15745 Project Milestone

Rohan Aggarwal & Amadou Ngom

Major Changes

None.

What Have You Accomplished So Far

We accomplished the following:

- Convert basic imperative code to MLIR.
- Convert simple SQL (without aggregations or joins) statements to MLIR.
- Make the code amenable to classical optimizations:
    - Constant Folding.
    - Common Subexpression Elimination.
    - Constant Propagation.
    - Loop Invariant Code Motion.
    - Function inlining.
- Examples are provided on the next page.

Meeting Your Milestone

We have not implemented relational optimization yet. But thanks to MLIR, we have many more imperative optimizations than we planned.

Surprises

None.

Revised Schedule

We plan to do filter ordering by the end of this week.
As for the steps after that, we want to discuss with Prof. Todd first.

Resources Needed

None.

Example Optimizations

Expression Rewriting: Detective Trivially True Filters.

The following shows that when MLIR detects a trivially true filter, it can skip the WHERE clause and directly perform the projection.

```
module {
  func @Select(%arg0: i64) -> i64 {
    br ^bb1
  ^bb1: // 3 preds: ^bb0, ^bb2, ^bb3
    %c37_i64 = constant 37 : i64
    %0 = sqlir.TableNext %c37_i64 : (i64) -> i1
    cond_br %0, ^bb2, ^bb4
  ^bb2: // pred: ^bb1
    %c37_i64_0 = constant 37 : i64
    %c73_i64 = constant 73 : i64
    %1 = cmpi "slt", %c37_i64_0, %c73_i64 : i64
    cond_br %1, ^bb3, ^bb1
  ^bb3: // pred: ^bb2
    %c37_i64_1 = constant 37 : i64
    %c1_i64 = constant 1 : i64
    %2 = sqlir.GetColumn %c37_i64_1, %c1_i64 : (i64, i64) -> i64
    %c37_i64_2 = constant 37 : i64
    %c2_i64 = constant 2 : i64
    %3 = sqlir.GetColumn %c37_i64_2, %c2_i64 : (i64, i64) -> i64
    %4 = muli %2, %3 : i64
    sqlir.FillResult %4 : i64
    %c37_i64_3 = constant 37 : i64
    %c1_i64_4 = constant 1 : i64
    %5 = sqlir.GetColumn %c37_i64_3, %c1_i64_4 : (i64, i64) -> i64
    %c37_i64_5 = constant 37 : i64
    %c2_i64_6 = constant 2 : i64
    %6 = sqlir.GetColumn %c37_i64_5, %c2_i64_6 : (i64, i64) -> i64
    %7 = addi %5, %6 : i64
    sqlir.FillResult %7 : i64
    br ^bb1
  ^bb4: // pred: ^bb1
    %c10000_i64 = constant 10000 : i64
    return %c10000_i64 : i64
  }
  func @Main(%arg0: i64) -> i64 {
    %cst = constant 3.777000e+01 : f64
    %c37_i64 = constant 37 : i64
    %0 = call @Select(%c37_i64) : (i64) -> i64
    return %0 : i64
  }
}
```

```
module {
  func @Select(%arg0: i64) -> i64 {
    %c1_i64 = constant 1 : i64
    %c37_i64 = constant 37 : i64
    %c2_i64 = constant 2 : i64
    %c10000_i64 = constant 10000 : i64
    br ^bb1
  ^bb1: // 2 preds: ^bb0, ^bb2
    %0 = sqlir.TableNext %c37_i64 : (i64) -> i1
    cond_br %0, ^bb2, ^bb3
  ^bb2: // pred: ^bb1
    %1 = sqlir.GetColumn %c37_i64, %c1_i64 : (i64, i64) -> i64
    %2 = sqlir.GetColumn %c37_i64, %c2_i64 : (i64, i64) -> i64
    %3 = muli %1, %2 : i64
    sqlir.FillResult %3 : i64
    %4 = addi %1, %2 : i64
    sqlir.FillResult %4 : i64
    br ^bb1
  ^bb3: // pred: ^bb1
    return %c10000_i64 : i64
  }
  func @Main(%arg0: i64) -> i64 {
    %c37_i64 = constant 37 : i64
    %0 = call @Select(%c37_i64) : (i64) -> i64
    return %0 : i64
  }
}
```

Expression Rewriting: Detecting Impossible Filters

The following shows what happens when a filter is detected as always false. MLIR is able to remove the projections block since it knows it will never be reached. However, there is still an empty loop over the table. We think it can be done by doing a landing pad transformation.

```
module {
  func @Select(%arg0: i64) -> i64 {
    br ^bb1
  ^bb1: // 3 preds: ^bb0, ^bb2, ^bb3
    %c37_i64 = constant 37 : i64
    %0 = sqlir.TableNext %c37_i64 : (i64) -> i1
    cond_br %0, ^bb2, ^bb4
  ^bb2: // pred: ^bb1
    %c37_i64_0 = constant 37 : i64
    %c73_i64 = constant 73 : i64
    %1 = cmpi "sgt", %c37_i64_0, %c73_i64 : i64
    cond_br %1, ^bb3, ^bb1
  ^bb3: // pred: ^bb2
    %c37_i64_1 = constant 37 : i64
    %c1_i64 = constant 1 : i64
    %2 = sqlir.GetColumn %c37_i64_1, %c1_i64 : (i64, i64) -> i64
    %c37_i64_2 = constant 37 : i64
    %c2_i64 = constant 2 : i64
    %3 = sqlir.GetColumn %c37_i64_2, %c2_i64 : (i64, i64) -> i64
    %4 = muli %2, %3 : i64
    sqlir.FillResult %4 : i64
    %c37_i64_3 = constant 37 : i64
    %c1_i64_4 = constant 1 : i64
    %5 = sqlir.GetColumn %c37_i64_3, %c1_i64_4 : (i64, i64) -> i64
    %c37_i64_5 = constant 37 : i64
    %c2_i64_6 = constant 2 : i64
    %6 = sqlir.GetColumn %c37_i64_5, %c2_i64_6 : (i64, i64) -> i64
    %7 = addi %5, %6 : i64
    sqlir.FillResult %7 : i64
    br ^bb1
  ^bb4: // pred: ^bb1
    %c10000_i64 = constant 10000 : i64
    return %c10000_i64 : i64
  }
  func @Main(%arg0: i64) -> i64 {
    %cst = constant 3.777000e+01 : f64
    %c37_i64 = constant 37 : i64
    %0 = call @Select(%c37_i64) : (i64) -> i64
    return %0 : i64
  }
}
```

```
module {
  func @Select(%arg0: i64) -> i64 {
    %c37_i64 = constant 37 : i64
    %c10000_i64 = constant 10000 : i64
    br ^bb1
  ^bb1: // 2 preds: ^bb0, ^bb2
    %0 = sqlir.TableNext %c37_i64 : (i64) -> i1
    cond_br %0, ^bb2, ^bb3
  ^bb2: // pred: ^bb1
    br ^bb1
  ^bb3: // pred: ^bb1
    return %c10000_i64 : i64
  }
  func @Main(%arg0: i64) -> i64 {
    %c37_i64 = constant 37 : i64
    %0 = call @Select(%c37_i64) : (i64) -> i64
    return %0 : i64
  }
}
```