C 代码优化（1） PDF转换可能丢失图片或格式，建议阅读原文

https://www.100test.com/kao_ti2020/449/2021_2022_C___E4_BB_A3_E7_A0_81_E4_c97_449188.htm 谈到优化，很多人都会直接想到汇编。难道优化只能在汇编层次吗？当然不是，C 层次一样可以作代码优化，其中有些常常是意想不到的。在C 层次进行优化，比在汇编层次优化具有更好的移植性，应该是优化中的首选做法。 确定浮点型变量和表达式是 float 型 为了让编译器产生更好的代码(比如说产生3DNow! 或SSE指令的代码)，必须确定浮点型变量和表达式是 float 型的。要特别注意的是，以＂；F＂；或＂；f＂；为后缀（比如：3.14f）的浮点常量才是 float 型，否则默认是 double 型。为了避免 float 型参数自动转化为 double，请在函数声明时使用 float。 使用32位的数据类型 编译器有很多种，但它们都包含的典型的32位类型是：int，signed，signed int，unsigned，unsigned int，long，signed long，long int，signed long int，unsigned long，unsigned long int。尽量使用32位的数据类型，因为它们比16位的数据甚至8位的数据更有效率。 明智使用有符号整型变量 在很多情况下，你需要考虑整型变量是有符号还是无符号类型的。比如，保存一个人的体重数据时不可能出现负数，所以不需要使用有符号类型。但是，如果是要保存温度数据，就必须使用到有符号的变量。 在许多地方，考虑是否使用有符号的变量是必要的。在一些情况下，有符号的运算比较快；但在一些情况下却相反。 比如：整型到浮点转化时，使用大于16位的有符号整型比较快。因为x86构架中提供了从有符号整型转化到浮点型的指令，但没有提供从无符号整型转

化到浮点的指令。看看编译器产生的汇编代码： 不好的代码 ： 编译前 编译后 double x； mov [foo 4], 0 unsigned int i； mov eax, i x = i； mov [foo], eax flid qword ptr [foo] fstp qword ptr [x] 上面的代码比较慢。不仅因为指令数目比较多，而且由于指令不能配对造成的FLID指令被延迟执行。最好用以下代码代替： 推荐的代码： 编译前 编译后 double x； fild dword ptr [i] int i； fstp qword ptr [x] x = i； 在整数运算中计算商和余数时，使用无符号类型比较快。以下这段典型的代码是编译器产生的32位整型数除以4的代码： 不好的代码 推荐的代码 编译前 编译后 int i； mov eax, i i = i / 4； cdq and edx, 3 add eax, edx sar eax, 2 mov i, eax 编译前 编译后 unsigned int i； shr i, 2 i = i / 4 ； 总结： 无符号类型用于： 除法和余数 循环计数 数组下标 有符号类型用于： 整型到浮点的转化 while VS. for 在编程中，我们常常需要用到无限循环，常用的两种方法是while (1) 和 for (； ；)。这两种方法效果完全一样，但那一种更好呢？然我们看看它们编译后的代码： 编译前 编译后 while (1)； mov eax,1 test eax,eax je foo 23h jmp foo 18h 编译前 编译后 for (； ；) ； jmp foo 23h 一目了然，for (； ；)指令少，不占用寄存器，而且没有判断跳转，比while (1)好。 使用数组型代替指针型 使用指针会使编译器很难优化它。因为缺乏有效的指针代码优化的方法，编译器总是假设指针可以访问内存的任意地方，包括分配给其他变量的储存空间。所以为了编译器产生优化得更好的代码，要避免在不必要的地方使用指针。一个典型的例子是访问存放在数组中的数据。C 允许使用操作符 [] 或指针来访问数组，使用数组型代码会让优化器减少产生不安全代码的可能性。比如，x[0] 和x[2] 不可能是同一个内存

地址，但 *p 和 *q 可能。强烈建议使用数组型，因为这样可能会有意料之外的性能提升。 不好的代码 推荐的代码 typedef struct { float x,y,z,w； } VERTEX； typedef struct { float m[4][4]； } MATRIX； void XForm(float* res, const float* v, const float* m, int nNumVerts) { float dp； int i； const VERTEX* vv = (VERTEX *)v； for (i = 0； i { dp = vv-> ；x * *m ； dp = vv-> ；y * *m ； dp = vv-> ；z * *m ； dp = vv-> ；w * *m ； *res = dp； // 写入转换了的 x dp = vv-> ；x * *m ； dp = vv-> ；y * *m ； dp = vv-> ；z * *m ； dp = vv-> ；w * *m ； *res = dp； // 写入转换了的 y dp = vv-> ；x * *m ； dp = vv-> ；y * *m ； dp = vv-> ；z * *m ； dp = vv-> ；w * *m ； *res = dp； // 写入转换了的 z dp = vv-> ；x * *m ； dp = vv-> ；y * *m ； dp = vv-> ；z * *m ； dp = vv-> ；w * *m ； *res = dp； // 写入转换了的 w vv ； // 下一个矢量 m -= 16； } } typedef struct { float x,y,z,w； } VERTEX； typedef struct { float m[4][4]； } MATRIX； void XForm (float* res, const float* v, const float* m, int nNumVerts) { int i； const VERTEX* vv = (VERTEX*)v； const MATRIX* mm = (MATRIX*)m； VERTEX* rr = (VERTEX*)res； for (i = 0； i { rr-> ；x = vv-> ；x * mm-> ；m[0][0] vv-> ；y * mm-> ；m[0][1] vv-> ；z * mm-> ；m[0][2] vv-> ；w * mm-> ；m[0][3]； rr-> ；y = vv-> ；x * mm-> ；m[1][0] vv-> ；y * mm-> ；m[1][1] vv-> ；z * mm-> ；m[1][2] vv-> ；w * mm-> ；m[1][3]； rr-> ；z = vv-> ；x * mm-> ；m[2][0] vv-> ；y * mm-> ；m[2][1] vv-> ；z * mm-> ；m[2][2] vv-> ；w * mm-> ；m[2][3]； rr-> ；w = vv-> ；x * mm-> ；m[3][0] vv-> ；y * mm-> ；m[3][1] vv-> ；z * mm-> ；m[3][2] vv-> ；w * mm-> ；m[3][3]； } } 100Test 下载频道开