522031910747+李若彬+hw9

实验环境

计算的第40个斐波那契数以及不同线程下的加速比

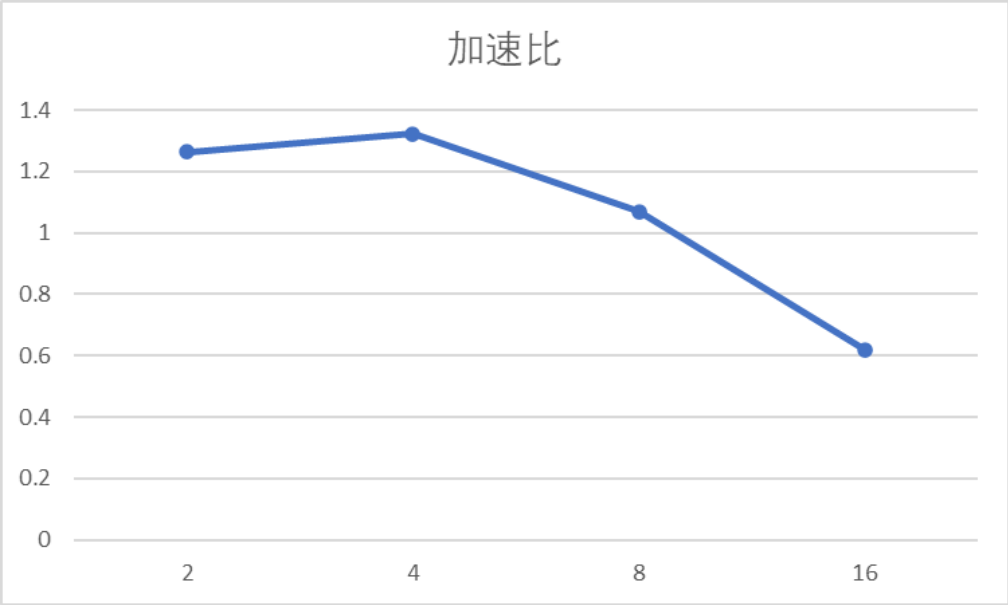
```
lrb@lrb-virtual-machine:/mnt/hgfs/SJTU/grade2-II/ADS/hw/hw9$ cd "/mnt/hgfs/SJTU/grade2-II/ADS/hw/hw9/" && g++ main.cpp -o main &&
"/mnt/hgfs/SJTU/grade2-II/ADS/hw/hw9/"main
Time to calculate fibonacci(40) by 1 thread: 5.49996 s
Result of fibonacci(40): 102334155

Time to calculate fibonacci(40) by 2 threads: 4.21787 s
Speedup of fibonacci(40) by 2 threads compared to 1 thread: 1.26428

Time to calculate fibonacci(40) by 4 threads: 4.03273 s
Speedup of fibonacci(40) by 4 threads compared to 1 thread: 1.32232

Time to calculate fibonacci(40) by 8 threads: 4.98408 s
Speedup of fibonacci(40) by 8 threads compared to 1 thread: 1.06992

Time to calculate fibonacci(40) by 16 threads: 8.5964 s
Speedup of fibonacci(40) by 16 threads compared to 1 thread: 0.620324
```

第40个斐波那契数：`102334155`

加速比：

`threadNum`=2	`threadNum`=4	`threadNum`=8	`threadNum`=16
1 : 1.26428	1 : 1.32232	1 : 1.06992	1 : 0.620324

分析



可以看出随着线程数量的增加，加速比并不是一直增加的。这是因为在这个特定的问题上，多线程并没有带来线性的加速效果。这种现象可能由以下几个原因造成：

1. **线程创建和管理开销**：每创建一个线程都会有一定的开销，包括线程创建、上下文切换等。在这个问题中，线程数量增加可能导致这些开销逐渐显现出来，超过了多线程带来的计算速度提升。
2. **线程间竞争**：多线程同时访问共享资源时会存在竞争，需要使用锁或其他同步机制来保护共享资源。这可能导致线程在某些时候需要等待，影响了整体的计算速度。
3. **任务划分不均匀**：在递归计算斐波那契数列的过程中，不同的子问题可能具有不同的计算复杂度，导致某些线程负载过重，而其他线程处于空闲状态。

为了更好地利用多线程带来的潜在性能提升，可以考虑以下几点：

- **任务划分和调度优化**：尝试将任务划分得更加均匀，避免部分线程负载过重。可以使用任务队列或其他调度算法来优化任务分配。
- **减少线程创建和销毁开销**：可以考虑使用线程池等机制来减少线程的创建和销毁开销，提高线程的重用率。
- **避免过度并行**：在某些情况下，并不是线程越多越好。需要根据具体情况调整线程数量，避免过度并行导致的额外开销。

通过这些优化，可以更好地利用多线程带来的性能提升，提高程序的效率。