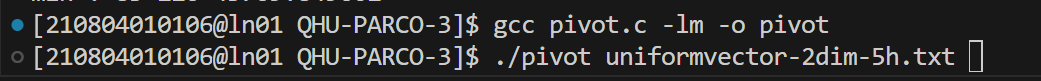
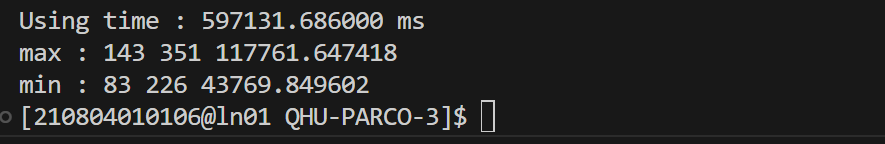
1. **实验准备**

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**。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。利用icc编译器对于英特尔处理器，可以通过较长的编译时间，换取更快的处理时间，利用icpc编译指令以及-qopenmp可以达到一个较好的加速比。除此之外，还需要对题目所要求实现的功能和数据存取有深入的了解，了解程序运行的原理，然后在算法层面或者数据结构层面对程序进行修改优化，如 **消除重复计算、排序优化**等等。

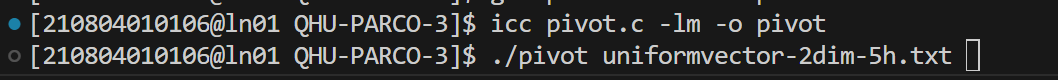
1. **实验步骤**

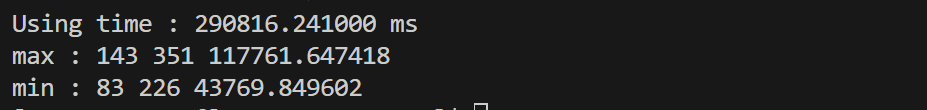
(1). 串行源程序执行时间为：597131.686000 ms





(2). 使用icc编译器，可以在英特尔处理器上产生更好的性能，特别是在进行数值计算和科学计算等计算密集型应用时，同时icc也支持自动向量化和内联汇编支持，大幅度加快程序运行的时间。



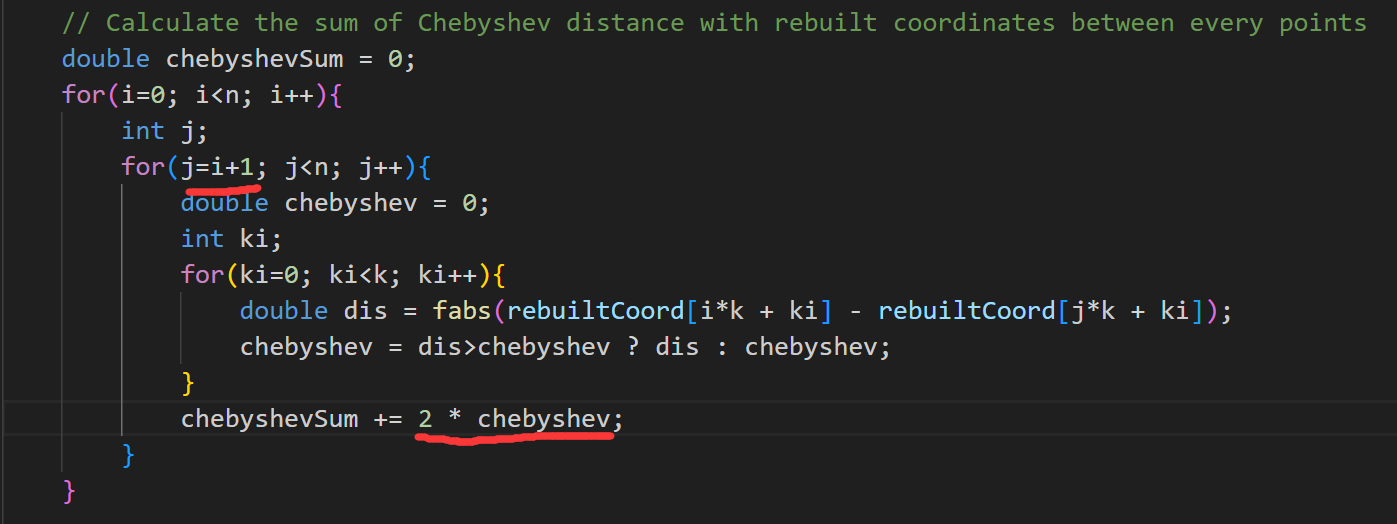


可以得出，加速比为597131.686000/ 290816.241000 = **2.053296**

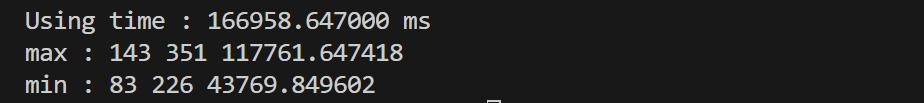
由此得出，icc对于英特尔处理的优化程度达到了很好的加速比，非常可观

(3). 重复计算消除

程序中该部分是计算每个点到其余点的切比雪夫距离之和的计算，其中存在着重复数据。举个例子，这个切比雪夫距离之和包含了：（x1,x4）(x4,x1) 的切比雪夫距离，所以我们只需要计算该矩阵的上三角部分，将结果乘以二即可。



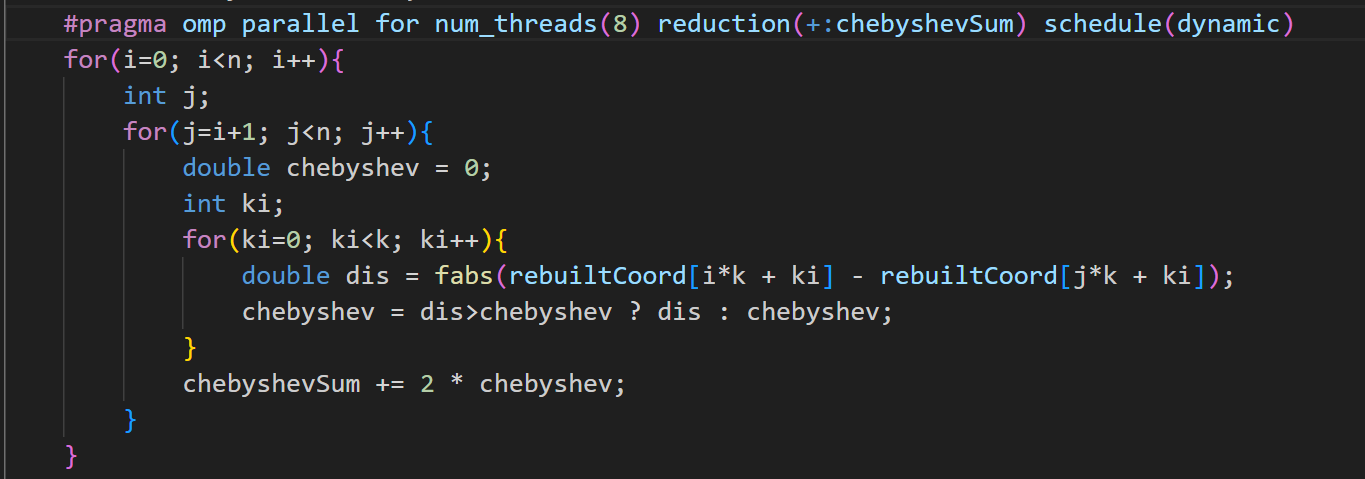
运行时间为：



可以得出，加速比为290816.241000 / 166958.647000 = **1.741846**

(3). Openmp优化

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。

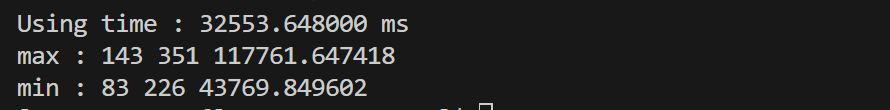


利用#pragma omp parallel 创建并行for循环，以及reduction执行规约操作，将各个线程计算得到的 chebyshevSum 变量的局部结果相加，以获得最终的总和。因为多个线程会并行更新 chebyshevSum，而 reduction 确保了线程之间的正确结果聚合。

经过一些尝试，线程数为8的时候加速比是最好的；同时运行时间，不指定分配策略 > static > dynamic。因此，openmp的最佳策略为

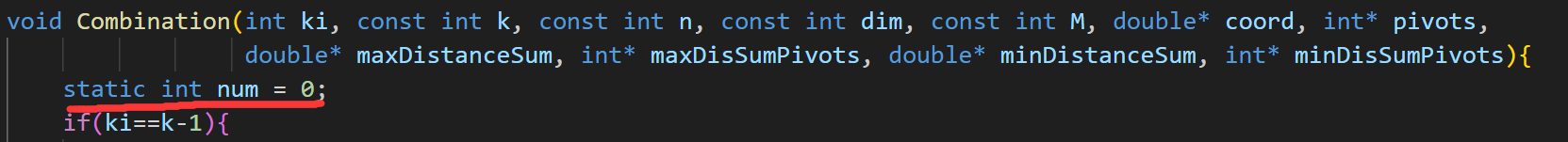
#pragma omp parallel for num\_threads(8) reduction(+:chebyshevSum) schedule(dynamic)

运行时间为：

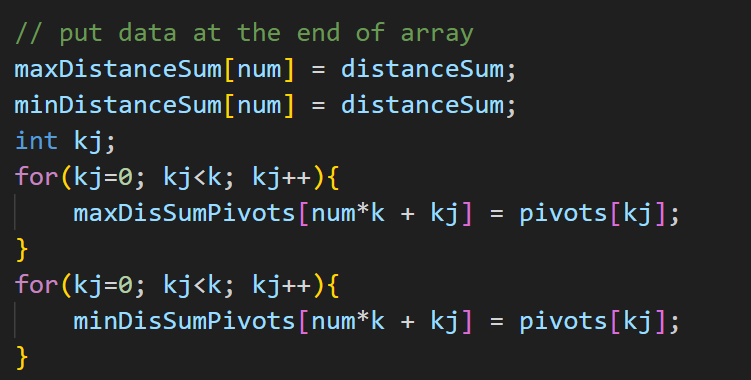


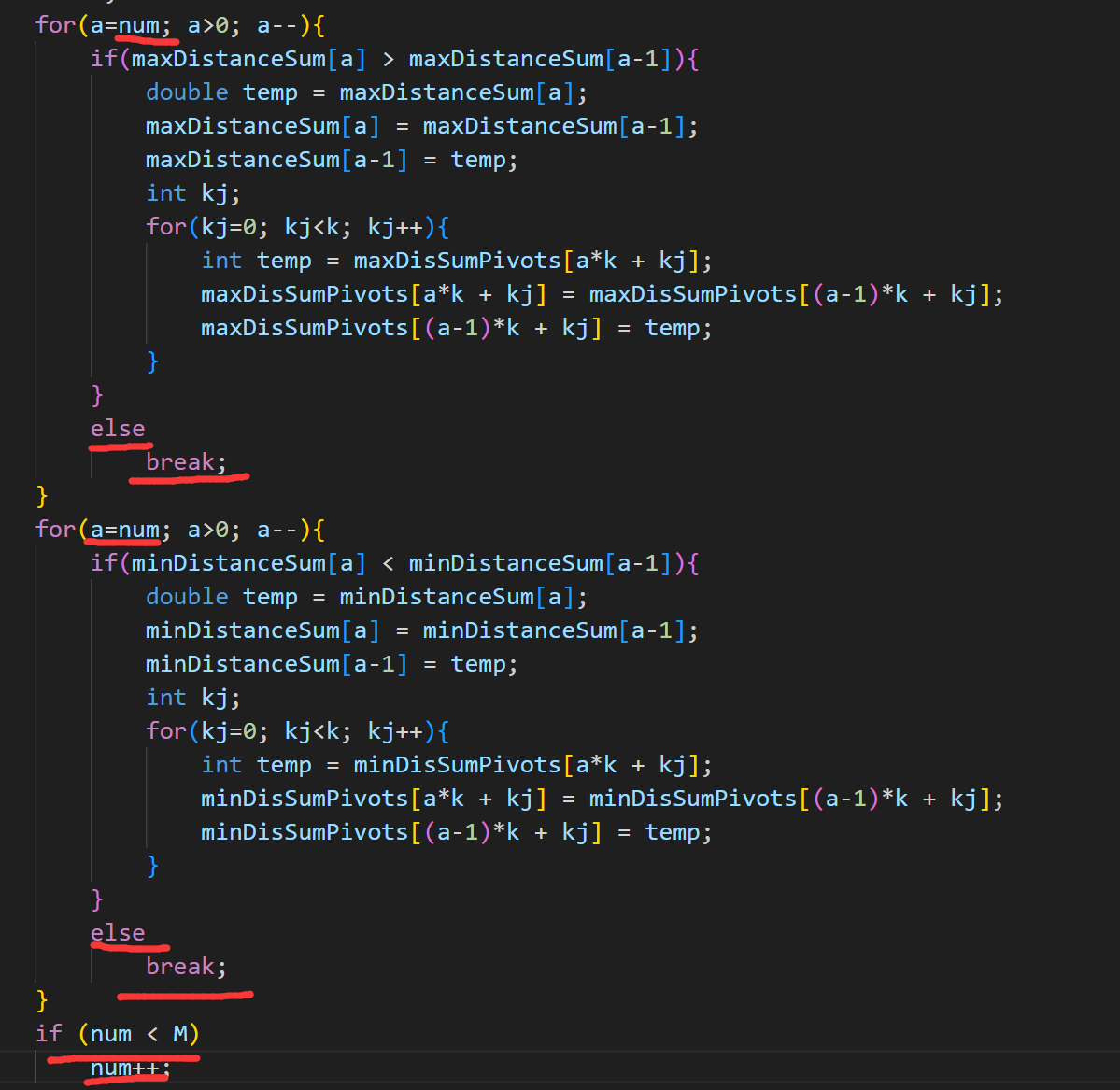
加速比为：166958.647000 / 32553.648000 = **5.128723**

(4). 排序优化



num用于记录已有多少个pivot被存储下来





不需要把数据存放到M下标了，因为从num+1到M都是没有数据存放的，直接从num到1进行if条件判断，swap sum值和pivot数组，这样子可以减少if的判断次数，减少无用的数据处理时间。



加速比为：

32553.648000 / 9968.663000 = **3.265598**

可以看出，优化的加速比非常可观，猜测原因可能为该数据集dfs所获得的所有组合数较小，因此消除排序的多余时间对于整体的加速比非常惊人，若换成k=5的数据集，则可能优化效果会有所降低。

1. **实验总结**

**整体的加速比为**：597131.686000 / 9968.663000 = **59.900880**

这次实验旨在优化一个计算密集型应用，通过使用OpenMP并行化任务、重复计算的消除、使用ICC编译器以及排序优化，以提高程序的性能。实验总结如下：

在实验准备阶段，我们首先学习了OpenMP的基本概念，包括并行区域、共享数据、并行循环和同步等基本概念。我们还尝试了不同的编译器选项和OpenMP指令集来优化程序的性能，特别是在英特尔处理器上使用ICC编译器，这显著提高了程序的性能。

首先，通过ICC编译器的使用，我们获得了大约2倍的性能提升，这表明优化编译器对于处理计算密集型任务非常有效。

其次，我们通过消除重复计算，仅计算距离矩阵的上三角部分，以减少计算量，获得了约1.74倍的性能提升。

然后，我们利用OpenMP并行化库，使用多核处理器的并行计算能力，获得了约5.13倍的性能提升。最佳线程数和调度策略是通过实验确定的，这进一步加速了程序的执行。

最后，通过排序优化，减少不必要的if条件判断次数，我们获得了约3.27倍的性能提升。这对于数据集中组合数较小的情况下表现尤为出色。

总体来说，整个优化过程实现了约59.90倍的性能提升，证明了优化技术的有效性。然而，需要指出的是，不同的数据集和应用场景可能需要不同的优化策略，因此在实际应用中需要根据具体情况进行调整。此实验为优化计算密集型应用提供了有益的经验和指导。