ECS223a Parallel Algorithms
Lecture Notes on BSP List Ranking

Recall that list ranking is a special case of performing parallel prefix on a linked list (the case where the operation is sum and the value at each vertex is one, so that we end up computing an index for each item that we could use to move the list into an array). This paper considers the general parallel prefix problem in the BSP model. It uses a randomized divide and conquer strategy. The entire paper gives an algorithm with $O(k \lg p)$ supersteps, where $k = O(\lg^* n)$, that is, essentially constant. We will only cover the beginning of the paper, which reviews a simple, practical algorithm due to Margaret Reid-Miller from 1993.

1 Sampling theorem

The input is given as a pointer array A. We use random sampling to break up the list into small parts, each of size at most $O(p \lg n)$. This uses the following (generally useful!) random sampling Theorem (the proof is in Section 2 of the paper).

Theorem 1 *Let c be a constant. Choosing n/p random samples from a sequence of length n divides it into pieces such that the length of the largest piece has length $3cp \lg n$ with probability at least $1 - 1/n^c$.*

2 Algorithm

Here is the outline of the algorithm.

```
1. Each processor gets an arbitrary piece of A of size n/p.
2. Each item in A chooses to join sample S with probability 1/p.
3. Do parallel prefix until every node has found the next node in S.
4. Every node in S sends its next node, and the prefix result, to processor 0.
5. Processor 0 solves the prefix problem for S and broadcasts the results.
6. Each processor solves the prefix problem for its portion of A.
```

In Step 1, we read the input; this requires $O(n/p)$ messages per processor. Note that each item can calculate which processor contains the destination of its pointer. Step 2 is $O(n/p)$ computation per processor. Step 3 requires $O(\lg p + \lg \lg n)$ rounds of communication, each consisting of $O(n/p)$ incoming and outgoing messages per processor, thanks the the sampling Theorem. In Step 4, processor 0 receives $O(n/p^2) \times p = O(n/p)$ messages. In Step 5, it spends time $O(n/p)$ computing the prefix operation sequentially, and broadcasts $O(n/p)$ messages. Finally, in Step 6, each processor does $O(n/p)$ sequential computation, using the results that it received in Step 5 and the results of the pointer jumping in Step 3. Thus, the entire algorithm requires $O(\lg p + \lg \lg n)$ supersteps, including $O(n/p)$ messages and $O(n/p)$ computation per step.

3 Evaluation

Since the sequential algorithm is $O(n)$, we'd be looking for an $O(n/p)$ parallel algorithm if we could get a perfect speedup; we'll never achieve this, since we have $O(n/p)$ operations in $O(\lg p + \lg\lg n)$ supersteps, so the algorithm is not optimal. But, when $O(\lg\lg n) = O(\lg p)$, i.e. $p = \Omega(\lg n)$, it requires only $O(\lg p)$ supersteps, which is pretty good. And, it is simple, practical and used in practice.