实验目的

0. 通过 Pthreads实现通用矩阵乘法

1. 基于Pthreads的数组求和

2. Pthreads求解二次方程组的根

3. 编写一个多线程程序来估算y=x^2曲线与x轴之间区域的面积，其中x的范围为[0,1]。

实验过程及代码

通过 Pthreads实现通用矩阵乘法

```
void *matrixMultiply(void *arg) {
    ThreadArgs *args = (ThreadArgs *)arg;
    int start_row = args->start_row;
    int end_row = args->end_row;

    for (int i = start_row; i < end_row; i++) {
        for (int j = 0; j < SIZE; j++) {
            int temp = 0;
            for (int k = 0; k < SIZE; k++) {
                temp += A[i][k] * B[k][j];
            }
            C[i][j] = temp;
        }
    }

    pthread_exit(NULL);
}
```

这段代码将任务分配给不同的线程，加速矩阵乘法的计算。

然后进行了相关测试，测试出不同线程数量和矩阵规模的执行时间，如下。

规模和线程	512	1024	2048
1	0.269	2.497	75.58
2	0.137	1.265	33.65
4	0.0621	0.671	16.18

可以明显地看出运算时间显著减少了。

基于Pthreads的数组求和

```
void *computeSum(void *arg) {
    int local_sum = 0;
    int index;

    while (1) {
        pthread_mutex_lock(&mutex); // 加锁
        index = global_index; // 获取下一个未加元素的全局下标
        global_index++;
        pthread_mutex_unlock(&mutex); // 解锁

        if (index >= ARRAY_SIZE) {
            break; // 所有元素已计算
        }

        local_sum += a[index];
    }

    pthread_mutex_lock(&second_mutex); // 加锁
    sum += local_sum; // 将局部总和添加到全局总和
    pthread_mutex_unlock(&second_mutex); // 解锁

    pthread_exit(NULL);
}
```

这是每个线程只能提取一个元素的情况.

```
void *computeSum(void *arg) {
    int local_group_index;
    int group_sum = 0;

    while (1) {
        pthread_mutex_lock(&mutex);
        local_group_index = global_group_index;
        // printf("global_group_index:%d\n",global_group_index);
        global_group_index++;

        pthread_mutex_unlock(&mutex);

        if (local_group_index >= NUM_THREADS) {
            break;
        }
printf("group_sum:%d\n",group_sum);
        int group_start = local_group_index * GROUP_SIZE;
        int group_end = (local_group_index + 1) * GROUP_SIZE;
        group_sum = 0;
        for (int i = group_start; i < group_end; i++) {
            // printf("a[%d]:%d",i,a[i]);

            group_sum += a[i];
        }
```

```
        pthread_mutex_lock(&second_mutex);

        sum += group_sum;
        // printf("sum:%d\n",sum);
        pthread_mutex_unlock(&second_mutex);

    }

    pthread_exit(NULL);
}
```

这是一次最多提取是个的情况，通过本地和全局下标进行数组的访问，每个线程进行10个元素的计算。

求解二次方程的根

```
void *thread_root(void *arg) {
    int num_of_thread = (int)arg;
    printf("#%d start.\n", num_of_thread);

    pthread_mutex_lock(&mutex);
    count++;
    if (count == THREAD_NUM) {
        count   = 0;
        bb      = b * b;
        four_ac = 4 * a * c;
        printf("#%d computed b*b and 4ac.\n", num_of_thread);
        pthread_cond_broadcast(&cond_var);
    } else {
        while (pthread_cond_wait(&cond_var, &mutex))
            ;
    }
    pthread_mutex_unlock(&mutex);

    pthread_mutex_lock(&mutex);
    count++;
    if (count == THREAD_NUM) {
        count = 0;
        two_a = 2 * a;
        printf("#%d computed 2a.\n", num_of_thread);
        pthread_cond_broadcast(&cond_var);
    } else {
        while (pthread_cond_wait(&cond_var, &mutex))
            ;
    }
    pthread_mutex_unlock(&mutex);

    pthread_mutex_lock(&mutex);
    count++;
    if (count == THREAD_NUM) {
        count = 0;
        sqrtd = sqrt(bb - four_ac);
```

```
            printf("#%d computed sqrt.\n", num_of_thread);
            pthread_cond_broadcast(&cond_var);
        } else {
            while (pthread_cond_wait(&cond_var, &mutex))
                ;
        }
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        count++;
        if (count == THREAD_NUM) {
            count = 0;
            add   = -b + sqrtd;
            sub   = -b - sqrtd;
            printf("#%d computed -b +/-.\n", num_of_thread);
            pthread_cond_broadcast(&cond_var);
        } else {
            while (pthread_cond_wait(&cond_var, &mutex))
                ;
        }
        pthread_mutex_unlock(&mutex);

        pthread_mutex_lock(&mutex);
        count++;
        if (count == THREAD_NUM) {
            count = 0;
            a_d2a = add / two_a;
            s_d2a = sub / two_a;
            printf("#%d computed x1 and x2.\n", num_of_thread);
            pthread_cond_broadcast(&cond_var);
        } else {
            while (pthread_cond_wait(&cond_var, &mutex))
                ;
        }
        pthread_mutex_unlock(&mutex);

        printf("#%d exit.\n", num_of_thread);
        pthread_exit(0);
    }
```

这种情况是先进行局部的项计算，最后再进行相加减，然后算出方程的根。中间利用`count`来实现线程的同步。

编写一个多线程程序来估算y=x^2曲线与x轴之间区域的面积，其中x的范围为[0,1]。

```
void* cast (void* args)
{
    int i;
    float x,y;
    int temp=0;
```

```
    for(i=0; i< (intervals/tnum); i++){
        x= (float)(rand() % 1001) * 0.001f;
        y= (float)(rand() % 1001) * 0.001f;
        if(y<x*x) temp++;
    }
    //加上
    pthread_mutex_lock(&mutex);
    sum += temp;
    pthread_mutex_unlock(&mutex);
}
```

这个实现比较简单，各个线程都进行随机模拟，然后把符合的相加。

实验结果

实验中验证Pthreads的可行性，并且发现比简单的串行具有更高的效率。

实验反思

多线程编程的debug比简单的串行编程难上不少，花费了不少的功夫，同时Pthreads比MPI编程更轻松的一点是，减少了通信，更简单。