中山大學
SUN YAT-SEN UNIVERSITY

并行程序设计与算法实验

Lab5-基于 OpenMP 的并行矩阵乘法

姓 名 李源卿

学 号 22336128

学 院 计算机学院

专 业 计算机科学与技术

2025 年 5 月 4 日

1 实验目的

- 掌握 OpenMP 编程的基本流程
- 掌握常见的 OpenMP 编译指令和运行库函数
- 分析调度方式对多线程程序的影响

2 实验内容

- 使用 OpenMP 实现并行通用矩阵乘法
- 设置线程数量（1-16）、矩阵规模（128-2048）、调度方式
  - 调度方式包括默认调度、静态调度、动态调度
- 根据运行时间，分析程序并行性能
- 选做：根据运行时间，对比使用 OpenMP 实现并行矩阵乘法与使用 Pthreads 实现并行矩阵乘法的性能差异，并讨论分析。

3 实验代码说明

使用 Openmp 来实现并行化很简单，只需要在矩阵乘法的外层 for 循环前加上预编译指令：

```
#pragma omp parallel for (schedule(guided|dynamic|default|static))?
```

即可。

```
case DYNAMIC: // 动态调度
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        double sum = 0.0;
        for (int k = 0; k < N; k++) {
            sum += A[i*N + k] * B[k*N + j];
        }
        C[i*N + j] = sum;
    }
}
break;
```

同样写了一下验证结果的模块：

```
int iserr=0;
for (int i = 0; i < N*N; i++)
{
    if (local_C[i] == C[i])
    {
        continue;
    }
    else
    {
        iserr = 1;
        printf("wrong!\n");
        break;
    }
}
if(!iserr){
    printf("right!\n");
}
```

验证了很多组数据，结果都是 true。

4 实验结果

表 1: 默认调度（时间单位：秒）

线程数	矩阵规模				
	128	256	512	1024	2048
1	0.0030	0.0505	0.4275	23.6886	190.1001
2	0.0016	0.0222	0.2095	11.8194	99.1248
4	0.0009	0.0125	0.0870	6.2486	54.3281
8	0.0011	0.0149	0.1715	6.2011	52.4640
16	0.0012	0.0124	0.1965	7.6892	67.4919

表 2: 默认调度加速比（基准：单线程）

线程数	矩阵规模				
	128	256	512	1024	2048
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.8750	2.2748	2.0406	2.0044	1.9183
4	3.3333	4.0400	4.9138	3.7913	3.4988
8	2.7273	3.3893	2.4927	3.8198	3.6233
16	2.5000	4.0726	2.1756	3.0808	2.8168

表 3: 静态调度

线程数	矩阵规模				
	128	256	512	1024	2048
1	0.0034	0.0498	0.4040	24.9018	192.3365
2	0.0018	0.0204	0.1821	12.1277	100.5978
4	0.0011	0.0118	0.0944	6.1797	51.6758
8	0.0011	0.0080	0.1400	6.3267	52.2564
16	0.0010	0.0128	0.2032	8.3740	69.5890

表 4: 静态调度加速比（基准：单线程）

线程数	矩阵规模				
	128	256	512	1024	2048
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.8889	2.4412	2.2186	2.0539	1.9119
4	3.0909	4.2203	4.2797	4.0300	3.7211
8	3.0909	6.2250	2.8857	3.9363	3.6813
16	3.4000	3.8906	1.9882	2.9739	2.7633

表 5: 动态调度

线程数	矩阵规模				
	128	256	512	1024	2048
1	0.0051	0.0524	0.4221	28.6079	201.0495
2	0.0017	0.0270	0.2100	12.0520	95.7423
4	0.0012	0.0114	0.0886	6.1280	50.7980
8	0.0008	0.0079	0.1002	6.2692	54.8217
16	0.0011	0.0156	0.1106	8.4593	69.4817

表 6: 动态调度加速比（基准：单线程）

线程数	矩阵规模				
	128	256	512	1024	2048
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	3.0000	1.9407	2.0100	2.3736	2.0998
4	4.2500	4.5965	4.7641	4.6685	3.9579
8	6.3750	6.6329	4.2126	4.5633	3.6666
16	4.6364	3.3589	3.8165	3.3818	2.8930

表 7: 同样环境下的 Pthreads 表现

线程数	矩阵规模				
	128	256	512	1024	2048
1	0.0035	0.0505	0.4295	29.1818	194.5260
2	0.0017	0.0228	0.1849	12.0011	99.3836
4	0.0011	0.0157	0.0950	6.2068	51.7878
8	0.0011	0.0084	0.1460	6.2817	52.7319
16	0.0015	0.0098	0.2315	8.1932	67.8994

表 8: Pthreads 加速比（基准：单线程）

线程数	矩阵规模				
	128	256	512	1024	2048
1	1.0000	1.0000	1.0000	1.0000	1.0000
2	2.0588	2.2149	2.3234	2.4316	1.9573
4	3.1818	3.2166	4.5211	4.7019	3.7562
8	3.1818	6.0119	2.9417	4.6456	3.6891
16	2.3333	5.1530	1.8553	3.5619	2.8648

5 实验分析

5.1 程序并行性能分析

- 首先是程序的拓展性，我们可以看到，在线程数 $\leq$ 物理核数且矩阵规模 $\geq$512 时，线程数加倍，时间是几乎会缩短一半的。至于在规模小的时候这个特性不明显，我觉得是执行 openmp 指令的准备工作和收尾工作导致的，比如在计算前需要创建线程等，这些都是需要消耗时间的，当计算时间过短的时候，这部分消耗的时间

就会作为主要部分。至于在线程数 > 物理核数之后，每个线程会轮流上物理核运行，增加了调度成本，所以时间消耗会变多。

- 其次是默认调度，静态调度和动态调度的性能差异。默认调度策略其实就是静态调度，所以理论上二者不应该有明显差异，实验结果也是符合预期的。由于本次矩阵乘法实验的规模都是线程数的整数倍，所以在划分任务的时候并不存在负载不均衡的情况，所以默认调度，静态调度和动态调度的差异不会很大，实验结果也是符合预期的。

5.2 Pthreads 与 Openmp 性能对比

上次做的 Pthreads 的加速比并不理想，大概率是实验环境是我自己的电脑 +WSL2 环境导致的。因为这次我在超算习堂上再次运行我的代码，得到了表 4，它的加速比是符合我的预期的。

然而我们可以看到二者并没有太多差别，这是因为我的 Pthreads 程序是对第一个 for 循环进行切分，而在 openmp 中，我也是对第一个 for 循环使用了 *#program* 语句，二者没有本质上的区别，区别只在于实现方式的不同，但是实现方式的不同并不影响整个计算矩阵乘法模块的性能，所以二者差异不大。