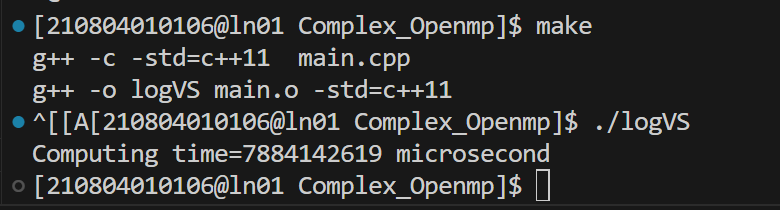
1. **实验准备**

在开始实验之前，需要先学习OpenMP的基本概念，包括指令和指令集，以及如何在代码中使用OpenMP来并行化任务，如：并行区域、共享数据、并行循环和同步等基本概念。除此之外，OpenMP是一种多线程编程模型，因此计算机必须支持多核处理器，这意味计算机至少需要具有双核处理器，以便能够看到运行多个核心，来实现并行效果。可能的话，尝试不同的编译器选项和OpenMP指令集，以优化程序的性能，GCC编译器支持不同的编译器选项来对程序进行自动地优化。除此之外，要确保您的计算机或服务器硬件支持AVX指令集。AVX通常在现代CPU中可用，但在较旧的处理器上可能不受支持。同时呢，还要使用支持AVX指令集的编译器。常见的C/C++编译器（如GCC、Clang、Visual C++）通常支持AVX。确保您的编译器已经配置为启用AVX优化。选择一种支持AVX的编程语言，如C或C++， 因为AVX指令通常通过内联汇编或特定的编译器指令嵌入到源代码中。

1. **实验步骤**

(1). 串行源程序执行时间为：7884142619 us

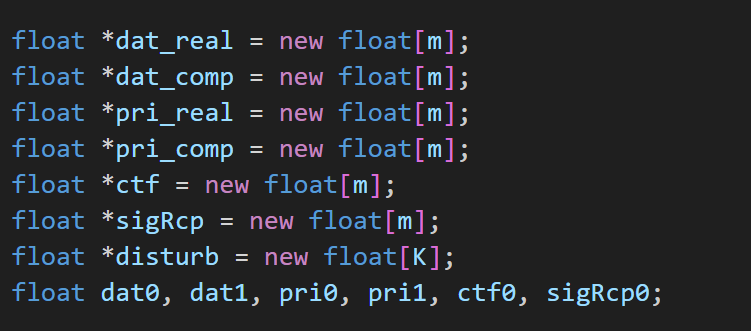


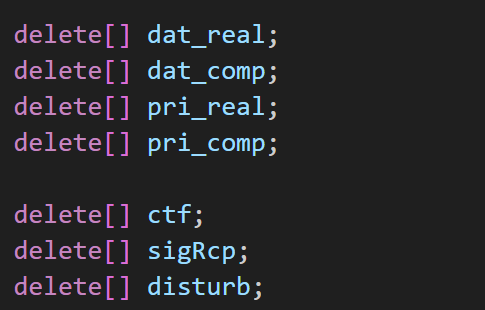
(2). 数据精度优化

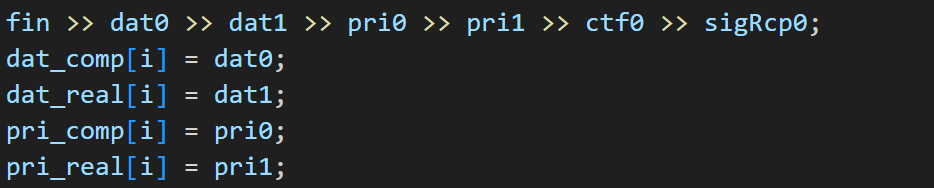
由于题目要求

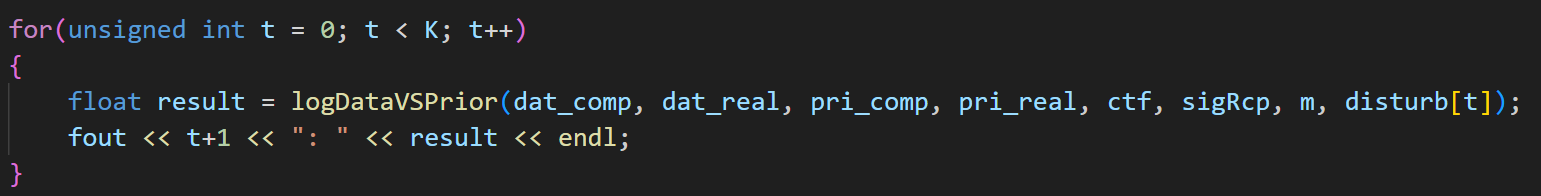
①．因此可以将程序中所有的double类型数据，无论是数组指针还是普通变量，都可以转用float储存，降低数据的精度来换取更小的存储空间和运行速度。

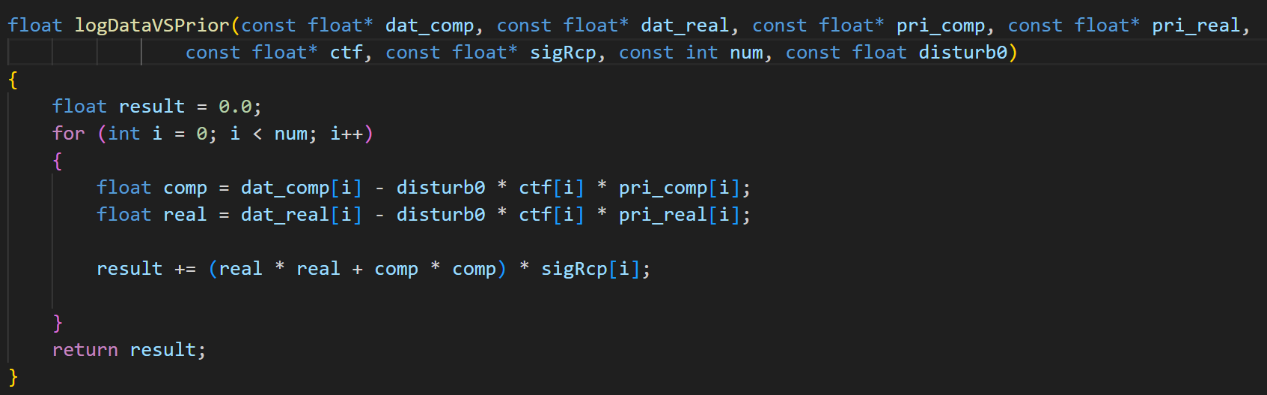
②．同时由于复数的操作会带来更长的运行时间和复杂度，我们可以把dat和pri两个复数数组根据虚部和实部进行拆分，这也是为了后续的AVX512优化铺路（向量化只能操作基本数据类型）。



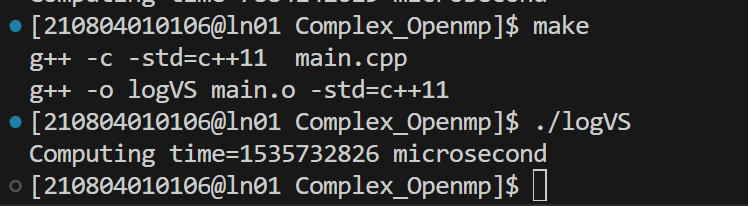








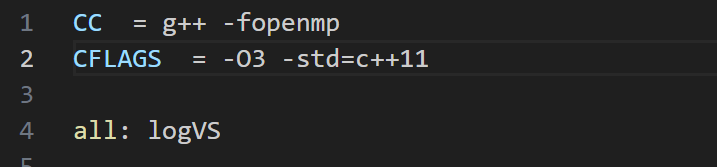
最后运行时间为：



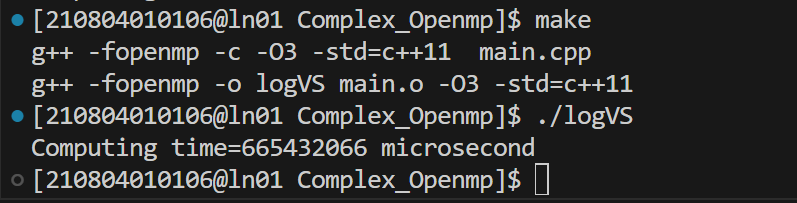
可以得出，加速比为7884142619 / 1535732826 = **5.13**   
64位的double转为32位的float，加速比还是十分可观的。

(3). 编译器指令优化

"O3" 通常表示编译器将进行更高级别的优化，以提高生成的代码的性能。这些优化可以包括内联函数、循环展开、寄存器分配优化、死代码消除和其他一系列性能优化技术。编译器的不同实现可能会在"O3" 优化级别下执行不同的优化。"O3" 通常表示编译器将进行更高级别的优化，以提高生成的代码的性能。这些优化可以包括内联函数、循环展开、寄存器分配优化、死代码消除和其他一系列性能优化技术。编译器的不同实现可能会在"O3" 优化级别下执行不同的优化。



运行时间为：

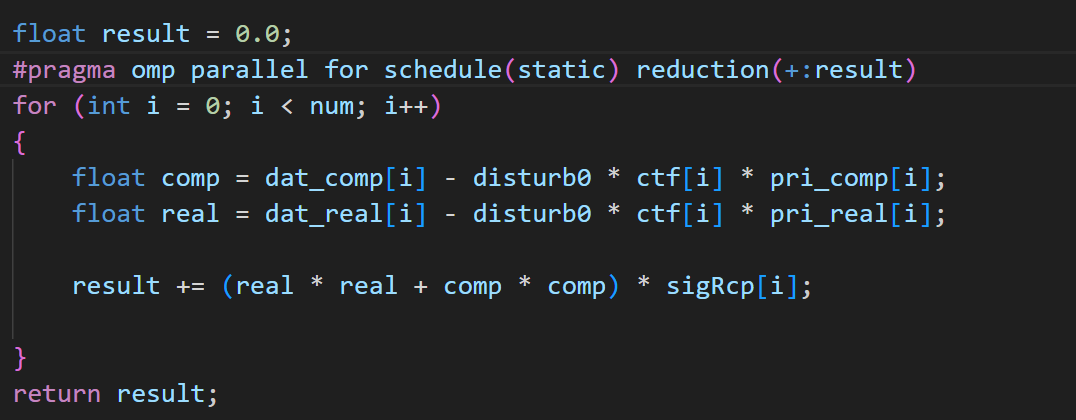


加速比为：1535732826 / 665432066 = **2.31**

可以看出，编译器自动帮我们进行了优化，耗费了更长的时间进行编译，但是在执行时大大加快了对数据的处理速度。

(4). Openmp并行优化

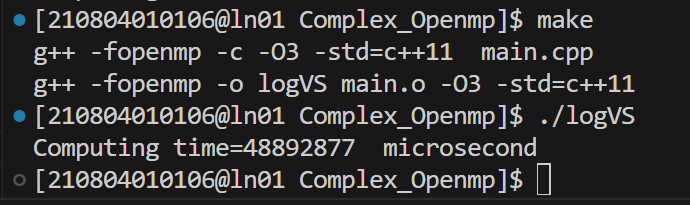
通过omp\_get\_max\_threads( ) ()函数可以得到，在执行openmp优化时，最多可以使用16个线程。



其中parallel for代表创建并行区域的for循环，编译器将自动指派for循环中的任务给线程，分别执行，schedule(static)代表静态调度，在编译时就把任务分配给进程，而不是执行时动态分配，最后reduction(+:result)表示规约操作，在每个循环中都加result。

利用openmp的并行库的精简操作核多核处理器的优势，来使大量数据的执行时间极大缩短。

运行时间为：

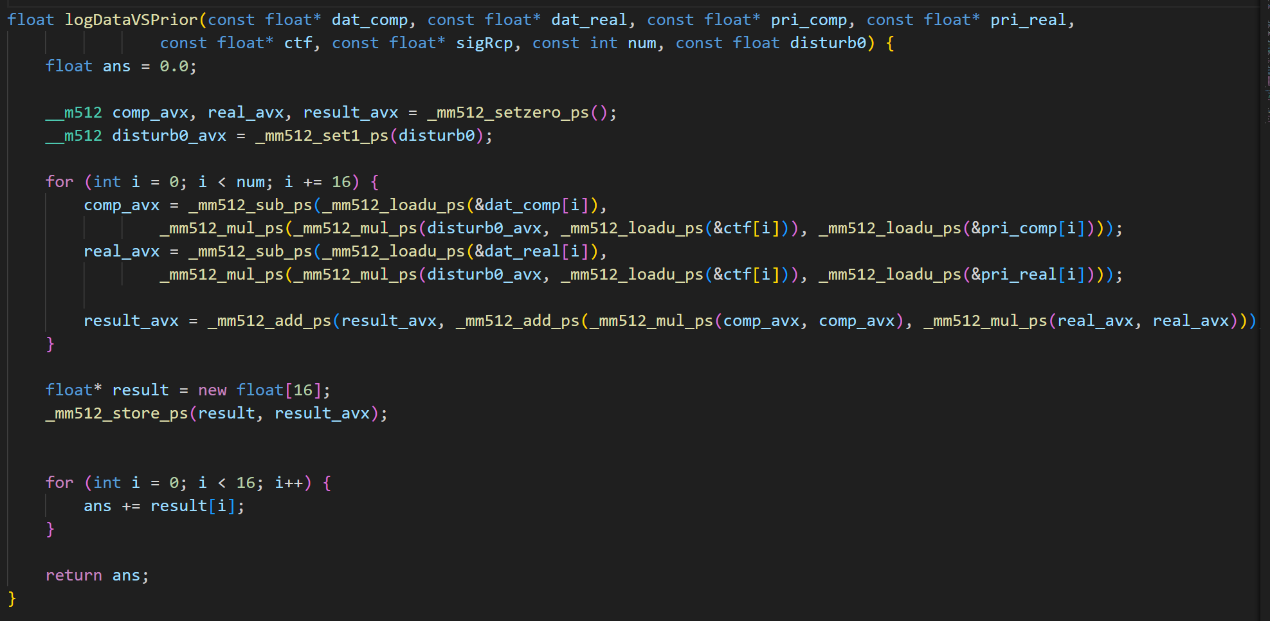


加速比为：665432066 / 48892877 = **13.61**

线程最大可用数可为16，但却没达到预想中的加速比，可能的原因为：①创建和管理线程本身会导致一定的开销。如果问题规模较小或者线程数过多，线程管理开销可能超过了性能提升。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。③**内存带宽限制**：在某些情况下，程序的性能可能受到内存带宽的限制，而不是CPU计算能力。

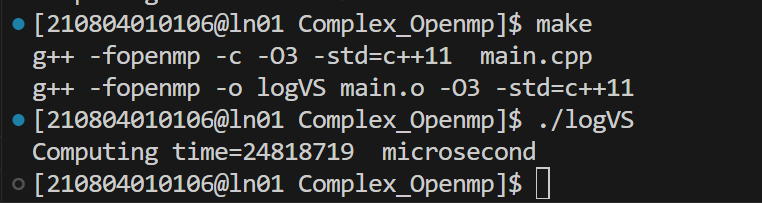
TIPS（在运行openmp时，需要包含omp.h的头文件核在g++或gcc编译器命令后指定-fopenmp，编译器才可以识别并编译openmp库）

(5). AVX512指令集优化



\_mm512表示512位的向量，由于float型数据有32位，因此一个向量可以存16个float数。利用set1(全赋为某个值), setzero(置零), load(从内存中根据首地址取数), mul, sub(操作函数) 等等，可以把float数向量化，然后一次性进行运算处理。它充分利用了SIMD计算，可以在一次指令执行中处理多个数据元素，从而提高了计算效率

运行时间为：



48892877 / 24818719 = **1.97**

1. **实验总结**

**整体的加速比为**：7884142619 / 24818719 = **317.67**

在这次实验中，我们进行了多层次的性能优化，从数据类型的精度降低到并行化以及编译器级别的优化，以及使用AVX512指令集的向量化优化。这一系列步骤充分展示了如何提高程序性能。

首先，我们通过将数据类型从64位的double转换为32位的float，我们牺牲了一些精度，但换取了更小的存储空间和更快的运行速度。这种权衡在某些应用中是合理的，尤其是当高精度不是首要考虑因素时。

其次，通过将复数操作分解为虚部和实部的操作，我们减少了运行时间和复杂性。这种分解在某些情况下可以加速程序，尤其是在处理大量复数数据时。

接下来，我们使用OpenMP进行并行优化，将工作分配给多个线程以充分利用多核处理器。尽管线程最大可用数为16，但由于一些原因（如线程管理开销、负载不平衡、内存带宽限制），最终加速比为13.61，仍然显著提高了程序性能。

此外，编译器级别的优化（使用-O3标志）也有助于提高执行效率。编译器可以自动进行多种优化，包括内联函数、循环展开等，最终加速比为2.90。

最后，通过使用AVX512指令集的向量化优化，我们充分利用了SIMD计算，一次性处理多个数据元素，从而将运行时间进一步缩短，最终加速比为1.97。

综合来看，实验中的不同优化层次相互协同，最终将串行执行的程序性能显著提升。通过这一系列优化步骤，我们学到了如何在性能和精度之间做权衡，如何利用多核处理器的潜力，以及如何使用SIMD指令集来加速计算。这些技能在高性能计算和科学计算中非常有用，有助于提高计算效率和减少计算时间。