Loop Lifting in MLIR

Download Poster DUMMY

Lorenzo Chelini*, Andi Drebes†, Oleksandr Zinenko‡, Albert Cohen‡, Henk Corporaal*, Tobias Grosser§ and Nicolas Vasilache‡
* TU Eindhoven †Inria and École Normale Supérieure ‡Google §ETH Zürich

1. MLIR

"A collection of modular and reusable software components which enables the progressing lowering of operations [1]".
But what about lifting?

2. Loop Lifting

- **Raise from the Affine dialect to the Linalg dialect by recognizing linear algebra operations in affine loop nests**
- **Leverage optimization strategies from high-level abstraction dialects:**
  - **➡ Calls to optimized libraries**
  - **➡ Instructions for specialized accelerators**

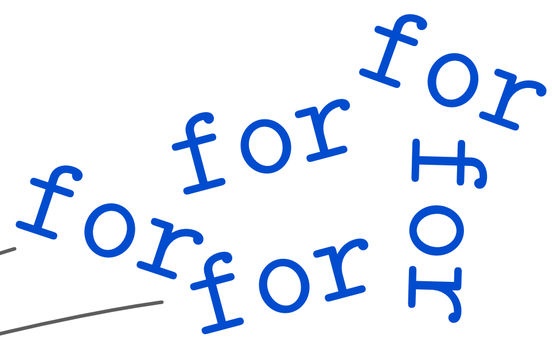
3. Example

```
func @some_function(%A: memref<42x42xf32>,
                    %B: memref<42x42xf32>,
                    %C: memref<42x42xf32>)
{
  affine.for %i = 0 to 42 {
    affine.for %j = 0 to 42 {
      affine.for %k = 0 to 42 {
        %0 = affine.load %A[%i, %k] : memref<42x42xf32>
        %1 = affine.load %B[%k, %j] : memref<42x42xf32>
        %2 = affine.load %C[%i, %j] : memref<42x42xf32>
        %3 = mulf %0, %1 : f32
        %4 = addf %2, %3 : f32
        affine.store %4, %C[%i, %j] : memref<42x42xf32>
      }
    }
  }
  return
}
```

linalg_matmul(%A, %B, %C)

4. Structural Matchers

```
auto body = [&](Operation *op) -> bool {
  auto loop = cast<AffineForOp>(op);
  // get induction variables i, j and k.
  return isGemmLike(
    loop.getLoopBody(),i, j, k).hasValue();
};
// structural matcher.
auto matcher = For(For(For(body)));
```

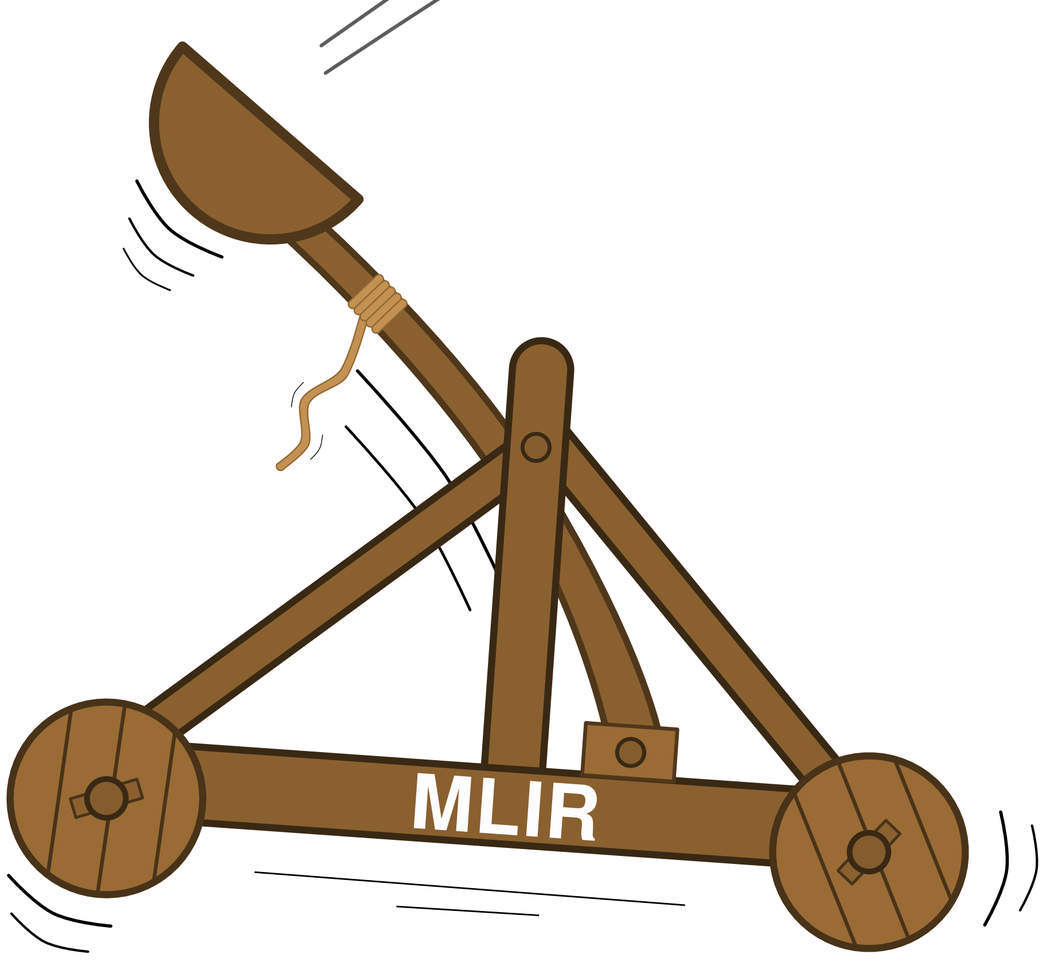


5. Access Matchers

```
auto isGemmLike (Region &body, Value i, Value j,
  Value k) {
  auto ctx = body.getContext();

  auto _i = placeholder(ctx, m_SpecificVal(i));
  auto _j = placeholder(ctx, m_SpecificVal(j));
  auto _k = placeholder(ctx, m_SpecificVal(k));

  // capture ‘A’, ‘B' and ‘C’ necessary for the builder.
  auto _A = arrayPlaceholder(m_Val(A));
  auto _B = arrayPlaceholder(m_Val(B));
  auto _C = arrayPlaceholder(m_Val(C));

  auto a = m_Op<AffineLoadOp>(_A({_i, _k}));
  auto b = m_Op<AffineLoadOp>(_B({_k, _j}));
  auto c = m_Op<AffineLoadOp>(_C({_i, _j}));
  auto gemm = m_Op<AddFOp>(c, m_Op<MulFOp>(a, b));

  auto store = dyn_cast<AffineStoreOp>(
    std::prev(body.front().end(), 2));
  auto add = dyn_cast<AddFOp>(
    store.getValueToStore()->getDefiningOp());
  if (!gemm.match(add))
    return matchFailure();
  return matchSuccess();
};
```

6. References

[1] MLIR website: https://mlir.llvm.org/



TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Inria INVENTEURS DU MONDE NUMÉRIQUE