Breaking the Linear-Memory Barrier in Massively Parallel Computing

MIS on Trees with Strongly Sublinear Memory

Sebastian Brandt, Manuela Fischer, Jara Uitto
ETH Zurich
Model: Massively Parallel Computing (MPC)
Model:
Massively Parallel Computing (MPC)

parallel computing framework
Model:
Massively Parallel Computing (MPC)

parallel computing framework
inspired by MapReduce
Model: Massively Parallel Computing (MPC)

parallel computing framework inspired by MapReduce

Karloff, Suri, Vassilvitskii [SODA’10]
Massively Parallel Computing (MPC) Model

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
Massively Parallel Computing (MPC) Model

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]

Synchronous Rounds

1. Local Computation
   at every machine
2. Global Communication
   between machines
Massively Parallel Computing (MPC) Model

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]

Synchronous Rounds
1. Local Computation at every machine
Massively Parallel Computing (MPC) Model

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]

Synchronous Rounds
1. Local Computation at every machine
2. Global Communication between machines
Massively Parallel Computing (MPC) Model

- $M$ machines
- $S$ memory per machine

Synchronous Rounds
1. Local Computation at every machine
2. Global Communication between machines
Massively Parallel Computing (MPC) Model

- $M$ machines
- $S$ memory per machine

Synchronous Rounds
1. Local Computation at every machine
2. Global Communication between machines
Massively Parallel Computing (MPC) Model

- $M$ machines
- $S$ memory per machine

Synchronous Rounds
1. Local Computation at every machine
2. Global Communication between machines

Complexity:
number of rounds
Massively Parallel Computing (MPC) Model

- $M$ machines
- $S$ memory per machine

Synchronous Rounds
1. Local Computation at every machine
2. Global Communication between machines

Complexity:
number of rounds
Massively Parallel Computing (MPC) Model

- **Local Computation** at every machine
- **Global Communication** between machines

**Complexity:** number of rounds

- $M$ machines
- $S$ memory per machine

**Synchronous Rounds**
- 1. Local Computation at every machine
- 2. Global Communication between machines
Massively Parallel Computing (MPC) Model

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O} \left( m + n \right) \]

Synchronous Rounds
1. Local Computation
   at every machine
2. Global Communication
   between machines

Complexity:
number of rounds
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]
Local Memory in MPC

\[ M \text{ machines} \]

\[ S \text{ memory per machine} \]

\[ M \cdot S = \tilde{O} (m + n) \]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]
Local Memory in MPC

\( M \) machines
\( S \) memory per machine
\( M \cdot S = \tilde{O}(m + n) \)

Superlinear Memory:
\( S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1 \)
Machines see all nodes.
Local Memory in MPC

M machines
$S$ memory per machine
$M \cdot S = \tilde{O}(m + n)$

Superlinear Memory:
$S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1$

Machines see all nodes.

often trivial
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

Superlinear Memory:
\[ S = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]

Machines see all nodes.

often trivial

for many problems, admits O(1)-round algorithms based on very simple sampling approach

Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

**Linear Memory:**
\[ S = \tilde{O}(n) \]
Machines see all nodes.

**Superlinear Memory:**
\[ S = \tilde{O}(n^{1+\delta}), \ 0 < \delta \leq 1 \]
Machines see all nodes.

Often trivial for many problems, admits \( O(1) \)-round algorithms based on very simple sampling approach.

*Lattanzi et al. [SPAA’11]*
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\( S = \tilde{O}(n^{\delta}) \)
- Linear Memory:
  - \( S = \tilde{O}(n) \)
  - Machines see all nodes.
  - usual assumption

\( S = \tilde{O}(n^{1+\delta}) \)
- Superlinear Memory:
  - \( S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1 \)
  - Machines see all nodes.
  - often trivial
  - for many problems, admits \( O(1) \)-round algorithms based on very simple sampling approach

Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\[ S = \tilde{O}(n^\delta) \]
\[ \tilde{\Theta}(n) \]
\[ \tilde{\Omega}(n^{1+\delta}) \]

**Linear Memory:**
\[ S = \tilde{O}(n) \]
Machines see all nodes.

- usual assumption
- often unrealistic

**Superlinear Memory:**
\[ S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1 \]
Machines see all nodes.

- often trivial
- for many problems, admits \( O(1) \)-round algorithms based on very simple sampling approach

*Lattanzi et al. [SPAA’11]*
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\[ S = \tilde{O}(n^\delta) \]

Linear Memory:
\[ S = \tilde{O}(n) \]
Machines see all nodes.

usual assumption
often unrealistic
\[ \tilde{O}(n) \text{ prohibitively large} \]

Superlinear Memory:
\[ S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1 \]
Machines see all nodes.

often trivial
for many problems, admits \( O(1) \)-round algorithms based on very simple sampling approach
Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\( S \) = \( \tilde{O}(n^\delta) \)

Linear Memory:
\( S = \tilde{O}(n) \)
Machines see all nodes.
usual assumption
often unrealistic
- \( \tilde{O}(n) \) prohibitively large
- sparse graphs trivial

Superlinear Memory:
\( S = \tilde{O}(n^{1+\delta}), 0 < \delta \leq 1 \)
Machines see all nodes.
often trivial
for many problems,
admits \( O(1) \)-round algorithms
based on very simple sampling approach
Lattanzi et al. [SPAA’11]
Local Memory in MPC

$M$ machines
$S$ memory per machine
$M \cdot S = \tilde{O}(m + n)$

Strongly Sublinear Memory:
$S = \tilde{O}(n^\delta)$, $0 \leq \delta < 1$
No machine sees all nodes.

Usual assumption

Linear Memory:
$S = \tilde{O}(n)$
Machines see all nodes.

Usual assumption

Superlinear Memory:
$S = \tilde{O}(n^{1+\delta})$, $0 < \delta \leq 1$
Machines see all nodes.

Often trivial

for many problems, admits $O(1)$-round algorithms based on very simple sampling approach
Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\( S \) \hspace{1cm} \( \tilde{O}(n^\delta) \) \hspace{1cm} \( \tilde{\Theta}(n) \) \hspace{1cm} \( \tilde{\Omega}(n^{1+\delta}) \)

### Strongly Sublinear Memory

\[ S = \tilde{O}(n^\delta), \; 0 \leq \delta < 1 \]

No machine sees all nodes.

For most problems, only direct simulation of LOCAL/PRAM algorithms known.

### Linear Memory

\[ S = \tilde{O}(n) \]

Machines see all nodes.

Usual assumption

Often unrealistic

- \( \tilde{O}(n) \) prohibitively large
- Sparse graphs trivial

### Superlinear Memory

\[ S = \tilde{O}(n^{1+\delta}), \; 0 < \delta \leq 1 \]

Machines see all nodes.

Often trivial

For many problems, admits O(1)-round algorithms based on very simple sampling approach

Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

**Strongly Sublinear Memory:**
\[ S = \tilde{O}(n^\delta), \ 0 \leq \delta < 1 \]
No machine sees all nodes.

Regular assumption

for most problems, only direct simulation of LOCAL/PRAM algorithms known

**Linear Memory:**
\[ S = \tilde{O}(n) \]
Machines see all nodes.

usual assumption

often unrealistic
- \( \tilde{O}(n) \) prohibitively large
- sparse graphs trivial

**Superlinear Memory:**
\[ S = \tilde{O}(n^{1+\delta}), \ 0 < \delta \leq 1 \]
Machines see all nodes.

often trivial

for many problems, admits O(1)-round algorithms based on very simple sampling approach

Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\[ \tilde{O}(n^\delta) \quad \tilde{\Theta}(n) \quad \tilde{\Omega}(n^{1+\delta}) \]

**Strongly Sublinear Memory:**
\[ S = \tilde{O}(n^\delta), \ 0 \leq \delta < 1 \]
No machine sees all nodes.

**Linear Memory:**
\[ S = \tilde{O}(n) \]
Machines see all nodes.

**Superlinear Memory:**
\[ S = \tilde{O}(n^{1+\delta}), \ 0 < \delta \leq 1 \]
Machines see all nodes.

for most problems, only direct simulation of LOCAL/PRAM algorithms known

usual assumption

often unrealistic
- \( \tilde{O}(n) \) prohibitively large
- sparse graphs trivial

often trivial

for many problems, admits O(1)-round algorithms based on very simple sampling approach

Lattanzi et al. [SPAA’11]
Local Memory in MPC

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

### Strongly Sublinear Memory

\[ S = \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \]
No machine sees all nodes.

- Algorithms have been stuck at this linear-memory barrier!
- Fundamentally?

### Linear Memory

\[ S = \tilde{O}(n) \]
Machines see all nodes.

- usual assumption
- often unrealistic
- \( \tilde{O}(n) \) prohibitively large
- sparse graphs trivial

### Superlinear Memory

\[ S = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]
Machines see all nodes.

- often trivial
- for many problems, admits \( O(1) \)-round algorithms
- based on very simple sampling approach
- Lattanzi et al. [SPAA’11]
Breaking the Linear-Memory Barrier:
Breaking the Linear-Memory Barrier:
Efficient MPC Graph Algorithms with Strongly Sublinear Memory
Breaking the Linear-Memory Barrier:

Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \text{ local memory} \]
\[ M = O(m/n^\delta) \text{ machines} \]
\[ \text{poly log log } n \text{ rounds} \]
Breaking the Linear-Memory Barrier:
Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \] local memory
\[ M = O\left(\frac{m}{n^\delta}\right) \] machines
\[ \text{poly log log } n \] rounds

Ghaffari, Kuhn, Uitto [FOCS’19]

Conditional Lower Bound
\[ \Omega(\log \log n) \] rounds
Breaking the Linear-Memory Barrier:
Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \text{ local memory} \]
\[ M = O\left(\frac{m}{n^\delta}\right) \text{ machines} \]
\[ \text{poly log log } n \text{ rounds} \]
Breaking the Linear-Memory Barrier:

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[ S = O(n^\delta) \] local memory

\[ M = O(m/n^\delta) \] machines

poly log log \( n \) rounds

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph
Breaking the Linear-Memory Barrier:

Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \] local memory
\[ M = O(m/n^\delta) \] machines
\[ \text{poly log log } n \] rounds

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph
Breaking the Linear-Memory Barrier:

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[
S = O(n^\delta) \text{ local memory}
\]
\[
M = O\left(\frac{m}{n^\delta}\right) \text{ machines}
\]
\[
\text{poly log log } n \text{ rounds}
\]

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph
Breaking the Linear-Memory Barrier:

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[
S = O(n^\delta) \text{ local memory}
\]
\[
M = O(m/n^\delta) \text{ machines}
\]
\[
\text{poly log log } n \text{ rounds}
\]

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph
Breaking the Linear-Memory Barrier:

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[ S = O(n^\delta) \] local memory
\[ M = O(m/n^\delta) \] machines
\[ \text{poly log log } n \] rounds

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
Breaking the Linear-Memory Barrier:

Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \] local memory
\[ M = O(m/n^\delta) \] machines
\[ \text{poly log log } n \] rounds

Imposed Locality:
machines see only subset of nodes, regardless of sparsity of graph

Our Approach to Cope with Locality:
enhance \textbf{LOCAL algorithms} with \textbf{global communication}
Breaking the Linear-Memory Barrier:
Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \] local memory
\[ M = O(m/n^\delta) \] machines
\[ \text{poly log log } n \] rounds

Imposed Locality:
machines see only subset of nodes, regardless of sparsity of graph

Our Approach to Cope with Locality:
enhance \text{LOCAL algorithms} with \text{global communication}
Efficient MPC Graph Algorithms with Strongly Sublinear Memory

Breaking the Linear-Memory Barrier:

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[ S = O(n^\delta) \] local memory

\[ M = O(m/n^\delta) \] machines

poly log log \(n\) rounds

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
enhance **LOCAL algorithms** with **global communication**
Breaking the Linear-Memory Barrier: 

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[ S = O(n^\delta) \text{ local memory} \]
\[ M = O\left(\frac{m}{n^\delta}\right) \text{ machines} \]
\[ \text{poly log log } n \text{ rounds} \]

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
- enhance **LOCAL algorithms** with **global communication**
  - exponentially faster than LOCAL algorithms due to shortcuts
Breaking the Linear-Memory Barrier:

**Efficient MPC Graph Algorithms with Strongly Sublinear Memory**

\[
S = O(n^\delta) \text{ local memory}
\]
\[
M = O(m/n^\delta) \text{ machines}
\]
\[
poly \log \log n \text{ rounds}
\]

**Imposed Locality:**
machines see only subset of nodes, regardless of sparsity of graph

**Our Approach to Cope with Locality:**
enhance **LOCAL algorithms** with **global communication**
- exponentially faster than LOCAL algorithms due to shortcuts

**best we can hope for**

GKU [FOCS’19]
Breaking the Linear-Memory Barrier:

Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^\delta) \] local memory
\[ M = O\left(\frac{m}{n^\delta}\right) \] machines
\[ \text{poly log log } n \] rounds

Imposed Locality:
machines see only subset of nodes, regardless of sparsity of graph

Our Approach to Cope with Locality:
enhance \text{LOCAL algorithms} with \text{global communication}
- exponentially faster than \text{LOCAL algorithms} due to shortcuts
Breaking the Linear-Memory Barrier:
Efficient MPC Graph Algorithms with Strongly Sublinear Memory

\[ S = O(n^{\delta}) \] local memory
\[ M = O(m/n^{\delta}) \] machines
\[ \text{poly log log } n \] rounds

Imposed Locality:
machines see only subset of nodes, regardless of sparsity of graph

Our Approach to Cope with Locality:
have LOCAL algorithms with global communication
- exponentially faster than LOCAL algorithms due to shortcuts
- polynomially less memory than most MPC algorithms
Problem: Maximal Independent Set (MIS)
Maximal Independent Set (MIS)
Maximal Independent Set (MIS)
Maximal Independent Set (MIS)
Maximal Independent Set (MIS)

Independent Set:
set of non-adjacent nodes
Maximal Independent Set (MIS)

**Independent Set:**
set of non-adjacent nodes

**Maximal:**
no node can be added without violating independence
MIS: State of the Art

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\[ \tilde{S} = \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \]
No machine sees all nodes.

\[ \tilde{S} = \tilde{O}(n) \]
Machines see all nodes.

\[ \tilde{S} = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]
Machines see all nodes.
MIS: State of the Art

$M$ machines
$S$ memory per machine
$M \cdot S = \tilde{O} (m + n)$

Strongly Sublinear Memory:
$S = \tilde{O} (n^\delta), 0 \leq \delta < 1$
No machine sees all nodes.

Linear Memory:
$S = \tilde{O} (n)$
Machines see all nodes.

Superlinear Memory:
$S = \tilde{O} (n^{1+\delta}), 0 < \delta \leq 1$
Machines see all nodes.

$Lattanzi et al. \ [SPAA’11]$
$O(1)$
MIS: State of the Art

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O} (m + n) \]

\[ S = \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \]
\[ \text{No machine sees all nodes.} \]

\[ S = \tilde{O}(n) \]
\[ \text{Machines see all nodes.} \]

\[ S = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]
\[ \text{Machines see all nodes.} \]

Ghaffari et al. [PODC’18]
\[ O (\log \log n) \]

Lattanzi et al. [SPAA’11]
\[ O (1) \]
MIS: State of the Art

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

- **Strongly Sublinear Memory:**
  \[ S = \tilde{O}(n^\delta), \ 0 \leq \delta < 1 \]
  No machine sees all nodes.

- **Linear Memory:**
  \[ S = \tilde{O}(n) \]
  Machines see all nodes.

- **Superlinear Memory:**
  \[ S = \tilde{O}(n^{1+\delta}), \ 0 < \delta \leq 1 \]
  Machines see all nodes.

- **Luby’s Algorithm**
  \[ O(\log n) \]

- **Ghaffari et al. [PODC’18]**
  \[ O(\log \log n) \]

- **Lattanzi et al. [SPAA’11]**
  \[ O(1) \]
MIS: State of the Art

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

Strongly Sublinear Memory:
\[ S = \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \]
No machine sees all nodes.

Linear Memory:
\[ S = \tilde{O}(n) \]
Machines see all nodes.

Superlinear Memory:
\[ S = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]
Machines see all nodes.

- Luby’s Algorithm
  \[ O(\log n) \]

- Ghaffari et al. [PODC’18]
  \[ O(\log \log n) \]

- Ghaffari and Uitto [SODA’19]
  \[ \tilde{O}(\sqrt{\log n}) \]

- Lattanzi et al. [SPAA’11]
  \[ O(1) \]
### MIS: State of the Art on Trees

<table>
<thead>
<tr>
<th>Strongly Sublinear Memory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \tilde{O}(n^\delta)$, $0 \leq \delta &lt; 1$</td>
</tr>
<tr>
<td>No machine sees all nodes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linear Memory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \tilde{O}(n)$</td>
</tr>
<tr>
<td>Machines see all nodes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Superlinear Memory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \tilde{O}(n^{1+\delta})$, $0 &lt; \delta \leq 1$</td>
</tr>
<tr>
<td>Machines see all nodes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luby’s Algorithm</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Ghaffari et al. [PODC’18]</td>
<td>$O(\log \log n)$</td>
</tr>
<tr>
<td>Ghaffari and Uitto [SODA’19]</td>
<td>$\tilde{O}(\sqrt{\log n})$</td>
</tr>
<tr>
<td>Lattanzi et al. [SPAA’11]</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

$M$ machines
$S$ memory per machine
$M \cdot S = \tilde{O}(m + n)$
MIS: State of the Art on Trees

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

Strongly Sublinear Memory:
\[ S = \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \]
No machine sees all nodes.

Linear Memory:
\[ S = \tilde{O}(n) \]
Machines see all nodes.

Superlinear Memory:
\[ S = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]
Machines see all nodes.

- **Luby’s Algorithm**
  \[ O(\log n) \]

- **Ghaffari and Uitto [SODA’19]**
  \[ \tilde{O}(\sqrt{\log n}) \]

- **Trivial solution**
  \[ O(1) \]

- **Trivial solution**
  \[ O(1) \]
MIS: State of the Art on Trees

\[ M \text{ machines} \]
\[ S \text{ memory per machine} \]
\[ M \cdot S = \tilde{O}(m + n) \]

\[
\begin{align*}
S &= \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \\
\text{No machine sees all nodes.}
\end{align*}
\]

\[
\begin{align*}
S &= \Theta(n) \\
\text{Machines see all nodes.}
\end{align*}
\]

\[
\begin{align*}
S &= \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \\
\text{Machines see all nodes.}
\end{align*}
\]

- **Strongly Sublinear Memory:**
  \[ S = \tilde{O}(n^\delta), \quad 0 \leq \delta < 1 \]
  No machine sees all nodes.

- **Linear Memory:**
  \[ S = \Theta(n) \]
  Machines see all nodes.

- **Superlinear Memory:**
  \[ S = \tilde{O}(n^{1+\delta}), \quad 0 < \delta \leq 1 \]
  Machines see all nodes.

**Our Result**

\[ O(\log^3 \log n) \]

**Ghaffari and Uitto [SODA’19]**

\[ \tilde{O}(\sqrt{\log n}) \]

**Trivial solution**

\[ O(1) \]
Our Result

$O(\log^3 \log n)$-round MPC algorithm with $S = \tilde{O}(n^\delta)$ memory that w.h.p. computes MIS on trees.
Our Result

\(\tilde{O}(\sqrt{\log n})\) rounds

\(S = \tilde{O}(n^\delta)\) memory

Ghaffari and Uitto [SODA'19]

\(O(\log^3 \log n)\)-round MPC algorithm

with \(S = \tilde{O}(n^\delta)\) memory that w.h.p. computes MIS on trees.
Our Result

\[ \tilde{O}\left(\sqrt{\log n}\right) \text{ rounds} \]
\[ S = \tilde{O}\left(n^\delta\right) \text{ memory} \]

\[ O(\log \log n) \text{ rounds} \]
\[ S = \tilde{O}(n) \text{ memory} \]

Ghaffari and Uitto [SODA’19]

\[ O(\log^3 \log n) \]-round MPC algorithm
with \[ S = \tilde{O}\left(n^\delta\right) \text{ memory} \] that w.h.p. computes MIS on trees.

Ghaffari et al. [PODC’18]
Our Result

\[ \tilde{O}(\sqrt{\log n}) \] rounds
\[ S = \tilde{O}(n^\delta) \] memory

\[ O(\log \log n) \] rounds
\[ S = \tilde{O}(n) \] memory

Ghaffari and Uitto [SODA’19]

\[ O(\log^3 \log n) \]-round MPC algorithm

with \[ S = \tilde{O}(n^\delta) \] memory that w.h.p. computes MIS on trees.

Conditional \[ \Omega(\log \log n) \]-round lower bound for \[ S = \tilde{O}(n^\delta) \]

Ghaffari, Kuhn, and Uitto [FOCS’19]
Algorithm
Algorithm Outline
Algorithm Outline
Algorithm Outline

1) Shattering

- Degree Reduction
- LOCAL Shattering

2) Post-Shattering

- Gathering of Components
- Local Computation
Algorithm Outline

1) Shattering

main LOCAL technique

Beck [RSA’91]
Algorithm Outline

1) Shattering
break graph into small components

main LOCAL technique
*Beck* [RSA’91]
Algorithm Outline

1) **Shattering**
   break graph into small components
   main LOCAL technique
   *Beck* [RSA’91]
Algorithm Outline

1) **Shattering**
   break graph into small components

   main LOCAL technique
   *Beck [RSA’91]*
Algorithm Outline

1) **Shattering**
   break graph into small components

   main LOCAL technique
   *Beck [RSA’91]*

2) **Post-Shattering**
   solve problem on remaining components
Algorithm Outline

1) **Shattering**
   break graph into small components
   
   main LOCAL technique
   *Beck [RSA’91]*

2) **Post-Shattering**
   solve problem on remaining components
   i) **Gathering of Components**
Algorithm Outline

1) Shattering
   break graph into small components

   main LOCAL technique
   Beck [RSA’91]

2) Post-Shattering
   solve problem on remaining components
   i) Gathering of Components
Algorithm Outline

1) Shattering
   break graph into small components
   main LOCAL technique
   Beck [RSA’91]

2) Post-Shattering
   solve problem on remaining components
   i) Gathering of Components
Algorithm Outline

1) **Shattering**
   break graph into small components

   main LOCAL technique
   *Beck* [RSA’91]

2) **Post-Shattering**
   solve problem on remaining components
   i) Gathering of Components
   ii) Local Computation
Algorithm Outline

1) **Shattering**  
   break graph into small components

   main LOCAL technique  
   *Beck [RSA’91]*

2) **Post-Shattering**  
   solve problem on remaining components

   i) **Gathering of Components**
   ii) **Local Computation**
Algorithm Outline

1) **Shattering**
   break graph into small components

2) **Post-Shattering**
   solve problem on remaining components
   i) **Gathering of Components**
   ii) **Local Computation**
Algorithm Outline

1) Shattering
   break graph into small components

   ii) LOCAL Shattering Ghaffari [SODA’16]

2) Post-Shattering
   solve problem on remaining components

   i) Gathering of Components
   ii) Local Computation
Algorithm Outline

1) **Shattering**
   break graph into small components
   
   i) **Degree Reduction**
   
   ii) **LOCAL Shattering** Ghaffari [SODA’16]

2) **Post-Shattering**
   solve problem on remaining components

   i) **Gathering of Components**
   
   ii) **Local Computation**
Polynomial Degree Reduction:
Subsample-and-Conquer
Polynomial Degree Reduction: 
**Subsample-and-Conquer**

**Subsample**

**Conquer**
Polynomial Degree Reduction: 
**Subsample-and-Conquer**

**Subsample**

**Conquer**
Polynomial Degree Reduction: Subsample-and-Conquer

**Subsample**
- subsample nodes independently

**Conquer**

- compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsampling nodes independently

**Conquer**
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
- subsample nodes independently

**Conquer**
- compute random MIS in subsampled graph
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
Polynomial Degree Reduction: Subsample-and-Conquer

Subsample
subsample nodes independently

Conquer
compute random MIS in subsampled graph
- gather connected components
Polynomial Degree Reduction: 
**Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
  
  ![Diagram](attachment:image.png)
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample **nodes** independently

**Conquer**
compute **random MIS** in **subsampled graph**
- gather connected components
- locally compute random 2-coloring
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**

subsample nodes independently

**Conquer**

compute random MIS in subsampled graph

- gather connected components
- locally compute random 2-coloring
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample *nodes* independently

**Conquer**
compute *random MIS* in *subsampled graph*
- gather connected components
- locally compute random 2-coloring
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS
Polynomial Degree Reduction: Subsample-and-Conquer

Subsample
subsample nodes independently

Conquer
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS
Polynomial Degree Reduction: 
**Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS
Polynomial Degree Reduction: Subsample-and-Conquer

Subsample
subsample nodes independently

Conquer
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS

Non-subsampled High-Degree Node
Polynomial Degree Reduction: Subsample-and-Conquer

Subsample
subsample nodes independently

Conquer
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS

Non-subsampled High-Degree Node
Polynomial Degree Reduction: \textbf{Subsample-and-Conquer}

\textbf{Subsample}

subsample \textit{nodes} independently

\textbf{Conquer}

compute \textit{random MIS} in \textit{subsampled graph}

- gather connected components
- locally compute random 2-coloring
- add a color class to \textit{MIS}

Non-subsampled \textbf{High-Degree Node}

- w.h.p. has many \textit{subsampled neighbors}
Polynomial Degree Reduction: 
**Subsample-and-Conquer**

**Subsample**
subsample *nodes* independently

**Conquer**
compute *random MIS* in *subsampled graph*
- gather connected components
- locally compute random 2-coloring
- add a color class to *MIS*

Non-subsampled **High-Degree Node**
- w.h.p. has many *subsampled neighbors*

independence due to restriction to trees!
Polynomial Degree Reduction:
**Subsample-and-Conquer**

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS

Non-subsampled **High-Degree Node**
- w.h.p. has many subsampled neighbors

independence due to restriction to trees!
Polynomial Degree Reduction: Subsample-and-Conquer

**Subsample**
subsample nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS

Non-subsampled High-Degree Node
- w.h.p. has many subsampled neighbors
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**
subsampling nodes independently

**Conquer**
compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS

**Non-subsampled High-Degree Node**
- w.h.p. has many subsampled neighbors
- thus w.h.p. has at least one MIS neighbor
Polynomial Degree Reduction: **Subsample-and-Conquer**

**Subsample**

subsample nodes independently

**Conquer**

compute random MIS in subsampled graph
- gather connected components
- locally compute random 2-coloring
- add a color class to MIS

**Non-subsampled High-Degree Node**
- w.h.p. has many subsampled neighbors
- thus w.h.p. has at least one MIS neighbor
- hence will be removed from the graph
Algorithm Outline

1) **Shattering**
   break graph into small components
   
   i) **Degree Reduction** *Iterated Subsample-and-Conquer*
   
   ii) **LOCAL Shattering** *Ghaffari* [SODA’16]

2) **Post-Shattering**
   solve problem on remaining components
   
   i) **Gathering of Components** *Distributed Union-Find*
   
   ii) **Local Computation**
Conclusion
and
Open Questions
MODEL: Sublinear-Memory MPC

\[ S = \tilde{O}(n^\delta) \] local memory
poly log log \( n \) rounds
MODEL: Sublinear-Memory MPC

\[ S = \tilde{O}(n^\delta) \] local memory
poly log log \( n \) rounds
**Model:** Sublinear-Memory MPC

\[ S = \tilde{O}(n^\delta) \] local memory
poly log log \( n \) rounds

**Approach:** LOCAL algorithms &
global communication
**MODEL:** Sublinear-Memory MPC

\[ S = \tilde{O}(n^\delta) \] local memory

poly log log \( n \) rounds

**APPROACH:** LOCAL algorithms &
global communication

**TECHNIQUE:** Shattering
MODEL: Sublinear-Memory MPC
\[ S = \tilde{O}(n^\delta) \text{ local memory} \]
\[ \text{poly log log } n \text{ rounds} \]

APPROACH: LOCAL algorithms &
global communication

TECHNIQUE: Shattering

PROBLEM: MIS
on trees
**MODEL:** Sublinear-Memory MPC

\[ S = \tilde{O}(n^\delta) \] local memory
poly log log \( n \) rounds

**APPROACH:** LOCAL algorithms &
global communication

**TECHNIQUE:** Shattering

**PROBLEM:** MIS

on trees

other graph problems?
more general graph families?
**Model:** Sublinear-Memory MPC

\[ S = \tilde{O}(n^\delta) \] local memory
dpoly log log \( n \) rounds

**Approach:** LOCAL algorithms &
global communication

**Technique:** Shattering

**Problem:** MIS

on trees

other graph problems?
more general graph families?

MIS & Matching for locally sparse graphs
in follow-up work [PODC’19]
**Model:** Sublinear-Memory MPC

\[ S = \tilde{O}(n^{\delta}) \] local memory
poly log log \( n \) rounds

**Approach:** LOCAL algorithms &
global communication

**Technique:** Shattering

**Problem:** MIS
on trees

other LOCAL techniques?

other graph problems?
more general graph families?

MIS & Matching for locally sparse graphs
in follow-up work [PODC’19]
<table>
<thead>
<tr>
<th><strong>Model:</strong></th>
<th>Sublinear-Memory MPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = \tilde{O}(n^\delta)$ local memory</td>
<td></td>
</tr>
<tr>
<td>poly log log log $n$ rounds</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Approach:</strong></th>
<th>LOCAL algorithms &amp; global communication</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Technique:</strong></th>
<th>Shattering</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Problem:</strong></th>
<th>MIS on trees</th>
</tr>
</thead>
</table>

- MIS & Matching for locally sparse graphs
- in follow-up work [PODC’19]
**Model:** Sublinear-Memory MPC

\[ S = \tilde{O}(n^{\delta}) \] local memory
poly log log \( n \) rounds

**Approach:** LOCAL algorithms &
global communication

**Technique:** Shattering

**Problem:** MIS
on trees

MIS & Matching for locally sparse graphs
in follow-up work [PODC’19]