# 15-210 Assignment CilkLab

Roy Sung
roysung@andrew.cmu.edu
Section E
5/2/2014

---

**6: Parallel Sorting**

---

**Task** 6.1
The work of this algorithm is $O(nlogn)$ and the span is $logn$. This is because if we assume that the split even divided the sequence, then this algorithm will follow closely to the reduce function. This means that there will be $logn$ levels with at each level there being approximately $n$ work being done at each level. This tells us that the work is $O(nlogn$ and the span is $logn$.

**Task** 6.24
It seems the speed up is not necessary linear, but more like based around the $log$ function. At the begining (starting from 1 thread) adding one more thread creates a good amount of speed up, but as you get closer to the number of actual logical cores in computer used, the speed up doesn't seem to get any better. Also if you go past the number of actual logical cores in the computer, the speed up decreases as the number of thread supposedly used gets higher.

**Task** 6.3
This is because we cannot parallelize on every single element. What I mean by this is that, if this algorithm was to go down to the base case, in the ideal situation there would be a processor working on each node. However for a large amount of elements, that would mean that there would be the same number of processors. This is ideal case, and for now not the practical case so then we have to preserve the processors that are in use so that they can be used for other parts of the tree as well. Thus when it gets down to a certain threshold, instead of using up another processor, we use the same one to compute the of the algorithm for the given sequence. That way the other processors can get to work on part parallel component of the algorithm.

**Task** 6.4
The first step which is choosing the pivots is determined in the code to be the number of pivots the algorithm is using. So in this case it would be $\frac{\sqrt{n}}{10}$. Thus the work and span would be $O(\sqrt{n})$. The second step consists of placing each element into a block or bucket that would be given to a processor. Thus the work being done here is $n$ and constant time for span. This is because we have to iterate over all the elements and assign it to a processor which happens to be the number of buckets. The third step consists of bucketing every single element into the right bucket. Thus we would have to iterate over all the elements and find which bucket it belongs to. In order to do this, the code specifically uses a binary search in order to find the bucket it belongs in. Thus each element, we would have to do a binary search. This would consist of $nlogn$ work and $log^2(n)$ span. Since the work and span of a binary search is $logn$. But since there are $O(\sqrt{n})$ buckets the work would actually be $O(nlog\sqrt{n})$ and the span would be $log^2\sqrt{n}$. The fourth step consists of sorting the buckets locally which is trivial compared to the rest of the algorithm(work and span wise) This is because the number of elements in each bucket should be signicantly smaller than the bigger elemnts. With all of these steps we would have it so that the work for sampleSort is $O(nlogn)$ and the span is $log^2(n)$.

**Task 6.5**

Yes it would scale well because then we can give each processor a single bucket of elements to process and sort. Then the span of the algorithm would be the run time.

**TAsk 7.1**

We can't do this because of race conditions. Each for loops would be competing with the other for loops and be acessing the same memory. Cilk already assumes that when using cilkfor the loops can be run in parallel without race conditions.