Molecular Computation
An Algorithmic Approach

Rati Gelashvili

Joint work with
Dan Alistarh (ETH), David Eisenstat (Google),
James Aspnes (Yale), Milan Vojnovic (MSR), Ron Rivest (MIT)
Distributed Systems

Ingredients:
Distributed Systems

Ingredients:

• Nodes
Distributed Systems

Ingredients:

• Nodes
• Communication
Distributed Systems

Ingredients:

• Nodes
• Communication
• Computation
Computational Model
Population Protocols [AADFP’04]
Computational Model
Population Protocols [AADFP’04]

- **Nodes** are *simple, identical agents*
  - Each node is *the same* finite state automaton
  - For example: a molecule
Computational Model
Population Protocols [AADFP’04]

• **Nodes** are *simple, identical agents*  
  • Each node is *the same* finite state automaton  
  • For example: a molecule

• **Interactions** are *pairwise*, and follow a *fair scheduler*  
  • Usually considered *uniform random*  
  • Nodes update their state following interactions
Computational Model
Population Protocols [AADFP’04]

- **Nodes** are *simple, identical agents*
  - Each node is *the same* finite state automaton
  - For example: a molecule

- **Interactions** are *pairwise*, and follow a *fair scheduler*
  - Usually considered *uniform random*
  - Nodes update their state following interactions

- **Computation** is performed *collectively*
  - The system *should converge* to configurations satisfying meaningful predicates
  - No “fixed” decision time
Computational Model
Population Protocols [AADFP’04]

• **Nodes** are *simple, identical agents*
  - Each node is *the same* finite state automaton
  - For example: a molecule

• **Interactions** are *pairwise*, and follow a *fair scheduler*
  - Usually considered *uniform random*
  - Nodes update their state following interactions

• **Computation** is performed *collectively*
  - The system *should converge* to configurations satisfying meaningful predicates
  - No “fixed” decision time

• A.k.a. Chemical Reaction Networks
Complexity

1. Time
   • **Round** = a *single pair* interacts
     • Chosen uniformly at random
   • **Parallel convergence time**
     • \#rounds to convergence / \# nodes
     • Alternative continuous-time definition exists
Complexity

1. Time
   • Round = a single pair interacts
     • Chosen uniformly at random
   • Parallel convergence time
     • #rounds to convergence / # nodes
     • Alternative continuous-time definition exists

2. Space
   • Number of distinct states per automaton
   • Alternatively, #memory bits to encode state
More Precisely: Communication

Courtesy of the Microsoft Research Biological Computation Group
More Precisely: Communication

Courtesy of the Microsoft Research Biological Computation Group
More Precisely: Communication

Courtesy of the Microsoft Research Biological Computation Group
What can we compute?

We can perform interactions of the type:
What can we compute?

We can perform interactions of the type:

Example: the **OR** function

- **Initial** states: 0 or 1
- **Final** state:
  - If there exists a 1, then all 1.
  - Otherwise, all 0
- **Protocol:**
What can we compute?

We can perform interactions of the type:

Example: the **OR** function

- **Initial** states: 0 or 1
- **Final** state:
  - If there exists a 1, then all 1.
  - Otherwise, all 0
- **Protocol:**

```
0  +  0
    ↓
0  0
```
What can we compute?

We can perform interactions of the type:

Example: the OR function

- **Initial** states: 0 or 1
- **Final** state:
  - If there exists a 1, then all 1.
  - Otherwise, all 0

- **Protocol:**
What can we compute?

We can perform interactions of the type:

Example: the **OR** function

- **Initial** states: 0 or 1
- **Final** state:
  - If there exists a 1, then all 1.
  - Otherwise, all 0
- **Protocol:**

```
0   0
↓   ↓
0   0
```

```
1   1
↓   ↓
1   1
```

```
0   1
↓   ↓
1   1
```

```
0   1
↓   ↓
1   1
```
What can we compute?

We can perform interactions of the type:

Example: the OR function

- **Initial** states: 0 or 1
- **Final** state:
  - If there exists a 1, then all 1.
  - Otherwise, all 0
- **Protocol:**

```
A + B
C ↓ D
```

```
0 + 0
0 0
1 + 1
1 1
0 + 1
1 1
1 + 0
1 1
```
The Majority Function

Majority ("Consensus")

• **Initial** states A, B

• **Output:**
  • A if $#A > #B$ initially.
  • B, otherwise.
The Majority Function

Majority ("Consensus")

• Initial states A, B

• Output:
  • A if #A > #B initially.
  • B, otherwise.

• Fundamental task
  • Complexity: [AAE08] & [DV12]; [PVV09] & [MNRS14]
  • Natural computation:
    the cell cycle switch implements approximate majority [CC12]
  • Implementation in DNA: [CDS+13, Nature Nanotechnology]
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

- Protocol:
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

• Protocol:
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

• Protocol:
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

• Protocol:
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

- Protocol:
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

• Protocol:

Discrepancy/margin: 
\[ \varepsilon = \frac{|\#A - \#B|}{n} \]
Can be as small as 
\[ \varepsilon = O\left(\frac{1}{n}\right) \].
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

• Protocol:

Discrepancy/margin: 
\[ \varepsilon = \frac{|\#A - \#B|}{n} \]
Can be as small as \( \varepsilon = O(1/n) \).

Theorem: Given \( n \) nodes and discrepancy \( \varepsilon \), the running time of 4EM is \( O( (\log n) / \varepsilon ) \).
Solving Majority

4-State Exact Majority [PVV09] [MNRS14]

- Protocol:

  Discrepancy/margin:
  \( \varepsilon = \frac{|\#A - \#B|}{n} \)
  Can be as small as \( \varepsilon = O(1/n) \).

**Theorem:** Given \( n \) nodes and discrepancy \( \varepsilon \), the running time of 4EM is \( O(\frac{\log n}{\varepsilon}) \).

Can be \( \Theta(n \log n) \) if \( \varepsilon = \text{constant} / n \).
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:
Solving Majority Approximately

• 3-state Approximate Majority [AAE08] [DV12]
• The protocol:
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:
Solving Majority \textit{Approximately}

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:

- Execution:
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:
- Execution:
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:

  - Execution:
Solving Majority *Approximately*

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:
  - Execution:

**Theorem:** Given $n$ nodes and discrepancy $\epsilon > \log n/\sqrt{n}$, the running time of 3AM is $O(\text{polylog } n)$, and the protocol is correct with high probability.
Solving Majority Approximately

- 3-state Approximate Majority [AAE08] [DV12]
- The protocol:
- Execution:

Theorem: Given $n$ nodes and discrepancy $\epsilon > \log n / \sqrt{n}$, the running time of 3AM is $O(\text{polylog } n)$, and the protocol is correct with high probability.

Error probability can be as high as constant for lower discrepancy.
## The Status

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reliability</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Four-State Protocol</td>
<td>Exact</td>
<td>Slow (super-linear)</td>
</tr>
<tr>
<td>The Three-State Protocol</td>
<td>Flaky (Up to Constant Error)</td>
<td>Fast (poly-logarithmic)</td>
</tr>
</tbody>
</table>
## Average&Conquer

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reliability</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Four-State Protocol</td>
<td>Exact</td>
<td>Slow (super-linear)</td>
</tr>
<tr>
<td>The Three-State Protocol</td>
<td>Flaky (Up to Constant Error)</td>
<td>Fast (poly-logarithmic)</td>
</tr>
<tr>
<td>Average&amp;Conquer [PODC 2015]</td>
<td>Exact</td>
<td>Fast (poly-logarithmic)</td>
</tr>
</tbody>
</table>
## Average&Conquer

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reliability</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Four-State Protocol</td>
<td>Exact</td>
<td>Slow (super-linear)</td>
</tr>
<tr>
<td>The Three-State Protocol</td>
<td>Flaky (Up to Constant Error)</td>
<td>Fast (poly-logarithmic)</td>
</tr>
<tr>
<td>Average&amp;Conquer [PODC 2015] (Super-Constant State Space)</td>
<td>Exact</td>
<td>Fast (poly-logarithmic)</td>
</tr>
</tbody>
</table>
The Plan

• Population Protocols
• The Majority Problem
  • 4EM
  • 3AM
  • Average-and-Conquer (AVC)
  • Quantized AVC
• Impossibility Results
• Open Questions
• Leader Election Problem
Simplified AVC: Main Ideas

- Each state corresponds to a value ("confidence level")
  - Strong states (non-negative value):
    - Positive $\rightarrow$ A
    - Negative $\rightarrow$ B
  - Weak: value +/- 0
- All nodes start with absolute value $m > 0$
  - $+m$ if A
  - $-m$ if B
- Two interaction types:
  - Averaging: strong (non-zero) nodes average out their values
  - Conquer: strong (non-zero) nodes bring weak nodes to "their side"
- Output:
  - If positive or $+0$, then A
  - If negative or $-0$, then B
AVC in Action

Initially: $+m$ or $-m$, odd integers
Strong states: non-zero absolute value.
Weak states: value zero ($+/-$).
AVC in Action

Initially: \(+m\) or \(-m\), odd integers
**Strong** states: non-zero absolute value.
**Weak** states: value zero (+/-).

Averaging:
- Whenever two strong nodes meet, they average values
AVC in Action

Initially: $+m$ or $-m$, odd integers

**Strong** states: non-zero absolute value.

**Weak** states: value zero (+/-).

**Averaging:**
- Whenever two strong nodes meet, they average values
AVC in Action

Initially: \(+m\) or \(-m\), odd integers

**Strong** states: non-zero absolute value.

**Weak** states: value zero (+/-).

Averaging:
- Whenever two strong nodes meet, they average values
AVC in Action

Initially: +m or -m, odd integers
Strong states: non-zero absolute value.
Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values
AVC in Action

Initially: +m or -m, odd integers
Strong states: non-zero absolute value.
Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values
AVC in Action

Initially: +m or -m, odd integers
Strong states: non-zero absolute value.
Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values
AVC in Action

Initially: $+m$ or $-m$, odd integers
**Strong** states: non-zero absolute value.
**Weak** states: value zero ($+/-$).

Averaging:
- Whenever two strong nodes meet, they average values

\[ +m \]
\[ +m - 1 \]
\[ \ldots \]
\[ +2 \]
\[ +1 \]
\[ +0 \]
\[ -m \]
\[ -m + 1 \]
\[ \ldots \]
\[ -2 \]
\[ -1 \]
\[ -0 \]
Initially: +m or -m, odd integers

Strong states: non-zero absolute value.

Weak states: value zero (+/-).

Averaging:

• Whenever two strong nodes meet, they average values
AVC in Action

Initially: +m or -m, odd integers
Strong states: non-zero absolute value.
Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values
AVC in Action

Initially: +m or -m, odd integers
**Strong** states: non-zero absolute value.
**Weak** states: value zero (+/-).

Averaging:
- Whenever two strong nodes meet, they average values
AVC in Action

Initially: +m or -m, odd integers

Strong states: non-zero absolute value.

Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values

Conquer:
• Strong nodes sway weak nodes towards their decision.
AVC in Action

Initially: +m or -m, odd integers
Strong states: non-zero absolute value.
Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values

Conquer:
• Strong nodes sway weak nodes towards their decision.
AVC in Action

Initially: +m or -m, odd integers
Strong states: non-zero absolute value.
Weak states: value zero (+/-).

Averaging:
• Whenever two strong nodes meet, they average values

Conquer:
• Strong nodes sway weak nodes towards their decision.
AVC in Action

Initially: +m or -m, odd integers

**Strong** states: non-zero absolute value.

**Weak** states: value zero (+/-).

**Averaging:**
- Whenever two strong nodes meet, they average values

**Conquer:**
- **Strong** nodes sway **weak** nodes towards their decision.

Note: For m = 1, we obtain a variant of 4EM.
AVC in Action

Initially: \(+m\) or \(-m\), odd integers
**Strong** states: non-zero absolute value.
**Weak** states: value zero (\(+/-\)).

Averaging:
• Whenever two strong nodes meet, they average values

Conquer:
• **Strong** nodes sway **weak** nodes towards their decision.

Note: For \(m = 1\), we obtain a variant of 4EM.

Disclaimer: original protocol is more complicated for technical reasons.
Summing up
Summing up

**Theorem 1 [AGV15]:** Given fixed $m < n$, AVC solves majority exactly in expected parallel time $O(\log n / (m \epsilon) + \log n \log m )$, using $s = O( m + \log n \log m )$ total states.
Summing up

Theorem 1 [AGV15]: Given fixed $m < n$, AVC solves majority exactly in expected parallel time $O(\log n / (m \varepsilon) + \log n \log m)$, using $s = O( m + \log n \log m)$ total states.

- In short:
  - If $m \approx 1 / \varepsilon$, then running time is always poly-logarithmic
  - If $\varepsilon = 1 / n$, then $m$ needs to be linear in $n$
  - $10^{23}$ molecules $\rightarrow O(10^{23})$ states?!

- $10^{23}$ molecules $\rightarrow O(23^2$ states $)$
- The idea: quantize integer states to powers of two
Summing up

**Theorem 1 [AGV15]:** Given fixed $m < n$, AVC solves majority exactly in expected parallel time $O(\log n / (m \epsilon) + \log n \log m )$, using $s = O( m + \log n \log m )$ total states.

- In short:
  - If $m \approx 1 / \epsilon$, then running time is always poly-logarithmic
  - If $\epsilon = 1 / n$, then $m$ needs to be linear in $n$
  - $10^{23}$ molecules $\rightarrow O(10^{23})$ states?!

**Theorem 2 [AAEGR16]:** logAVC solves majority exactly in expected parallel time $O(\log^3 n )$, using $s = O( \log^2 n )$ total states.

- $10^{23}$ molecules $\rightarrow O(23^2 \text{ states })$
- The idea: **quantize** integer states to powers of two
Is AVC any good?

Results are for $\epsilon = O(1 / n)$

Legend:
- **Blue** = 3AM
- **Green** = 4EM
- **Yellow** = AVC / logAVC
Is AVC any good?

Results are for $\epsilon = O(1 / n)$
Legend:
- Blue = 3AM
- Green = 4EM
- Yellow = AVC / logAVC

Is AVC implementable?
Is AVC any good?

Results are for $\varepsilon = O(1 / n)$
Legend:
Blue = 3AM
Green = 4EM
Yellow = AVC / logAVC

Challenging: currently, small constant number of states implementable.
Time-Space Trade-Offs
Time-Space Trade-Offs

Theorem A: Any protocol using \( s < \frac{1}{2} \log \log n \) states per node and solving majority with discrepancy \( \varepsilon \) must have expected stabilization time greater than \( \frac{n}{(2^s + \varepsilon n)^2} \).
Time-Space Trade-Offs

Theorem A: Any protocol using $s < \frac{1}{2} \log \log n$ states per node and solving majority with discrepancy $\epsilon$ must have expected stabilization time 
\[ > \frac{n}{(2^s + \epsilon n)^2}. \]

In particular:

- If $s = \text{constant}$ and $\epsilon n = \text{constant}$, then stabilization time linear in $n$
- If $s = O(\log \log n)$ and $\epsilon n = \text{constant}$, then stabilization time $> \frac{n}{\text{polylog } n}$
Time-Space Trade-Offs

Theorem A: Any protocol using $s < \frac{1}{2} \log \log n$ states per node and solving majority with discrepancy $\epsilon$ must have expected stabilization time $\geq n / (2^s + \epsilon n)^2$.

- In particular:
  - If $s = \text{constant}$ and $\epsilon n = \text{constant}$, then stabilization time linear in $n$
  - If $s = O(\log \log n)$ and $\epsilon n = \text{constant}$, then stabilization time $> n / \text{polylog } n$

Complex molecules are needed for deterministic computation.
Discussion
Molecular computation is fertile ground for algorithmic research.
Molecular computation is fertile ground for algorithmic research.

There are inherent space-time trade-offs when designing deterministic population protocols.
Discussion

Molecular computation is fertile ground for algorithmic research.

There are inherent space-time trade-offs when designing deterministic population protocols.

Open Challenges:

• Tighter trade-off bounds
• Other problems: plurality, approximate counting
• Modeling faulty interactions (leaks)
• Large-scale simulation of molecular algorithms
Leader Election

• **Input:** All nodes start in the same initial state
• **Output:**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of States</th>
<th>Convergence Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial Leader Election</td>
<td>2</td>
<td>$\Omega(n^2)$</td>
</tr>
<tr>
<td>Leader-Minion [AG, ICALP 2015]</td>
<td>$O(\log^3 n)$</td>
<td>$O(\log^3 n)$</td>
</tr>
<tr>
<td>Lottery Leader Election [AAEGR16]</td>
<td>$O(\log^2 n)$</td>
<td>$O(\log^{5.3} n \log log n)$</td>
</tr>
</tbody>
</table>
Leader Election

- **Input:** All nodes start in the same initial state
- **Output:**
  - Exactly one node is in a “leader” state, remains leader forever

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of States</th>
<th>Convergence Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial Leader Election</td>
<td>2</td>
<td>$\Omega(n^2)$</td>
</tr>
<tr>
<td>Leader-Minion [AG, ICALP 2015]</td>
<td>$O(\log^3 n)$</td>
<td>$O(\log^3 n)$</td>
</tr>
<tr>
<td>Lottery Leader Election [AAEGR16]</td>
<td>$O(\log^2 n)$</td>
<td>$O(\log^{5.3} n \log\log n)$</td>
</tr>
</tbody>
</table>
The Impossibility Result

**Theorem A**: Any protocol using $< \frac{1}{2} \log \log n$ states per node and electing $L$ leaders will have expected stabilization time $> \frac{n}{C \text{ polylog } n L^2}$.
The Impossibility Result

Theorem A: Any protocol using \(< \frac{1}{2} \log \log n\) states per node and electing \(L\) leaders will have expected stabilization time \(> \frac{n}{(C \text{ polylog } n \cdot L^2)}\).

Example:
- \(O(\log \log n)\) states / node, one leader
- Stabilization time \(> \frac{n}{\text{ polylog } n}\) (quasi-linear)
- Generalizes a recent result by Doty and Soloveichik [DISC15] to super-constant states
Bonus: A Cute Algorithm

• The goal: approximate \( n \)

• The state:
  • A flip bit \( F \), initially 0
  • A counter “variable” \( C \), initially 0

• The algorithm:
  • Stage 1: do four interactions, updating \( F = 1 - F' \)
  • Stage 2: increment counter \( C \) until you first see \( F' = 1 \)
  • Stage 3: exchange \( C \) with interaction partner, setting \( C = \max \left( C, C' \right) \)

• The guarantee:
  • The convergence value is
    \[
    (1 - \varepsilon) \log n < C < (1 + \varepsilon) \log n,
    \]
    with high probability