**并行程序设计大作业报告**

范洪宇 软博18班 2018312541

1. **选题介绍**

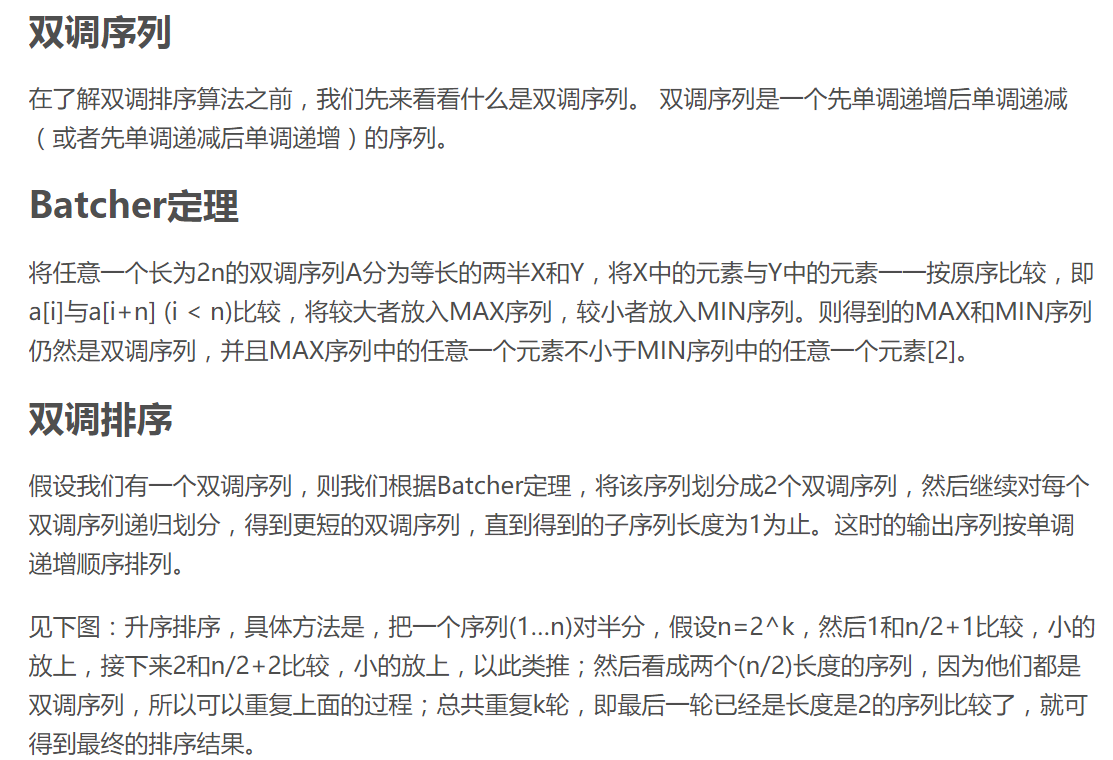
* **分段双调排序算法**

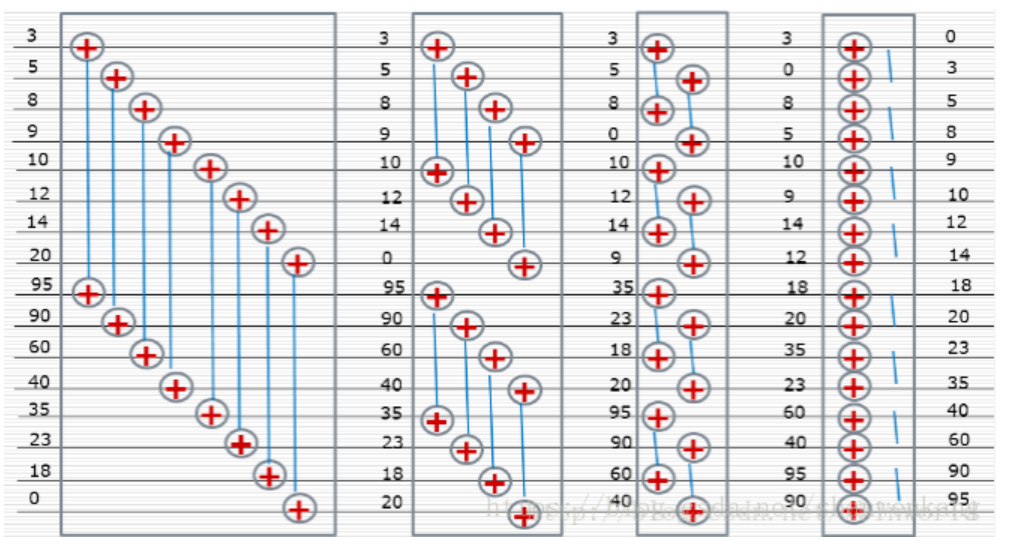
分段双调排序算法是一种非常适合并行化的排序算法，算法即给出分成m段的n个浮点数，输入的数据按照段号有序，但是每个段内部是无序的。经过分段双调排序算法处理后，每个段中数据按照递增或者递减的顺序排序。在单线程情况下，算法的时间复杂度是，并行处理后可以大大加速。

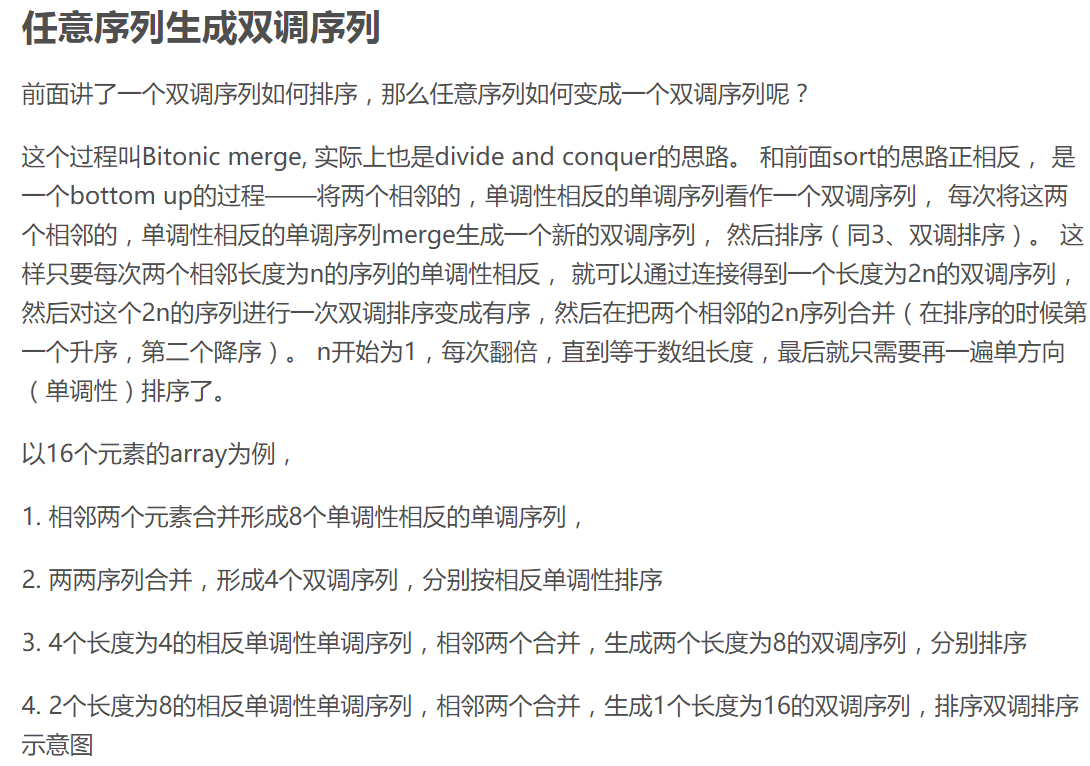
因为分段双调排序段间不影响的特点，非常适合并行化，常被并行化后用在网络拓扑排序中。除了并行化，大作业中还有一些创新点，除了输入数据申请空间进行存储，在排序的过程中算法并未申请任何额外的内存空间，仅借助已有的内存空间进行排序，从而能够保证空间安全。

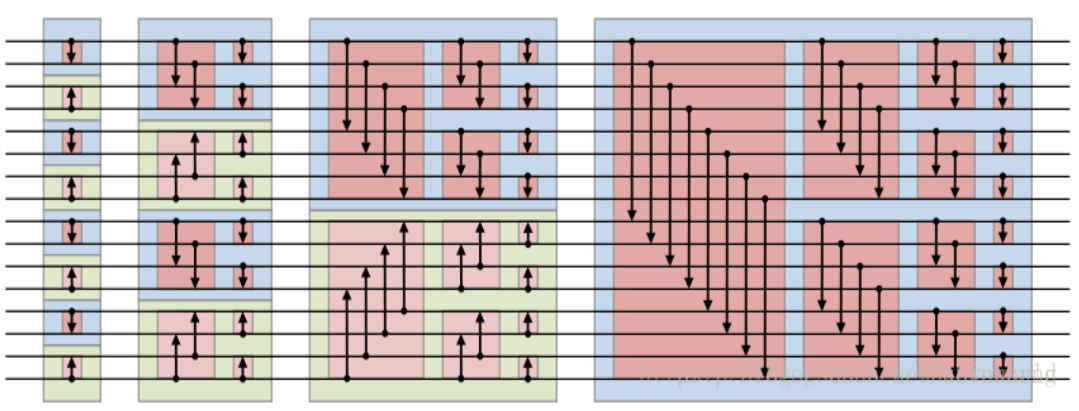
* **双调排序算法**

分段双调排序每个段之间没有数据关联，排序算法执行过程中，每个段都是使用的双调排序算法，下面会对双调排序算法进行一个简单的介绍，资料来源(https://blog.csdn.net/shanwenkang/article/details/82811130)：

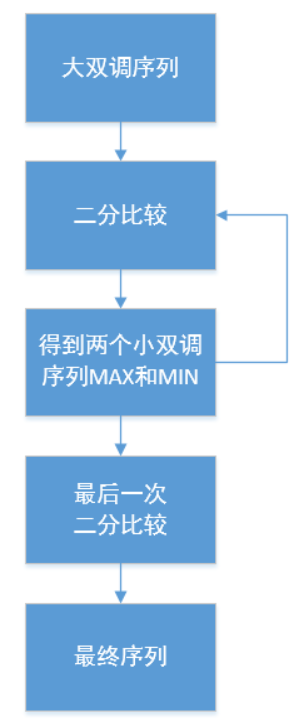
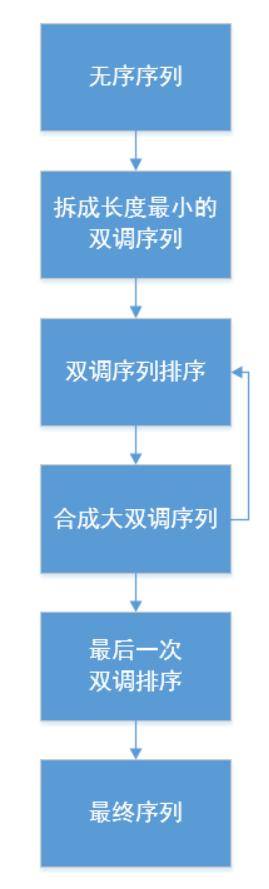








算法实现的核心流程如下左侧的流程图所示，双调序列排序每一次处理双调序列的过程如下图右图所示：



* **并行化选择**

本作业选择使用C++语言实现，使用的多线程库函数为Pthread库，为了最后代码检查简单，所有的代码实现在一个文件中，支持g++的操作系统都可以编译运行。

1. **算法实现环境以及实现方案**

* **运行平台及环境**

实验测试的台式机环境是Ubuntu 16.04 LTS，CPU配置为Inter-i7-7700，4核八线程，16GB内存，240GB SSD。

实验测试使用的服务器环境是Ubuntu 16.04 LTS，CPU配置为Inter-E5-2620，12核24线程，32GB内存，2TB HD。

实验基本上是在命令行里面运行的，调试环境使用到了VS code。实验中使用g++编译即可，无需其他依赖。此外因为台式机和服务器上面配置不同，不同机器上实验数据不具备可比性，我们只把同一台机器上获取的结果进行对比分析。

* **实验编译以及运行**

所有代码都在segBitonicSort.cpp文件中，只需要在命令行运行



就可以生成可执行文件。

Windows下执行 sort.exe

Linux类系统下执行 ./sort

即可，输入输出测例已经在代码内部写好了

* **实现方案**

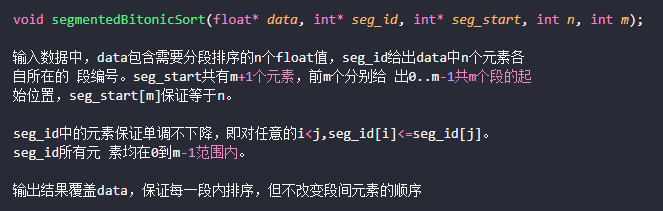
算法的原理和简单的例子已经在第一章介绍双调排序中给出了，这里不再重复介绍，仅介绍一些实现的技术细节和流程。

体现并行排序算法的提升需要一定规模的数据，所以首先是数据生成问题。

算法中实现了生成数据的部分，生成的数据作为排序算法的输入。生成数据可以指定生成数据的个数以及分段的段数，比如设置生成1千万个数据，分成10个段。生成的数据会保存在data.txt文件中，其中包括1千万个数据以及每个数据所在的段的信息。

接下来data.txt会作为排序算法的输入，因为体现并行的优势，所以进行了单线程和多线程情况的对比，最后会生成三个输出文件，分别为output\_single.txt，output\_mt.txt以及result.txt。output\_single.txt文件是在单线程情况下排序后的输出文件，output\_mt.txt文件中是在多线程线程下排序后的输出文件，这两个文件的内容是完全一样的。result.txt中存放了两种不同情况下运行时间以及耗时对比。

程序的核心接口描述如下：



程序的核心并行单元为Void\* bitonicSort(Void\* para)，其中封装了双调排序的算法实现，给定无序数组段的起始和结束位置，算法运行后能够给出排序后的数组。大作业的创新点体现在，排序算法会利用输入数据的数组空间进行排序，在整个排序的过程中不会动态申请任何的内存空间，防止不安全的访问，能够在并行网络拓扑环境中限制空间申请。

算法中同时进行了单线程排序以及多线程排序，线程的数量可以提前设置，如果线程设置数量多于段的数量，则线程数量即为段的数量，在运行过程中保证线程数量不会超过CPU支持的线程数量。

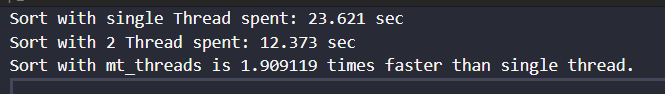
算法中还提供了格式化输出和时间监控的部分，能够将单线程和多线程情况下的排序结果输出到文件，并且对不同过程的时间进行记录和对比。

1. **实验结果对比分析**

* **1千万数据2数据段 && 单线程VS双线程**

第一组对比试验的配置如下，在自己的台式机上面，随机生成1千万个数值在0-11234567之间的浮点数，并且随机分成2个数据段，然后在单线程与双线程的情况下进行实验。

已知在单线程情况下，算法的时间复杂度为，1千万个数据分成2段。假设CPU处理1千万次的时间约为1s，那么估算可得排序的时间约为40s，经过试验对比，得到如下结果：

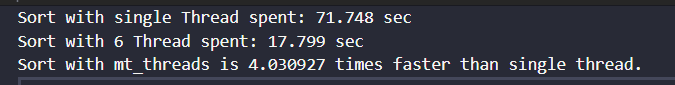


由图片可得，单线程情况下处理1千万个数据，2个数据段的排序花费了23.6s左右的时间，双线程并行情况下花费了12.3s左右的时间，基本在1.9倍的提升。分析结果，CPU性能越来越好，单线程情况下CPU每1s可以处理的数据量应该超过1千万次，所以单线程情况下为23.6s，而不是40s也是合理的实验结果，时间在数量级上满足逻辑即可。此外，数据量占用量并不大，CPU和内存比较宽裕，能够比较靠近理论上的提升。

* **3千万数据6数据段 && 单线程VS六线程**

第二组对比试验的配置如下，在自己的台式机上面，随机生成3千万个数值在0-11234567之间的浮点数，并且随机分成6个数据段，因为CPU是4核八线程，在单线程与六线程的情况下进行实验。

和第一组对比试验一样，检测测试了一下CPU处理1千万次数据所花费的时间，之前估算在1s左右，经过测试，只需要0.6s左右，因为之前衡量算法的时候，基本上默认是1千万次运算CPU时间约为1s，不过已经是几年前了，CPU处理的性能变得更强了，所以耗费的时间更少了，是合理的。所以根据双调排序的时间复杂度，可以估算出理论上单线程，3千万数据分成6个数据段下排序的时间约为80.1s。经过实验对比，得到如下的结果：

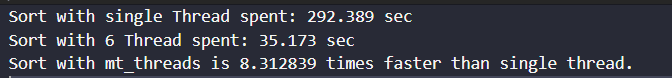


由图片可得，在单线程情况下，处理3千万数据，6个数据段需要消耗71.7s，但是在六个线程情况下需要消耗17.8s，速度提升了4.03倍。速度的提升比起理论上6倍速度还是有一定的差距，因为CPU竞争，内存访问，文件读写等都可能会存在等待和调度，并且不同数据排序本身耗时不同，时间复杂度仅代表一个普遍情况。综上所述，时间是合理的。

* **1亿数据10数据段 && 单线程VS十线程**

第三组对比试验的配置如下，第三组实验是在服务器上跑的，随机生成1亿个数值在0-11234567之间的浮点数，并且随机分成10个数据段，然后在单线程与十线程的情况下进行实验。

首先估算一下时间复杂度，CPU处理1千万次运算时间约为0.6s。目前的配置是1亿个数据，分成10个数据段，那么时间复杂度可以计算为，经过计算可得约为294s。那么经过对比试验，可以得到如下的结果：

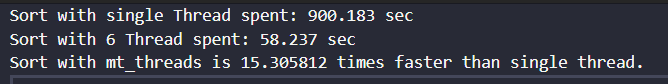


由图可得，1亿个数据分成10个数据段，在单线程情况下排序时间为292.4s，跟理论上排序时间基本相同，在十线程并行情况下排序花费了35.1s的时间，加速比为8.31倍。分析结果，与预期结果基本相符，提升非常明显，但是无法达到理论上10倍的速度，因为数据段之间情况不同，先结束的线程需要等待后续线程，并且对于存储，内存等访问也有竞争和调度，有一定的影响。总而言之，实验结果是符合预期的，也体现出了分段双调排序并行的可行性与高效性。

* **3亿数据20数据段 && 单线程VS二十线程**

第四组对比试验的配置如下，第四组实验是在服务器上跑的，随机生成3亿个数值在0-11234567之间的浮点数，并且随机分成20个数据段，然后在单线程与二十线程的情况下进行实验。

首先估算一下时间复杂度，目前的配置是3亿数据，分成2个数据段，在单线程情况下排序的理论时间计算约为933s。经过对比试验，得到结果为：

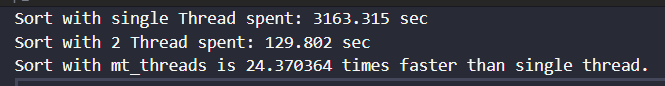


由上图可得，在单线程情况下，排序时间为900.2s，符合预期，在二十个线程的情况下排序花费的时间为58.3s，加速比约为15倍。实验结果基本符合预期的目标。

* **10亿数据50数据段 && 单线程VS二十二线程**

第五组对比试验的配置如下，第五组实验是在服务器上跑的，随机生成10亿个数值在0-11234567之间的浮点数，并且随机分成50个数据段，然后在单线程与二十线程的情况下进行实验。相比于前四组实验，这组实验比较特殊，因为首先数据量非常大，而且服务器支持的核心数是小于段落数的，所以即使多线程，后续段落也需要等待才能分配到资源进行排序。本组实验作为前几组实验的对比试验进行观察与总结。

因为数据段的数量多于线程数量，所以并行加速比应该无法按照原始的情况进行衡量。单线程情况下，处理10亿数据，50个数据段，经过估算的理论时间约为3197s。经过对比试验之后，实验结果为：



由上图可得，10亿数据50个数据段在单线程情况下进行排序所花费的时间为3613.3s，在22个线程情况下花费的时间是129.8s。加速比为24.37倍。因为数据段数量是多于允许使用的线程的数量的，所以并行化一开始运行式有很多数据段是需要等待的，等待前面线程释放之后再进行排序。所以加速的比例肯定是无法到达接近50倍的。目前的结果应该也在合理的范围之内。

* **注意事项**

因为每次运行都是动态的随机生成数据文件，1亿个数据文件大小就已经超过1GB了。所以提交的时候不提交生成的数据里，最后提交的代码里面保存了1千万数据，两个数据段。单线程和双线程的对比试验，如果运行测试的话，大约半分钟能够运行完。其余的可以自行修改主函数里面的配置参数来进行修改。

1. **总结和感想**

经过对分段双调排序的实现以及并行化处理，还是学到了并行程序设计实现方面的一些知识和方法，并且自己动手收获很多。

首先是双调排序的时间复杂度为，算不上是性能最优的排序算法，但是其先天的可并行化处理的优势使其经常被使用在并行环境之下，综合性能良好。其次的感受就是使用多个线程，受限于线程调度，资源竞争，数据传输等影响，提升的效率是要小于预期的，这也是可以预见的，而且实验结果也是比较合理的，满足预期。

大作业实现过程中还是遇到了很多的问题，最难的在于算法的实现，其次就是基于创新点，不动态申请任何内存空间，需要对现有的数据空间进行划分，在排序过程中不断地覆盖和重复利用，保证里空间访问的安全性。在这一点的实现上调试花费了比较多的时间。

最后感谢向东老师的授课，以及助教的辛勤付出，通过《并行程序设计》这门课，复习了很多之前学习过的算法，排序，矩阵运算，图论中的一些方法等，而且对于并行程序实现进行了一些了解和动手实践，收获很多。