中山大学计算机院本科生实验报告

学号	姓名
20319045	刘冠麟

实验一：

目的：

OpenMP实现通用矩阵乘法

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：ABC三个矩阵，及矩阵计算所消耗的时间t

要求：

使用OpenMP多线程实现并行矩阵乘法，设置不同线程数量（1-16）、矩阵规模（128-2048）、调度模式（默认、静态、动态调度），通过实验分析程序的并行性能。

实验过程与核心代码

- openmp实现并行矩阵乘法

```
    void open_mul(){
     #pragma omp parallel for num_threads(Thread_NUM) schedule(static, 1)
     for(int m=0;m<M;m++){
         for(int n=0;n<N;n++){
             for(int k=0;k<K;k++){
                 C[m*K+k]+=A[m*N+n]*B[n*K+k];
             }
         }
     }
}
```

切换不同调度：

```
  #pragma omp parallel for num_threads(Thread_NUM) 默认
  #pragma omp parallel for num_threads(Thread_NUM) schedule(static, 1) 静态
  #pragma omp parallel for num_threads(Thread_NUM) schedule(dynamic, 1) 动态
```

实验结果

编译执行

```
gcc -fopenmp openmp_mul.c -o open_mul
./open_mul 128 128 128 1
```

运行结果

静态:

线程数\规模	128	256	512	1024	2048
1	0.018203	0.14362	1.2936	9.42581	78.62
2	0.019524	0.212628	1.33065	9.86254	80.12
4	0.021525	0.167289	1.2865	9.74325	83.65
8	0.025625	0.166128	1.26595	9.15233	81.56
16	0.026348	0.17852	1.23548	9.05492	81.72

动态:

线程数\规模	128	256	512	1024	2048
1	0.024762	0.15923	1.3102	10.3031	74.29
2	0.014623	0.16035	13069	11.125	78.65
4	0.023575	0.15635	1.4326	10.569	82.67
8	0.026824	0.1608	1.2851	10.435	83.48
16	0.023548	0.17133	1.2566	10.223	76.43

默认:

线程数\规模	128	256	512	1024	2048
1	0.02035	0.1432	1.2742	9.3251	82.99
2	0.01964	0.1523	1.2643	9.2645	78.82
4	0.02443	0.1625	1.2006	9.3541	79.66
8	0.02474	0.1547	1.3083	9.2546	84.62
16	0.02550	0.1723	1.2882	9.5127	83.26

实验感想

- 从线程：

  无论是静态调度还是动态调度，随着线程数量的增加，程序的运行时间并没有显著减少，反而有所增加。这可能是因为：

  1. **线程管理开销**：创建和管理多线程会带来一定的开销。当线程数量增加时，这种开销也会增加，可能会抵消并行计算带来的性能提升。
  2. **资源竞争**：当线程数量增加时，线程之间可能会发生资源竞争（如内存、缓存等）可能导致性能下降。
  3. **负载不均衡**：在静态调度中，循环迭代在并行区域开始之前就已经被分配给各个线程。如果各个迭代的计算量不均等，可能会导致一些线程早于其他线程完成任务，而其他线程还在继续计算，这会导致性能下降。而在动态调度中，虽然可以在一定程度上解决负载不均衡的问题，但是由于需要动态分配迭代，因此会增加一些额外的开销。

- 从调度方式：

  对于较小的问题规模动态调度的运行时间通常比静态调度的运行时间短。这可能是因为动态调度能够更好地处理负载不均衡的问题，即使在问题规模较小的情况下也是如此。

  对于较大的问题规模静态调度的运行时间通常比动态调度的运行时间短，可能是因为在这种情况下，负载不均衡的问题不太明显，而动态调度的额外开销（例如，动态分配迭代）可能会导致性能下降。