COMS E6998-9: Algorithms for Massive Data (Fall'23) Nov 20, 2023

Lecture 21: MPC (Parallel) Algorithms

Instructor: *Alex Andoni* Scribes: *Parker Williams*

1 Introduction

Recall some notation established last lecture:

P - number of machines
S - space available per machine
N - input size
ptime - number of rounds (our measure of performance)
$P \cdot S = \Theta(N)$

Also from last time: Addition and Prefix Sum - solvable in $O(\log_s(N))$ where the main idea was an n-ary tree on N leaves.

2 Sorting Problem

- **Description:** Sorting of records based on a key.
- **Time Complexity:** $O(\log_S(N))$.
- **Input:** Pairs (a_i, I_e), where a_i is the key, and I_e is the record.
- **Output:** Tuples (a_i, r_i, I_e), where r_i is the rank order of the record post-sorting. These are ordered on each machine in order r_i.

Example 1. *Say some numbers a_i are consecutive.*

$...M_i$	M_{i+1}	$M_{i+2}...$
.., 3, 3	*3, ..., 3*	*3, 3, 5, 5, ...*

Suppose the problem is counting the number of distinct elements.

Algorithm Description:

1. *Sort all the a_i's.*
2. *Each machine locally computes the number of distinct elements d_i.*

3. *Each machine sends its count of distinct elements d_i, along with the first and last element (f_i, l_i), to M_1.*

4. *On M_1, output the total number of distinct elements. This is calculated as the summation of the counts of distinct elements from each machine, minus the number of times the first element of a machine's range was equal to the last element of the previous machine's range: $\Sigma_i(d_i - \mathbf{1}[l_i = f_{i+1}])$.*

3 Graph Problems

We now move to graph problems, because parallelizing these is much less straightforward than the other problems we have seen so far.

3.1 Connectivity on a graph with n nodes and $m \geq n$ edges

Note: $N = \Theta(m)$

3.1.1 Dense regime

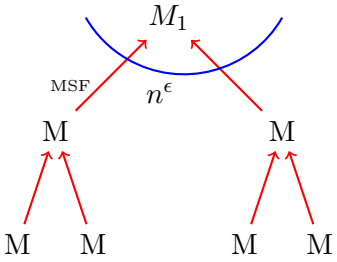
We are in a dense regime when $S \geq n^{1+\epsilon}$, for $\epsilon > 0$. Namely, each machine can maintain a table for all the nodes. We also think of $N = m = n^{1+\delta}$, for $\delta > \epsilon$.

Theorem 2. *Connectivity in the dense regime is solvable in $O(\frac{\delta}{\epsilon})$ time.*

Proof. **Alg: in $O(\frac{\delta}{\epsilon})$ rounds**

1. locally compute the **minimum spanning forest (MSF)**
2. aggregate MSF from n^ϵ machines into 1

End: M_1 computes the connected components of the union of input



Correctness: Whenever an edge is discarded, we keep an alternative path (because of the property of the MSF).
Runtime: In one round, we decrease the number of edges by a factor of n^ϵ. So after $T = 2\lceil\frac{\delta}{\epsilon}\rceil$ rounds, the number of edges decreases from $m \leq n^2$ to $\frac{n^2}{(n^\epsilon)^T} < 1$. □

3.1.2 Sparse regime

We are in a sparse regime when $S << n$.

3.1.3 Algorithm 1:

BFS in time $O(D)$ where D is the diameter of any connected component of the graph (diameter can be thought of as the maximal distance between two nodes). A motivating example could be social graphs, where there is thought to be only ≈ 6 degrees of separation (making the diameter 6).

Idea: Simulate 1 layer of the BFS wave in 1 round. So the wave is parallelized across machines, and then after D rounds we have saturated.

Algorithm:

- Keep $mark(i) \in \{0, 1\}$.
- For $\forall_i$, keep incident edges (perhaps on a few machines $\leq \lceil \frac{d_i}{s} \rceil + 1$ for degree d).
- Start(i) and end(i) correspond to the first and last machines storing the edges.

Preprocessing:

- Locally duplicate edges $((u, v) \to (u, v), (v, u))$.
- Sort edges lexicographically (in other words, sort by the first node so that everything is consecutive, then sort by the second node).
- Collect start(i) and end(i) (each machine is responsible for $\frac{n}{p}$ nodes).

BFS algorithm on MPC:

1. $Mark(S) = 1$, $mark(\forall \neq S) = 0$.
2. While we can (or in $\leq D$ rounds):
   - for $\forall u$ marked, send "push" to start(u)/end(u)
   - if $|start(u) - end(u)| > 1$, then distribute "push" to machines between start(u) & end(u) in $O(log_s(n))$ rounds.
   - each machine with "push" from u information, and edge (u, v), generate a message "(v,marked)"
   - sort all "(v,marked)", remove duplicates
   - send "(v,marked)" to the machine responsible for node v
   - set $mark(v) = 1$
3. Collect start(i) and end(i) (each machine is responsible for $\frac{n}{p}$ nodes).

3.1.4 Algorithm 2:

Suppose the diameter is bad. We can instead obtain a runtime of $O(log(n))$ rounds.

Idea: in each stage, contract $\Omega(n)$ edges.

Before contraction of edge e After contraction

(We make a side note that B collapsed into A)

The next question is how we choose the edge to contract. If we picked randomly this is not ideal, because if we were to randomly pick a chain, we are stuck in the scenario from before, where we need to essentially do BFS. So the idea is to break chains so that no such scenario could be picked.

Algorithm: for $t = O(log(n))$ iterations:

1. $\forall u$, pick "leader" with probability $\frac{1}{2}$.
2. $\forall u$ which is not a leader, choose the adjacent leader with the smallest index.
3. Contract all edges of the form: (leader, non-leader). The non-leader is contracted into the leader, so the leader maintains its name.

Claim 3. *After $t = O(log(n))$ iterations, any fixed connected component is collapsed into a single node.*

Proof. $\Pr[u \text{ is contracted}] \geq \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$.

$\mathbb{E}[\#$ of remaining nodes after t stages$] \leq n * (\frac{3}{4})^t < 1$ (for $t = O(log(n))$).

We may apply the Markov bound to obtain high confidence that we have contracted to at most 1 node. □

Implementation details (storage):

- $\forall u$: $start(u)$, $end(u)$, $active(u) = \{yes, (no, v)\}$ where v is the node u was contracted into.
- Edges.
- (Referring to 2 from the algorithm) Each leader does a push onto its neighbors. The neighbors learn the smallest index leader from this information.
- (Referring to 3 from the algorithm) Each (u,v) [(non-leader, leader)] to contract. $active(u) = (no, v)$. From u we push the information that (u,$\forall$j) is to be relabeled into (v,$\forall$j).