先进编译实验室
Advanced Compiler

循环优化系列第五讲

循环不变量外提

嘉宾：柴赟达

先进编译实验室
Advanced Compiler

循环不变量外提

基础概念

循环不变量是指在循环迭代空间内值不发生变化的变量。由于循环不变量的值在循环的迭代空间内不发生变化，因此可将其外提到循环外仅计算一次，避免其在循环体内重复计算。

```
for (int i = 1; i < N; i++)
    for (int j = 1; j < M; j++)
        U[i] = U[i] + W[i]*W[i]* D[j] / (dt * dt);
```

```
T1 = 1 / (dt * dt);
  for (int i = 1; i < N; i++) {
      T2 = W[i]*W[i];
      for (int j = 1; j < M; j++)
          U[i] = U[i] + T2 * D[j] * T1;
  }
```


合法性：

① 变换不能影响源程序的语义

优点：

① 削弱计算

先进编译实验室
Advanced Compiler

循环不变量外提

编译器中的循环不变量外提

```
#include <stdio.h>
int main() {
    const int M = 256;
    const int N = 256;
    float U[M], W[M], D[M];
    float dt = 5.0;
    for (int i = 1; i < N; i++) {
        U[i] = i;
        W[i] = i + 1;
        D[i] = i + 2;
    }
    for (int i = 1; i < N; i++)
        for (int j = 1; j < M; j++)
            U[i] = U[i] + W[i]*W[i]* D[j] / (dt * dt);
    printf("%f", U[1]);
}
```

```
[llvm@2022] clang 1.c -O1 -Rpass=licm
1.c:14:20: remark: hoisting getelementptr [-Rpass=licm]
1.c:14:27: remark: hoisting getelementptr [-Rpass=licm]
            U[i] = U[i] + W[i]*W[i]* D[j] / (dt * dt);
                          ^
1.c:14:27: remark: hoisting load [-Rpass=licm]
1.c:14:31: remark: hoisting fmul [-Rpass=licm]
            U[i] = U[i] + W[i]*W[i]* D[j] / (dt * dt);
                                ^
1.c:14:18: remark: Moving accesses to memory location out of the loop [-Rpass=licm]
            U[i] = U[i] + W[i]*W[i]* D[j] / (dt * dt);
                  ^
```

循环不变量外提

编译器中的循环不变量外提

① 确定循环：

- 计算该循环的标头（header）结点和支配（dominate）结点和定义（define）结点。
- 寻找循环不变语句。
- 找到循环的出口（exit）块：在循环外有后继节点（successor）的块。

② 找出符合以下条件的代码并进行外提：

- 是循环不变量。
- 且位于支配出口的块中。
- 且位于支配循环中所有使用其计算值块的基本块中。
- 且这个变量在循环内只有一次赋值。

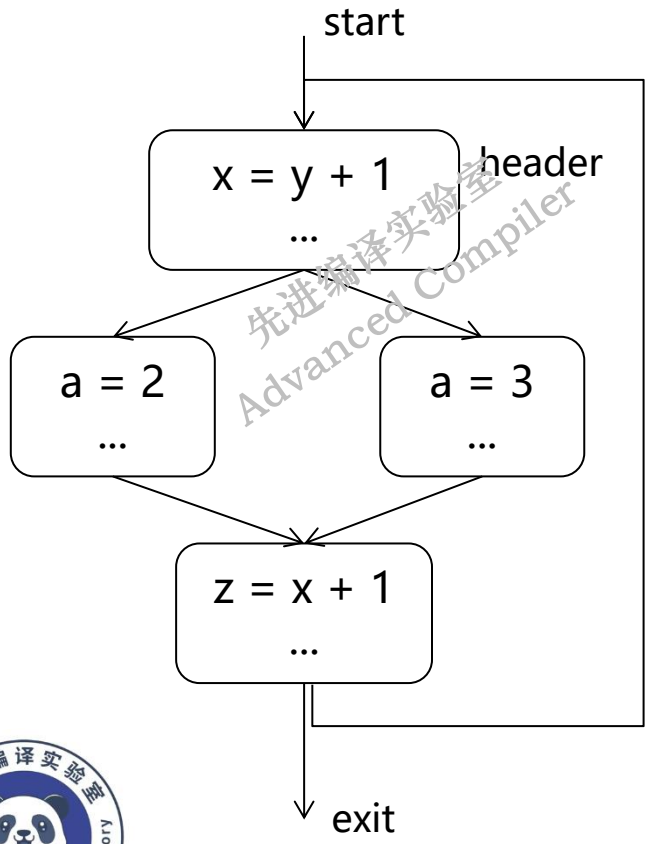
③ 对循环采用深度优先算法，进行搜索，从候选项中选出不变量：

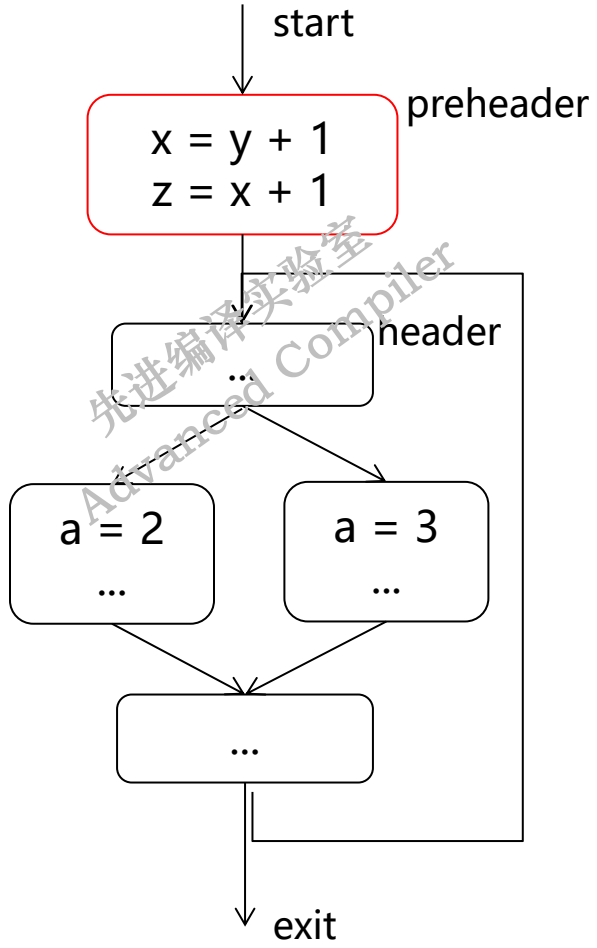
- 如果该不变量所依赖的所有不变操作都已移动，则把它移动到preheader块里面，完成优化算法。

循环不变量外提

循环不变量外提的编译器实现

- 示例





- y+1 中 y 的定义在循环之外)

循环不变量外提

循环不变量外提的编译器实现

- **中间代码示例**

```
define void @func(i32 %i) {
Entry:
        br label %Loop
Loop:
        %j = phi i32 [ 0, %Entry ], [ %Sum, %Loop ]
        %loopinvar = add i32 %i, 12
        %Sum = add i32 %j, %loopinvar
        %cond = icmp eq i32 %Sum, 0
        br i1 %cond, label %Exit, label %Loop
Exit:
        ret void
}
```

```
define void @func(i32 %i) {
Entry:
  %loopinvar = add i32 %i, 12
  br label %Loop

Loop:
; preds = %Loop, %Entry
  %j = phi i32 [ 0, %Entry ], [ %Sum, %Loop ]
  %Sum = add i32 %j, %loopinvar
  %cond = icmp eq i32 %Sum, 0
  br i1 %cond, label %Exit, label %Loop

Exit:
; preds = %Loop
  ret void
}
```

先进编译实验室
Advanced Compiler

分享完毕，感谢聆听！

参考文献：

[1] Optimizing Compilers for Modern Architectures: A Dependence-Based Approach [Book Review][J]. Computer,2002,35(4).

[2] 多伦多大学cscd70编译原理课程课件。

https://www.slidestalk.com/u70/Lecture5LICMandStrengthReduction0208201837108

[3] https://web.cs.wpi.edu/~kal/PLT/PLT10.2.1.html

[4] https://learning.oreilly.com/library/view/llvm-essentials/9781785280801/ch05.html#ch05lvl1sec35

先进编译实验室
Advanced Compiler