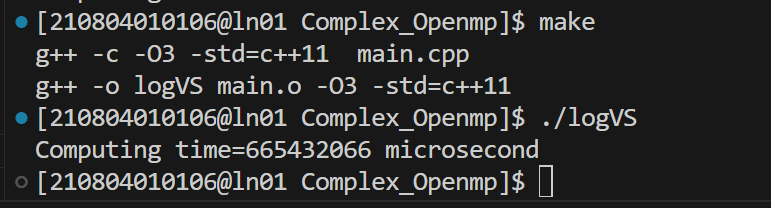
1. **实验准备**

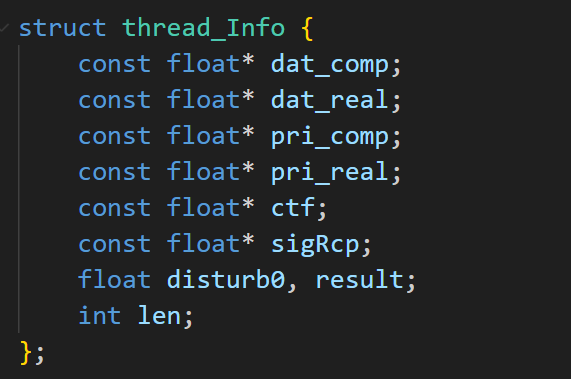
在开始实验之前，如果计划在多核处理器上进行优化实验，要确保系统具有多核CPU，并了解系统的核心数量和硬件拓扑，这将有助于在编写多线程代码时进行合理的线程分配和负载均衡。在编写多线程代码时，要考虑并发问题，如竞态条件、死锁、资源争夺等，学习如何使用互斥锁（mutex）和条件变量（condition variable）来管理并发。在涉及复杂的并行算法和数据结构时，要确保充分了解这些领域的知识，并行编程通常涉及到任务划分、数据共享和通信等问题。最后，当编写多线程代码时，要注意线程安全性和潜在的漏洞。确保您的程序没有内存泄漏、缓冲区溢出或其他安全问题。

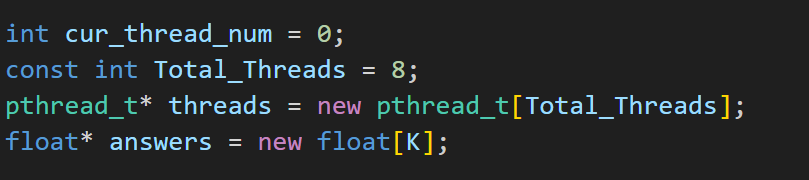
**二、实验步骤**

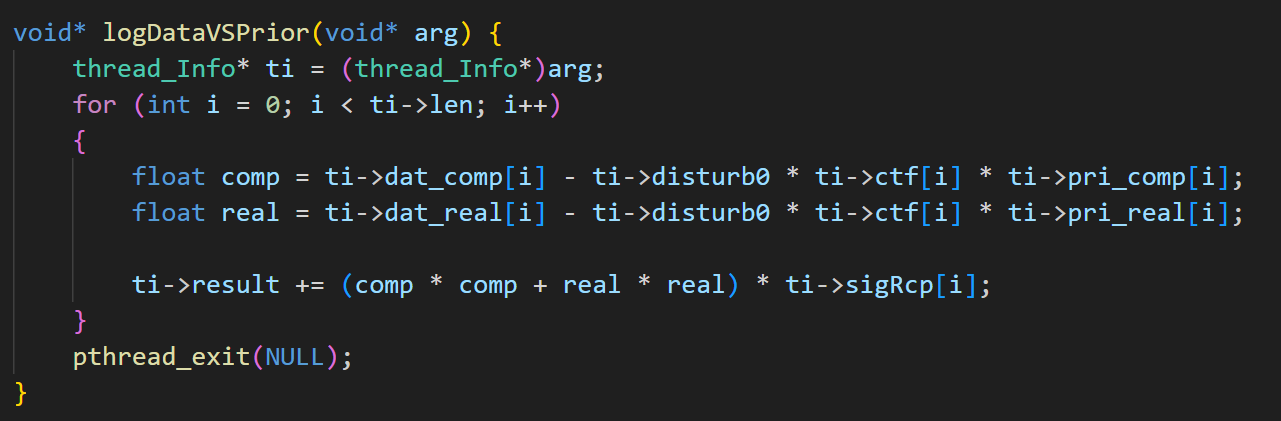
(1).经过精度降低、复数拆分和-O3优化后执行时间为：665432066 us

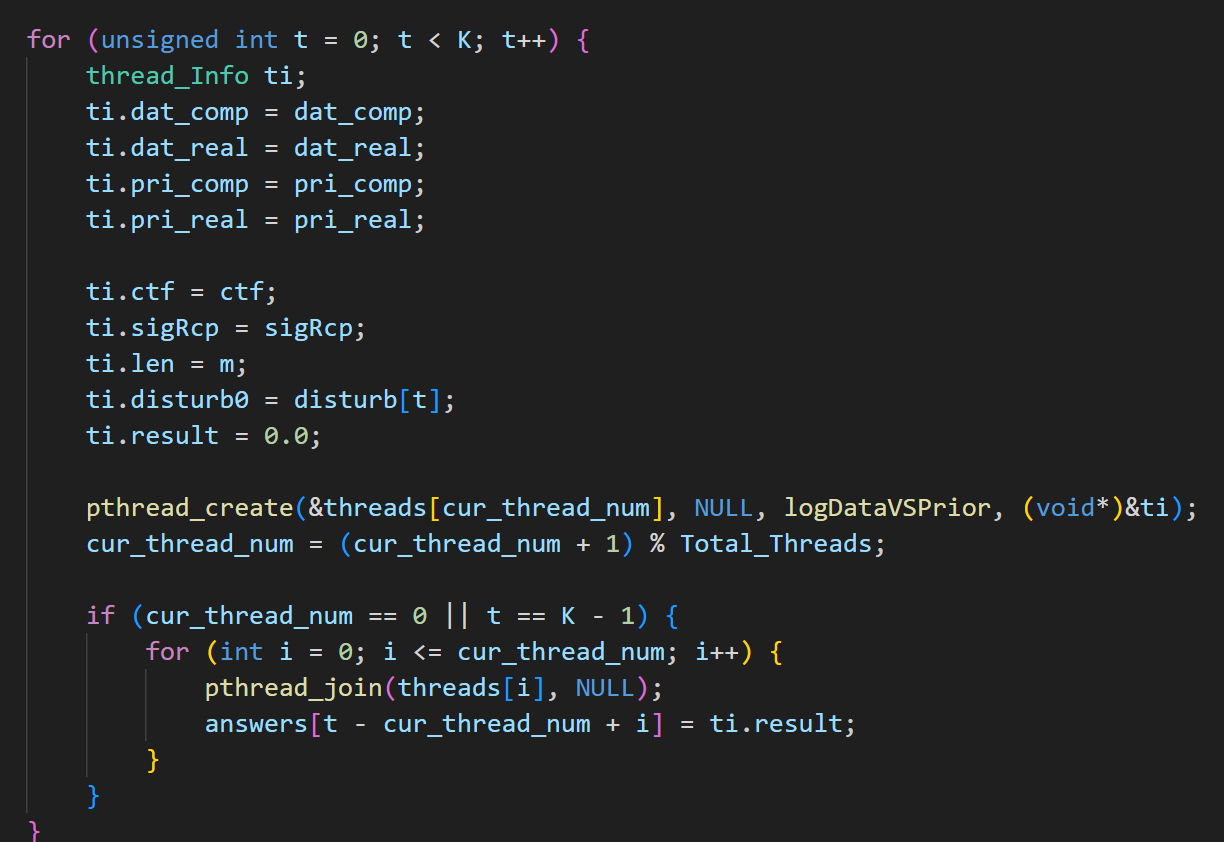


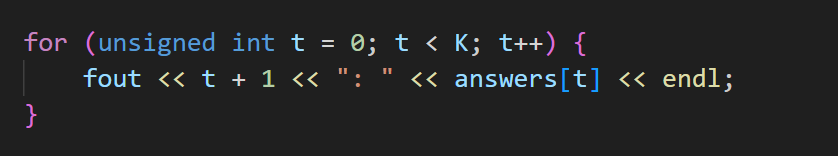
(2). Pthread优化





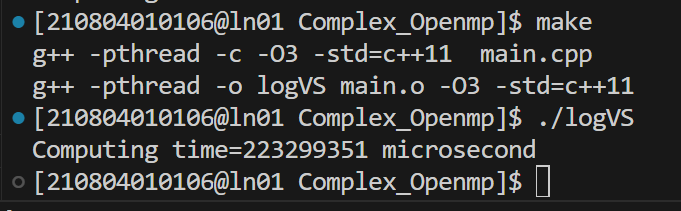






定义结构体thread\_Info来存储每一个线程所要处理的数据（包括各个float数组和操作的数据大小），然后通过pthread\_create根据pthread\_t线程号来创建线程，指定要执行的任务函数，传入void\*型指针强制类型转换为需要的threa\_Info类型，最后在函数中计算出结果，并储存到answers数组中。因为无法控制线程的运行顺序，所以只能存储数据后，等全部运行完毕再输出。注意需要使用pthread\_join来等待所有线程执行完毕后，才可以执行主函数串行代码，否则主函数执行后会强制关闭子线程。

运行时间为



加速比为：665432066 / 223299351 = **2.98**

和openmp一样，虽然线程最大数为8，但没有达到预想中的加速比效果，如①硬件的性能特性也会影响加速比。不同的CPU架构和硬件拓扑对并行性能有不同的影响。某些硬件可能对线程争用更敏感，而某些硬件可能更适合多线程工作负载。②负载不平衡。一些线程可能比其他线程更快完成工作，导致部分线程处于空闲状态。这会导致没有充分利用所有可用的计算资源。

**三、实验总结**

通过本次实验，我们学到了如何通过精度降低、复数拆分、编译器级别的优化以及Pthreads多线程编程来提高程序的性能。总的来说，通过这些优化手段，程序的执行时间从初始的较长时间显著减少，最终达到了快速运行的状态。尽管Pthreads多线程编程提供了并行执行任务的潜力，但在实际应用中，我们仍然遇到了一些挑战，如硬件特性、线程争用、负载不平衡等，这些因素可能限制了加速比的提高。在实验总结中，我们意识到了在多线程编程中需要综合考虑并解决这些挑战，以实现更好的性能优化。此外，我们学到了如何使用互斥锁和条件变量来管理并发，以确保线程安全性。综合来看，多线程编程的性能优化需要细致的分析、不断的测试和经验积累，以更好地发挥多核处理器的潜力。