COMS 6998: Algorithms for Massive Data (Fall'25) Oct 27, 2025

Lecture 16

Instructor: *Alex Andoni* Scribes: *Roberto Brera*

1 MPC model recap

We work in the Massively Parallel Computation (MPC) model (board summary at the start of lecture).

Definition 1 (MPC parameters)**.** *An MPC computation is parameterized by:*

- ***Input size*** N *(number of words needed to encode the input).*
- ***Per-machine space / I/O budget*** S *(each machine has* S *words of local memory; per round it can send/receive* $\tilde{O}(S)$ *words).*
- ***# machines*** M*, typically assumed large enough to store the input:* $M \geq N/S$ *(so total memory* $MS \geq N$*).*
- ***# rounds*** R*, the main complexity measure.*

Each round consists of local computation plus a communication/shuffle step respecting the per-machine I/O budget.

2 Problem: pointer jumping / k-hop

The lecture used *pointer jumping* as a warm-up and as a bridge to "k-hop" computation.

Definition 2 (Pointer jumping / k-hop on a functional graph)**.** *Let* $A : [n] \to [n]$ *be a function (equivalently, a directed graph with out-degree* 1*). Given* $k \in \mathbb{N}$ *and a start vertex* $s \in [n]$*, output*

$$A^k(s) \quad \textit{where} \quad A^0(s) = s, \;\; A^{t+1}(s) = A(A^t(s)).$$

2.1 Doubling idea

We cover the standard doubling (a.k.a. pointer-doubling) algorithm. First assume k is a power of two.

Definition 3 (Doubling tables)**.** *For* $i \geq 0$*, define an array* $B_i : [n] \to [n]$ *by*

$$B_0(j) := A(j), \qquad B_i(j) := B_{i-1}\big(B_{i-1}(j)\big) \quad (i \geq 1).$$

Intuitively, $B_i(j)$ *is the result of jumping* 2^i *steps forward from* j*.*

Claim 4. *For all* $i \geq 0$ *and all* $j \in [n]$*,*

$$B_i(j) = A^{2^i}(j).$$

Proof. By induction on i.

Base case $i = 0$: $B_0(j) = A(j) = A^{2^0}(j)$.

Inductive step: assume $B_{i-1}(j) = A^{2^{i-1}}(j)$ for all j. Then

$$B_i(j) = B_{i-1}\big(B_{i-1}(j)\big) = A^{2^{i-1}}\Big(A^{2^{i-1}}(j)\Big) = A^{2^i}(j).$$

□

Thus if $k = 2^\ell$, we can return $B_\ell(s) = A^k(s)$.

2.2 General k via binary decomposition

We now cover the standard "scan bits of k" trick (example shown: $k = [10010]_2 = 16 + 2$).

Let $k = \sum_{i=0}^{\lfloor \log_2 k \rfloor} b_i 2^i$ with $b_i \in \{0, 1\}$. Compute all B_i as above, and then:

$$p \leftarrow s; \quad \text{for } i = 0, 1, \ldots, \lfloor \log_2 k \rfloor : \ \text{ if } b_i = 1 \text{ then } p \leftarrow B_i(p).$$

At the end, $p = A^k(s)$.

2.3 MPC implementation and round complexity

The lecture emphasized how to implement the recurrence

$$B_i(j) = B_{i-1}\big(B_{i-1}(j)\big)$$

in MPC using a simple routing / join step.

- Think of a machine "responsible for index j" as storing the pair $(j, B_{i-1}(j))$.
- To compute $B_i(j)$, the machine for j needs to learn the value stored at index $B_{i-1}(j)$, namely $B_{i-1}(B_{i-1}(j))$.
- In one communication step, the machine for j sends a request keyed by $B_{i-1}(j)$ to the machine responsible for that index; the recipient replies with the needed value.

With $S = O(1)$, each index stores $O(1)$ words and sends/receives $O(1)$ messages per iteration, so each doubling level costs $O(1)$ MPC rounds.

Theorem 5 (Pointer jumping in MPC)**.** *For the pointer jumping / k-hop problem, there is an MPC algorithm using $S = O(1)$ local space per machine and*

$$R = O(\log k)$$

rounds (via the doubling algorithm above).

3 Connection to attention / transformers

The lecture then pivoted to a connection between MPC-style computation and transformers (*"Connection to attention/transformers"* and referencing "Sanford–Hsu–Telgarsky '24").

3.1 Self-attention primitive

Definition 6 (Softmax attention map)**.** *Given vectors $q_1, \ldots, q_N \in \mathbb{R}^d$ (queries), $k_1, \ldots, k_N \in \mathbb{R}^d$ (keys), and $v_1, \ldots, v_N \in \mathbb{R}^{d_v}$ (values), define for each token i:*

$$\mathrm{Att}(q_i) = \frac{\sum_{j=1}^{N} e^{\langle q_i, k_j \rangle} v_j}{\sum_{j=1}^{N} e^{\langle q_i, k_j \rangle}}.$$

A (simplified) transformer layer alternates global mixing via attention with per-token local computation via an MLP; stacking L layers yields depth L.

3.2 Simulating MPC with transformers

The key stated message was:

Theorem 7 (MPC $\Rightarrow$ Transformer)**.** *Suppose a problem P can be solved in the MPC model using N machines, local memory*

$$S = n^{\varepsilon}$$

and in R rounds. Then there exists a transformer with

$$L = R$$

layers that solves P, with internal dimensions d, d_v bounded by $S^{O(1)}$.

Corollary 8 (k-hop depth bound)**.** *There exists a transformer that solves k-hop / pointer jumping using*

$$O(\log k)$$

layers.

Interpretation. The depth L of the transformer plays the role of MPC rounds: each attention layer can be viewed as a global communication step (routing information between "positions"), while MLPs emulate local computation.

Training dynamics viewpoint: if we provide many supervised examples

$$(A, k, s) \mapsto A^k(s),$$

a transformer may learn an algorithm resembling doubling / binary decomposition (hence the emphasis on the binary representation of k).

4 MPC algorithms for geometric graphs: MST and clustering

The last part of the lecture moved to MPC algorithms for *geometric graphs*.

4.1 Problem setup

We are given n points in $\mathbb{R}^d$. The implicit graph is the complete weighted graph on these points with edge weights given by Euclidean distance. Although the complete graph has

$$m = \binom{n}{2}$$

edges, the input size is just $N = O(n)$ words (the coordinates), so the goal is to avoid materializing all edges.

The lecture highlighted:

- **MST** (minimum spanning tree),
- **clustering**, with the note *"MST $\approx$ single-linkage clustering"*.

Definition 9 (Euclidean MST cost)**.** *For a tree T on the n points,*

$$\text{cost}(T) \;=\; \sum_{(u,v)\in T} \|u - v\|_2.$$

Definition 10 ($(1+\varepsilon)$-approximate MST)**.** *Given $\varepsilon \in (0,1)$, output a spanning tree T such that*

$$\text{cost}(T) \le (1+\varepsilon) \cdot \text{cost}(\text{MST}).$$

4.2 Stated result (approximate MST in MPC)

Approximate MST guarantee in low dimension:

Theorem 11 (Approx. Euclidean MST in MPC)**.** *For points in $\mathbb{R}^d$, one can compute a $(1+\varepsilon)$-approximation to the Euclidean MST in*

$$R = \left(\log_S n\right)^{O(1)}$$

rounds, assuming the local memory satisfies

$$S \;\ge\; (1/\varepsilon)^{O(d)}.$$

4.3 High-level idea: "sketch & solve" via geometric sparsification

- **Sketch / sparsify:** Build a sparse graph H (a geometric *spanner*) on the n points so that distances in H approximate Euclidean distances up to factor $(1+\varepsilon)$. The drawings suggested using a grid (or quadtree-style) decomposition at one or more scales, and keeping a small number of *representative* points per cell/region. The number of representatives per region can be on the order of $(1/\varepsilon)^d$.
- **Solve on the sketch:** Run an MPC MST algorithm (e.g. a Borůvka-style contraction routine) on the sparse graph H instead of on the complete graph. Since H preserves distances up to $(1+\varepsilon)$, the MST of H yields a $(1+\varepsilon)$-approximation to the Euclidean MST.

WLOG assumptions. For the planar case ($d = 2$) the lecture wrote a simplifying assumption along the lines of: points have integer coordinates in a bounded grid (e.g. within $[n^2]^2$) and there are no duplicate points. These assumptions help avoid degeneracies/ties and keep discretizations well-behaved.