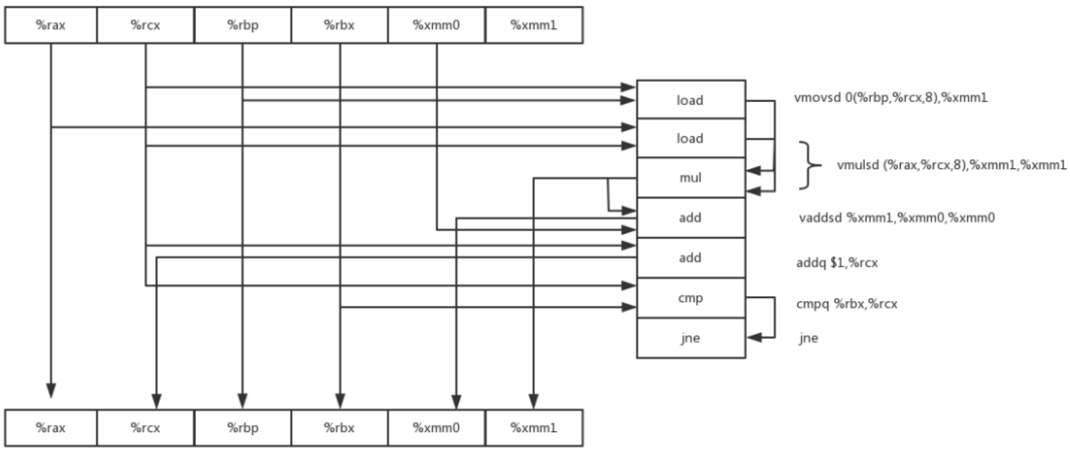
第五章 优化程序性能-家庭作业

——1170300825 李大鑫

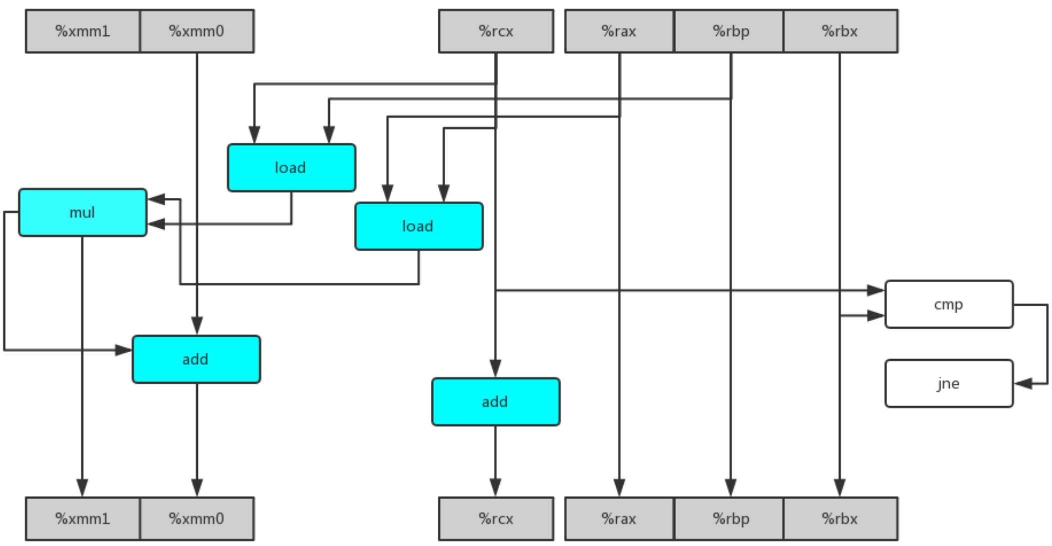
5.13

A:

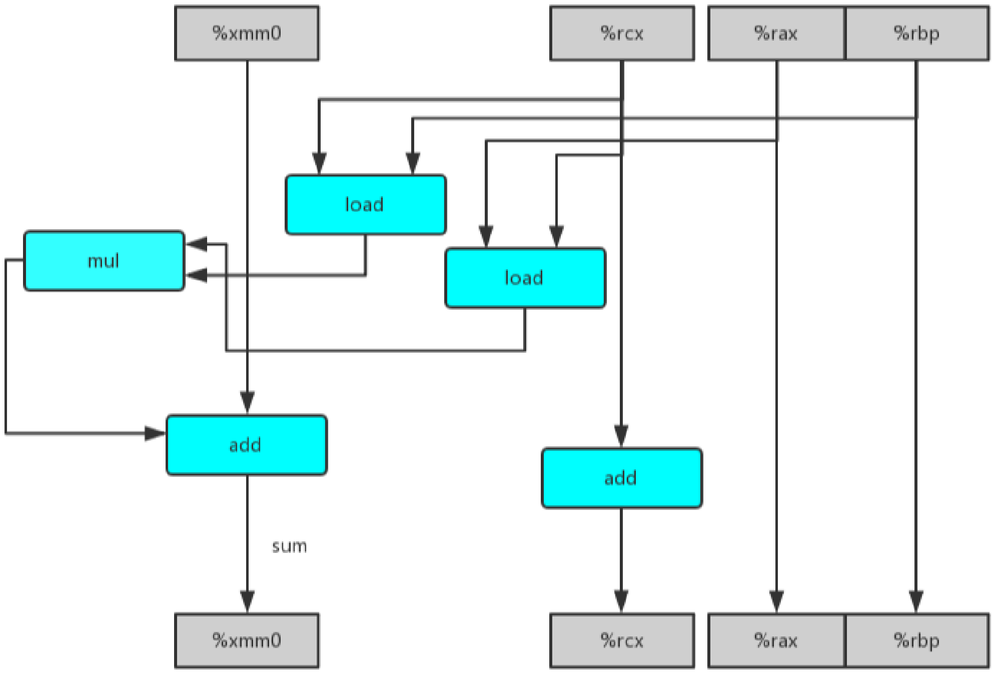
5-13 类似图



5-14（a）类似图



5-14（b）类似图



B

对于 double 类型，浮点相乘的需要 5 个时钟周期，浮点数相加需要 3 个时钟周期，则根据两条数据相关的关键路径可以得知这条关键路径决定 CPE 下界的是**浮点数加法的延迟**，CPE 下界是：3

C

对于整数数据，整数相乘需要 3 个时钟周期，整数相加需要 1 个时钟周期，决定这两条数据相关的关键路径 CPE 下界的是整数相加的延迟，CPE 下界是：1

D

从 A 中所列出的三个数据流图中我们可以看出，虽然浮点数相乘需要用 5 个时钟周期来完成，但是真正的关键路径中并没有包含 mul，这是因为，每次进行的 mul 运算都是独立的，也就是不需要依赖上一次乘法的结果，所以这些 mul 可以并行计算，而 add 运算是依赖于上一次运算的结果的，所以真正决定关键路径延时的是浮点数的加法，浮点数加法的延迟是 3 个时钟周期，所以 CPE 是 3.00。

5.15

A

代码:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#define LEN 24

#define data_t long

/* Create abstract date type for vector */
typedef struct {
    long len;
    data_t *data;
} vec_rec, *vec_ptr;
/* Create vector of specific length */
vec_ptr new_vec(long len) {
    /* allocate header structure */
    vec_ptr result = (vec_ptr) malloc(sizeof(vec_rec));
    if (!result)
        return result;

    result->len = len;
    data_t *data = NULL;
    /* allocate array */
    if (len > 0) {
        data = (data_t*) calloc(len, sizeof(data_t));
        if (!data) {
            free((void*) result);
            return NULL;
        }
    }
    result->data = data;
    return result;
}

/*
 * Retrieve vector element and store at dest
```

```c
 * return 0 (out of bounds) or 1 (successful)
 */
int get_vec_element(vec_ptr v, long index, data_t *dest) {
    if (index < 0 || index >= v->len) {
        return 0;
    }
    *dest = v->data[index];
    return 1;
}

/* return length of vector */
long vec_length(vec_ptr v) {
    return v->len;
}

/* expose data */
data_t* get_vec_start(vec_ptr v) {
    return v->data;
}

/* set data */
void set_vec_start(vec_ptr v, data_t *data) {
    v->data = data;
}

/* inner product. accumulate in temporary */
void inner4(vec_ptr u, vec_ptr v, data_t *dest) {
    long i;
    long length = vec_length(u);
    data_t *udata = get_vec_start(u);
    data_t *vdata = get_vec_start(v);
    data_t sum = (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;

    //利用运算的独立性
    //6x6循环展开
    //6次循环展开x6个累积变量
    //每次循环保证6个数的序号都在序列之中
    for (i = 0; i < length-6; i+=6) {
        sum = sum + 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];
    }
    //剩余的单独相乘 累积到sum1
    for(; i < length; i++) {
        sum = sum + udata[i] * vdata[i];
    }
    //结果相加
    *dest = sum + sum1 + sum2 + sum3 + sum4 + sum5;
}

int main(int argc, char* argv[]) {
    vec_ptr u = new_vec(LEN);
    vec_ptr v = new_vec(LEN);

    data_t *arr = (data_t*) malloc(sizeof(data_t) * LEN);
    memset(arr, 0, sizeof(data_t) * LEN);
    arr[0] = 0;
    arr[11] = 1;
    arr[2] = 2;
    arr[23] = 3;

    set_vec_start(u, arr);
    set_vec_start(v, arr);

    data_t res;
    inner4(u, v, &res);

    printf("ans : %d",res);
    return 0;
}
```

代码测试:

```
E:\CodeTraining\ProgramingLanguageTest\C\csapp_chapter5\cmake-build-debug\csapp_chapter5.exe

ans : 14

Process finished with exit code 0
```

B

什么因素制约了性能达到 CPE 等于 1.00? :

在关键路径上，虽然此时可以进行流水线并行计算，但是对于浮点数和整数而言，他们的功能单元的发射时间 issue_time 和容量 capacity 都是 1，那么这样的话，最多每个时钟周期完成 issue_time/capacity=1 个加法操作，所以此时 CPE 的下界是 1，不能再低了。

5.17

代码：

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

void* basic_memset(void *s, int c, size_t n) {
  size_t cnt = 0;
  unsigned char *schar = s;
  while (cnt < n) {
    *schar++ = (unsigned char) c;
    cnt++;
  }
  return s;
}

/*
 * K = long的长度
 * cs 存储K个memset需要的char
 */
void* effective_memset(void *s, unsigned long cs, size_t n) {
  /* K 对齐 写 的要求*/
  size_t K = sizeof(unsigned long);
  size_t cnt = 0;
  unsigned char *schar = s;
  //首先进行字节写 直到schar的地址是K的倍数 停止 这样保证了以后写的K对齐
  while (cnt < n) {
    if ((size_t)schar % K == 0) {
      break;
    }
    *schar++ = (unsigned char)cs;
    cnt++;
  }

  /* 使用字级的写，一次性写K */
```

```c
  //获得指向目标的long型（字级）的指针，这样每次slong+1就是跨越一个字，对于
  unsigned long *slong = (unsigned long *)schar;
  size_t rest = n - cnt;      //剩余字节数
  size_t loop = rest / K;     //可以进行的循环次数
  size_t tail = rest % K;     //循环补充的字节数

  //字节级的写 一次性写K个字节
  for (size_t i = 0; i < loop; i++) {
    *slong++ = cs;
  }

  /* 循环补充的 字节级别的写 */
  schar = (unsigned char *)slong;
  for (size_t i = 0; i < tail; i++) {
    *schar++ = (unsigned char)cs;
  }
  return s;
}


int main(int argc, char* argv[]) {
  size_t space = sizeof(char) * 65537;

  void* basic_space = malloc(space);
  void* effective_space = malloc(space);

  int basic_fill = 0xFF;
  unsigned long effective_fill = ~0;

  basic_memset(basic_space, basic_fill, space);
  effective_memset(effective_space, effective_fill, space);

  //assert 断言判断最终结果是否正确
  assert(memcmp(basic_space, effective_space, space) == 0);

  free(basic_space);
  free(effective_space);
  return 0;
}
```

代码测试:

```
E:\CodeTraining\ProgramingLanguageTest\C\csapp_chapter5\cmake-build-debug\5-17.exe


Process finished with exit code 0
```

5.19

代码:

```c
/*
 * 5.19.c
 */
#include <stdio.h>
#include <assert.h>

void psum1a(float a[], float p[], long n) {
  long i;
  float last_val, val;
  last_val = p[0] = a[0];
  for (i = 1; i < n; i++) {
    val = last_val + a[i];
    p[i] = val;
    last_val = val;
  }
}

/* 4x4a版本*/
void psum_4_4a(float a[], float p[], long n) {
  long i;
  float  last_val;
  float tmp, tmp1, tmp2, tmp3;
  last_val = p[0] = a[0];

  /* 循环展开优化 */
  for (i = 1; i < n - 4; i+=4) {
    tmp = last_val + a[i];
    tmp1 = tmp + a[i+1];
    tmp2 = tmp1 + a[i+2];
    tmp3 = tmp2 + a[i+3];

    /*多个累积变量*/
    p[i] = tmp;
    p[i+1] = tmp1;
    p[i+2] = tmp2;
    p[i+3] = tmp3;
```

```
    /* 重新结合优化 */
    last_val = last_val + (a[i] + a[i+1] + a[i+2] + a[i+3]);
  }
  //循环补充
  for (; i < n; i++) {
    last_val += a[i];
    p[i] = last_val;
  }
}

#define LOOP 1000
#define LEN  1000

int main(int argc, char* argv[]) {
  float a[5] = { 1, 2, 3, 4, 5 };
  float p[5];
  psum1a(a, p, 5);
  assert(p[4] == 15);

  float q[5];
  psum_4_4a(a, q, 5);
  //使用断言测试结果的正确性
  //将4x1a版本与1a版本进行正确性比较
  assert(q[4] == 15);

  for (int i = 0; i < LOOP; i++) {
    float s[LEN];
    float d[LEN];
    psum1a(s, d, LEN);
    psum_4_4a(s, d, LEN);
  }
  return 0;
}
```

代码测试:

```
E:\CodeTraining\ProgramingLanguageTest\C\csapp_chapter5\cmake-build-debug\5-19.exe


Process finished with exit code 0
```