中山大学计算机院本科生实验报告

（2024 学年秋季学期）

课程名称：高性能计算程序设计　　**批改人：**

实验	基于 Pthread 的多线程编程	专业（方向）	信息与计算科学
学号	22336044	姓名	陈圳煌
Email	2576926165@qq.com	完成日期	2024.10.28

1.实验目的

从通用矩阵乘法、数组求和、 求解二次方程组的根以及 Monte-carlo 方法四个小任务来探究基于 GCC_8.1.0 环境下内置 Pthread 库的多线程编程。

2.实验过程和核心代码

· 通用矩阵乘法（代码文件为 task0.c）

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<time.h>

int thread_count;
int m, n, k;
double **A;
double **B;
double **C;

void* Pth_mat_vect(void* rank){
    long my_rank=(long)rank;
    int local_m=m/thread_count;
    int begin_row=my_rank*local_m;
    int end_row=(my_rank+1)*local_m;
    for(int i=begin_row;i<end_row;i++){
        for(int j=0;j<k;j++){
            C[i][j]=0.0;
            for(int l=0;l<n;l++){
                C[i][j]+=A[i][l]*B[l][j];
            }
        }
    }
    return NULL;
```

```
}

int main(int argc, char* argv[]){
    printf("Input matrix scale:\n");
    scanf("%d %d %d",&m,&n,&k);
    long thread;
    pthread_t* thread_handles;

    thread_count=strtol(argv[1], NULL, 10);
    thread_handles=malloc(thread_count*sizeof(pthread_t));

    srand(time(NULL));
    A=malloc(m*sizeof(double*));
    B=malloc(n*sizeof(double*));
    C=malloc(m*sizeof(double*));
    for(int i=0;i<m;i++){
        A[i]=malloc(n*sizeof(double));
        C[i]=malloc(k*sizeof(double));
    }
    for(int i=0;i<n;i++){
        B[i]=malloc(k*sizeof(double));
    }
    //initialize
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            A[i][j]=rand()*5.53/(double)RAND_MAX;
            // A[i][j]=0.05;
        }
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<k;j++){
            B[i][j]=rand()*3.35/(double)RAND_MAX;
            // B[i][j]=0.05;
        }
    }

    clock_t start=clock();
    for(thread=0;thread<thread_count;thread++){
        pthread_create(&thread_handles[thread], NULL, Pth_mat_vect,
(void*)thread);
    }

    for(thread=0;thread<thread_count;thread++){
```

```
        pthread_join(thread_handles[thread], NULL);
    }
    clock_t end=clock();
    double my_time=((double)(end-start))*1000.0/CLOCKS_PER_SEC;

    printf("Used time: %.4lf ms",my_time);
    //free
    for(int i=0;i<n;i++){
        free(A[i]);
    }
    free(A);
    for(int i=0;i<k;i++){
        free(B[i]);
        free(C[i]);
    }
    free(B);
    free(C);
    free(thread_handles);
}
```

首先创建 n 个线程，根据线程序号对矩阵进行划分，分成 n 个 M/n 行的分块矩阵（列数不变），分别进行矩阵乘法，但是不需要为每个线程额外分配空间。

· 数组求和（代码文件为 task1.c）

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<time.h>

int thread_count;
int global_index=0;
int a[1000];
int a_sum=0;
pthread_mutex_t mutex;

void* add(void* rank){
    long my_rank=(long)rank;

    while(1){
        //mutex
        pthread_mutex_lock(&mutex);
        if(global_index>=1000){
```

```
            pthread_mutex_unlock(&mutex);
            return NULL;
        }
        for(int i=global_index;i<global_index+10;i++){
            a_sum+=a[i];
        }
        global_index+=10;
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}

int main(int argc, char* argv[]){
    long thread;
    pthread_t* thread_handles;
    pthread_mutex_init(&mutex, NULL);
    thread_count=strtol(argv[1], NULL, 10);
    thread_handles=malloc (thread_count*sizeof(pthread_t));

    srand(time(NULL));
    //initialize
    for(int i=0;i<1000;i++){
        a[i]=rand()*100/RAND_MAX;
    }

    for(thread=0;thread<thread_count;thread++){
        pthread_create(&thread_handles[thread], NULL, add,
(void*)thread);
    }

    for(thread=0;thread<thread_count;thread++){
        pthread_join(thread_handles[thread], NULL);
    }
    int sum=0;
    for(int i=0;i<1000;i++){
        sum+=a[i];
    }
    printf("Global index: %d\n",global_index);
    printf("Sum of arr is: %d\n",a_sum);
    printf("Read sum is: %d\n",sum);

    pthread_mutex_destroy(&mutex);
```

```
    free(thread_handles);
    return 0;
}
```

创建 n 个线程，每个线程根据线程序号，每次从全局变量 Global_index 作为起始位置，往后选取 10 个数进行求和运算。由于线程在临界区访问全局变量，因此多个线程同时访问则会出现同步问题，因此使用互斥锁 mutex 处理，当有线程进入临界区访问全局变量时，其他线程就无法进入临界区，直到在临界区的线程离开。为了避免除不尽的问题，每次线程进入临界区都会检测全局变量 Global_index 是否超过 1000，是则解开互斥锁并且结束线程。

· 求解二次方程组（代码文件为 task2.c）

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<time.h>
#include<math.h>

int thread_count=8;
struct EQ{
    double a;
    double b;
    double c;
    double value[8];
    //-b,b^2,4ac,2a,det, √b^2-4ac,-b+,-b-
} eq;

int isok[8];

void* solve(void* rank){
    long my_rank=(long)rank;
    //busy-waiting
    while(isok[my_rank]!=1);

    switch(my_rank){
        case 0://-b
            eq.value[0]=-1*eq.b;
            break;
        case 1://b^2
            eq.value[1]=pow(eq.b,2);
            break;
        case 2://4ac
```

```
            eq.value[2]=4*eq.a*eq.c;
            break;
        case 3://2a
            eq.value[3]=2*eq.a;
            break;
        case 4://det
            eq.value[4]=eq.value[1]-eq.value[2];
            break;
        case 5:// √det
            if(eq.value[4]<0){
                break;
            }
            eq.value[5]=sqrt(eq.value[4]);
            break;
        case 6://(-b- √det)/2a
            if(eq.value[4]<0){
                break;
            }
            eq.value[6]=(eq.value[0]-eq.value[5])/eq.value[3];
            break;
        case 7://(-b+ √det)/2a
            if(eq.value[4]<0){
                break;
            }
            eq.value[7]=(eq.value[0]+eq.value[5])/eq.value[3];
            break;
    }
    isok[(my_rank+1)%thread_count]=1;
    return NULL;
}

int main(int argc, char* argv[]){
    long thread;
    pthread_t* thread_handles;
    printf("Please enter the coefficients of the quadratic
equation.(a!=0):\n");
    scanf("%lf %lf %lf",&eq.a,&eq.b,&eq.c);

    // thread_count=strtol(argv[1], NULL, 10);
    thread_handles=malloc (thread_count*sizeof(pthread_t));

    for(int i=0;i<8;i++){
        isok[i]=0;
```

```
    }
    isok[0]=1;

    for(thread=0;thread<thread_count;thread++){
        pthread_create(&thread_handles[thread], NULL, solve,
(void*)thread);
    }

    for(thread=0;thread<thread_count;thread++){
        pthread_join(thread_handles[thread], NULL);
    }

    printf("(%lf)x^2 + (%lf)x + (%lf) ",eq.a,eq.b,eq.c);
    if(eq.value[4]<0){
        printf("Has no solution!\n");
    }else if(eq.value[4]==0){
        printf("Has 1 solution: x= %.6lf\n",eq.value[6]);
    }else{
        printf("Has 2 solution:
x1= %.6lf  x2= %.6lf\n",eq.value[6],eq.value[7]);
    }

    free(thread_handles);
    return 0;

}
```

固定采用 8 个线程，将求解的步骤分步进行，需要使用忙等待，后一个线程需要等待前一个线程计算的部分完成才能执行。

· Monte-carlo 方法

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<time.h>

int thread_count;
//y=x^2 与 x 轴面积,x∈[0,1]，积分运算结果为 1/3
int point_num=0;
int MAX_POINT_NUM=10000000;
int under=0;//在函数曲线下方的点数
pthread_mutex_t mutex;
```

```c
double f(double x){
    return x*x;
}

void* set_point(void* rank){
    long my_rank=(long)rank;

    srand(time(NULL));
    while(1){
        pthread_mutex_lock(&mutex);
        if(point_num>=MAX_POINT_NUM){
            pthread_mutex_unlock(&mutex);
            return NULL;
        }
        double x=rand()*1.0/RAND_MAX;
        double y=rand()*1.0/RAND_MAX;
        // printf("(%.2lf,%.2lf)\n",x,y);
        if(y<=f(x)){
            under+=1;
        }
        point_num+=1;
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main(int argc, char* argv[]){
    long thread;
    pthread_t* thread_handles;
    pthread_mutex_init(&mutex, NULL);

    thread_count=strtol(argv[1], NULL, 10);
    thread_handles=malloc (thread_count*sizeof(pthread_t));

    for(thread=0;thread<thread_count;thread++){
        pthread_create(&thread_handles[thread], NULL, set_point,
(void*)thread);
    }

    for(thread=0;thread<thread_count;thread++){
        pthread_join(thread_handles[thread], NULL);
    }
```

```
    double area=under*1.0/MAX_POINT_NUM;
    printf("Used Point: %d\n",MAX_POINT_NUM);
    printf("Used Pthread: %d\n",thread_count);
    printf("Area: %.6lf\n",area);

    pthread_mutex_destroy(&mutex);
    free(thread_handles);
    return 0;
}
```

与任务 1 的实现类似，每个线程每次生成一个在正方形范围内的随机点，判断是否在函数图像 y=x^2 和 x 轴包围内，最后通过计算落入范围内的点数通过蒙德卡罗近似求 y=x^2 与 x 轴包围的区域的面积。

3. 实验结果

· 通用矩阵乘法

C 语言中使用时钟滴答数除以程序每秒钟时钟滴答数(CLOCK_PER_SEC)得到运行时间（结果 x1000 得到毫秒级）

Comm_size (num of processes)	Order of Matrix (milliseconds)				
	128	256	512	1024	2048
1	2	52	448	5401	60771
2	2	27	246	2873	33778
4	1	14	146	1517	17081
8	2	13	87	940	9150
16	2	9	80	776	7785

加速比

Comm_size (num of processes)	Order of Matrix (Speedups)				
	128	256	512	1024	2048
1	1.0	1.0	1.0	1.0	1.0
2	1.0	1.93	1.82	1.88	1.80
4	2.0	3.71	3.07	3.56	3.56
8	1.0	4	5.15	5.75	6.64
16	1.0	5.78	5.6	6.96	7.81

并行效率

Comm_size (num of processes)	Order of Matrix				
	128	256	512	1024	2048
1	1.0	1.0	1.0	1.0	1.0
2	0.5	0.97	0.91	0.94	0.9
4	0.5	0.93	0.77	0.89	0.89
8	0.13	0.5	0.64	0.72	0.83
16	0.06	0.36	0.35	0.44	0.49

可以看到，随着运算规模增大，线程数多的任务加速比和效率逐渐提高。与使用 MPI 实现通用矩阵乘法相比，MPI 涉及到多个进程间的通信，内存不共享，且需要手动分配任务，比较适合大规模的运算；而 Pthread 共享内存环境，线程之间的通信开销较低，但是需要处理数据冒险和死锁的问题。

· 数组求和

采用随机生成的 1000 个取值范围为[0,100]之间的数进行计算。

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task1 1
Global index: 1000
Sum of arr is: 49273
Read sum is: 49273
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task1 2
Global index: 1000
Sum of arr is: 49482
Read sum is: 49482
```

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task1 3
Global index: 1000
Sum of arr is: 49003
Read sum is: 49003
```

通过改变线程数，分成多个子任务进行数组求和，并与采用直接累加求和的结果进行对比，可以看到，最终的结果一致，由于数组规模较小，因此不进行时间比较。（以上结果测试表明无论线程数是否整除 1000，计算的任务分配不会出现错误）

· 求解二次方程组

测试 1：a=1,b=-4,c=4

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task2
Please enter the coefficients of the quadratic equation.(a!=0):
1 -4 4
(1.000000)x^2 + (-4.000000)x + (4.000000) Has 1 solution: x= 2.00
```

测试 2：a=1,b=6,c=9

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task2
Please enter the coefficients of the quadratic equation.(a!=0):
1 6 9
(1.000000)x^2 + (6.000000)x + (9.000000) Has 1 solution: x= -3.00
```

测试 3：a=122,b=126624,c=46.5

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task2
Please enter the coefficients of the quadratic equation.(a!=0):
122 126624 46.5
(122.000000)x^2 + (126624.000000)x + (46.500000) Has 2 solution: x1= -1037.901272 x2= -0.000367
```

经验证，得到的结果符合方程的解，证明了分线程求解方程的可行性。

· Monte-carlo 方法

测试 1：点数为 100

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task3 8
Used Point: 100
Used Pthread: 8
Area: 0.350000
```

测试 2：点数为 10000

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task3 8
Used Point: 10000
Used Pthread: 8
Area: 0.336500
```

测试 3：点数为 10000000

```
PS C:\Users\陈圳煌\Desktop\计算机学院\计算机学院（大三上）\高性能程序设计\test3> ./task3 8
Used Point: 10000000
Used Pthread: 8
Area: 0.333781
```

而 y=x^2 在[0,1]上的定积分为 1/3，可以看到随着模拟点数的增加，最终的结果越来越接近 1/3，经过验证，分线程实现 Monte-carlo 方法成功。

4. 实验感想

在这次实验中，我了解了 Pthread 并且学习了基于这个函数库的多线程编程。在进行实验之前，原本被如何配置环境困扰，后来发现 GCC-8.1.0 版本具备多线程编程功能，内置 Pthread 函数库。在实现 task1 过程中，我遇到了线程同步问题，导致了 Global_index 到达 1000 后，线程直接退出并未解除互斥锁导致死锁问题，最后通过定点输出发现问题所在。测试实验结果时，我将本次实验与上次实验的 MPI 进程通信进行对比，让我对并行计算有了更多的了解。