

第六章：中间代码优化（2）



公共表达式节省

□ 设有下列语句：

$t := b * c;$

$e := \underline{b * c} + \underline{b * c};$

$c := \underline{b * c} + 10;$

$d := \underline{b * c} + d;$

□ 优化后，可得到下列语句：

$t := b * c; e := t + t;$

$c := t + 10; d := b * c + d;$

公共表达式节省

- 公共表达式：是指两个表达式中的子表达式他们的计算结果相同。
- 公共表达式节省在局部进行的时候都是针对基本块的。
- ❖ 相似性方法*
- ❖ 编码方法

基于相似性的公共表达式节省

- 必经代码：称 d_i 代码为 d_j 代码的必经代码，如果执行 d_j 代码时 d_i 代码一定已经被执行过了，在一个基本块中若 $i < j$ 则 d_i 代码是 d_j 代码的必经代码
- 活跃代码：称 $d_i: (\omega, A, B, t_1)$ 在 d_j 处是活跃的，如果 d_i 是 d_j 的必经代码，且从 d_i 到 d_j 期间均不改变 A 、 B 的值

基于相似性的公共表达式节省

- ECC代码（可节省的公共代码）：称一个运算代码 d_j 是可节省的代码并记为ECC，如果存在其一个必经活跃代码 d_i ，并且计算结果相同。（值得注意的是即使 d_i 和 d_j 代码完全相同， d_j 也不一定是可节省的）
- 相似代码：如果 d_i 和 d_j 运算符、运算分量都相同，则称 d_i 和 d_j 为相似代码

基于相似性的公共表达式节省

□ ECC定理：假设当前代码为运算代码dj：
(ω_1, A_1, B_1, t_1)，如果存在相似的必经活跃代
码，则dj一定是ECC代码，例如基本块：

1:(+,A,B,T1)

优化为：

2:(+,A,B,T2)

1:(+,A,B,T1)

3:(*,T1,T2,T3)

3:(*,T1,T1,T3)

这是判定ECC的充分条件，而不是充要条件

基于相似性的公共表达式节省

- 判断 d_j 为相似性ECC的两个要点:
 - ❖ 判断是否存在其相似的必经活跃代码 d_i , 相似、必经很容易判断, 构建活跃代码表来判定 d_i 在 d_j 处是否活跃;
 - ❖ 等价替换, 当 d_j 代码被节省后, d_j 的结果 t_j 在后续代码中均被替换为 t_i , 因此需要建立等价变量表, 其元素具有形式 (t_i, t_j) ;

基于相似性的公共表达式节省

□ 算法流程

1. 以基本块为单位
2. 创建：活跃运算代码表、等价变量表
3. 每当扫描新代码，首先根据等价变量表进行等价替换
4. 若当前代码是运算代码，则进行判断和优化工作，并更新等价变量表
5. 若当前代码为赋值代码，则进行注销活跃运算代码工作

例子：
 $d = d + c * b$
 $a = d + c * b$
 $c = d + c * b$
 $a = d + c * b$

序号	优化前	优化后	活跃代码表	等价变量表
1	(*,c,b,t1)	(*,c,b,t1)	1	
2	(+,d,t1,t2)	(+,d,t1,t2)	1 2	
3	(=,t2,-,d)	(=,t2,-,d)	1	
4	(*,c,b,t3)	——	1	(t1,t3)
5	(+,d,t3,t4)	(+,d,t1,t4)	1 5	(t1,t3)
6	(=,t4,-,a)	(=,t4,-,a)	1 5	(t1,t3)
7	(*,c,b,t5)	——	1 5	(t1,t3)(t1,t5)
8	(+,d,t5,t6)	——	1 5	(t1,t3)(t1,t5) (t4,t6)
9	(=,t6,-,c)	(=,t4,-,c)	5	"
10	(*,c,b,t7)	(*,c,b,t7)	5 10	"
11	(+,d,t7,t8)	(+,d,t7,t8)	5 10 11	"
12	(=,t8,-,d)	(=,t8,-,d)	5 10 11	"

基于值编码的公共表达式节省

- 扩大ECC：将相似代码的定义扩充为不局限于分量的名字相同。例如：

$a = b * c;$

$d = b;$

$d = d * c$

尽管 $b * c$ 和 $d * c$ 看起来不同，但是实际上具有相同的值，故应处理为ECC

基于值编码的公共表达式节省

- 值编码优化方法的主要思想:
对中间代码中出现的每个分量（常量和变量）确定一个值编码，使得具有相同值的分量编码值相等.

基于值编码的公共表达式节省

- 编码原理:
- 用到一张值编码表, 表示分量当前值编码
- 若当前考察的代码为 $dk: (\omega, u_1, u_2, u_3)$:
 - 若值编码表中已有 (u_1, m_i) 则令 $dk: u_i$ 的值编码为 $m_i, i=1, 2, 3$
 - 否则为 $dk: u_i$ 创建一个新编码 m 填入编码表
- 若考察代码为 $dk: (=, u_1, -, u_2)$:
 - u_1 的处理处理与之前相同
 - 令 $dk: u_2$ 的编码与 $dk: u_1$ 的编码相同

基于值编码的公共表达式节省

□ 值编码性质

1. 不同分量上的相同常量一定具有相同值编码
2. 不同分量上的相同变量未必具有相同编码
3. 不同常量的编码值一定不同，对变量来说并非如此
4. 每当一个变量 x 被赋值， x 将得到一个新的编码，使得后面代码中的 x 分量取编码值 n ，直至 x 再被赋值

基于值编码的公共表达式节省

- 在分析的过程中对应每条四元式还要生成一个编码四元式。
- $\mu(x)$ 表示任意运算分量 x 的值编码;
- 把 $(\omega, \mu(A), \mu(B), \mu(t))$ 叫作 (ω, A, B, t) 的映象码;

基于值编码的公共表达式节省

算法流程：从基本块的第一条四元式开始

1. 等价变量表替换
2. 对四元式中的分量进行编码
3. 值编码表替换，生成编码四元式
4. 遇到运算型四元式，往前查编码四元式中与当前编码四元式运算符、运算分量都相同的，若有则说明确定ECC，删去当前四元式，把等价的名字填入等价表
5. 遇到赋值(=,a,-,b),b的编码赋值为a的编码

例: $a = b * c + b * c$; $d = b$;
 $e = d * c + b * c$

[illegible]

循环不变式外提

- ◆ 循环不变式: 如果一个表达式E在一个循环中不改变其值, 则称它为该循环的不变表达式。
- ◆ 循环不变式外提: 将循环不变式提到循环外面进行。

不变式: 是式值 = 调用函数.
指针

例如，假设有程序段：

```
i:=1;// t=x*y;
```

```
while i<=1000 do
```

```
  begin a[i]:= x*y //t ;i:=i+1 end;
```

则 $x*y$ 是该循环的不变表达式，可以把它提到循环外边

矩阵乘法：有a， b为10*10数组

```
for(i=1~10)
```

```
  for(j=1~10)
```

```
    for(k=1~10)
```

```
      c[i][j]=c[i][j]+a[i][k]*b[k][j];
```

a[i]地址的计算与j、k循环无关，c[i][j]地址的计算与k循环无关

循环不变式外提

- 解决两个问题
 - ❖ 检查循环体中哪些变量的值被改变过
 - ❖ 根据这个结果来看哪些表达式是不变的表达式
- 建立变量定值表，将循环体中值被改变的变量都填到表里，若某运算型四元式中两个运算分量都不出现在这个表里，就说明其值不发生改变，可以进行外提。

循环不变式外提

- ◆ 对循环体四元式进行第一遍扫描，把有定值的变量填到变量定值表中，若它是一个运算型四元式(ω_1, A_1, B_1, t_1)，则把 t_1 填到表中，若为赋值型四元式($=, a, -, b$)则把 b 填入表中
- ◆ 循环不变式外提为第二遍扫描，每遇到一个运算型四元式(ω_1, A_1, B_1, t_1)，若 A_1 、 B_1 都不在变量定值表中，则将其提到循环体外，同时在变量定值表中删去 t_1

```

i:=1
while i<=100 do
  begin
    z:=i*k*5;
    a:=2*k+2*k*2;
    i:=i+1;
  end


```

LoopDef
 ={t1, t2, t3, z,
 t4, t5, t6, t7, a, t8, i}

优化前四元式序列:

1. (ASSIG, 1, —, i)
2. (WHILE, —, —, —)
3. (LE, i, 100, t1)
4. (DO, t1, —, —)
5. (MULTI, i, k, t2)
6. (MULTI, t2, 5, t3)
7. (ASSIG, t3, —, z)
8. (MULTI, 2, k, t4)
9. (MULTI, 2, k, t5)
10. (MULTI, t5, t4, t6)
11. (ADDI, t4, t6, t7)
12. (ASSIG, t7, —, a)
13. (ADDI, i, 1, t8)
14. (ASSIG, t8, —, i)
15. (ENDWHILE, —, —, —)

循环优化前四元式序列:

1. (ASSIG, 1, —, i)
2. (WHILE, —, —, —)
3.  (LE, i, 100, t1)
4. (DO, t1, —, —)
5. (MULTI, i, k, t2)
6. (MULTI, t2, 5, t3)
7. (ASSIG, t3, —, z)
8. (MULTI, 2, k, t4)
9. (MULTI, t4, 2, t6)
10. (ADDI, t4, t6, t7)
11. (ASSIG, t7, —, a)
12. (ADDI, i, 1, t8)
13. (ASSIG, t8, —, i)
14. (ENDWHILE, —, —, —)

LoopDef={t1, t2, t3, z,
t4, t6, t7, a, t8, i}

~~X~~ ~~X~~ ~~X~~

循环优化前四元式序列:

1. (ASSIG, 1, —, i)
2. (WHILE, —, —, —)
3. (LE, i, 100, t1)
4. (DO, t1, —, —)
5. (MULTI, i, k, t2)
6. (MULTI, t2, 5, t3)
7. (ASSIG, t3, —, z)
8. (MULTI, 2, k, t4)
9. (MULTI, t4, 2, t6)
10. (ADDI, t4, t6, t7)
11. (ASSIG, t7, —, a)
12. (ADDI, i, 1, t8)
13. (ASSIG, t8, —, i)
14. (ENDWHILE, —, —, —)

循环优化后四元式序列:

1. (ASSIG, 1, —, i)
2. (MULTI, 2, k, t4)
3. (MULTI, t4, 2, t6)
4. (ADDI, t4, t6, t7)
5. (WHILE, —, —, —)
6. (LE, i, 100, t1)
7. (DO, t1, —, —)
8. (MULTI, i, k, t2)
9. (MULTI, t2, 5, t3)
10. (ASSIG, t3, —, z)
11. (ASSIG, t7, —, a)
12. (ADDI, i, 1, t8)
13. (ASSIG, t8, —, i)
14. (ENDWHILE, —, —, —)

循环不变式外提中注意的问题

1. 多层循环问题中，一个四元式从里层开始可以被外提若干次，里层变量定值表属于外层变量定值表
2. 除法不外提
3. 赋值绝不外提
4. 非良性循环（函数调用和地址应用型参数赋值）不做外提优化

例子

例2:已知:a:array[1..10][1..1000] of integer;
设有如下循环语句:

```
j:=1  
while j<=1000 do  
  begin a[i][j]:=0;  
    j:=j+1;  
  end;
```

优化前的四元式序列:

```
(ASSIG, 1, —, j)
(WHILE, —, —, —)
(LE, j, 1000, t1)
(DO, t1, —, —)
(SUBI, i, 1, t2)
(MULTI, t2, 1000, t3)
(AADD, a, t3, t4)
(SUBI, j, 1, t5)
(MULTI, t5, 1, t6)
(AADD, t4, t6, t7)
(ASSIG, 0, —, t7)
(ADDI, j, 1, t8)
(ASSIG, t8, —, j)
(ENDWHILE, —, —, —)
```

LoopDef={t1, t2, t3, t4, t5,
t6, t7, t8, j}

优化后的四元式序列:

```
(ASSIG, 1, —, j)
(SUBI, i, 1, t2)
(MULTI, t2, 1000, t3)
(AADD, a, t3, t4)
(WHILE, —, —, —)
(LE, j, 1000, t1)
(DO, t1, —, —)
(SUBI, j, 1, t5)
(MULTI, t5, 1, t6)
(AADD, t4, t6, t7)
(ASSIG, 0, —, t7)
(ADDI, j, 1, t8)
(ASSIG, t8, —, j)
(ENDWHILE, —, —, —)
```

LoopDef={t1, t5, t6, t7, t8}

```
j:=1
while j<=1000 do
  begin a[i][j]:=0;
        j:=j+1;
  end;
```