Loop normalizations

Simone Campanoni
simone.campanoni@northwestern.edu
Outline

• Canonical form

• Loop-closed SSA form

• Other forms
Let’s look at a problem that loop normalizations will solve
Let’s say we want to add some code to be executed just before jumping into a loop:
- (Incorrect) Add code to a predecessor of the header outside the loop
- (incorrect) Add code to all predecessors of the header
包括<stdio.h>

int main (){ 
  for (int i=0; i < 10; i++) { 
    printf("Hello world\n"); 
  } 
  return 0; 
}

Let’s say we want to add some code to be executed just before every iteration.
- (Incorrect) Add code to the successor of the header that is within the loop.
We need to normalize loops so CATs can expect a single pre-defined shape!
First normalization: adding a pre-header

- Optimizations often require code to be executed once before the loop
- Create a pre-header basic block for every loop
Common loop normalization
Common loop normalization
Loop normalization in LLVM

• The loop-simplify pass normalize natural loops
• Output of loop-simplify:
  • **Pre-header**: the only predecessor of the header
Loop normalization in LLVM

• The loop-simplify pass normalize natural loops
• Output of loop-simplify:
  • **Pre-header**: the only predecessor of the header
  • **Latch**: node executed just before starting a new loop iteration
Loop normalization in LLVM

• The loop-simplify pass normalize natural loops
• Output of loop-simplify:
  • **Pre-header**: the only predecessor of the header
  • **Latch**: single node executed just before starting a new loop iteration
  • **Exit node**: ensures it is dominated by the header

![Diagram of loop normalization with nodes n1, n2, n3, nX, Pre-header, Header, Body, Latch, exit]
Loop normalization in LLVM

• The loop-simplify pass normalize natural loops
• Output of loop-simplify:
  • Pre-header: the only predecessor of the header
  • Latch: single node executed just before starting a new loop iteration
  • Exit node: ensures it is dominated by the header
Loop normalization in LLVM

- **Pre-header** \( \text{llvm::Loop::getLoopPreheader()} \)
- **Header** \( \text{llvm::Loop::getHeader()} \)
- **Latch** \( \text{llvm::Loop::getLoopLatch()} \)
- **Exit** \( \text{llvm::Loop::getExitBlocks()} \)

```
opt -loop-simplify bitcode.bc -o normalized.bc
```
Outline

• Canonical form

• Loop-closed SSA form

• Other forms
Further normalizations in LLVM

• Loop representation can be further normalized:
  • \textit{loop-simplify} normalize the shape of the loop
  • What about definitions in a loop?

• Problem: updating code in loop might require to update code outside loops for keeping SSA
Loop pass example

A pass needs to add a conditional definition of d
Loop pass example

This is not in SSA anymore: we must fix it
Further normalizations in LLVM

• Loop representation can be further normalized:
  • `loop-simplify` normalize the shape of the loop
  • What about definitions in a loop?

• Problem: updating code in loop might require to update code outside loops for keeping SSA
  • Keeping SSA form is expensive with loops
  • Loop-closed SSA form: no var is used outside of the loop in that it is defined
  • `lcssa` insert phi instruction at loop boundaries for variables defined in a loop body and used outside
    • Outside code only refers to these PHIs
  • Isolation between optimization performed in and out the loop
  • Faster keeping the SSA form
    • Propagation of code changes outside the loop blocked by phi instructions
Loop pass example

```
while (){
    d = ...
}
... = d op ...
... = d op ...
call f(d)
```

Lcssa normalization

```
while (){
    d = ...
}
... = d1 = phi(d...)
... = d1 op ...
... = d1 op ...
call f(d1)
```

```
while (){
    d = ...
    ...
    if (...){
        d2 = ...
    }
    d3=phi(d,d2)
}
d1 = phi(d...)
... = d1 op ...
... = d1 op ...
call f(d1)
```

```
while (){
    d = ...
    ...
    if (...){
        d2 = ...
    }
    d3=phi(d,d2)
}
d1 = phi(d3...)
... = d1 op ...
... = d1 op ...
call f(d1)
```
Loop-closed SSA form in LLVM

```
opt -lcssa bitcode.bc -o transformed.bc

llvm::Loop::isLCSSAForm(DT)

formLCSSA(...)```
Outline

• Canonical form

• Loop-closed SSA form

• Other forms
Further normalizations in LLVM

• Scalar evolution normalization

\{0, +, 1\} \rightarrow \frac{1}{\frac{1}{\frac{1}{%1 = \text{phi} [\%2, \%\text{Latch}], [0, \%\text{PreHeader}]}}{\%2 = \text{add} \%1, 1}}

\begin{align*}
... &= \%2 \\
... &= \%1 \\
\end{align*}

• Whilifier
Always have faith in your ability

Success will come your way eventually

Best of luck!