中山大学计算机学院本科生实验报告

课程名称：并行程序设计与算法

实验	**Pthreads并行矩阵乘法与数组求和**	专业（方向）	计算机科学与技术
学号	21307035	姓名	邓栩瀛
Email	dengxy66@mail2.sysu.edu.cn	完成日期	2024.4.8

1、实验目的

1.并行矩阵乘法

使用Pthreads实现并行矩阵乘法，并通过实验分析其性能。

输入：m,n,k三个整数，每个整数的取值范围均为[128, 2048]

问题描述：随机生成m×n的矩阵A及n×k的矩阵B，并对这两个矩阵进行矩阵乘法运算，得到矩阵C.

输出：A,B,C三个矩阵，及矩阵计算所消耗的时间t。

要求：1. 使用Pthread创建多线程实现并行矩阵乘法，调整线程数量（1-16）及矩阵规模（128-2048），根据结果分析其并行性能（包括但不限于，时间、效率、可扩展性）。2. 选做：可分析不同数据及任务划分方式的影响。

2. 并行数组求和

使用Pthreads实现并行数组求和，并通过实验分析其性能。

输入：整数n，取值范围为[1M, 128M]

问题描述：随机生成长度为n的整型数组A，计算其元素和$s = \sum_{i=1}^{n} A_i$

输出：数组A，元素和s，及求和计算所消耗的时间t。

要求：1. 使用Pthreads实现并行数组求和，调整线程数量（1-16）及数组规模（1M, 128M），根据结果分析其并行性能（包括但不限于，时间、效率、可扩展性）。2. 选做：可分析不同聚合方式的影响。

2、实验过程和核心代码

执行命令

```
clang++ -std=c++11 -pthread main.cpp -o main.out
```

核心代码

1.并行矩阵乘法

```
    pthread_t threads[MAX_THREADS]; //存储线程的标识符
    ThreadArgs thread_args[MAX_THREADS];//存储每个线程的参数

    for (int i = 0; i < num_threads; ++i) //循环创建多个线程
    {
        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;
        pthread_create(&threads[i], NULL, multiply, (void*)&thread_args[i]);
    }

    for (int i = 0; i < num_threads; ++i) //循环等待每个线程执行完毕
    {
        pthread_join(threads[i], NULL);
    }
```

2.并行数组求和

创建线程

```
    pthread_t threads[NUM_THREADS]; //存储线程的标识符
    ThreadData threadData[NUM_THREADS]; //存储每个线程的参数
    int chunkSize = n / num_threads;
    for (int i = 0; i < num_threads; ++i) //循环创建多个线程
    {
        threadData[i].array = array;
        threadData[i].start = i * chunkSize;
        threadData[i].end = (i == num_threads - 1) ? n : (i + 1) * chunkSize;
        pthread_create(&threads[i], NULL, sumArray, (void*)&threadData[i]);
    }
```

连接线程并计算部分和

```
    long long totalSum = 0;
    for (int i = 0; i < num_threads; ++i) {
        pthread_join(threads[i], NULL);
        totalSum += threadData[i].sum;
    }
```

3、实验结果

1.并行矩阵乘法

线程数	矩阵规模				
	128	256	512	1024	2048
1	0.018121	0.126202	0.879359	6.58175	112.664
2	0.020349	0.147105	0.887256	6.6904	119.91
4	0.020277	0.147617	0.942781	6.80991	133.562
8	0.023345	0.15305	1.14385	8.43453	183.479
16	0.021787	0.168542	1.10103	8.41133	139.031

实验结果分析：

1. 随着矩阵规模的增加，运行时间也随之增加，因为大型矩阵乘法需要更多的计算时间。
2. 在每个矩阵规模下，随着线程数量的增加，运行时间并不总是线性减少，可能是因为线程创建和管理带来了额外的开销，并且矩阵乘法的计算本身可能存在一定的串行性质。
3. 对于较小的矩阵（128x128），增加线程数量并没有显著改善性能，可能是因为矩阵太小，无法充分利用多线程的优势；当矩阵规模增大到2048x2048时，16个线程的性能反而略有上升。

2.并行数组求和

线程数	数组规模				
	1M	4M	16M	64M	128M
1	0.001107	0.004466	0.016947	0.065957	0.131481
2	0.001107	0.004885	0.017118	0.065882	0.133381
4	0.001274	0.004895	0.018004	0.070281	0.141728
8	0.001581	0.004678	0.020014	0.097768	0.161229
16	0.002145	0.005131	0.020618	0.081596	0.163816

实验结果分析：

1. 在每个数组规模下，随着线程数的增加，运行时间并不总是减少。比如，在64M和128M数组规模下，从单线程到双线程的运行时间增加了，可能是由于线程间的通信和同步造成的额外开销，对于小规模问题而言，这种开销可能超过了并行计算的好处。
2. 在小规模问题的情况下，增加线程数并不会明显改善性能，因为问题规模太小，无法充分利用多线程的优势。
3. 在大规模问题情况下，随着线程数的增加，运行时间有所减少。这表明在处理大规模问题时，并行计算可以更有效地利用系统资源，但增加线程数可能会带来一些额外的开销。

4、实验感想

1. 实验中可以观察到，在处理大规模数据时，并行计算能够显著提高计算速度。
2. 合理选择线程数量对于获得最佳性能至关重要，虽然增加线程数量可以提高并行性，但过多的线程可能会增加系统开销，导致性能下降。
3. 矩阵规模对并行性能有着显著影响，对于小规模矩阵，多线程并不一定会带来性能提升，因为线程创建和管理的开销可能会超过并行计算的收益，而对于大规模矩阵，则更容易实现并行计算的性能提升。