Efficient Data-Flow Analysis on Region-Based Control Flow in MLIR

Weiwei Chen
weiwei.chen@modular.com
EuroLLVM 2024
Agenda

01 Data-flow Analysis

02 Region-based Control Flow Representation in MLIR

03 An Efficient SCCP

04 Conclusions

05 Questions?
Data-flow Analysis

- Gathers information that is propagated along the control-flow graph (CFG) of a program.
- Static analysis that covers all the edges of how data is flowed in the program.
- Analysis states can be use for optimizations like Sparse Conditional Constant Propagation (SCCP), Value Range Analysis, Bit-Vector Analysis, etc.
Classic Data-flow Analysis

Sparse

\[ S(v_{i+2}) = f_{op}(S(v_i), S(v_{i+1})) \]

Dense

\[ S_{i+1} = f_{op}(S_i) \]

MLIR Dataflow Analysis by Jeff Niu, Tom Eccles, 2023 EuroLLVM
Classic Data-flow Analysis States

Sparse

\[ S(\text{arg}(\text{BB}_i, n)) = S(\text{out}(\text{BB}_i, n)) \lor S(\text{out}(\text{BB}_{i+1}, n)) \]

Dense

\[ S(\text{begin}(\text{BB}_i)) = S(\text{end}(\text{BB}_i)) \lor S(\text{end}(\text{BB}_{i+1})) \]

Analysis State Lattice

\begin{align*}
\top \lor X &= \top \\
\bot \lor X &= X \\
\top \land X &= X \\
\bot \land X &= \bot \\
\end{align*}

\( X_i \lor X_j = \text{unique UB}(X_i, X_j) \)

\( X_i \land X_j = \text{unique LB}(X_i, X_j) \)

Boolean Constraints

Data-flow Analysis in LLVM and MLIR

- **LLVM:** SCCP, IPSCCP, etc.
  - SCCPSolver, Clang Dataflow framework
- **MLIR:**
  - Dead Code Analysis, IntegerRangeAnalysis, LivenessAnalysis, etc.
  - Extensible and composable DataFlowFrameWork

Analysis follows the \textbf{general} control flow graph (CFG):

- $\phi$ nodes add complexity 😞
- CFG can be irreducible 😱
- Logically difficult to debug 😞

Iterates an analysis state solver to fix point:

\[
S_{i+1}(P_{n+1}) = S_i(P_{n+1}) \lor f_{op}(S_i(P_n))
\]

\[
S_{i+1}(P_{n+1}) = S_i(P_{n+1})
\]

\[1\] Data flow analysis: an informal introduction Clang Documentation.
\[2\] MLIR Dataflow Analysis by Jeff Niu, Tom Eccles, 2023 EuroLLVM.
Region-based Control Flow Representation in MLIR

- Structured Control Flow Representation (like mlir.scf)
- Support early exits:
  - `break`, `continue`.
  - exits in the middle of basic blocks.
  - pure region-based representation.
- No arbitrary control flow, only branch back to parent region(s).
- High-level control flow representation matches well with program logic.
- Easy for frontends to emit directly, i.e. Mojo

```mlir
func.func @foobar() {
  rcf.loop {
    %0 = call @rand_bool() : () → i1
    rcf.if %0 {
      rcf.break
    }
    call @do_something() : () → ()
    rcf.continue
  }
  return
}
```
Region-based Control Flow Representation in MLIR

- Region operations:
  - rcf.loop, rcf.if, rcf.for, ...
- Region terminators:
  - rcf.yield, rcf.break, rcf.continue
- Control flow interfaces for passes use abstraction.
- Co-exist with CFG and mlir.scf.

```cpp
class RCFNode : public mlir::OpInterface<ControlFlowNode, _> {
public:
  // Given potential constant values of the operands of this operation, return
  // the indices of the entry region of the operation, which is the region to
  // the beginning of which control-flow branches upon visiting the start of
  // this operation, and the operands with which to branch to that region.
  // Return 'None' to indicate that control-flow branches directly to after the
  // operation.
  void GetEntryTargets(ArrayRef<Attribute> operands,
                        SmallVectorImpl<RCFTarget> & targets);
  
  // Verifier.
  static mlir::LogicalResult verify(mlir::Operation *op);  
};

class RCF Terminator : public mlir::OpInterface<ControlFlow Terminator, _> {
public:
  // This method is invoked on the proper ancestors of a control-flow
  // terminator to determine the nearest valid parent operation. The method
  // should return true if the provided operation is a valid parent operation
  // to the terminator, and false to keep searching.
  bool isParentNode(Operation *op);
  
  // Return the branch target of the terminator relative to its control-flow
  // parent and the operands with which to branch to that region. For instance,
  // to branch to the beginning of the first region, the method should return
  // "0". To branch to the after the parent operation, the method should return
  // "None".
  void GetBranchTargets(ArrayRef<Attribute> operands,
                        SmallVectorImpl<RCFTarget> & targets);
  
  // Verifier.
  static mlir::LogicalResult verify(mlir::Operation *op);  
};
```
Sparse Conditional Constant Propagation

\[ T \lor \text{any} = T \]
\[ \perp \lor \text{any} = \text{any} \]
\[ C_i \lor C_k = C_i \iff C_i = C_k \]
\[ C_i \lor C_k = T \iff C_i \neq C_k \]

L(x, y, a, b)

\[ x = 1 \]
\[ y = 2 \]
\[ \text{if } x > \text{foo}(x) \]
\[ \text{while } a > 4 \]
\[ a = \text{foo}(a) \]
\[ b = a \cdot a \]
Sparse Conditional Constant Propagation

```
\begin{align*}
\text{a = 1} \\
\text{while a < 3} \\
\text{if foo(a)} \\
\text{a = 3} \\
\text{break} \\
\text{++ a} \\
\text{continue} \\
\text{if a > 1} \\
\text{c = 2} \\
\text{c = 4}
\end{align*}
```
Sparse Conditional Constant Propagation

```
a = 1
while a < 3
  if foo(a)
    a = 3
    break
  ++ a
  continue
if a > 1
  c = 2
  c = 4
```

`c` is over-defined
Sparse Conditional Constant Propagation

```
(Sparse Conditional Constant Propagation)

while a < 3
  if foo(a)
    if a > 1
      c = 2
    else
      c = 4
  else
    ++ a
    break
    continue

a = 1

(Sparse Conditional Constant Propagation)
```

The diagram shows a flowchart for a program with conditions and assignments. The flowchart includes decisions based on boolean expressions and operations on variables. The final output shows that `c` is over-defined.
**Sparse Conditional Constant Propagation**

1. \( a = 1 \)
2. \( \text{while } a < 3 \)
3. \( \text{if } \text{foo}(a) \)
   - \( T \)
     - \( a = 3 \)
     - \( \text{break} \)
   - \( F \)
     - \( ++ a \)
     - \( \text{continue} \)
4. \( \text{if } a > 1 \)
   - \( T \)
     - \( c = 2 \)
   - \( F \)
     - \( c = 4 \)

*Note: c must be 2?*
Sparse Conditional Constant Propagation

Interpret Loop based on control flow

iter = 1
Sparse Conditional Constant Propagation

Interpret Loop based on control flow

iter = 2
Sparse Conditional Constant Propagation

Interpret Loop based on control flow

iter = 3

```
while a < 3
  if foo(a)
    a = 3
    break
  ++ a
  continue
```

```
if a > 1
  c = 2
  F
  c = 4
  T
```
Interpret Loop based on control flow

iter = 3

Sparse Conditional Constant Propagation

```plaintext
while a < 3
    if foo(a)
        a = 3
        ++ a
        continue
    else
        break
    if a > 1
        c = 2
        if a > 1
            c = 4
        else
            c = 2

(c must be 2!)
```
Sparse Conditional Constant Propagation

Interpret Loop based on control flow
- More accurate result. ☑️
- Can explode compilation time ⚠️
  - Large loop iterations.
  - Nested loops.
  - Use heuristics.

```
(a = 1)
while a < 3
    if foo(a)
        a = 3
        break
    ++ a
    continue
if a > 1
    c = 2
    T
    F
    c = 4
```

c must be 2!
SCCP on Region-based CF

Fix-point solver:
- Arbitrary update order
- O(M*N)

- Converge SCC first
- Then update tail
- O(M + N)

- Loops are SCCs
- Forward linear analysis outside of SCCs
- For SCCs:
  - Localize analysis within SCC
  - Join SCC output and input states
  - Up to 2x linear analysis within SCC
- Complexity: $O(2x \text{ #operations})$.
- Theoretical SCCP complexity: \[ O(\# \text{ SSA edges}) + O(\# \text{ control flow edges}) \]
- Heuristics based loop interpretation for better analysis.

---

[1] Constant propagation with conditional branches by M. Wegman, F. K. Zadeck
## Experiments

<table>
<thead>
<tr>
<th>Model name</th>
<th>QPS wo/sccp</th>
<th>QPS w/sccp</th>
<th>Compilation Time (s) wo/sccp</th>
<th>Compilation Time (s) w/sccp</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlrm-rm2-multihot</td>
<td>39.66</td>
<td>39.87</td>
<td>201.163 s ± 0.475 s</td>
<td>200.471 s ± 0.481 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1.006x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>resnet50-v1.5</td>
<td>110.29</td>
<td>111.59</td>
<td>38.092 s ± 11.945 s</td>
<td>38.316 s ± 12.509 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1.012x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gpt2</td>
<td>124.71</td>
<td>125.34</td>
<td>30.873 s ± 0.432 s</td>
<td>29.189 s ± 0.139 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1.005x)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Benchmark environment:**

- c5n.metal
- Disable hyper-threading and turbo-boost.
- CPU freq: 2.9G Hz.

**Benchmark Methodology:**

- Run each model multiple time for a set period of time.
- Statistical results.
Conclusions

- Structured region-based control flow representation:
  - Allows early exits.
  - Can co-exist with mlir.scf and CFG.
  - Reducible control flow that guarantees best case complexity for data-flow analysis.
  - Logically easy to debug due to close match to the high-level programming language.
  - Applicable to other efficient analyses: range value, bit-vector, memory scoping, ...

- We are planning to upstream:
  - Region-based control flow representation — RFC.
  - First-class support for successors and predecessors.
  - Data-flow analyses based on the control flow representation.

[RFC] Region-based control-flow with early exits in MLIR

Mogball

Region-based control-flow with early exits

This RFC proposes the additional of a new region-based control-flow paradigm to MLIR, but one that enables early exits via operations like break or continue in contrast with SCF.
Questions?