Homework9: Parallel

赵楷越 522031910803

2024 年 4 月 25 日

1 多线程代码

```cpp
//递归计算斐波那契数列
inline int fibonacci(int n, int threadNum) {
    if (n <= 1) {
        return n;
    }
    if (threadNum > 1) {
        int x, y;
        std::thread t1([&] { x = fibonacci(n - 1, threadNum / 2); });
        std::thread t2([&] { y = fibonacci(n - 2, threadNum / 2); });
        t1.join();
        t2.join();
        return x + y;
    } else {
        return fibonacci(n - 1, threadNum) + fibonacci(n - 2, threadNum);
    }
}
```

2 计算的第 40 个斐波那契数

计算的第 40 个斐波那契数：102334155

3 运行结果及测得的耗时比及分析

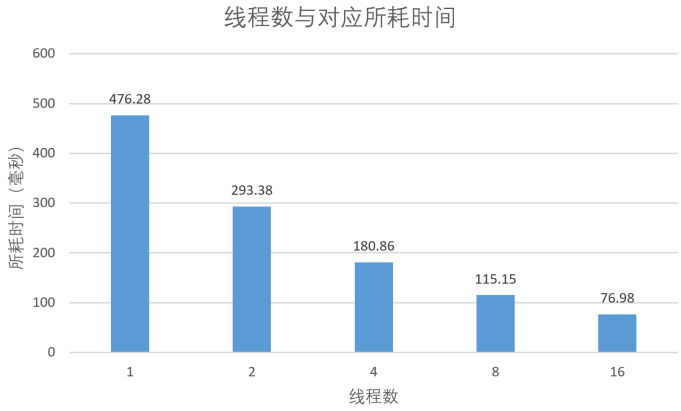
将最大线程数为 1，2，3，8，16 的情况均运行了 100 次，取了运行 100 次后的平均值得到如下运行结果。

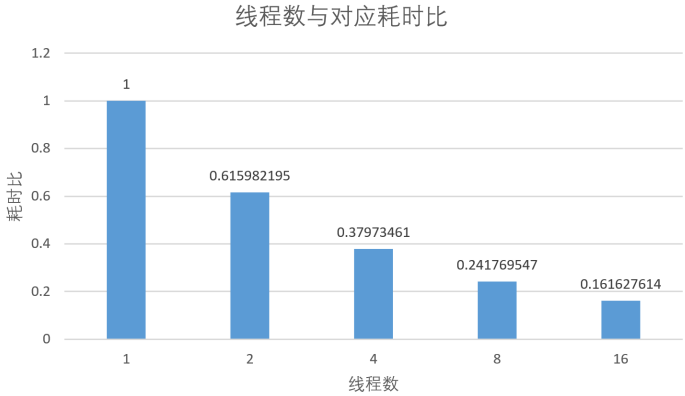
```
PS D:\Horizon\Projects\Ongoing\DS\hw9> g++ -std=c++11 -pthread parallel.cpp -o parallel
PS D:\Horizon\Projects\Ongoing\DS\hw9> ./parallel.exe
Using 1 thread: 102334155time taken: 476.28 milliseconds
Using 2 threads: 102334155time taken: 293.38 milliseconds
Using 4 threads: 102334155time taken: 180.86 milliseconds
Using 8 threads: 102334155time taken: 115.15 milliseconds
Using 16 threads: 102334155time taken: 76.98 milliseconds
```

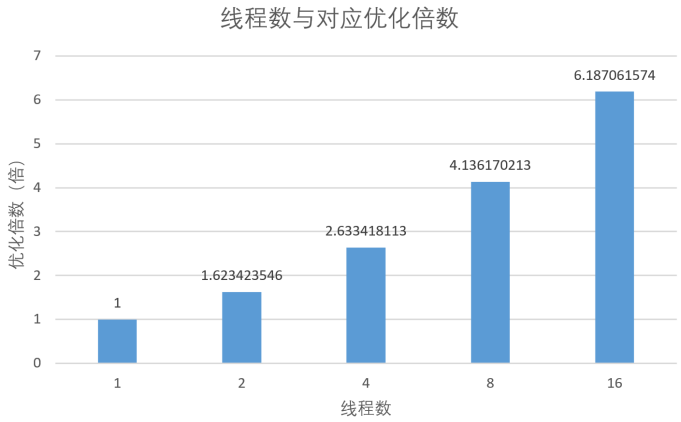
制得具体数据及耗时比情况如下表所示。

线程数	1	2	4	8	16
所耗时间（毫秒）	476.28	293.38	180.86	115.15	76.98
耗时比	1	0.615	0.379	0.241	0.161
优化倍数	1	1.623	2.633	4.136	6.187

绘制如下图像可清晰直观地展现出多线程的优化效果。







对如上结果进行分析：

1. 发现随着线程数的增加，对应的所耗时间确实减少了。说明多线程处理能对递归计算斐波那契数列的程序进行一定的优化。

2. 但是观察对应的耗时比，发现增加对应的线程数之后，并没有相应地提升响应的优化倍数，随着线程数的指数级增长，对应的优化倍数仅仅呈现线性增长的趋势。分析这种原因是因为随着线程数的增加，线程的创建和管理开销也会增加，这可能会导致耗时比没有增加到对应的线程数倍。