver	durbin	1.0	1.0	1.0
solver	gramschmidt	104.3	5.5	0.1
solver	lu	107.7	19.2	0.2
solver	ludcmp	1.8	1.9	1.0
solver	trisolv	2.6	0.2	0.1
medley	deriche	1.3	0.3	0.2
medley	floyd-warshall	12.6	12.4	1.0
medley	nussinov	7.7	7.7	1.0
		30/30	27/30	14/30

Thank you for your Attention!