第五章书面作业

5.14

编写习题 5.13 中描述的内积过程的一个版本，使用 6*1 循环展开。对于 x86-64，我们对这个展开的版本的测试得到，对整数数据 CPE 为 1.07，而对两种浮点数据 CPE 仍然为 3.01。

A．解释为什么在 Intel Core i7Haswell 上运行的任何（标量）版本的内积过程都不能达到比 1.00 更小的 CPE 了。

B．解释为什么对浮点数据的性能不会通过循环展开而得到提高。

代码：

```
1. void inner4(vec_ptr u, vec_ptr v, data_t *dest){
2.     long i;
3.     long length = vec_length(u);
4.     data_t *udata = get_vec_start(u);
5.     data_t *vdata = get_vec_start(v);
6.     data_t sum = (data_t) 0;
7.
8.     for (i = 0; i < length - 6; i += 6) {
9.         sum = sum + udata[i] * vdata[i];
10.            + udata[i + 1] * vdata[i + 1];
11.            + udata[i + 2] * vdata[i + 2];
12.            + udata[i + 3] * vdata[i + 3];
13.            + udata[i + 4] * vdata[i + 4];
14.            + udata[i + 5] * vdata[i + 5];
15.    }
16.
17.    for (; i < length; i ++){
18.        sum = sum + udata[i] * vdata[i];
19.    }
20.
21.    *dest = sum;
22.}
```

答：

A．因为只有一个微处理器能执行加操作，C=1，而发射时间为 1 个周期，故程序至少运行 N*I/C=N 个周期，此时 CPE=N/N=1。故任何版本都不能达到比 1.00 更小的 CPE 了。

B．因为此时没有发生流水线，每次迭代需要 6*3=18 个周期，所以 CPE=18/6=3，没有得到提高。

5.15

编写习题 5.13 中描述的内积过程的一个版本，使用 6*6 循环展开。对于 x86-64，我们对这个函数的测试得到对整数数据的 CPE 为 1.06，对浮点数据的 CPE 为 1.01。什么因素制约了性能达到 CPE 等于 1.00？

代码：

```
1. void inner4(vec_ptr u, vec_ptr v, data_t *dest){
2.     long i;
3.     long length = vec_length(u);
4.     data_t *udata = get_vec_start(u);
5.     data_t *vdata = get_vec_start(v);
6.     data_t sum = (data_t) 0;
7.     data_t sum1 = (data_t) 0;
8.     data_t sum2 = (data_t) 0;
9.     data_t sum3 = (data_t) 0;
10.    data_t sum4 = (data_t) 0;
11.    data_t sum5 = (data_t) 0;
12.
13.    for (i = 0; i < length - 6; i += 6) {
14.        sum = sum + udata[i] * vdata[i];
15.        sum1 = sum1 + udata[i + 1] * vdata[i + 1];
16.        sum2 = sum2 + udata[i + 2] * vdata[i + 2];
17.        sum3 = sum3 + udata[i + 3] * vdata[i + 3];
18.        sum4 = sum4 + udata[i + 4] * vdata[i + 4];
19.        sum5 = sum5 + udata[i + 5] * vdata[i + 5];
20.    }
21.
22.    for (; i < length; i ++){
23.        sum = sum + udata[i] * vdata[i];
24.    }
25.
26.    sum = sum + sum1 + sum2 + sum3 + sum4 + sum5;
27.    *dest = sum;
28.}
```

答：指令的发射时间和加法/乘法处理器个数制约了性能达到 CPE=1.00。

5.17

库函数 memset 的原型如下：

void *memset(void *s, int c, size_t n);

这个函数将从 s 开始的 n 个字节的内存区域都填称为 c 的地位字节。例如，通过将参数 c 设置为 0，可以用这个函数来对一个内存区域清零，不过用其他值也是可以的。

下面是 memset 最直接的实现：

```
1. /* Basic implementation of memset */
2. void *basic_memset(void *s, int c, size_t, n) {
3.     size_t cnt = 0;
4.     unsigned char *schar = s;
5.     while (cnt < n) {
6.         *schar ++ = (unsigned char) c;
7.         cnt ++;
8.     }
9.     return s;
10.}
```

实现该函数一个更有效的版本，使用数据类型为 unsigned long 的字来装下 8 个 c，然后用字级的写遍历目标内存区域。你可能发现增加额外的循环展开会有所帮助。在我们的参考机上，能够把 CPE 从直接实现的 1.00 降低到 0.127。即，程序每个周期可以写 8 个字节。

代码：

```
1. void *advanced_memset(void *s, int c, size_t n){
2.     size_t K = sizeof(unsigned long);
3.     size_t sc = sizeof(unsigned char);
4.     unsigned long cs = 0;
5.     for (size_t i = 0; i < K; i ++) {
6.         cs <<= (sc << 2);
7.         cs += (unsigned char) c;
8.     }
9.     size_t cnt = 0;
10.    unsigned char *schar = s;
11.    while(cnt < n) {
12.        if(schar % K == 0) break;              // align 对齐
   unsigned long 的位置
13.        *schar ++ = (unsigned char) c;
14.        cnt ++;
15.    }
16.    size_t lst = n - cnt;
17.    size_t m = lst / K;
18.    size_t rest = lst % K;
19.    unsigned long *slong = (unsigned long *) schar; // 一次进行8
   字节的覆盖
20.    for (size_t i = 0; i < m; i ++) {
```

```
        *slong ++ = cs;
    }
    schar = (unsigned char *) slong;
    for (size_t i = 0; i < rest; i ++) {
        *schar ++ = (unsigned char) c;
    }
    return s;
}
```