

Lab5 - OpenMP并行矩阵乘法

实验要求

使用OpenMp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，每个整数取值范围均为[128, 2048]

问题描述：随机生成 $m \times n$ 的A矩阵及 $n \times k$ 的B矩阵，矩阵相乘得到C矩阵

输出：A, B, C三个矩阵，以及矩阵计算时间 t

要求：

1. 使用OpenMP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），调度方式：默认调度、静态调度、动态调度。根据运行时间分析其并行性能。
2. 选做：对比使用OpenMP和使用Pthread的性能差异

实验过程

一、并行矩阵乘法

1. 实现思路

与上一个Pthread实验类似，将A划分到各个线程与矩阵B进行矩阵相乘，再汇总到矩阵C中。

2. 代码实现

主函数

```
clock_t start = clock(); // 开始计时
# pragma omp parallel num_threads(num_threads)
matrix_multiply(A, B, C, m, n, k, num_threads);
clock_t end = clock(); // 结束计时
```

线程执行

```
void matrix_multiply(double* A, double* B, double* C, int m, int n, int k, int num_threads) {
# pragma omp for schedule(static,1)
for (int B_i = 0; B_i < n; B_i++) {
    for (int A_i = 0; A_i < m; A_i++) {
        C[A_i * n + B_i] = 0;
        for (int A_j = 0; A_j < k; A_j++) {
            C[A_i * n + B_i] += A[A_i * k + A_j] * B[A_j * n + B_i];
        }
    }
}
}
```

3. 运行结果

```
gcc -g -Wall -fopenmp -o opm opmatrix.c
./opm 4
```

```
ehpc@61583b2ed2e2:~/data$ gcc -g -Wall -fopenmp -o opm opmatrix.c
ehpc@61583b2ed2e2:~/data$ ./opm 4
Enter values for m, k, n (128-2048): 128 128 128
Initializing data for matrix multiplication C=A*B for matrix
A(128x128) and matrix B(128x128)

Computations completed.
Top left corner of matrix A:
  76      83      2      28      15      62
  80      49      97      58      45      50
  85      42      10      4      79      39
   9      24      93      46      18      98
  54      37      94      18      21      81
  95      70      54      62      81      20

Top left corner of matrix B:
  35      77      96      32      61      29
  75      2      73      73      74      52
  70      20      70      13      54      47
  33      23      72      8      73      52
  92      44      53      45      42      8
  79      60      27      62      99      47

Top left corner of matrix C:
 3.2545E+05  3.2153E+05  2.9517E+05  3.237E+05  3.5483E+05  3.1423E+05
 3.3455E+05  3.4746E+05  3.4212E+05  3.4477E+05  3.4538E+05  3.2112E+05
 3.2366E+05  3.0937E+05  2.9253E+05  3.268E+05  3.1678E+05  2.8423E+05
 2.8167E+05  2.9358E+05  2.7474E+05  2.7464E+05  2.7252E+05  2.9053E+05
 3.1587E+05  2.9487E+05  3.0553E+05  3.0763E+05  3.3281E+05  2.9937E+05
 3.3655E+05  3.354E+05  3.162E+05  3.3859E+05  3.536E+05  3.1768E+05
Time taken: 0.007204 seconds
```

4. 性能分析

- 记录不同调度方式时间开销,取矩阵规模128, 线程数8为参考

默认调度: 23.25 s

静态调度: (static,1) 35.55s (static,8)24.89s (static,16)24.89s

动态调度: 22.04 s

可见静态调度在size参数较小时会出现耗时长于默认调度的情况, 其余调度方式对比在此处动态调度占优, 但总体耗时相差不大。

- 以动态调度方式, 记录不同线程数量 (1-16) 及矩阵规模 (128-2048) 下的时间开销, 时间单位: 毫秒。

线程数\矩阵规模	128	256	512	1024	2048
1	16.527	221.595	1827.268	20807.161	256867.789
2	17.052	230.584	1990.194	21149.835	270678.155
4	20.012	237.903	2039.067	21578.352	322256.694
8	28.462	273.120	2389.153	22148.1889	385644.818
16	76.978	377.487	3100.155	46479.330	525547.198

Lab3中Pthread的数据情况如下:

线程数\矩阵规模	128	256	512	1024	2048
----------	-----	-----	-----	------	------

线程数\矩阵规模	128	256	512	1024	2048
1	21.616	275.948	2138.426	27905.705	267350.834
2	21.160	270.047	2486.53	29149.927	275.937894
4	34.696	308.223	2384.006	28602.850	322226.751
8	25.558	301.801	2803.819	30186.458	341741.185
16	34.326	313.104	2804.56	33789.139	354035.642

可以看到，在线程数较少的时候，OpenMp的耗时总体较低，性能较优，而随着线程数增加，线程达到8以上时，OpenMP的耗时要明显高于Pthread的耗时，即在线程数较大的时候，Pthread的性能具有明显优势。