Faster and Simpler Concurrent Algorithms for Graph Problems

Sixue Liu
Princeton

Joint work with Robert Tarjan
and Peilin Zhong (Columbia)
Two most basic graph problems

Connectivity: In an undirected graph, find all the connected components (maximal set of vertices pairwise connected by a path).

Reachability*: In a directed graph, find all the strongly connected components (maximal set of vertices pairwise reachable by a directed path).
Two most basic graph problems

Sequential algorithms:

• Connectivity: any graph search, DFS or BFS
• Reachability: [Tarjan’72] or [Gabow’00]

- For a (directed) graph with $n$ vertices and $m$ edges (arcs), solvable in $O(m + n)$ time.
- Optimal
A new view

- Graphs from real-world applications grow too large
- Computational resource is relatively cheap
- Hadoop (MapReduce), Spark...

- Concurrent algorithms

A friends graph of social network
[http://niftythings.paulnickerson.net/2011/03/facebook-friends-graph.html]
Two theoretical models for concurrent algorithms

- Massively Parallel Computation (MPC) Model
- Parallel Random Access Machines (PRAM) Model
Two theoretical models for concurrent algorithms

• Massively Parallel Computation (MPC) Model

  ![Diagram](image)

  - Input $N$ words
    - $N = O(m + n)$ for graph problems
  - Each processor holds $s = N/p = O(N^\delta)$ words
    - Constant $\delta \in (0,1)$
  - Computation proceeds in rounds
  - Computation in a single processor is free
  - Minimize the number of (communication) rounds

[figure From Peilin Zhong]
Two theoretical models for concurrent algorithms

- Parallel Random Access Machines (PRAM) Model

  - Unlimited shared memory
  - Each processor holds constant words
  - Each processor can read/write a cell in the shared memory in constant time
  - Trade-off between the time (span) of computation and the number of processors
  - $\text{Time} \times \text{processors number} = \text{work}$
    - At most the $|\text{work}|$ number of cells are used
  - Minimize the time, work, or processors number

[http://cs.uef.fi/~penttone/parallel/pram.html]
Two theoretical models for concurrent algorithms

- PRAM models (in increasing power):
  - Exclusive-read Exclusive-write (EREW)
  - Concurrent-read Exclusive-write (CREW)
  - Concurrent-read Concurrent-write (CRCW)
    - COMMON
    - ARBITRARY
    - PRIORITY
    - COMBINING: min, max, add
Two theoretical models for concurrent algorithms

In most cases, MPC is more powerful than PRAM

• OR of $n$ bits: $\Omega(\log n)$ time on a CREW PRAM [Cook, Dwork & Reischuk’86]
  $\Rightarrow$ Connectivity: $\Omega(\log n)$ time on a CREW PRAM

• PARITY of $n$ bits: $\Omega(\log n / \log \log n)$ time on a PRIORITY CRCW PRAM with polynomial processors [Beame & Hastad’87]

• Sort $n$ integers: $\Omega(\sqrt{\log n})$ time on a PRIORITY CRCW PRAM [Heide & Wigderson’87]
  ▪ All three problems can be easily solved on MPC in $O(1/\delta)$ time [Goodrich’99]
    ▪ MPC with total space $N$ can simulate any PRAM algorithm with work at most $N$
# Results on connectivity

<table>
<thead>
<tr>
<th>Reference</th>
<th>Time (w.h.p.)</th>
<th>#Processors (total memory)</th>
<th>Model</th>
<th>Rd./det.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shiloach &amp; Vishkin'82</td>
<td>$O(\log n)$</td>
<td>$O(m + n)$</td>
<td>ARB. CRCW</td>
<td>det</td>
</tr>
<tr>
<td>Awerbuch &amp; Shiloach’87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cole &amp; Vishkin’91</td>
<td>$O(\log n)$</td>
<td></td>
<td>ARB. CRCW</td>
<td>det</td>
</tr>
<tr>
<td>Iwama &amp; Kambayashi’94</td>
<td>$O((m + n)\alpha(m, n)/ \log n)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gazit’91</td>
<td>$O(\log n)$</td>
<td>$O((m + n)/ \log n)$</td>
<td>ARB. CRCW</td>
<td>rd</td>
</tr>
<tr>
<td>Johnson &amp; Metaxas’97</td>
<td>$O((\log n)^{3/2})$</td>
<td>$O(m + n)$</td>
<td>CREW</td>
<td>det</td>
</tr>
<tr>
<td>Chong &amp; Lam’95</td>
<td>$O(\log n \log \log n)$</td>
<td>$O(m + n)$</td>
<td>EREW</td>
<td>det</td>
</tr>
<tr>
<td>Halperin &amp; Zwick’96</td>
<td>$O(\log n)$</td>
<td>$O((m + n)/\log n)$</td>
<td>EREW</td>
<td>rd</td>
</tr>
<tr>
<td>Liu &amp; Tarjan</td>
<td>$O(\log n)$</td>
<td>$O(m + n)$</td>
<td>COMB. CRCW or MPC</td>
<td>det</td>
</tr>
<tr>
<td>(extremely simple algorithm)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andoni et al. ’18</td>
<td>$O(\log d \log \log_{m/n} n)$</td>
<td>$O(m + n)$</td>
<td>MPC</td>
<td>rd</td>
</tr>
<tr>
<td>Liu, Tarjan &amp; Zhong</td>
<td>$O(\log d \log \log_{m/n} n)$</td>
<td>$O(m + n)$</td>
<td>ARB. CRCW</td>
<td>rd</td>
</tr>
</tbody>
</table>

*When $m = n^{1+\Omega(1)}$, runs in $O(\log d)$ time w.p. $1 - 1/poly(n)$  

*d: diameter  

*m: #edges  

*n: #vertices
## Results on single-source reachability

<table>
<thead>
<tr>
<th>Reference</th>
<th>Work</th>
<th>Time</th>
<th>Nearly work efficient (work = $\tilde{O}(m)$)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel BFS</td>
<td>$O(m)$</td>
<td>$\tilde{O}(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Parallel Transitive Closure</td>
<td>$\tilde{O}(n^\omega)$</td>
<td>$\tilde{O}(1)$</td>
<td>No</td>
</tr>
<tr>
<td>Spencer’97</td>
<td>$\tilde{O}(m + np^2)$</td>
<td>$\tilde{O}(n/p)$</td>
<td>If $p = \tilde{O}(\sqrt{m/n})$</td>
</tr>
<tr>
<td>Ullman &amp; Yannakakis’91</td>
<td>$\tilde{O}(mp + p^4/n)$</td>
<td>$\tilde{O}(n/p)$</td>
<td>If $p = \tilde{O}(1)$</td>
</tr>
<tr>
<td>Fineman’18</td>
<td>$\tilde{O}(m)$</td>
<td>$\tilde{O}(n^{2/3})$</td>
<td>Yes</td>
</tr>
<tr>
<td>Jambulapati et al. ’19</td>
<td>$\tilde{O}(m)$</td>
<td>$\tilde{O}(n^{1/2})$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* $m$: #arcs * $n$: #vertices * $\tilde{O}$: omit polylog factors * $\omega$: matrix multiplication exponent

- [Fineman’18] is the first algorithm to achieve truly sublinear time and nearly work efficient
  - Can be extended to solve all-pairs reachability and strongly connected components with $\tilde{O}(1)$ overhead [Schudy’08]
- [Liu & Tarjan]: an $\tilde{O}(n^{(2/3)} \ln^2)$ lower bound for Fineman’s algorithm and an $\tilde{O}(n^{(1/2)} \ln^2)$ lower bound for JLS.
Outline of the talk

✓ Connectivity and reachability
✓ MPC and PRAM models
✓ The previous and our results

➢ Much simpler algorithms for connectivity in $O(\log n)$ time
  ▪ General ideas in analyzing one of our algorithms
• A high-probability $O(\log d \log \log_{m/n} n)$-time ARB. CRCW PRAM algorithm for connectivity
❖ Fineman’s algorithm for reachability and our idea on lower bounds
How to represent components?

Label all vertices in each component with a unique vertex in the component: can test if two vertices are in the same component by comparing their labels.

Assume $n$ vertices: $1, \ldots, n$; $m$ edges

Minimum labeling: the minimum vertex in the component.
How to represent components?
How to represent components?

1 1
2 1
3 1
4 1
5 1
6 1
7 1
How to represent components during execution?
The vertices \( v \) and the arcs \((v, v.p)\) define a directed graph (digraph)

If the only cycles are loops (arcs of the form \((v, v)\)),
the digraph consists of a set of rooted trees:

- \( v.p \) is the parent of \( v \)
- \( v \) is a root iff \( v = v.p \)

If labels never increase, all cycles are loops
Our results

Seven extremely simple algorithms on COMB. CRCW PRAM with min, or MPC:

• Two of them run in $\Theta((\log n)^2)$ time.
• For another three algorithms, one runs in $O((\log n)^2)$ time, and two of them run in $O(d)$ time.
• Another two of them run in $O(\log n)$ time.
Algorithm \textbf{R} (for root-connect)

for each \( v \) do \( v.p = v \);

repeat

\{for each arc \((v, w)\) do if \( v.p < w.p \) \& \( w.p.p = w.p \) then \( w.p.p = v.p \);

for each \( v \) do \( v.p = v.p.p \}\}

until no parent changes

*Here use min as the COMBINING function

*For simplicity, treat each edge \((v, w)\) as two directed arc \((v, w)\) and \((w, v)\)
Algorithm **R** (for root-connect)

for each $v$ do $v.p = v$
repeat
  {for each $(v, w)$ do if $v.p < w.p$ & $w.p.p = w.p$ then $w.p.p = v.p$;
   for each $v$ do $v.p = v.p.p$}
until no parent changes
Algorithm **R** (for root-connect)

for each $v$ do $v.p = v$

repeat

  {for each $(v, w)$ do if $v.p < w.p$ & $w.p.p = w.p$ then $w.p.p = v.p$;
   for each $v$ do $v.p = v.p.p$}

until no parent changes
Our bound and main idea

• Algorithm $R$ runs in $O(\log n)$ time.
  - The sum of the heights of certain trees decreases by a constant factor every constant number of rounds.
After two rounds all trees contain at least two vertices (except in components of one vertex)

• A tree is **passive** in a round if it does not change during that round, **active** if it does

• The potential $\Phi(T)$ of an active tree $T$ is its height plus one, plus one more if flat

• The potential of a passive tree is zero
Analysis of algorithm $\mathcal{R}$

- Let $T$ be an active tree at the end of round $k$
- The constituent trees of $T$ at the end of round $j < k$ are those at the end of round $j$ whose vertices are in $T$
- The potential of $T$ in round $j$ is the sum of the potentials of its constituent trees

Lemma: $\Phi(T_{k-1}) \geq \Phi(T)$, and if $k - j \geq 5$, $\Phi(T_j) \geq (4/3)\Phi(T)$
Analysis of algorithm R

- A shortcut on a non-flat tree reduces the height by a constant factor
- A calculation gives $\Phi(T_{k-1}) \geq \Phi(T)$, and $\Phi(T_{k-1}) \geq (6/5)\Phi(T)$ if $T$ has height at least 4
- If $T$ has height at most 3 and has at least one active constituent tree of sufficient height, or at least two constituent trees, the lemma holds
- Otherwise $T$ only has one active constituent tree
- After at most four rounds, all constituent trees are combined, and $T$ is passive, a contradiction
Idea in analysis of our other algorithms

- Red-green coloring on vertices
- Tree-path partition
- Path-compression from union-find data structure

Seven extremely simple algorithms on COMB. CRCW PRAM with min or MPC:

- Two of them run in $\Theta((\log n)^2)$ time.
- For another three algorithms, one runs in $O((\log n)^2)$ time, and two of them run in $O(d)$ time.
- Another two of them run in $O(\log n)$ time.
Outline of the talk

✓ Connectivity and reachability
✓ MPC and PRAM models
✓ The previous and our results
✓ Much simpler algorithms for connectivity in $O(\log n)$ time

➢ A high-probability $O(\log d \log \log_{m/n} n)$-time ARB. CRCW PRAM algorithm for connectivity

❖ Fineman’s algorithm for reachability and our idea on lower bounds
The algorithm in high-level

Repeat

\{ VOTE; // identifies a set of leaders
    LINK; // link every non-leader to a leader in its component
    SHORTCUT; ALTER;
    // ALTER replaces every edge \((v, w)\) by \((v.p, w.p)\) \}

until no edge exists other than loops

Invariant: at the end of each round, every tree is flat and only roots have incident edges.

- Ignore the loops, a new graph (of roots) at each round
- Measure the progress by \#vertices in the new graph: terminates when there is only one vertex per component
The algorithm in high-level

Repeat

\{VOTE; // identifies a set of leaders
LINK; // link every non-leader to a leader in its component
SHORTCUT; ALTER;
  // ALTER replaces every edge (v, w) by (v.p, w.p) \}

until no edge exists other than loops

A naïve try: VOTE sets each vertex as a leader w.p. 1/2;
LINK only examines direct neighbors (arcs).
• Each vertex in the current graph is not in the new graph w.p. ≥ 1/4
• An high-probability $O(\log n)$-time ARB. CRCW algorithm [Reif’84]

Key idea: expand the neighbor set s.t. voting for leaders with much smaller prob.
still guarantees for each vertex a leader to link to. [Andoni et al. ‘18]
A toy (extreme) case

Key idea: expand the neighbor set s.t. voting for leaders with much smaller prob. still guarantees for each vertex a leader to link to.

EXPAND takes $O(\log d)$ time: repeatedly doubling the neighbor set
A toy (extreme) case

Key idea: expand the neighbor set s.t. voting for leaders with much smaller prob. still guarantees for each vertex a leader to link to.
A toy (extreme) case

Key idea: expand the neighbor set s.t. voting for leaders with much smaller prob. still guarantees for each vertex a leader to link to.

*Require every vertex to know about all vertices in its component*
What if a vertex does not have enough space to know about all other vertices in its component?

Each vertex processor only holds \( \leq 3 \) words for neighbors

Measure the progress by vertex degree
Measure the progress

Repeat

\{\text{EXPAND; VOTE; LINK; SHORTCUT; ALTER;}\} // a round takes \(O(\log d)\) time

until no edge exists other than loops

- Assume the graph is connected and \(m = bn\) (\(b\) is large enough)
- EXPAND takes \(O(\log d)\) time and tries to increase each degree to \(b^{0.5}\)
  - EXPAND at most squares the degree, thus total space \(O(m)\)
- VOTE sets each vertex as a leader w.p. \(b^{-0.1}\)
- W.h.p. each vertex has a leader as a neighbor and links to it
- In the next round, \#vertices\(\approx n' = nb^{-0.1} = n^{1.1}/m^{0.1}\)
- EXPAND now has the space to increase each degree to: \((m/n')^{0.5} = (m/n)^{1.1/2} = (b^{0.5})^{1.1}\)
- Double-exponential progress on the degree lower bound: at least \(b^{1.1^k/2}\) in round \(k\)
- Base \(m/n\), finally at most \(n \Rightarrow O(\log \log_{m/n} n)\) rounds
- \(O(\log d \log \log_{m/n} n)\) total time
From MPC to PRAM

Repeat

\{EXPAND; VOTE; LINK; SHORTCUT; ALTER;\}

until no edge exists other than loops

• Straightforward (but technical) to implement this algorithm on MPC [Andoni et al. ‘18]
  ▪ Use the $O(1/\delta)$-time sorting, indexing, and prefix-sum
• There is no such strong primitive in a PRAM (lower bounds exist)

We will present:
1. A COMB. CRCW PRAM algorithm with min and add functions
2. An ARB. CRCW PRAM algorithm
From MPC to PRAM

Repeat

\{EXPAND; VOTE; LINK; SHORTCUT; ALTER;\}

until no edge exists other than loops

• LINK, SHORTCUT, and ALTER are straightforward

• How to do a VOTE?
  • VOTE sets each vertex as a leader w.p. $b^{-0.1}$, where $b$ depends on #vertices in the current graph
    ▪ Easy with the add function
    ▪ On an ARB. CRCW, roughly knows an upper bound, then use concentration inequalities somewhere…
  ▪ What if $v$ in a small clique with no leader after the first VOTE? Needs to fix one common parent for all
    ▪ Easy with the min function
    ▪ On an ARB. CRCW, for every neighbor $u$ of $v$, if $u<v$ set $v$ as a non-leader: the min one survives as a leader
From MPC to PRAM

Repeat

\{\text{EXPAND};\ \text{VOTE};\ \text{LINK};\ \text{SHORTCUT};\ \text{ALTER};\}

until no edge exists other than loops

• How to do an EXPAND?
• Each vertex needs $b$ space (processors), the current #vertices $n$ is assumed known
• Use hashing to re-allocate processors: divide $m$ processors into $m/b$ blocks, hash vertices to blocks; when $b = o(m/n)$, almost every vertex owns a block
• How to implement a neighbor set?
• Use another hashing to hash neighbors of neighbors into the table
• Need w.h.p. each vertex obtains enough neighbors after EXPAND
  • Choosing table size carefully, collisions should be low
  • But collisions can broadcast: a vertex might stop expansion due to a collision in its neighbor’s table, thus not obtaining enough neighbors
  • Use a delicate distance-doubling argument, the above bad case happens with small probability
Our results

Theorem: There is an ARB. CRCW PRAM algorithm using $O(m + n)$ processors that computes the connected components of any given graph. W.p. $1 − 1/poly(\delta + \log n)$, it runs in $O(\log d \log \log \delta n)$ parallel time, where $\delta = (m + n)/n$.

- Using a more careful EXPAND (two tables, one for tentative expansion) to construct a shortest path tree, one can obtain the same result for spanning forest.
- When $m = n^{1+\Omega(1)}$, our algorithms run in $O(\log d)$ time w.p. $1 − 1/poly(n)$
Outline of the talk

✓ Connectivity and reachability
✓ MPC and PRAM models
✓ The previous and our results
✓ Much simpler algorithms for connectivity in $O(\log n)$ time
✓ A high-probability $O(\log d \log \log m/n)$-time ARB. CRCW PRAM algorithm for connectivity

❖ Fineman’s algorithm for reachability and our idea on lower bounds
Fineman’s algorithm for reachability

1. **Reduce the diameter** of the (directed) graph through the addition of *shortcuts* (arcs whose addition does not change the transitive closure)

2. Run parallel BFS on the shortcutted graph: if the diameter is reduced to $d$, BFS takes $\tilde{O}(d)$ parallel time

---

The BFS starting at 1 used to take 3 hops, now 2 hops
Fineman’s algorithm for reachability

• Fineman analyzes a sequential $\tilde{O}(m)$-time algorithm to reduce the diameter to $\tilde{O}(n^{2/3})$ through the addition of $\tilde{O}(n)$ shortcuts
  ▪ Reducing the diameter below $O(n^{1/11})$ requires adding $\Omega(m)$ shortcuts [Huang & Pettie’18]

• Also provides a parallel version of this algorithm with $\tilde{O}(m)$ work and $\tilde{O}(n^{2/3})$ time, thus along with the parallel BFS and Schudy’s algorithm, giving an $\tilde{O}(m)$-work $\tilde{O}(n^{2/3})$-time parallel algorithm for reachability
The sequential diameter-reducing algorithm

\[
\text{DR}(G = (V, E)):
\]

- if \( V = \emptyset \) then return \( \emptyset \)
- select a pivot \( x \in V \) u.r.
- let \( R^+ \) be the set of vertices reachable from \( x \)
- let \( R^- \) be the set of vertices that can reach \( x \)
- \( S = \{(x, v) \mid v \in R^+\} \cup \{(u, x) \mid u \in R^-\} \) // add shortcuts
- \( V_F \leftarrow R^+ \setminus R^- \), \( V_B \leftarrow R^- \setminus R^+ \), \( V_U \leftarrow V \setminus (R^+ \cup R^-) \)
- return \( \text{DR}(G[V_F]) \cup \text{DR}(G[V_B]) \cup \text{DR}(G[V_U]) \)
The sequential diameter-reducing algorithm

What we need:

• For a given path $P$:

Remark: any vertex on $P$ is a bridge of $P$
The sequential diameter-reducing algorithm

\[ \text{DR}(G = (V, E)):\]

\[
\begin{align*}
\text{if } V &= \emptyset \text{ then return } \emptyset \\
\text{select a pivot } x &\in V \text{ u.r.} \\
\text{let } R^+ &\text{ be the set of vertices reachable from } x \\
\text{let } R^- &\text{ be the set of vertices that can reach } x \\
S &= \{(x, v) \mid v \in R^+\} \cup \{(u, x) \mid u \in R^-\} \quad \text{// add shortcuts} \\
V_F &\leftarrow R^+ \setminus R^-, \quad V_B \leftarrow R^+ \setminus R^-, \quad V_U \leftarrow V \setminus (R^+ \cup R^-) \\
\text{return } \text{DR}(G[V_F]) \cup \text{DR}(G[V_B]) \cup \text{DR}(G[V_U])
\end{align*}
\]

- Choosing an ancestor decreases the expected #ancestors by half
- Choosing a descendant decreases the expected #descendants by half
- Choosing a bridge shortcuts a path
The sequential diameter-reducing algorithm

- Using a potential function, the expected length (at most \#bridge+\#ancestors+\#descendants) at level $k$ is at most $n/2^{k/2}$
- At level $k$, the path is partitioned into $2^k$ subpaths: requiring $2^k$ concatenation arcs
- Choosing $k = (2/3) \log n$ minimizes the sum
- The shortcutted path has expected length at most $n^{2/3}$
- Repeat the algorithm for polylog times: w.h.p. the diameter is $O(n^{2/3})$
- Each run takes $\tilde{O}(m)$ time and adds $\tilde{O}(n)$ shortcuts $\Rightarrow$ a total of $\tilde{O}(m)$ time, $\tilde{O}(n)$ shortcuts
An example: a directed grid

- All arcs go from top-left to bottom-right
An example: a directed grid

- Suppose we choose an ancestor $x$
An example: a directed grid

- Choose an ancestor $x$: partition the graphs into the 3 subproblems $G[V_B]$, $G[V_F]$, and $G[V_U]$
- $V_Uf$ is far from the path, and $V_Uc$ is close to the path
An example: a directed grid

- $G[V_B]$ is not path-relevant
- Vertices from $V_{uf}$ cannot partition the path: the only relevant part is in $V_F$
- Any vertex on a higher row than $x$ ceases to be an ancestor in subproblems
An example: a directed grid

- Any vertex on a row higher than $x$ ceases to be an ancestor in subproblems

$V_{Uc}$

$V_F$

Higher

Lower

Ancestors

Descendants
An example: a directed grid

- If $x$ is an ancestor, any vertex on a higher row ceases to be an ancestor in subproblems.
- If $x$ is a descendant, any vertex on a lower row ceases to be a descendant in subproblems.
Analysis on grid

![Diagram of grid and graph levels](image)

- $P_1$
- $P_2$
- $x$

Level 1
- $P$

Level 2
- $P_1$
- $P_2$

Level 3
Analysis on grid

• The probability of choosing a bridge in any subproblem at level $k$ is independent of the subpath length
• Show that the number of rows at some critical level is $n^\delta$, then the probability of choosing a bridge at level $O(\log n)$ is $o(1)$
Analysis on grid

• Any vertex on a row lower than x ceases to be an ancestor in subproblems

• At the next level, either #ancestor_rows times a uniform distributed random variable $U \sim (0,1)$ or #descendant_rows times a $U \sim (0,1)$.

• Equivalent to think that we first reveal random variable $U$ then reveal the random bits for deciding $x$ to be ancestor or descendant.

⇒At the next level, (#ancestor_rows*#descendant_rows) times a $U \sim (0,1)$.

Erlang distribution after taking log ⇒Var=𝔼 ⇒Chebyshev’s for concentration!
Analysis on grid

• Erlang distribution after taking log \(\Rightarrow \text{Var} = \mathbb{E} \Rightarrow \) Chebyshev’s for concentration.

• By AM-GM, #rows is \(\Omega(n^{\delta})\) w.h.p.

• Sampling a bridge w.p. \(o(1)\) before the critical level \(k = (2/3) \ln n\).

• By order statistics, w.h.p. the path is partitioned into \(2^k\) subpaths after level \(k\).

• The diameter of an \(n^{2/3}\)-row, \(n^{1/3}\)-column grid cannot be reduced to \(n^{(2/3)\ln 2}\) in one execution w.h.p.

• The algorithm is repeated for \(\tilde{O}(1)\) times. Make \(\ln n\) copies of grids, each diameter cannot be reduce to \(n^{(2/3)\ln 2}\) w.h.p.

• For the diameter of the entire graph to be below \(n^{(2/3)\ln 2}\), each grid must have diameter below this, which happens w.p. \((o(1))^{\ln n} < n^{-2}\).

• Union bound over all executions: w.h.p. cannot reduce the diameter below \(n^{(2/3)\ln 2}\).
Summary

✓ Connectivity and reachability
✓ MPC and PRAM models
✓ The previous and our results

✓ Much simpler algorithms for connectivity in $O(\log n)$ time
✓ A high-probability $O(\log d \log \log m/n \ n)$-time ARB. CRCW PRAM algorithm for connectivity
✓ Fineman’s algorithm for reachability and our idea on lower bounds
Thanks!