COMS E6998-9: Algorithms for Massive Data (Fall'25) Oct 20, 2025

Lecture 14: Massively Parallel Computing

Instructor: *Alex Andoni* Scribes: *Sadi Gulcelik*

1 Massively Parallel Computing

Model parameters recap:

- N — input size
- S — space per machine (or $O(S)$), typically we think of it as N^δ for constant $\delta > 0$
- M — number of machines (e.g., $\Theta(N/S)$)
- R — number of rounds (parallel time)

From past lecture:

- **XOR / ADD:** $R = O(\log_S N)$
- **Prefix Sum:** $R = O(\log_S N)$

2 Parameter Setting and Questions

We typically assume $S = N^\delta$ for some constant $0 < \delta < 1$. Then

$$\log_S N = \frac{\log N}{\log N^\delta} = \frac{1}{\delta},$$

which is a constant — so many problems can be solved in $O(1)$ rounds.

Examples and questions:

- **PageRank:** can we parallelize it efficiently given the many iterations?
- **Distinct elements, mode:** example problems for MPC.
- **Reconstruction:** what kinds of problems can't be reconstructed in MPC?

3 Sorting with MPC

We'll now consider **sorting** with MPC.

Goal: show that sorting can be done in

$$R = O(\log_S N)$$

rounds.

Setup

We'll take $S = N^{3/4}$, so the number of machines is $O(N^{1/4})$.

Idea

We'll use a **quicksort-style** approach: take $N^{1/4}$ random pivots — as in standard quicksort, we pick random pivots and split the array accordingly.

Inputs and Outputs

- **Input:** (i, a_i), where i is the index
- **Output:** (i, a_i, r_i), where r_i is the rank of a_i

4 Sorting with MPC — Quicksort-Style Approach

At the start, each machine holds $O(S) = O(N^{3/4})$ input pairs. We'll run a quicksort-style algorithm with $N^{1/4}$ pivots.

Step 1: Sampling pre-pivots

For each element $i \in [N]$, mark i as a **pre-pivot** with probability $N^{-2/3}$.

- Expected number of pre-pivots: $N^{1/3}$.
- With high probability, the number of pre-pivots is within a constant factor of $N^{1/3}$ (Chernoff bounds).

Step 2: Sharing pre-pivots

Each machine sends all of its pre-pivots to every other machine.

Each machine sends $O(N^{1/3})$ items to each of the $M = O(N^{1/4})$ machines. Total data per machine: $O(N^{7/12}) < O(S)$, so this fits the per-machine constraint.

Step 3: Local rank computation

For each machine j and each pivot p, compute the local rank $r_j(p)$ — the number of items on machine j smaller than p.

Step 4: Sending local ranks to controller

Each machine sends all its local ranks $r_j(p)$ to a controller machine M_1.

Step 5: Global rank computation

The controller M_1 computes the global rank of each pre-pivot p:

$$r(p) = \sum_j r_j(p) + 1$$

After this, M_1 knows the global ranks of all pre-pivots.

Justification. To perform this computation, M_1 must receive $O(N^{1/3})$ local ranks from each of the $M = O(N^{1/4})$ machines. The total data received is $O(N^{1/3}) \times O(N^{1/4}) = O(N^{7/12})$, which is strictly less than the available memory $S = O(N^{3/4})$. Hence, M_1 can safely aggregate all local ranks without exceeding its space bound.

Step 6: Choosing final pivots

From the pre-pivots, M_1 selects $N^{1/4}$ final pivots.

For the j-th pivot p_j:

$$\text{rank}(p_j) \in \left[\frac{jN}{N^{1/4}}, \frac{jN}{N^{1/4}} + O\left(\frac{N}{N^{1/4}}\right)\right]$$

Claims:

$$|r(p_{j+1}) - r(p_j)| = O(N^{3/4}) = O(S)$$

so each partition fits within one machine's memory. With high probability, we can find suitable pivots for all j.

Justification. There are $N^{1/3}$ pre-pivots in total, and we select $N^{1/4}$ final pivots from them. The expected number of data items between two consecutive final pivots is $N/N^{1/4} = N^{3/4}$. Since we sampled significantly more pre-pivots ($N^{1/3}$), the distribution of their global ranks approximates the true data distribution well. Chernoff bounds guarantee that actual ranks concentrate around expectations, so M_1 can select $N^{1/4}$ pivots such that each partition contains $O(N^{3/4}) = O(S)$ items with high probability. This ensures that every partition fits in a single machine's memory.

Step 7: Broadcasting pivots

The controller M_1 sends chosen pivots p_j (with their ranks) to all machines.

Step 8: Redistributing data

Each machine sends (i, a_i) to machine M_j such that $p_{j-1} < a_i \leq p_j$.

After this, machine j holds all elements in $(p_{j-1}, p_j]$.

Step 9: Local sort and rank reconstruction

Each machine locally sorts its elements and computes their global ranks by adding the global rank of the previous pivot p_{j-1}.

Justification. Machine M_j holds all elements in $(p_{j-1}, p_j]$ and knows the global rank $r(p_{j-1})$ broadcast by M_1. By sorting locally, it obtains the local rank $r_{\text{local}}(a_i)$ of each element. The global rank is then computed as

$$r(a_i) = r(p_{j-1}) + r_{\text{local}}(a_i),$$

ensuring every element's final rank is globally consistent.

At this point, every element (i, a_i) has an associated global rank r_i, and the array is sorted.

5 Problem 4 — Counting Distinct Elements

Input: a_i, for $i = 1, \ldots, N$ **Output:** total number of distinct elements

Algorithm

1. **Sort the input:** using the MPC sorting algorithm. After sorting, the data is distributed across machines $(M_1, M_2, \ldots)$.
2. **Local distinct count:** Each machine computes the number of distinct elements in its local block and outputs: c_j (count), l_j (leftmost), r_j (rightmost). Handle duplicates across machine boundaries.
3. **Aggregation phase (S-ary tree):** Combine results using an S-ary tree structure: sum counts, subtract duplicates across block boundaries. Continue until the root computes the global number of distinct elements.

Result: Final count at root = total number of distinct elements.

$$R = O(\log_S N)$$

Complexity Justification.

1. Sorting takes $O(\log_S N)$ rounds (as shown above).
2. Local counting is purely local and takes 0 additional rounds.
3. Aggregation uses an S-ary tree over $M = O(N/S)$ machines, whose depth is $\log_S(N/S) = \log_S N - 1$.

Hence, total round complexity remains $O(\log_S N)$.

6 Problem 5 — Graph Connectivity in MPC

Graphs with n nodes and m edges.

Input size $N = O(n + m)$, typically $m \geq n$ so $N = \Theta(m)$.

Input: (a_i, b_i) for $i = 1, \ldots, m$. Problem: connectivity in an undirected, unweighted graph.

Typically we have $S = N^\delta = m^\delta$. Depending on how S compares to n, we distinguish dense vs sparse regime.

Dense Regime

Dense regime means $S \geq n^{1+\varepsilon}$.

Since $m = N \gg S \geq n^{1+\varepsilon}$, we have $m \geq n^{1+\varepsilon}$.

Theorem 1. *We can solve connectivity in*

$$O\left(\frac{\delta}{\varepsilon}\right)$$

rounds, where $m = n^{1+\delta}$.

Algorithm Sketch

1. **Local computation:** Each machine computes its local connected components (spanning trees). This reduces m, since internal edges no longer matter globally.
2. **Aggregation across machines:** Merge local spanning trees using an n^ε-ary tree. Each round reduces the total number of edges by n^ε.
3. **Round complexity:** After $\frac{\delta}{\varepsilon}$ rounds, $m < n$. The remaining edges fit on a single machine, and connectivity is computed locally.

Justification. Let m_k denote the number of edges remaining after k rounds. Initially $m_0 = n^{1+\delta}$. Each aggregation round merges n^ε components, reducing the number of inter-component edges by the same factor:

$$m_k = \frac{m_0}{(n^\varepsilon)^k} = n^{1+\delta-k\varepsilon}.$$

The process stops when $m_k \leq n$, i.e.

$$n^{1+\delta-k\varepsilon} \leq n \quad \Rightarrow \quad 1+\delta-k\varepsilon \leq 1 \quad \Rightarrow \quad k \geq \frac{\delta}{\varepsilon}.$$

Thus, $O(\delta/\varepsilon)$ rounds suffice for full connectivity computation.

Result:

$$R = O\left(\frac{\delta}{\varepsilon}\right)$$

rounds in the MPC model when $m = n^{1+\delta}$.

Note

Next lecture — we'll look at the **sparse regime**, where $S \ll n$.