作业 1

陈岩@NJU

一． 想法

归并排序的过程比较简单，之前上并行计算课时写过 Java 版本的并行的归并排序程序，所以这里实现一个 Go 语言版本的，伪代码如下：

---

threshold：一个线程直接排序的子数组最大长度；

merge(array, i, mid, j)：将分别有序的数组 array[i…mid]和 array[mid+1…j]合并为一个整体有序的数组 array[i…j]；

输入：无序数组 array[1…n]；

输出：有序数组 array[1…n]；

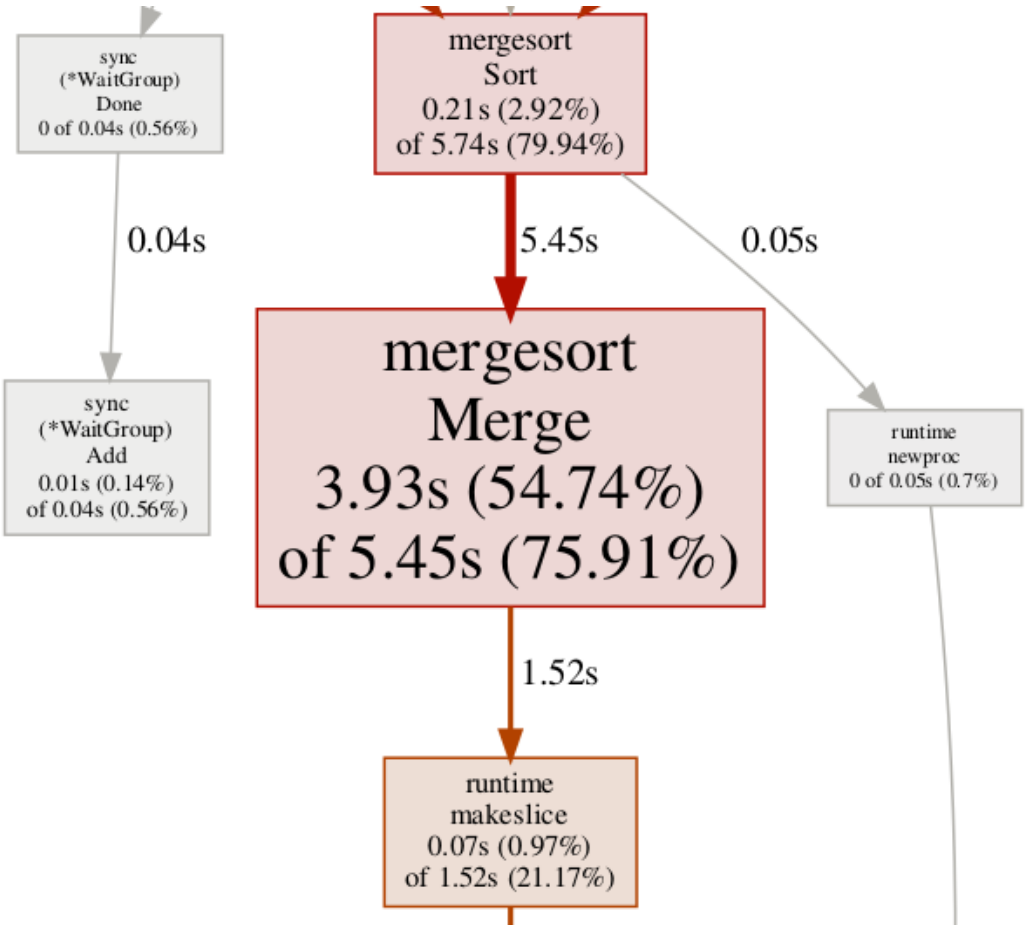
```
Begin
    para_mergesort(array, 1, n, threshold,)
End


procedure para_mergesort(array, i, j, threshold)
Begin
    if (j – i) <= threshold
        call mergesort(array, i, j)
    else
        mid = (i + j) / 2
        start 2 new threads t1 and t2
        t1: para_mergesort(array, i, mid, threshold)
        t2: para_mergesort(array, mid + 1, j, threshold)
        merge(array, i, mid, j)
End
```

这个并行算法的思路非常简单，相比串行算法，只是在需要排列的子数组中元素个数大于某个给定的阈值 threshold 时，启动两个线程来分别处理子数组分为两个部分以后的数组；否则直接调用串行的归并排序处理。threshold 的值可以作为参数进行调节，以减少开启线程的代价超出直接串行处理两个子数组的代价的情况。
其时间复杂度大致满足 T(n)=T(n/2)+n，根据主定理，T(n)=O(n)。

二． 优化

实现的初始版本大致依据上述伪代码实现，并行部分使用 Go 语言的 go 原语创建线程，并使用 WaitGroup 实现线程之间的同步。

单独运行 bench_test.go 中的 BenchmarkMergesort 函数，并生成 cpuprofile 文件，然后通过 go tool pprof 生成 pdf 视图，其中一部分如下：

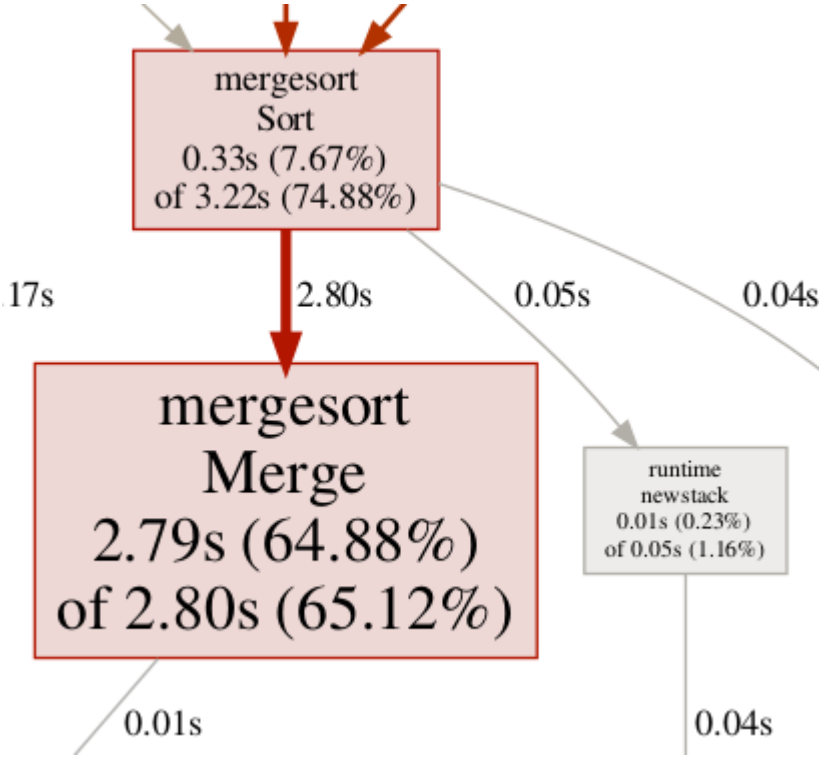


从中可以看出 Merge 过程占了较多的时间，这是意料之中的，因为归并排序主要的排序过程就是 Merge 实现的，且并行化也没有涉及 Merge 的过程，考虑对这一过程进行优化。

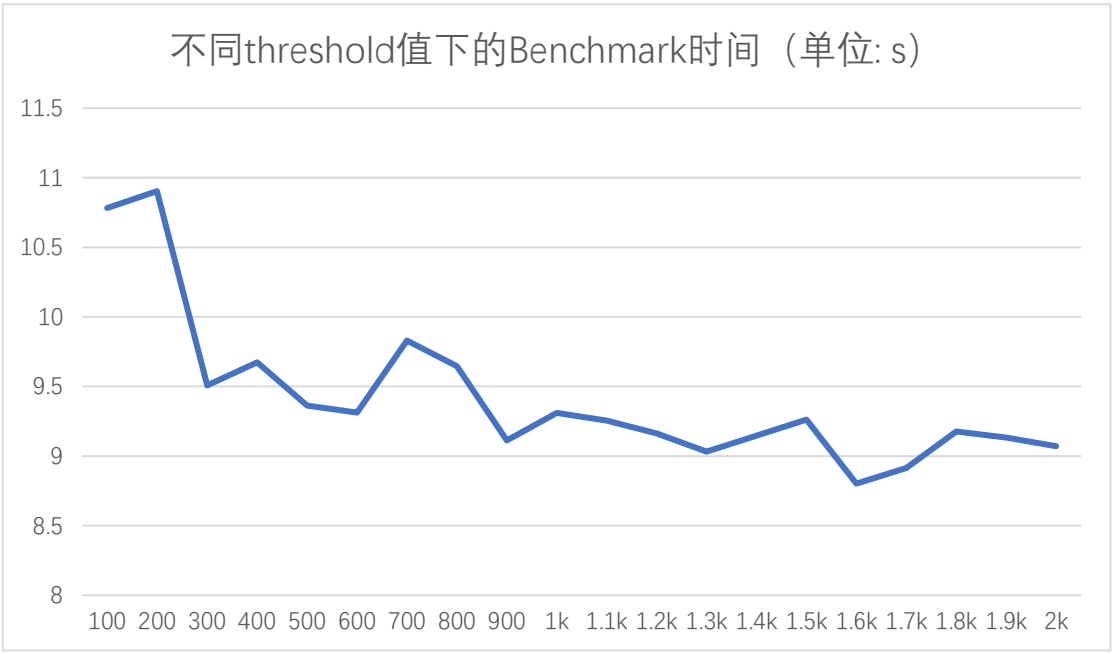
1. 注意到在对子数组 A 和 B 进行 Merge 操作时，由于 A 和 B 已经是有序的，所以如果 A 的最后一个元素小于 B 的第一个元素，则只需要把 A 和 B 前后拼接起来即可，通过这一判断可以在满足条件的情况下省去频繁比较元素大小的花销。
2. 在初始的实现中，Merge 函数会在内部创建一个辅助数组，这样导致了大量的创建数组的花销以及内存的花销（因为创建的数组有大量的重叠）。于是将其修改为在主函数中创建一个和待排序数组相同大小的辅助数组，并将其作为参数传递给排序函数。这样做可以节省大量的时间和空间。
3. 初始实现中 Merge 函数在最后会将辅助数组再复制到原始的数组中。可以采取一个简单的 trick 来省去这一复制过程，即在逐层排序的时候依次交换辅助数组和原始数组的角色，并保证在最后一次归并时是将数组排列好并放入原始的待排序数组即可。为了实现这一点，初始创建辅助数组时需要将原数组的内容复制给辅助数组，并在递归函数进行递归调用时交换原始数组和辅助数组的位置。这样仅以一次数组

复制的代价，节省了大量的数组复制操作。

优化后 cpuprofile 生成的性能图中 Merge 部分如下图所示，可以看出一定的优化效果：



另外，对程序的超参数 threshold 也尝试进行了一定的优化。以运行 5 次 BenchmarkMergesort 的时间为主要指标，对 threshold 在[100, 2000]之间以 100 为间隔进行采样，得到的结果如下：



上述测试结果只是象征性的，并不一定准确，首先测试的范围和精度有待进一步提升，其次每一次的运行结果与测试环境有关，最后每一次生成的随机测试样例都互不相同。尽管如此，还是根据上述结果将 threshold 的值设为 1600。

经过上述优化后运行 benchmark 的结果如下：

```
nju@ubuntu:~/talent-plan/tidb/mergesort$ make bench
go test -bench Benchmark -run xx -count 5 -benchmem
goos: linux
goarch: amd64
pkg: pingcap/talentplan/tidb/mergesort
BenchmarkMergeSort-4                   1        1287190767 ns/op        137046768 B/op     27681 allocs/op
BenchmarkMergeSort-4                   1        1254554431 ns/op        134942576 B/op     22204 allocs/op
BenchmarkMergeSort-4                   1        1268218315 ns/op        134776688 B/op     20107 allocs/op
BenchmarkMergeSort-4                   1        1224207267 ns/op        134921616 B/op     21923 allocs/op
BenchmarkMergeSort-4                   1        1231904354 ns/op        134943184 B/op     22202 allocs/op
BenchmarkNormalSort-4                  1        4696703951 ns/op              64 B/op          2 allocs/op
BenchmarkNormalSort-4                  1        4690031104 ns/op              64 B/op          2 allocs/op
BenchmarkNormalSort-4                  1        4701706927 ns/op              64 B/op          2 allocs/op
BenchmarkNormalSort-4                  1        4717037067 ns/op              64 B/op          2 allocs/op
BenchmarkNormalSort-4                  1        4704698516 ns/op              64 B/op          2 allocs/op
PASS
ok      pingcap/talentplan/tidb/mergesort       35.230s
```

可以看出所实现的归并排序要快于 sort.Slice()函数。