中山大学计算机院本科生实验报告

（2023 学年秋季学期）

课程名称：高性能计算程序设计　　批改人：

实验	通用矩阵乘法	专业（方向）	信息与计算科学
学号	21311359	姓名	何凯迪
Email	hekd@mail2.sysu.edu.cn	完成日期	2023/8/28

1. 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实现基于 Monte-carlo 方法的积分估值

2. 实验过程和核心代码

（0） 通过 Pthreads 实现通用矩阵乘法

定义数据结构：

```
// 定义单精度浮点数矩阵结构
typedef struct {
    int rows;
    int cols;
    float** data;
} Matrix;

// 数据结构传递给线程的参数
typedef struct {
    int thread_id;
    int num_threads;
    const Matrix* A;
    const Matrix* B;
    Matrix* C;
} ThreadArgs;
```

创建随机矩阵：

```
// 创建一个随机矩阵
Matrix createRandomMatrix(int rows, int cols) {
    Matrix mat;
    mat.rows = rows;
    mat.cols = cols;
    mat.data = (float**)malloc(rows * sizeof(float*));

    for (int i = 0; i < rows; i++) {
        mat.data[i] = (float*)malloc(cols * sizeof(float));
        for (int j = 0; j < cols; j++) {
            mat.data[i][j] = (float)rand() / RAND_MAX; // 生成0到1之间的随机浮点数
        }
    }

    return mat;
}
```

矩阵乘法函数：

```
// 用于线程的矩阵乘法函数
void* matrixMultiplyThread(void* thread_args) {
    ThreadArgs* args = (ThreadArgs*)thread_args;
    const Matrix* A = args->A;
    const Matrix* B = args->B;
    Matrix* C = args->C;
    int num_threads = args->num_threads;
    int thread_id = args->thread_id;

    int start_row = (A->rows / num_threads) * thread_id;
    int end_row = (thread_id == num_threads - 1) ? A->rows : start_row + (A->rows / num_threads);

    for (int i = start_row; i < end_row; i++) {
        for (int j = 0; j < B->cols; j++) {
            for (int k = 0; k < A->cols; k++) {
                C->data[i][j] += A->data[i][k] * B->data[k][j];
            }
        }
    }

    return NULL;
}
```

创建和启动线程：

```
    // 创建和启动线程
    for (int i = 0; i < num_threads; i++) {
        thread_args[i].thread_id = i;
        thread_args[i].num_threads = num_threads;
        thread_args[i].A = &A;
        thread_args[i].B = &B;
        thread_args[i].C = &C;

        if (pthread_create(&threads[i], NULL, matrixMultiplyThread, &thread_args[i]) != 0) {
            perror("pthread_create");
            return 1;
        }
    }
```

等待线程结束：

```
    // 等待线程结束
    for (int i = 0; i < num_threads; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("pthread_join");
            return 1;
        }
    }
```

打印矩阵并释放内存：

```
// 打印矩阵的内容，只输出部分内容，用省略号代替其余部分
void printMatrix(const Matrix* mat, int maxRows, int maxCols) {
        int numRows = mat->rows;
        int numCols = mat->cols;

        for (int i = 0; i < numRows && i < maxRows; i++) {
                for (int j = 0; j < numCols && j < maxCols; j++) {
                        printf("%f ", mat->data[i][j]);
                }

                if (numCols > maxCols) {
                        printf("..."); // 用省略号代替超出部分的列
                }

                printf("\n");
        }

        if (numRows > maxRows) {
                printf("...\n"); // 用省略号代替超出部分的行
        }
}

// 释放矩阵内存
void freeMatrix(Matrix* mat) {
    for (int i = 0; i < mat->rows; i++) {
        free(mat->data[i]);
    }
    free(mat->data);
}
```

（1） 基于 Pthreads 的数组求和

求和函数：

```
// 用于线程的求和函数
void* threadSum(void* arg) {
    int local_group_sum = 0;

    while (1) {
        int group;

        // 加锁以获取下一个组的索引
        pthread_mutex_lock(&mutex);
        group = global_group;
        global_group++;
        pthread_mutex_unlock(&mutex);

        if (group < NUM_GROUPS) {
            int group_start = group * NUM_ELEMENTS_PER_GROUP;
            int group_end = group_start + NUM_ELEMENTS_PER_GROUP;
            for (int i = group_start; i < group_end; i++) {
                local_group_sum += a[i];
            }
        } else {
            break; // 所有组已经被处理
        }
    }

    // 加锁以更新总和
    pthread_mutex_lock(&mutex);
    sum += local_group_sum;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}
```

创建和启动线程：

```
    // 创建和启动线程
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], NULL, threadSum, NULL) != 0) {
            perror("pthread_create");
            return 1;
        }
    }
```

等待线程结束：

```
    // 等待线程结束
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("pthread_join");
            return 1;
        }
    }
```

一次提取十个连续的数：

```
#define ARRAY_SIZE 1000
#define NUM_ELEMENTS_PER_GROUP 10

int NUM_GROUPS = ARRAY_SIZE / NUM_ELEMENTS_PER_GROUP;

int a[ARRAY_SIZE];
int group_sums[ARRAY_SIZE / NUM_ELEMENTS_PER_GROUP];
int global_group = 0; // 修正全局变量名
int sum = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

// 用于线程的求和函数
void* threadSum(void* arg) {
    int local_group_sum = 0;

    while (1) {
        int group;

        // 加锁以获取下一个组的索引
        pthread_mutex_lock(&mutex);
        group = global_group;
        global_group++;
        pthread_mutex_unlock(&mutex);

        if (group < NUM_GROUPS) {
            int group_start = group * NUM_ELEMENTS_PER_GROUP;
            int group_end = group_start + NUM_ELEMENTS_PER_GROUP;
            for (int i = group_start; i < group_end; i++) {
                local_group_sum += a[i];
            }
        } else {
            break; // 所有组已经被处理
        }
    }

    // 加锁以更新总和
    pthread_mutex_lock(&mutex);
    sum += local_group_sum;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}
```

（2） Pthreads 求解二次方程组的根

相关定义：

```c
// 定义二次方程的系数
double a, b, c;
double x1, x2;
int solutions_found = 0;

// 定义互斥锁和条件变量
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

// 定义线程的返回值结构
typedef struct {
    double b_squared;
    double four_a_c;
} ThreadResult;
```

计算中间量函数：

```c
// 用于线程的求解函数，计算b²的部分
void* calculateBSquared(void* arg) {
    ThreadResult* result = (ThreadResult*)arg;
    result->b_squared = b * b;
    pthread_exit(NULL);
}

// 用于线程的求解函数，计算4ac的部分
void* calculateFourAC(void* arg) {
    ThreadResult* result = (ThreadResult*)arg;
    result->four_a_c = 4 * a * c;
    pthread_exit(NULL);
}
```

创建和启动线程：

```c
    // 创建并启动两个线程，分别计算b²和4ac的部分
    pthread_create(&thread_b_squared, NULL, calculateBSquared, &result_b_squared);
    pthread_create(&thread_four_a_c, NULL, calculateFourAC, &result_four_a_c);
```

等待线程结束：

```c
    // 等待两个线程结束
    pthread_join(thread_b_squared, NULL);
    pthread_join(thread_four_a_c, NULL);
```

用得到的中间量求解：

```
    // 计算根
    double discriminant = result_b_squared.b_squared - result_four_a_c.four_a_c;

    if (a == 0) {
        if (b != 0) {
            x1 = x2 = -c / b;
            solutions_found = 1;
        } else if (c == 0) {
            solutions_found = -1; // 无穷多解
        }
    } else {
        if (discriminant > 0) {
            x1 = (-b + sqrt(discriminant)) / (2 * a);
            x2 = (-b - sqrt(discriminant)) / (2 * a);
            solutions_found = 2;
        } else if (discriminant == 0) {
            x1 = x2 = -b / (2 * a);
            solutions_found = 1;
        }
    }
```

（3） 编写一个多线程程序来实现基于 Monte-carlo 方法的积分估值。

Monte-carlo 估算：

```
// 用于线程的估算函数
void* estimateArea(void* arg) {
    int points_per_thread = NUM_POINTS / NUM_THREADS;
    int points_inside = 0;

    for (int i = 0; i < points_per_thread; i++) {
        double x = (double)rand() / RAND_MAX; // 随机生成 x 值
        double y = (double)rand() / RAND_MAX; // 随机生成 y 值

        if (y <= x * x) {
            points_inside++;
        }
    }

    pthread_mutex_lock(&mutex);
    area += (double)points_inside / points_per_thread;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}
```

创建和启动线程：

```
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_create(&threads[i], NULL, estimateArea, NULL) != 0) {
            perror("pthread_create");
            return 1;
        }
    }
```

等待线程结束：

```
    for (int i = 0; i < NUM_THREADS; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            perror("pthread_join");
            return 1;
        }
    }
```

3. 实验结果

（0） 通过 Pthreads 实现通用矩阵乘法（以 2048 阶不同线程数为例）

2048 阶单线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./matrix_mult 2048 2048 2048 1
N=2048, M=2048, K=2048
num_threads=1
Execution time: 41644.746311 milliseconds

Matrix A:
0.664541 0.071188 ...
0.238904 0.449839 ...
...

Matrix B:
0.698296 0.778810 ...
0.298551 0.231386 ...
...

Matrix C (Result of A * B):
504.742401 524.495300 ...
499.321564 531.235291 ...
...
```

2048 阶双线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./matrix_mult 2048 2048 2048 2
N=2048, M=2048, K=2048
num_threads=2
Execution time: 20711.451160 milliseconds

Matrix A:
0.485899 0.096496 ...
0.377412 0.970775 ...
...

Matrix B:
0.075690 0.494485 ...
0.792588 0.914933 ...
...

Matrix C (Result of A * B):
504.523682 503.721741 ...
530.625671 523.728149 ...
...
```

2048 阶四线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./matrix_mult 2048 2048 2048 4
N=2048, M=2048, K=2048
num_threads=4
Execution time: 10344.407864 milliseconds

Matrix A:
0.094539 0.900865 ...
0.628698 0.240278 ...
...

Matrix B:
0.149012 0.779351 ...
0.601046 0.504262 ...
...

Matrix C (Result of A * B):
498.317627 503.112823 ...
505.723267 505.684448 ...
...
```

2048 阶八线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./matrix_mult 2048 2048 2048 8
N=2048, M=2048, K=2048
num_threads=8
Execution time: 5872.120856 milliseconds

Matrix A:
0.324706 0.006511 ...
0.371849 0.718293 ...
...

Matrix B:
0.156940 0.530319 ...
0.431922 0.201414 ...
...

Matrix C (Result of A * B):
512.574829 515.496094 ...
504.359985 503.916138 ...
...
```

2048 阶十六线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./matrix_mult 2048 2048 2048 16
N=2048, M=2048, K=2048
num_threads=16
Execution time: 5672.593769 milliseconds

Matrix A:
0.774938 0.438854 ...
0.224475 0.056797 ...
...

Matrix B:
0.984979 0.338149 ...
0.040983 0.911305 ...
...

Matrix C (Result of A * B):
504.857910 521.138000 ...
491.913055 505.551239 ...
...
```

可见正常运行。

（1） 基于 Pthreads 的数组求和

1~1000 求和，四线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./sum 4
Sum: 500500
Execution time: 0.269075 milliseconds
```

1~1000 求和，八线程：

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./sum 8
Sum: 500500
Execution time: 0.358563 milliseconds
```

1~1000 求和，四线程：（一次提取十个数）

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./sum_ten 4
Sum: 500500
Execution time: 0.248908 milliseconds
```

1~1000 求和，八线程：（一次提取十个数）

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./sum_ten 8
Sum: 500500
Execution time: 0.358072 milliseconds
```

可见正常运行且结果正确，一次提取十个数的运行时间少于一个一个提取。

（2） Pthreads 求解二次方程组的根

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ gcc -o quadratic quadratic.cpp -lm -lpthread
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./quadratic
Enter the coefficients of the quadratic equation (a, b, c): 2 1 -1
Two real solutions: x1 = 0.500000, x2 = -1.000000
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./quadratic
Enter the coefficients of the quadratic equation (a, b, c): 1 2 1
One real solution: x = -1.000000
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./quadratic
Enter the coefficients of the quadratic equation (a, b, c): 1 0 1
No real solutions
```

可见正常运行且结果正确。

（3） 编写一个多线程程序来实现基于 Monte-carlo 方法的积分估值。

```
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ gcc -o monte-carlo monte-carlo.cpp -lpthread
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./monte-carlo 1
Estimated area: 0.332854
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./monte-carlo 2
Estimated area: 0.334090
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./monte-carlo 4
Estimated area: 0.332593
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./monte-carlo 8
Estimated area: 0.333675
kaddy@kaddy-VirtualBox:~/HPC/exp_2$ ./monte-carlo 16
Estimated area: 0.333237
```

由数学知识可得，所求积分应为 1/3，结果接近。

4. 实验感想

//可以写写过程中遇到的问题，你是怎么解决的。以及可以写你对此次实验的一些理解……

本次实验较上次而言，简单方便了许多，pthread 用得我神清气爽！！