链滴

优化程序性能

作者：zhuhonglin
原文链接：https://ld246.com/article/1502286724292
来源网站：链滴

优化程序性能

前言

很多情况下，除了保证我们的程序可以正确运行外，让程序运行的快也是一个需要考虑的重要因素。而一个程序要跑的快，是多个领域多个角度共同作用的结果，往往涉及许多程序优化的技术，这些太复杂难以理解，我希望通过学习，能够简单的总结几种代码优化的方案，从而帮助程序能够快速的执。

以下的例子使用 gcc 编译器编译的 c 程序

循环的优化

首先可以看一个循环的代码

```
// 将字符串中的字符全部变成小写： 比较慢的循环

void  lower(char  *s)
{
   long i;
   for(i = 0; i < strlen(s); i++)
      if (s[i] >= 'A' && s[i] <= 'Z')
         s[i] -= ('A' - 'a');
}

// 将字符串中的字符变成小写： 优化后的循环

void  lower_faster(char  *s)
{
   long i;
   long len = strlen(s);
   for(i = 0; i < len; i++)
      if (s[i] >= 'A' && s[i] <= 'Z')
         s[i] -= ('A' - 'a');
}
```

两段代码之间的差距就是我们提前使用一个变量接收了字符串的长度，并且之后用在了 for 循环中，什么这种方式可以优化代码?

几乎每一种编程语言都会有 for 循环，不过毫无疑问所有的 for 循环最后都变成了一句句指令代码，此只要搞明白这个指令代码是怎么回事，就可以明白为什么后者的循环效率更高，速度更快。

for 循环执行流程：

```
init i
test i <= n
update i++
statements in for block...
```

每一次 for 循环，都需要比较一次条件，如果在循环的条件中写入函数，那么每一次比较都需要调用数，而后者编译器只是直接拿出这个地址就可以。如果说 strlen 作为内建函数，编译器能够发现问题

行优化，那么要是其他的函数写入循环中，就有可能造成浪费。

内存引用优化

以下例子

```
void  dosomething(data_type v[], data_type *dest)
{
   long i;
   long length = get_length(v)
   for (i = 0; i < length; i++)
   {
      *dest = *dest + v[i];
   }
}
```

我们避免再循环中使用外部调用，从而加快性能，但是实际性能没有明显提升。说明是其他的因素限了性能。

将这段代码编译成汇编语言，可以发现，每一次循环，都需要从 dest 指向的内存中读出值，计算完之后再写入内存，第二次循环又读出值，而实际上，这些步骤是可以避免的，因为上一次写入的值和次读出的值并没有什么不同，白白消耗了读写的时间。

为了消除这种读写，最好能够有一个存储在寄存器中的变量，暂时的接受这些循环迭代，直到循环结以后，一次写入 dest 所指的内存。因此引入一个临时变量

```
void  dosomething(data_type v[], data_type *dest)
{
   long i;
   long length = get_length(v)
   data_type acc = INIT_VALUE;
   for (i = 0; i < length; i++)
   {
      acc = acc + v[i];
   }
   *dest = acc;
}
```

临时变量 acc（寄存器中） 暂存每次迭代的值，从而避免读写内存，直到循环结束，一次写入内存。

对高速缓存友好的代码

1. 最常见的情况需要运行的快（核心代码要快）
2. 尽量命中

对于命中的情况可以使用二元数组举例子

```
int  sum_array(int a[M][N])
{
   int i, j, sum = 0;
```

```
    for (i = 0; i < M; i++)
        for(j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

这段代码首先使用了临时变量，确保速度上没有问题，现在考虑命中的问题，假设高速缓存一次缓存个值，也就是读取 a[0][0]时，不命中，接着a[0][1] ~ a[0][3]载入缓存，接下来读取这三个值时，全命中。之后循环这个过程。无论如何命中率始终是4中3。

而如果循环调换顺序，先读取每一列，造成的结果是每一次都是不命中，这对缓存太不友善了，循环执行一次，缓存就不得不重新加载四个值，其中三个还浪费了。

参考书籍《深入理解计算机系统》