USER MANUAL

COpt : A High Level Domain Specific Language to Perform Compiler Optimizations

Compilation : compile < path > / < coptinputfile.txt >

Execution : run < path > / <llvmirinputfile.ll > -< params >

<params> :

licm	- to perform loop invariant code motion
gvn	- to perform global value numbering
gcse	- to perform global common subexpression elimination
lcse	- to perform local common subexpression elimination
loopunroll	- to perform loop unrolling
constprop	- to perform constant folding and propagation
tailcallelim	- to perform tail recursive call elimination
indvarsim	- to perform induction variable simplification
fninline	- to perform function inlining

1.Basic Keywords

1. opt

   Every optimization begins with this keyword followed by the name of the optimization and enclosing braces { }.

2. iterate

   This keyword allows the programmer to iterate through functions, basic blocks, instructions and loops.

3. repeat

   This keyword should be followed by an integer and braces. The code within the braces { } is generated the specified number of times.

4. precondition

   Certain functions may need preconditions before they are performed. This keyword is passed as a parameter to these functions.

5. dag

   This keyword represents/holds/stores a directed acyclic graph of expressions that is used for local optimizations.

6. availableexprs

   This keyword represents/holds/stores the results of performing the available expressions dataflow analysis that is performed across the control flow graph.

7. reachingdefs

   This keyword represents/holds/stores the results of performing the reaching definitions dataflow analysis that is performed across the control flow graph.

8. reversepostorder

   This keyword indicates that the traversal needs to be done in reverse post order.

2.Common Functions

1. runOn(iterable)

   LLVM provides a method that walks through LLVM IR and runs on the specified iterable. Every optimizations needs to begin with this function which will specify what the code is for.

2. makeDAG( )

   This function creates a DAG to hold binary expressions.

3. findAvailableExpressionsInfo(function)

   This function performs available expressions analysis on the funcion passed to it.

4. findReachingDefinitionsInfo(function)

   This function performs reaching definitions analysis on the funcion passed to it.

5. eraseUnusedInstructions( )

   This function removes the instructions that were made dead by an optimization.

3.Iterables

1. module
2. cfg
3. function
4. basicblock
5. loop
6. instruction
7. expression

4.Optimization Names

1. FunctionInlining
2. ConstantFolding
3. DeadCodeElimination
4. LocalCSE
5. IndVarSimplify
6. GlobalVN
7. LoopInvariantCodeMotion
8. TailCallElim
9. Loop Unrolling
10. GlobalCSE

5.Function Inlining Functions and Thresholds

1. makeInline(function, thresholds)

   This function takes different thresholds as parameters and inlines functions that satisfy them. If no thresholds are mentioned, all functions are inlined.

2. removeInlinedFunctions( )

   The presence of this function indicates that the programmer wishes to remove the function definitions of inlined functions.

3. noLines

   This is a threshold that checks the instruction count.

4. noUses

   This is a threshold that checks the number of times the function is called.

6.Constant Folding and Propagation Functions

1. foldpropagate(instruction, precondition)

   This function performs constant folding and propagation on an instruction provided that the precondition is satisfied.

2. foldable(instruction)

   This function checks if an instruction is foldable.

7.Dead Code Elimination Functions

1. deadcodeelim( )

   This function eliminates dead code.

8.Local CSE Functions and Thresholds

1. lcse(basicblock, dag, thresholds)

   This function performs local common subexpression elimination on basic blocks using a previously created DAG.

2. noExprs

   This is a threshold that checks the number of binary expressions in a basic block.

9.Induction Variable Simplification Functions

1. indvarsimplify(loop)

   This function rewrites the loop in a manner that allows induction variable elimination to be easily performed.

10.Global Value Numbering Functions

1. valuenumber(instruction)

   This function performs value numbering for the instructions in the program.

2. eliminateInstructions(function)

   This keyword eliminates instructions based on value numbering and their congruence classes.

11.Loop Invariant Code Motion Keywords and Functions

1. loopinvstmts

   This is a keyword which contains all the loop invariant statements in the loop.

2. motionstmts

   This keyword is a set of all statements that can be hoisted above the loop.

3. findLoopInvariantStatements(loop, reachingdefs)

   This function takes loop, reaching definitions to the function as parameters, and it returns loopinvstmts.

4. findMotionCandidates(loop, loopinvstmts)

   This function takes loop, loop invariant statements to the function as parameters, and it returns motionstmts.

5. applyCodeMotion(loop, motionstms)

This function moves the motion candidates that has been obtained from the previous functions outside the loop.

12.Tail Recursion Elimination Functions

1. tailcallelim(function, precondition)

   This function performs tail recursion elimination on the function passed provided that the precondition is satisfied.

2. hasRecursiveTailCall(function)

   This function checks if the passed function has a tail recursive call.

13.Loop Unrolling Functions

1. unrollLoop(loop, noLines)

   This function takes a loop and the threshold, which is the number of lines with its value, and it unrolls the loop based on this threshold.

14.Global Common Subexpression Elimination Functions

1. gcse(function, dag, thresholds)

   This function performs global common subexpression elimination on functions using the previously computed available expressions information.

2. noExprs

   This is a threshold that checks the number of binary expressions in a basic block.