

第10章 代码优化



LI Wensheng, SCS, BUPT

知识点：基本块优化
循环优化

代码优化

10.1 代码优化概述

10.2 基本块优化

10.4 循环优化

10.5 窥孔优化

小结

10.1 代码优化概述

■ 代码优化程序的任务

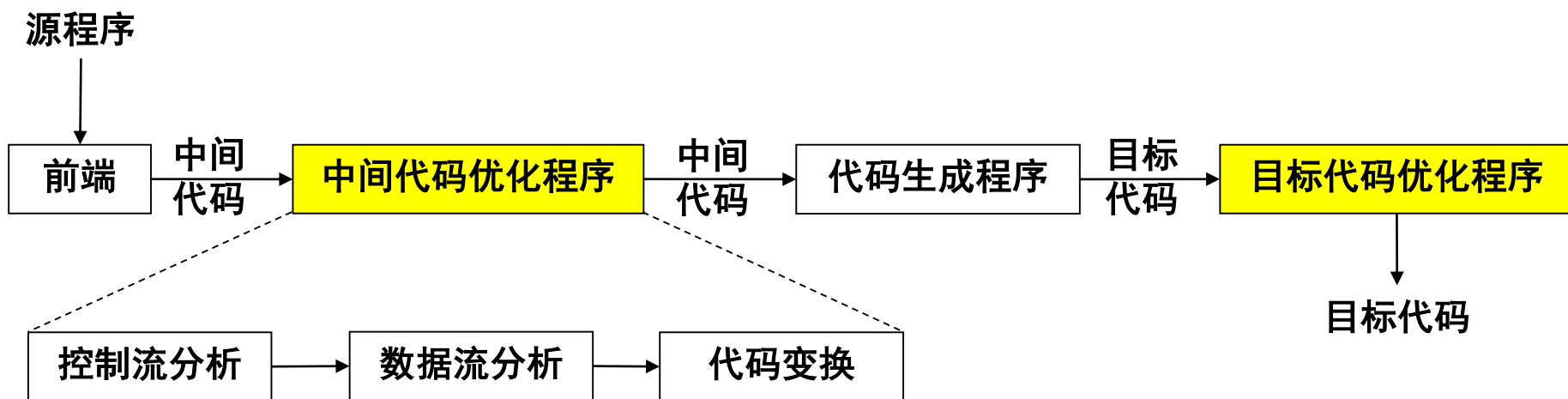
- ◆ 对中间代码或目标代码进行等价变换，使变换后的代码质量更高。

■ 对代码优化程序的要求

- ◆ 等价变换
- ◆ 提高目标代码的执行速度
- ◆ 减少目标代码占用的空间

代码优化程序的位置

- 代码优化可在两个不同阶段进行：
 - ◆ 对中间代码进行，与目标机器无关的优化
 - ◆ 对目标代码进行，与目标机器有关的优化



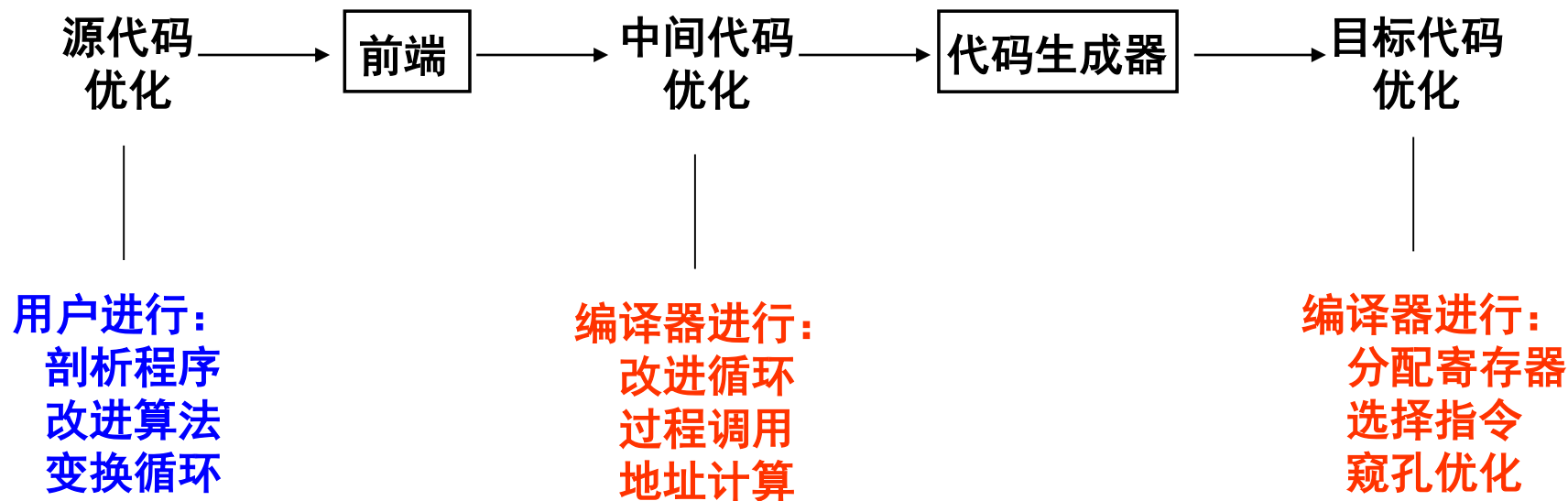
中间代码优化

目标代码优化

优化分类

- **源程序优化：**由用户进行，在编制源程序时设计或选用时间复杂度最佳的算法。
 - 如对N个数据进行排序：
 - “插入排序” 算法需要时间 $2.02N^2\mu s$
 - “快速排序” 算法需要时间 $12N\log_2 N\mu s$
 - $N=100$ 时，快速排序比插入排序快2.5倍
 - $N=100,000$ 时，快速排序比插入排序要快100倍以上。
- **中间代码优化：**由编译器完成，经过编译的各分析阶段之后生成中间代码时，力求生成优化的中间代码。
- **目标代码优化：**由编译器完成，经过编译的各综合阶段之后，生成目标代码时，在目标代码一级应充分利用目标机器的资源

用户和编译器可改进的地方



代码优化的主要种类

■ 中间代码优化

◆ 基本块优化

- 在基本块内进行的优化。
- 常数合并与传播、删除公共子表达式、复制传播、削弱计算强度、改变计算次序等。

◆ 循环优化

- 在循环语句所生成的中间代码序列上进行的优化。
- 循环展开、代码外提、削弱计算强度、删除归纳变量等。

◆ 全局优化

- 在非线性程序段上（含多个基本块）进行的优化。

■ 目标代码优化

◆ 窥孔优化

- 在目标代码上进行局部改进的优化。
- 删除冗余指令、控制流优化、代数化简等。

10.2 基本块优化

对表达式序列求值.

确定该基本块出口处活动名字的值

- 基本块的功能：对表达式序列求值，确定该基本块出口处活动名字的值
- 基本块的值：基本块出口处活动名字的值
- 对基本块进行等价变换要保证基本块的值不变

10.2.1 常数合并及常数传播

10.2.2 删除公共表达式

10.2.3 复制传播

10.2.4 削弱计算强度

10.2.5 改变计算次序

10.2.1 常数合并及常数传播

- 常数合并：将在编译时可计算出值的表达式用其值替代。

$x=2+3+y$ 可代之以： $x=5+y$

- 常数传播：用在编译时已知的变量值代替程序正文中对这些变量的引用。

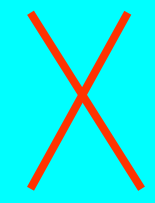
$PI:=3.14;$

$D-to-R:= 0.01744$


- 常数合并与传播主要是一种局部优化技术，通常只用于不带下标的变量

```
i:=0
10: i:=i+1
...
if i<10 goto 10
```

```
i:=0
10: i:=0+1
...
if i<10 goto 10
```



```
...
a[i]:=9.0
...
a[j]:=3.0
b:=a[i]
```



常数合并的实现

- 在符号表中增加两个信息域
 - ◆ 标志域：指示当前该变量的值是否存在。
 - ◆ 常数域：如果变量值存在，则该域存放的即是该变量的当前值。
- 常数合并时，注意事项：
 - ◆ 不能将结合律与交换律用于浮点表达式。
 - 浮点运算的精度有限，这两条定律并非是恒真的。
 - ◆ 不应将任何附加的错误引入。

10.2.2 删除公共表达式

- 在一个基本块中，当第一次对表达式E求值之后，如果E中的运算对象都没有改变，再次对E求值，则除E的第一次出现之外，其余的都是冗余的公共表达式。
- 删除冗余的公共表达式，用第一次出现时的求值结果代替重复求值的结果。

(1) **a:=b+c**

(2) **b:=a-d**

(3) **c:=b+c**

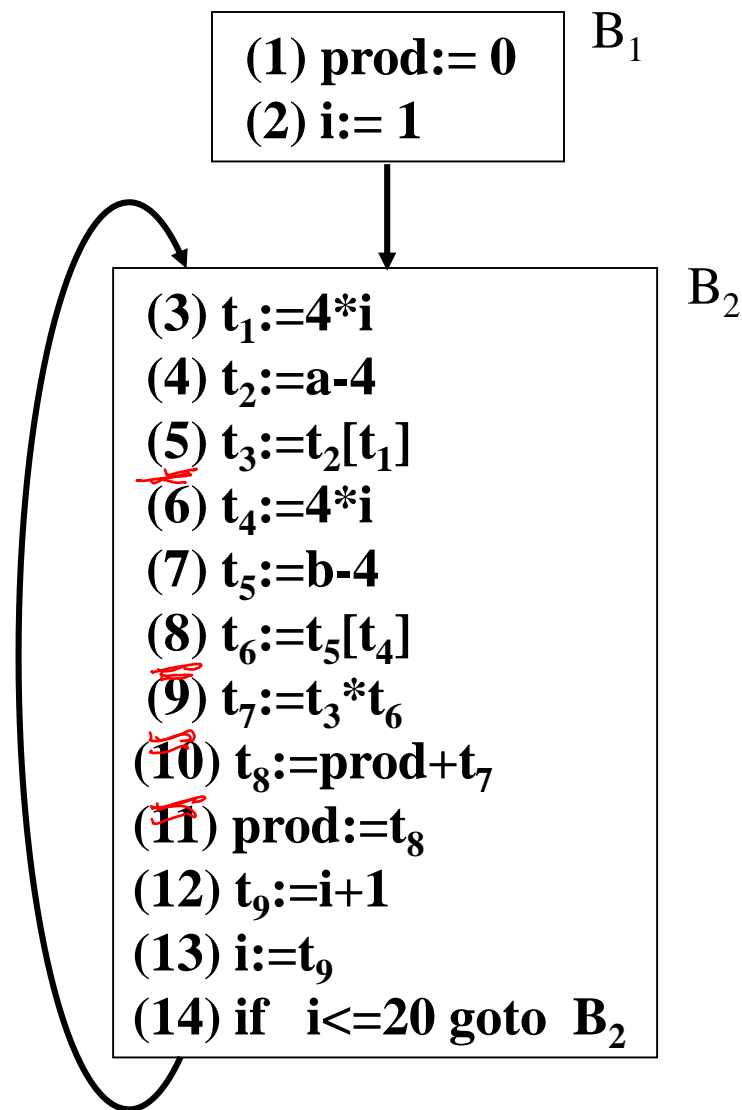
(4) **d:= b**

示例

■ 计算向量点积的程序

```
prod=0;  
i=1;  
do {  
    prod=prod+a[i]*b[i];  
    i++;  
}while (i<=20);
```

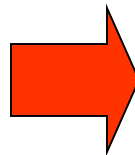
■ 程序的控制流图：



对基本块B₂删除公共表达式

删除公共表达式

(3) $t_1 := 4 * i$
(4) $t_2 := a - 4$
(5) $t_3 := t_2[t_1]$
(6) $t_4 := 4 * i$
(7) $t_5 := b - 4$
(8) $t_6 := t_5[t_4]$
(9) $t_7 := t_3 * t_6$
(10) $t_8 := \text{prod} + t_7$
(11) $\text{prod} := t_8$
(12) $t_9 := i + 1$
(13) $i := t_9$
(14) if $i \leq 20$ goto B₂

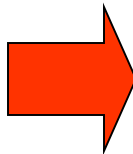


(3) $t_1 := 4 * i$
(4) $t_2 := a - 4$
(5) $t_3 := t_2[t_1]$
(6') $t_4 := t_1$
(7) $t_5 := b - 4$
(8) $t_6 := t_5[t_4]$
(9) $t_7 := t_3 * t_6$
(10) $t_8 := \text{prod} + t_7$
(11) $\text{prod} := t_8$
(12) $t_9 := i + 1$
(13) $i := t_9$
(14) if $i \leq 20$ goto B₂

10.2.3 复制传播

- 为减少重复计算，可以利用复制传播来删除公共表达式
- 思想：在复制语句 $f := g$ 之后，尽可能用 g 代替 f

```
(3)  $t_1 := 4 * i$   
(4)  $t_2 := a - 4$   
(5)  $t_3 := t_2[t_1]$   
(6')  $t_4 := t_1$   
(7)  $t_5 := b - 4$   
(8)  $t_6 := t_5[t_4]$   
(9)  $t_7 := t_3 * t_6$   
(10)  $t_8 := \text{prod} + t_7$   
(11)  $\text{prod} := t_8$   
(12)  $t_9 := i + 1$   
(13)  $i := t_9$   
(14) if  $i \leq 20$  goto  $B_2$ 
```



```
(3)  $t_1 := 4 * i$   
(4)  $t_2 := a - 4$   
(5)  $t_3 := t_2[t_1]$   
(6')  $t_4 := t_1$  这句话没必要存在了  
(7)  $t_5 := b - 4$   
(8')  $t_6 := t_5[t_4]$   
(9)  $t_7 := t_3 * t_6$   
(10)  $t_8 := \text{prod} + t_7$   
(11)  $\text{prod} := t_8$   
(12)  $t_9 := i + 1$   
(13)  $i := t_9$   
(14) if  $i \leq 20$  goto  $B_2$ 
```

删除死代码

- 死代码：如果对一个变量 x 求值之后却不引用它的值，则称对 x 求值的代码为死代码。
- 死块：控制流不可到达的块称为死块。
 - ◆ 如果一个基本块是在某一条件为真时进入执行的，经数据流分析的结果知该条件恒为假，则此块是死块。
 - ◆ 如果一个基本块是在某个条件为假时才进入执行，而该条件却恒为真，则这个块也是死块。
- 在确定一个基本块是死块之前，需要检查转移到该块的所有转移语句的条件。
- 死块的删除，可能使其后继块成为无控制转入的块，这样的块也成为死块，同样应该删除。

死块

后继块也是死块

对基本块 B_2 删除死代码

(3) $t_1 := 4 * i$
(4) $t_2 := a - 4$
(5) $t_3 := t_2[t_1]$
(6') $t_4 := t_1$
(7) $t_5 := b - 4$
(8') $t_6 := t_5[t_1]$
(9) $t_7 := t_3 * t_6$
(10) $t_8 := \text{prod} + t_7$
(11) $\text{prod} := t_8$
(12) $t_9 := i + 1$
(13) $i := t_9$
(14) if $i \leq 20$ goto B_2



(3) $t_1 := 4 * i$
(4) $t_2 := a - 4$
(5) $t_3 := t_2[t_1]$
(7) $t_5 := b - 4$
(8') $t_6 := t_5[t_1]$
(9) $t_7 := t_3 * t_6$
(10) $t_8 := \text{prod} + t_7$
(11) $\text{prod} := t_8$
(12) $t_9 := i + 1$
(13) $i := t_9$
(14) if $i \leq 20$ goto B_2



(3) $t_1 := 4 * i$
(4) $t_2 := a - 4$
(5) $t_3 := t_2[t_1]$
(7) $t_5 := b - 4$
(8') $t_6 := t_5[t_1]$
(9) $t_7 := t_3 * t_6$
(11') $\text{prod} := \text{prod} + t_7$
(13') $i := i + 1$
(14) if $i \leq 20$ goto B_2

? 还可以这样

10.2.4 削弱计算强度

- 对基本块的代数变换：对表达式中的求值计算用代数上等价的形式替换，以便使复杂的运算变换成为简单的运算。

$x := y ** 2$

可以用代数上等价的乘式（如： $x := y * y$ ）代替

- $x := x + 0$ 和 $x := x * 1$
 - ◆ 执行的运算没有任何意义
 - ◆ 应将这样的语句从基本块中删除。

10.2.5 改变计算次序

- 考虑语句序列：

$t_1 := b + c$

$t_2 := x + y$

- 如果这两个语句是互不依赖的，即 x 、 y 均不为 t_1 ， b 、 c 均不为 t_2 ，则交换这两个语句的位置不影响基本块的执行结果。
- 对基本块中的临时变量重新命名不会改变基本块的执行结果。

如：语句 $t := b + c$

改成语句 $u := b + c$

把块中出现的所有 t 都改成 u ，不改变基本块的值。

次序和结果

10.4 循环优化

- 为循环语句生成的中间代码包括如下4部分：
 - ◆ **初始化部分**：对循环控制变量及其他变量赋初值。此部分组成的基本块位于循环体语句之前，可视为构成循环的第一个基本块。
 - ◆ **测试部分**：测试循环控制变量是否满足循环终止条件。这部分的位置依赖于循环语句的性质，若循环语句允许循环体执行0次，则在执行循环体之前进行测试；若循环语句要求循环体至少执行1次，则在执行循环体之后进行测试。
 - ◆ **循环体**：由需要重复执行的语句构成的一个或多个基本块组成。
 - ◆ **调节部分**：根据步长对循环控制变量进行调节，使其增加或减少一个特定的量。可把这部分视为构成该循环的最后一个基本块。
- 循环结构中的**调节部分**和**测试部分**也可以与**循环体**中的其他语句一起出现在基本块中。

循环优化的主要技术

10.4.1 循环展开

10.4.2 代码外提/频度削弱

10.4.3 削弱计算强度

10.4.4 删除归纳变量

10.4.1 循环展开

- 以空间换时间的优化过程。
 - ◆ 循环次数在编译时可以确定
 - ◆ 针对每次循环生成循环体（不包括调节部分和测试部分）的一个副本。
- 进行循环展开的条件：
 - ◆ 识别出循环结构，而且编译时可以确定循环控制变量的初值、终值、以及变化步长。
 - ◆ 用空间换时间的权衡结果是可以接受的。
- 在重复产生代码时，必须确保每次重复产生时，都对循环控制变量进行了正确的合并。

示例： 考虑C语言的循环语句：
for (i=0; i<10; i++)
 x[i]=0;

假定：
int x[10];
其存储空间基址： x

■ 生成三地址代码：

```
100: i:=0
101: if i<10 goto 103
102: goto 107
103: t1:=4*i
104: x[t1]:=0
105: i:=i+1
106: goto 101
107: ...
```

7条语句

完成循环需要执行53条语句

■ 循环展开：

```
100: x[0]:=0
101: x[4]:=0
102: x[8]:=0
103: x[12]:=0
104: x[16]:=0
105: x[20]:=0
106: x[24]:=0
107: x[28]:=0
108: x[32]:=0
109: x[36]:=0
```

10条赋值语句

10.4.2 代码外提/频度削弱

- 降低计算频度的优化方法，减少循环中代码总数的一个重要方法。
- 将循环结构中的与循环无关代码提到循环结构的外面（通常提到循环结构的前面），从而减少循环中的代码总数。
- 如C语言程序中的语句：
while (i<=limit-2) {
...
}
如果limit的值在循环过程中保持不变，则limit-2的计算与循环无关。

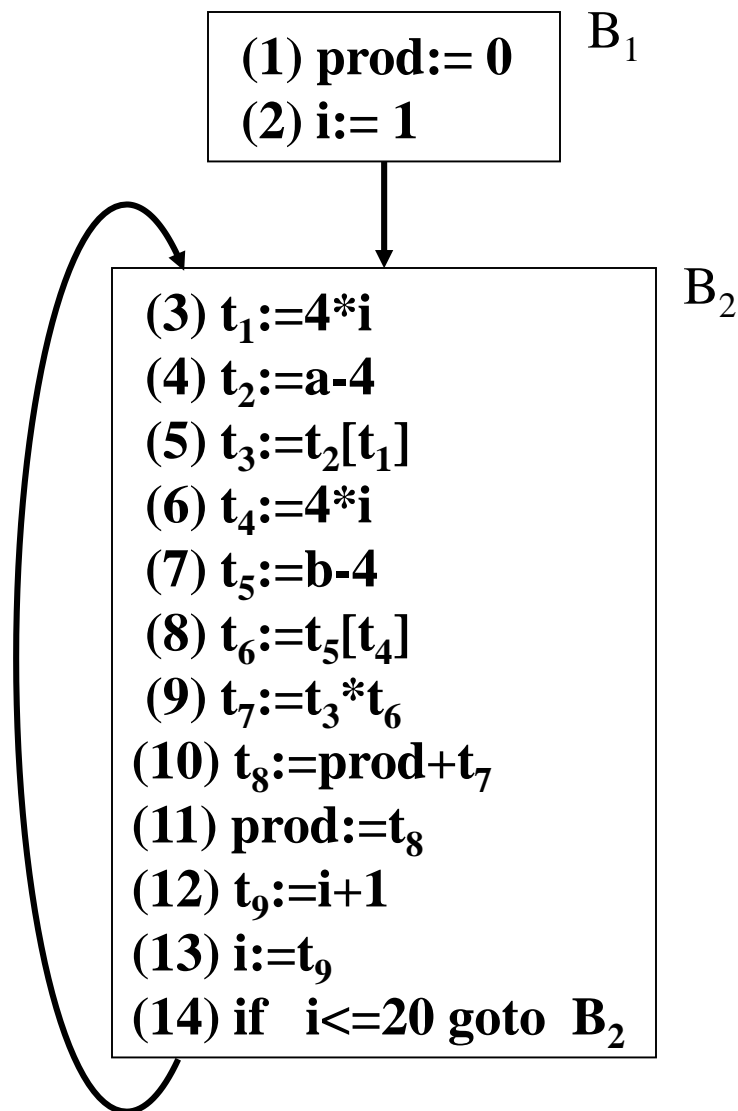
```
t:=limit-2;  
while (i<=t) {  
...  
}
```

例如：

■ 计算向量点积的程序

```
prod=0;  
i=1;  
do {  
    prod=prod+a[i]*b[i];  
    i++;  
}while (i<=20);
```

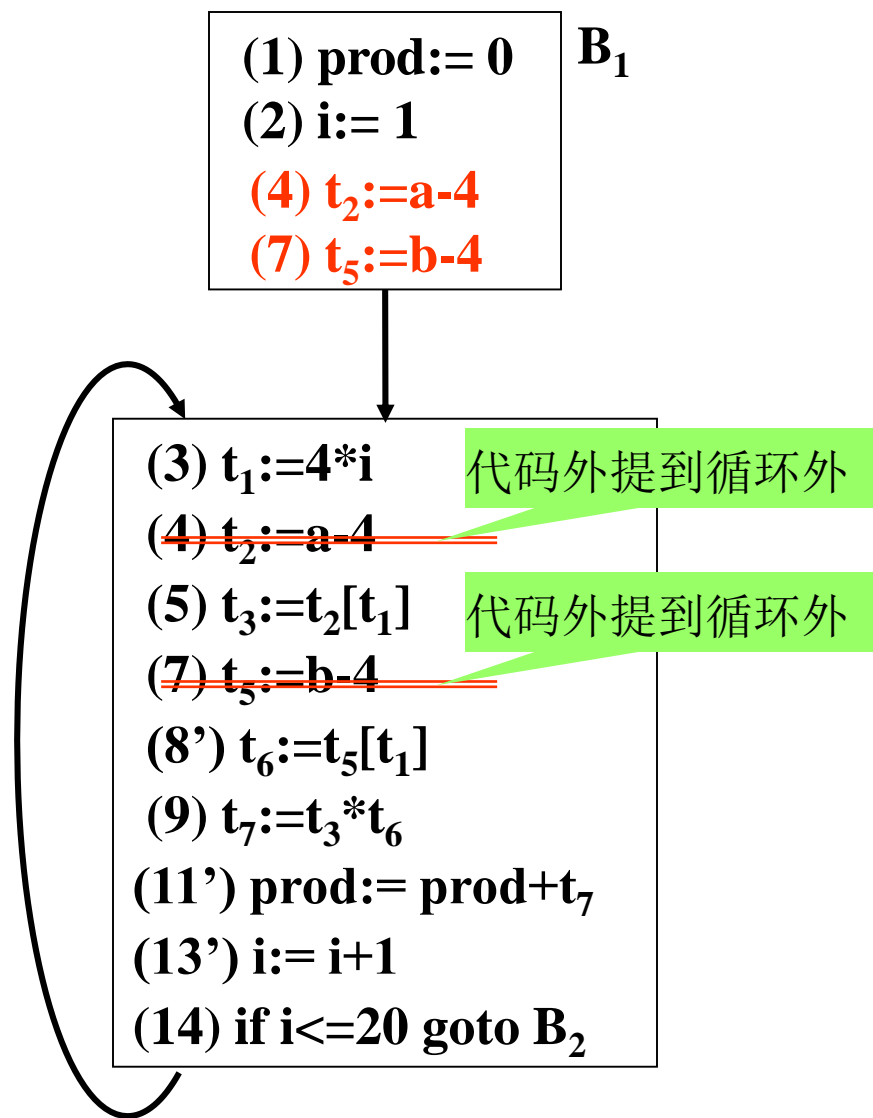
■ 程序的控制流图：



■ B_2 块经删除公共表达式和删除死代码后:

```
(3)  $t_1 := 4 * i$   
(4)  $t_2 := a - 4$   
(5)  $t_3 := t_2[t_1]$   
(7)  $t_5 := b - 4$   
(8')  $t_6 := t_5[t_1]$   
(9)  $t_7 := t_3 * t_6$   
(11')  $prod := prod + t_7$   
(13')  $i := i + 1$   
(14) if  $i \leq 20$  goto  $B_2$ 
```

■ 对 B_2 块进行代码外提:

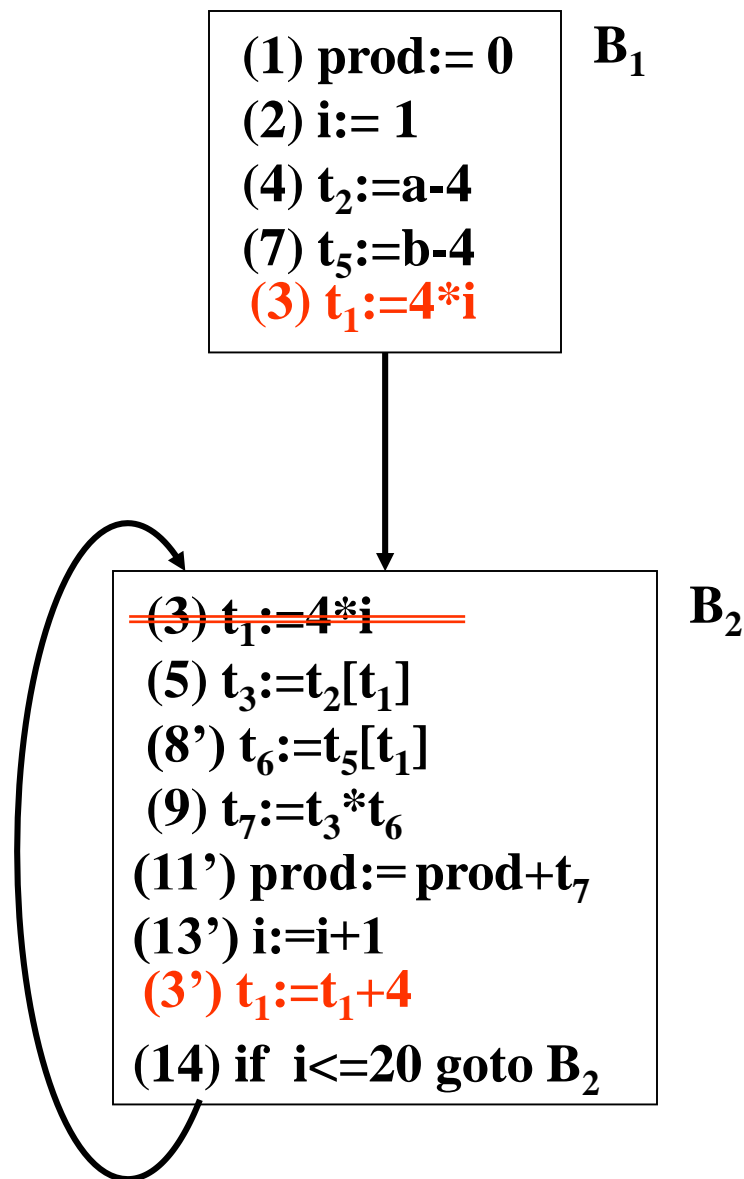


10.4.3 削弱计算强度

- 将当前运算类型代之以需要较少执行时间的运算类型的优化方法。
- 大多数计算机上乘法运算比加法运算需要更多的执行时间。
- 如可用 '+' 代替 '*', 则可节省许多时间, 特别是当这种替代发生在循环中时更是如此。

例如：

- B_2 块中每循环一次 i 的值增加1， t_1 的值始终与 i 保持着线性关系，每循环一次值增加4。因此可以把循环中计算 t_1 值的乘运算，变换成在循环前进行一次乘运算，而在循环中进行加法运算。

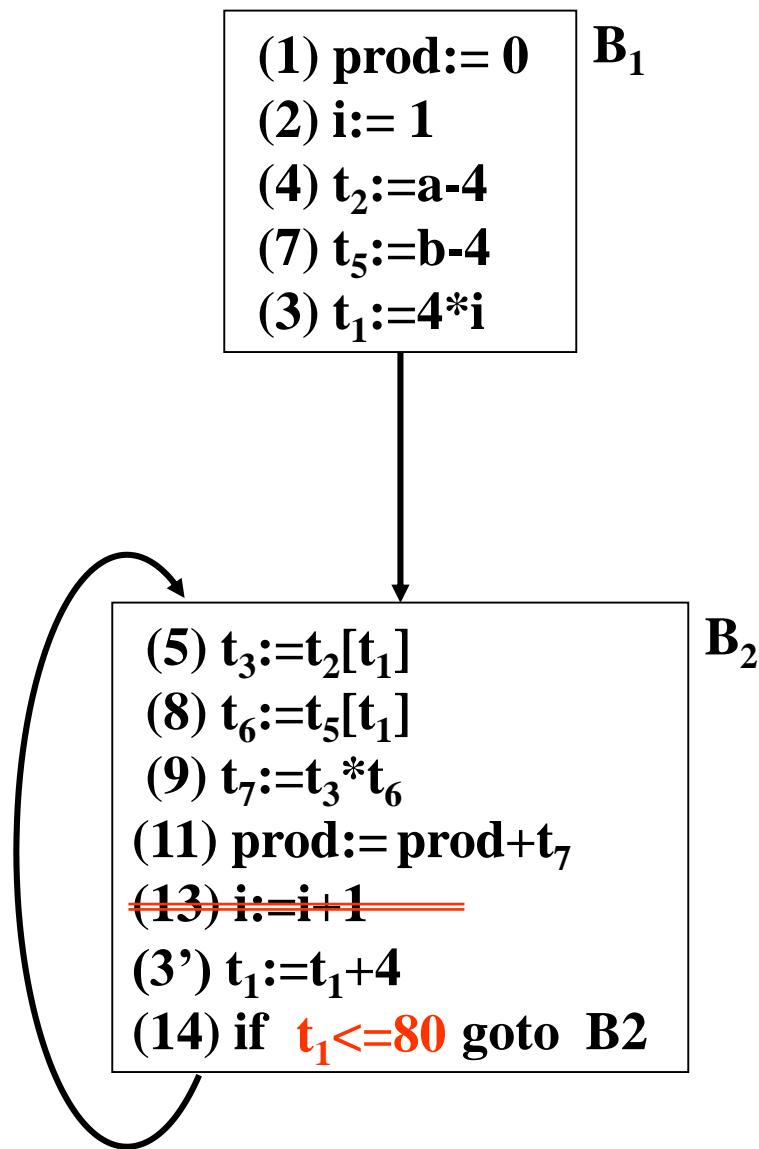


10.4.4 删除归纳变量

- 如果循环中对变量 i 只有唯一的形如 $i:=i+c$ 的赋值，并且 c 为循环不变量，则称 i 为循环中的基本归纳变量。
- 如果 i 是循环中的一个基本归纳变量， j 在循环中的定值总可以化归为 i 的同一线性函数，即 $j:=c_1*i+c_2$ ，这里 c_1 和 c_2 都是循环不变量，则称 j 是归纳变量，并称 j 与 i 同族。
- 通常，一个基本归纳变量除用于其自身的递归定值外，往往只用于计算其他归纳变量的值、以及用来控制循环的进行。

删除归纳变量 (续)

- 当循环中含有归纳变量时, 可视情况删除其中的一个或几个, 以减少存在的个数。
- 例如: 在 B_2 块中,
 - ◆ 无论在循环前和循环中, i 和 t_1 的值始终保持着 $t_1=4*i$ 的线性关系, 所以把循环控制条件 $i \leq 20$ 变换成 $t_1 \leq 80$, 对整个程序来说, 其运行结果是一样的
 - ◆ i 和 t_1 是一组归纳变量, 在 B_2 块中可以删除 i



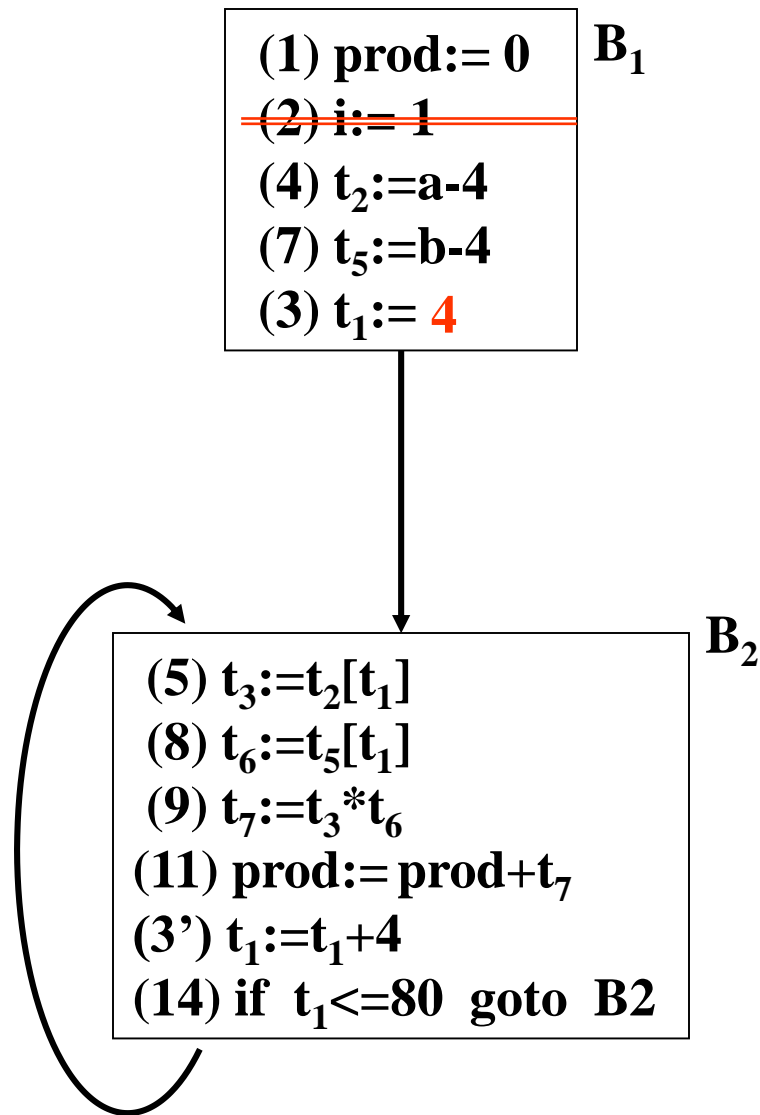
对B₁进行优化

■ 常数传播

- ◆ (2) 中，i赋值为1，(4)和(7)没有改变i的值，因此(3)中4*i的值为4

■ 删除死代码

- ◆ i的赋值已是无用赋值（因为i的值没有再引用），可以把(2)删除



10.5 窥孔优化

- 对目标代码进行局部改进的简单有效的技术
- 窥孔：在目标程序上设置的一个可移动的小窗口。
 - ◆ 通过窥孔，能看到目标代码中有限的若干条指令。
 - ◆ 窥孔中的代码可能不连续。
- 窥孔优化：依次考察通过窥孔可以见到的目标代码中很小范围内的指令序列，只要有可能，就代之以较短或较快的等价的指令序列。
 - ◆ 特点：每个改进都可能带来新的改进机会。
 - ◆ 通常需要对目标代码重复扫描。
- 常用技术：删除冗余指令、删除死代码、控制流优化、削弱计算强度及代数化简。
- 常作为改进目标代码质量的技术，也可用于中间代码的优化。

10.5.1 删除冗余的传送指令

- 如果窥孔中出现如下指令：
 - (1) `MOV a, R0`
 - (2) `MOV R0, a`
- 若这两条指令在同一基本块中，删除(2)是安全的。
 - ◆ 指令(1)的执行已经保证a的当前值同时存放在其存储单元和寄存器R₀中。
- 如果指令(2)是一个基本块的入口语句，则不能删除
 - ◆ 不能保证指令(2)紧跟在(1)之后执行。

10.5.2 删除死代码

- 死代码：程序中控制流不可到达的一段代码。
- 如果无条件转移指令的下一条指令没有标号，即没有控制转移到此语句，则它是死代码，应该删除。
 - ◆ 删除死代码的操作有时会连续进行，从而删除一串指令。
- 如果条件转移语句中的条件表达式的值是个常量，则生成的目标代码势必有一个分支成为死代码。
 - ◆ 为了调试一个较大的C语言程序，通常需要在程序里插入一些用于跟踪调试的语句，当调试完成后，可能不删除这些语句，而只令其成为死代码。

示例：程序里插入的跟踪调试语句

```
#define debug 0
```

```
...
```

```
if debug {
```

```
...
```

```
/* 输出调试信息 */
```

```
}
```

- 翻译该 if 语句，得到的中间代码可能是：

```
if debug=1 goto L1
```

```
goto L2
```

```
L1: ... /* 输出调试信息 */
```

```
L2: ...
```

- 需要把从 if 到 L₂所标识的语句之前的全部语句删除。

10.5.3 控制流优化

- 连续跳转的goto语句:

goto L₂

...

L₁: goto L₂

- 条件转移语句:

if a<b goto L₂

...

L₁: goto L₂

- 如果控制结构为:

goto L₁

...

L₁: if a<b goto L₂

L₃: ...

- 如果只有这一个语句转移到L₁:

if a<b goto L₂

goto L₃

...

L₃: ...

10.5.4 强度削弱及代数化简

- 削弱计算强度：用功能等价的执行速度较快的指令代替执行速度慢的指令。
 - ◆ 特定的目标机器上，某些机器指令比其它一些指令执行要快得多。如：
 - 用 $x*x$ 实现 x^2 比调用指数函数要快得多。
 - 用移位操作实现定点数乘以2或除以2的幂运算比进行乘/除运算要快。
 - 浮点数除以常数用乘以常数近似实现要快等，如： $x/5$ 变为 $x*0.2$ 。
- 窥孔优化时，有许多代数化简可以尝试，但经常出现的代数恒等式只有少数几个。
 - ◆ 如： $x:=x+0$ 或 $x:=x*1$
 - ◆ 在简单的中间代码生成算法中经常出现这样的语句，它们很容易由窥孔优化删除。

充分利用目标机器的特点

- 目标机器可能有高效实现某些专门操作的硬指令，找出允许使用这些指令的情况可明显缩短执行时间。
- 如：
某些机器有加1或减1的硬件指令（INC/DEC），用这些指令实现语句 $i:=i+1$ 或者 $i:=i-1$ ，可大大改进代码质量。

小 结

- 代码优化程序的功能
 - ◆ 等价变换
 - ◆ 执行时间
 - ◆ 占用空间
- 代码优化程序的组织
 - ◆ 控制流分析
 - ◆ 数据流分析
 - ◆ 代码变换
- 优化种类
 - ◆ 基本块优化
 - ◆ 循环优化
 - ◆ 窥孔优化

小结（续）

- 基本块优化的主要技术
 - ◆ 常数合并与常数传播
 - ◆ 删除冗余的公共表达式
 - ◆ 复制传播
 - ◆ 删除死代码
 - ◆ 削弱计算强度
 - ◆ 改变计算次序

小结 (续)

- 循环优化的主要技术
 - ◆ 循环展开
 - ◆ 代码外提/频度削弱
 - ◆ 削弱计算强度
 - ◆ 删除归纳变量
- 窥孔优化的主要技术
 - ◆ 删除冗余指令
 - ◆ 删除死代码
 - ◆ 控制流优化
 - ◆ 削弱计算强度及代数化简