Table of Contents

第五章家庭作业

李一鸣 1160300625

code

github: upupming/CSAPP/chapter5

All solutions and lab reports are in Chinese.

serve locally

clone code

```
git clone https://github.com/upupming/CSAPP.git
cd CSAPP/chapter5
```

install nodejs

```
sudo apt install nodejs nodejs-legacy
```

install `gitbook-cli`

```
npm i -g gitbook-cli
```

install plugins

```
gitbook install
```

serve

```
gitbook serve --no-watch
```

visit link

```
http://localhost:4000
```

generate ebook

prerequisite

- ebook-convert

```
sudo apt install calibre
```

generate book

```
gitbook pdf ./ ./CSAPP-chapter5.pdf
gitbook mobi ./ ./CSAPP-chapter5.mobi
gitbook epub ./ ./CSAPP-chapter5.epub
```

ref: gitbook toolchain: ebook

5.13

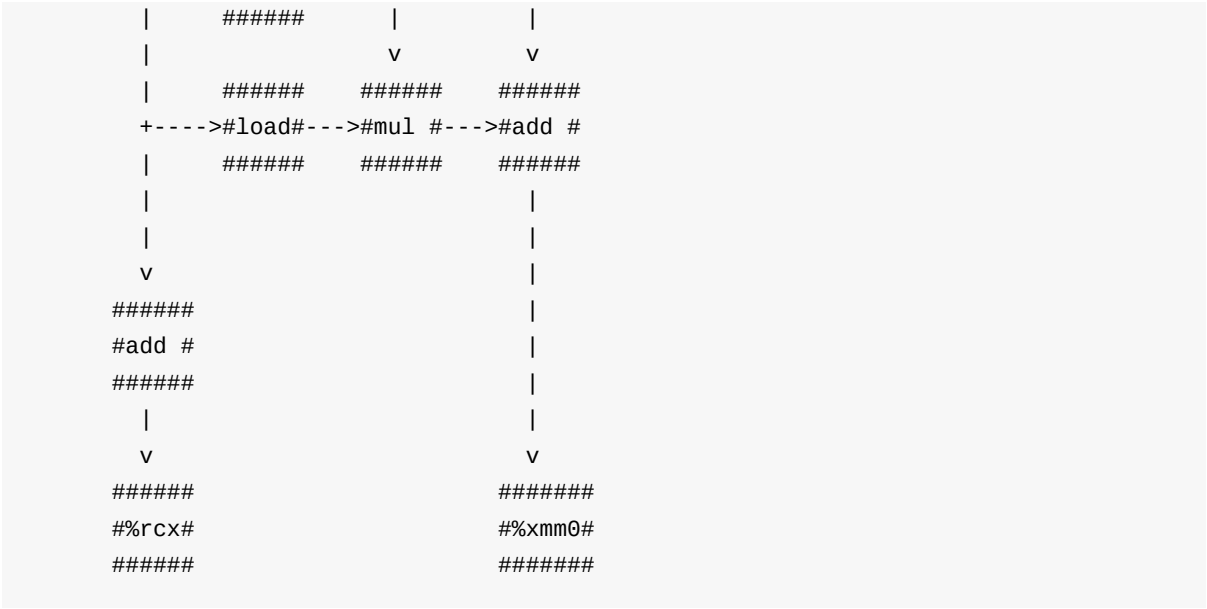
A.

先根据机器代码参照图5-13画出数据流图。 下图中 `|` 、 `-` 代表连线， `+` 代表连接点， `>` 表示指向。

```
#######前一次迭代产生的寄存器值########
#%rbp|%rcx|%rax|%rbx|%xmm1|%xmm0|
##|####|####|####|####################
  +----|----|----|----------|--->#    #
  |    +----|----|----------|--->#load#      vmovad 0(%rbp,%rcx,8),%xmm1
  |    |    |    |    +-----|----#    #
  |    |    |    |    |     |    ######
  |    +----|----|----------|--->#    #
  |    |    |    |    |     |    #load#---+  vmulsd (%rax,%rcx,8), ...
  |    |    +----|----------|--->#    #   |
  |    |    |    |    |     |    ######   |
  |    |    |    |    +-----|--->#    #<--+
  |    |    |    |          |    #mul #      vmulsd ..., %xmm1, %xmm1
  |    |    |    |    +-----|----#    #
  |    |    |    |    |     |    ######
  |    |    |    |    +-----|--->#    #
  |    |    |    |    |     +--->#add #      vaddsd %xmm1,%xmm0,%xmm0
  |    |    |    |    |     +----#    #
  |    |    |    |    |     |    ######
  |    +----|----|----------|--->#    #
  |         |    |    |     |    #add #      addq $1, %rcx
  |    +----|----|----------|----#    #
  |    |    |    |    |     |    ######
  |    +----|----|----------|--->#    #
  |    |    |    |    |     |    #cmp #---+  cmpq %rbx, %rcx
  |    |    |    +----|-----|--->#    #   |
  |    |    |    |    |     |    ######   |
  |    |    |    |    |     |    #    #   |
  |    |    |    |    |     |    #jne #<--+  jne .L15
  |    |    |    |    |     |    #    #
  |    |    |    |    |     |    ######
  v    v    v    v    v     v
|%rbp|%rcx|%rax|%rbx|%xmm1|%xmm0|
#######产生用于下一次迭代的寄存器值####
```

再参照图5-14去掉不属于某个循环寄存器之间的相关链，得到关键路径。

```
      ######                      #######
      #%rcx#                      #%xmm0#
      ##|###                      ##|####
        |                           |
        |     ######                |  <--------- 关键路径
        +---->#load#------+         |
```



可以看到：左边是整数加法，延迟为1个周期；右边是浮点加法，延迟为3个周期。 右边的链是关键路径。

B.

浮点加法决定了CPE下界为3.00。

C.

整数加法决定了CPE下界为1.00。

D.

乘法操作不在关键路径上，CPE下界由关键路径上的浮点加法操作决定。

5.14

A.

使用6x1循环展开，迭代次数变为1/6，但是每次迭代有6个顺序加法操作，关键路径变为原来6倍。 因此只是减少了循环开销，使得CPE接近且不低于其延迟界限1.00。

B.

跟A类似。 但是循环开销相对于浮点加法已经可以忽略不计了，所以CPE基本没有降低。

```
/*
 * 5.14.c 求内积6x1循环展开
 */
/* Inner product. Accumulate in temporary */
void inner4_6_1(vec_ptr u, vec_ptr v, data_t *dest){
    long i;
    long length = vec_length(u);
    long limit = length-5;// i<length-5, i+5<length
    data_t *udata = get_vec_start(u);
    data_t *vdata = get_vec_start(v);
    data_t sum = (data_t) 0;

    for(i = 0; i < limit; i+=6){
        sum = sum + udata[i] * vdata[i] +
             udata[i+1] * vdata[i+1] +
             udata[i+2] * vdata[i+2] +
             udata[i+3] * vdata[i+2] +
             udata[i+4] * vdata[i+3] +
             udata[i+5] * vdata[i+4]
            );
    }

    // 多出来的部分
    for(; i < length; i++){
        sum = sum + udata[i] * vdata[i];
    }
    *dest = sum;
}
```

5.15

1. 对于整数来说：循环开销增大，k值（=6）不够大。
2. 对于浮点数来说：浮点加法的吞吐量界限为1.00，CPE不可能低于它。

```
/*
 * 5.15.c 求内积6x6循环展开
 */
/* Inner product. Accumulate in temporary */
void inner4_6_6(vec_ptr u, vec_ptr v, data_t *dest){
    long i;
    long length = vec_length(u);
    long limit = length-5;// i<length-5, i+5<length
    data_t *udata = get_vec_start(u);
    data_t *vdata = get_vec_start(v);
    data_t sum0 = (data_t) 0;
    data_t sum1 = (data_t) 0;
    data_t sum2 = (data_t) 0;
    data_t sum3 = (data_t) 0;
    data_t sum4 = (data_t) 0;
    data_t sum5 = (data_t) 0;

    for(i = 0; i < limit; i+=6){
        sum0 = sum0 + udata[i] * vdata[i];
        sum1 = sum1 + udata[i+1] * vdata[i+1];
        sum2 = sum2 + udata[i+2] * vdata[i+2];
        sum3 = sum3 + udata[i+3] * vdata[i+3];
        sum4 = sum4 + udata[i+4] * vdata[i+4];
        sum5 = sum5 + udata[i+5] * vdata[i+5];
    }

    // 多出来的部分
    for(; i < length; i++){
        sum0 = sum0 + udata[i] * vdata[i];
    }
    *dest = sum0 + sum1 + sum2 + sum3 + sum4 + sum5;
}
```

5.16

```
/*
 * 5.16.c 求内积6x1a循环展开
 */
/* Inner product. Accumulate in temporary */
void inner4_6_1a(vec_ptr u, vec_ptr v, data_t *dest){
    long i;
    long length = vec_length(u);
    long limit = length-5;// i<length-5, i+5<length
    data_t *udata = get_vec_start(u);
    data_t *vdata = get_vec_start(v);
    data_t sum = (data_t) 0;

    for(i = 0; i < limit; i+=6){
        sum = sum +
            (
             udata[i] * vdata[i] +
             udata[i+1] * vdata[i+1] +
             udata[i+2] * vdata[i+2] +
             udata[i+3] * vdata[i+2] +
             udata[i+4] * vdata[i+3] +
             udata[i+5] * vdata[i+4]
            );
    }

    // 多出来的部分
    for(; i < length; i++){
        sum = sum + udata[i] * vdata[i];
    }
    *dest = sum;
}
```

5.17

```c
/* 5.17.c
 * memset8x8循环展开
 */
void my_memset(void *s, unsigned long cs, size_t n){
    // 数据对其，开始部分以单字节写如
    size_t K = sizeof(unsigned long);
    size_t cnt = 0;
    unsigned char *schar = s;
    while(cnt < n){
        if((size_t)schar % K == 0){
           // 找到对齐起始点
           break;
        }
        *schar++ = (unsigned char)cs;
        cnt++;
    }

    // 每次复制8字节
    unsigned long *slong = (unsigned long *)schar;
    // size_t比较不可用减法
    for(; cnt + K < n; cnt += K){
        *slong = cs;
    }

    // 剩余部分，以单字节写入
    schar = (unsigned char)slong;
    for(; cnt < n; cnt++){
        *schar++ = (unsigned char)cs;
    }
    return s;
}
```

5.18

使用6x3a循环展开优化。

```
/* 5.18.c
 * 多项式求值6x3a循环展开
 */
double poly_6_3a(double a[], double x, long degree){
    long i;
    long limit = degree-5;// i<=degree-5, i+5<=degree
    double result0 = a[0];
    double result1 = 0;
    double result2 = 0;

    // x^1 x^3 x^5
    double xpwr0 = x;
    double xpwr1 = x * x * x;
    double xpwr2 = xpwr1 * x * x;

    // x^6
    double xpwr_step = xpwr1 * xpwr1;

    // 循环展开
    for(i = 1; i <= limit; i+=6){
        result0 = result0 + (a[i]*xpwr0 + a[i+1]*xpwr0*x);
        result1 = result1 + (a[i+2]*xpwr1 + a[i+3]*xpwr1*x);
        result2 = result2 + (a[i+4]*xpwr2 + a[i+5]*xpwr2*x);

        xpwr0 *= xpwr_step;
        xpwr1 *= xpwr_step;
        xpwr2 *= xpwr_step;
    }
    // 剩余部分
    for(; i <= degree; i++){
        result0 = result0 + a[i]*xpwr0;
        xpwr0 *= x;
    }

    return result0 + result1 + result2;
}
```

5.19

使用6x6a循环展开。 浮点加法的延迟界限为3.00,吞吐量界限为1.00。6x6a循环展开可以使CPE接近于1.00。

```
/*
 * 5.12.c 前置和计算6x6a循环展开
 */
void psum_6_6a(float a[], float p[], long n){
    long i;
    /* last_val holds p[i-1]; val holds p[i] */
    float last_val;
    float val0, val1, val2, val3, val4, val5;
    last_val = p[0] = a[0];

    // i < n-5, i+5 < n
    long limit = n-5;

    //循环展开
    for(i = 1; i < limit; i++){
        val0 = last_val + a[i];     p[i] = val0;
        val1 = val0 + a[i+1];       p[i+1] = val1;
        val2 = val1 + a[i+2];       p[i+2] = val1;
        val3 = val2 + a[i+3];       p[i+3] = val1;
        val4 = val3 + a[i+4];       p[i+4] = val1;
        val5 = val4 + a[i+5];       p[i+5] = val1;

        // 这里很有技巧，本来应该写成：
        // last_val = last_val + (a[i] + a[i+1] + a[i+2] + a[i+3] + ...);
        // 但是前置和比较特殊，可以不用重复加了
        last_val = val5;
    }
    // 剩余部分
    for(; i < n; i++){
        last_val += a[i];
        p[i] = last_val;
    }
}
```