Algorithm description and analysis

General description

The quick sort algorithm basically repeats two operations:

1. Partition the array into two parts: Elements $\leq$ pivot and Elements $>$ pivot;
2. Recursively apply the quick sort algorithm to both parts of the array, respectively.

Both operations are parallelized. To avoid spawning too much threads on dealing with small base cases, a parameter $G = 1024$ is used to stop spawning threads when input array size is less than G.

I parallel two recursive calls simply by using $task$ directive in openmp.

Partition algorithm

Description

Let's say we would like to partition an array A of length N according to a $pivot$. The partition parts consist of 3 steps:

1. Create two array leq and gt and initialize all elements as 0. Iterate over the original array. If an element is $\leq$ pivot, mark corresponding position of leq as 1. Otherwise, mark corresponding position of gt as 1.
2. Update leq and gt to be their inclusive prefix sum, respectively.
3. Create an array $temp$. Iterate over the original array A. If an element A[i] $\leq$ pivot, lookup leq to find out its index in $temp$, which is leq[i]-1. Otherwise, lookup gt to find out its index in $temp$, which is N-gt[I]. Set $temp$ with the value in A. After the iteration, overwrite A with $temp$.

Below is the pseudocode for this algorithm:

```
#STEP1
for i=0 to N-1 do:
  if A[i] <= pivot: leq[i] = 1
  else: gt[i] = 1

#STEP2
leq = inclusive-scan(leq)
gt = inclusize-scan(gt)

#STEP3
for i=0 to N-1 do:
  if A[i] <= pivot: tmp[leq[i]-1] = A[i]
```

```
    else: tmp[N-gt[i]] = A[i]
copy tmp to A
```

All 3 steps are paralleled by #pragma omp parallel for.

Auxiliary storage

Obviously, the parallel partition algorithm above requires auxiliary storage. All 3 steps requires using leq and gt array of size N. Both STEP2 and STEP3 requires using a tmp array of size N, but can be destroy after the step is finished.

Hence, the overall auxiliary storage for this parallel partition algorithm should be $3N \in O(N)$.

Work and Depth

Both STEP1 and STEP3 have $O(N)$ work. For STEP2, I use parallel scan algorithm #2 in the slide, so the work is $O(NlgN)$. Hence, the total work should be$W = W_1 + W_2 + W_3 = O(NlgN)$.

Both STEP1 and STEP3 are parallel for loop, so $D_1 = D_3 = O(lgN)$. Parallel scan algorithm #2 has depth $D_2 = O(lgN)$. 3 steps run serially, so total depth $D = D_1 + D_2 + D_3 = O(lgN)$.