Self-Assembling Distributed Internet Software

Yuriy Brun

http://www.cs.washington.edu/homes/brun
brun@cs.washington.edu
Biological Systems

- Self-healing
- Fault-tolerant
- Decentralized
In Contrast: Software

- Less complexity
- Fault-tolerance is “intelligently designed”
- Not expected to recover from catastrophes
Computation is Evolving

**Privacy**: no entity should gain access to the data.

**Security**: no entity may undetectably compromise the computation.
Parts of Real-World Problems Are Computational

- Protein folding
- Resource allocation
- Scheduling
- Encryption breaking
- ...
Distributed Computation

- Computation on the Internet
  - SETI@home [KWA+96]
  - Folding@Home [LSSP02]
  - Rosetta@home [Ros07]

- Grid Computing & Clouds
  - MapReduce [DG04]
  - OrganicGrid [CB04]

- Do not preserve privacy
My Approach: Intelligent Distribution

Obstacle: Discreet classical NP-hard computation is impossible [Chi05].

Solution: Distribute a computation so that no small group of machines knows too much or has too much power.
Outline

1. Private Distributed Computation
2. Self-Assembly
3. Tile Architectural Style
4. Contributions
Outline

1. Private Distributed Computation
2. Self-Assembly
3. Tile Architectural Style
4. Contributions
Self-Assembly in Nature
Self-Assembly in Nature
Self-Assembly in Nature
Self-Assembly in Nature
Self-Assembly in Nature
Self-Assembly in Nature
Tile Assembly Model [Win98b]

- Tile: a square with labels
- Each label has a strength
- Tiles attach if labels are strong enough

Tiles are Turing Universal [Win98b]
Tile Assembly Model [Win98b]

- **Tile**: a square with labels
- **Each label has a strength**
- **Tiles attach if labels are strong enough**

Tiles are Turing Universal [Win98b]
Efficiency in Tile Systems

- Assemble
  - linear polymers \([ACG^+01]\)
  - squares \([RW00, AGHM02, ACG^+02]\)
  - computable shapes \([SW07]\)

- Count \([Win98a, Moi05, BRW05]\)

- Compute Binomial Coefficients \([Win98a, RPW04]\)
Efficient Computation

- Add [Bru07]
- Multiply [Bru07]
- Factor [Bru08a]
- Solve SubsetSum [Bru08b]
- Solve k-SAT [Bru08c]
Efficient Computation

- Add [Bru07]
- Multiply [Bru07]
- Factor [Bru08a]
- Solve SubsetSum [Bru08b]
- Solve k-SAT [Bru08c]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\((x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\)
Solving 3-SAT

\((x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\)
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Solving 3-SAT

\[(x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor \neg x_1 \lor \neg x_0) \land (\neg x_2 \lor x_1 \lor x_0)\]
Can Tiles Implement More-Efficient Algorithms?

\[ O^*(1.8393^n) \] 3-SAT Solution [Bru09]
Can Tiles Implement More-Efficient Algorithms?

\[ O^*(1.8393^n) \text{ 3-SAT Solution [Bru09]} \]
Outline

1. Private Distributed Computation
2. Self-Assembly
3. Tile Architectural Style
4. Contributions
Tile Style Intuition
Software Architecture

“Software architecture: the set of principal design decisions made about a system.” [TMD09]
Converting the Model to an Architecture

Architectural Elements [MRMM02]
- Components: tiles
- Interfaces: side labels
- Topology: 2-D grid
- Behaviors: identifying nodes, recruiting attachments, replicating, and reporting the solution
- Interaction: recruitment data exchange
Node Operations [BM07]

- Initiation (by the client)
- Node Discovery
- Replication
- Recruitment
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Node Discovery

Goal: Selecting a uniformly random node.

A random walk on the graph provably returns a uniformly random node after only $\Theta(\log N)$ requests [MR95].
Data: Each Node Knows Very Little

Less than 1 bit of information per tile.
Data: It Is Hard to Control the Entire Input

\[ 1 - (1 - c^n)^s \]

\( n \) — bits in input  \( c \) — compromised fraction  \( s \) — number of seeds
Data: It Is Hard to Control the Entire Input

\[
1 - (1 - cn)^s
\]

- \(n\) — bits in input
- \(c\) — compromised fraction
- \(s\) — number of seeds

TeraGrid Case Study

- \(\sim 100,000\) machines
- 17-variable 100-clause 3-SAT problem

<table>
<thead>
<tr>
<th>Compromised Fraction</th>
<th>Confidence Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\frac{1}{8})</td>
<td>(1 - 10^{-10})</td>
</tr>
<tr>
<td>(\frac{1}{4})</td>
<td>(1 - 10^{-5})</td>
</tr>
<tr>
<td>(\frac{1}{3})</td>
<td>(1 - 10^{-3})</td>
</tr>
</tbody>
</table>
Experimental Setup

- **Mahjong**: tile style implementation
  - Java, 3K LoC
  - Leverages Prism-MW [MMRM05]
  - Download: [http://csse.usc.edu/~ybrun/Mahjong](http://csse.usc.edu/~ybrun/Mahjong)
Experimental Setup

- **Mahjong**: tile style implementation
  - Java, 3K LoC
  - Leverages Prism-MW [MMRM05]
  - Download: [http://csse.usc.edu/~ybrun/Mahjong](http://csse.usc.edu/~ybrun/Mahjong)

- **Networks**
  - 11-node private cluster (P4 1.5GHz, 512MiB, WinXP/2000)
  - 186-node USC HPCC cluster [Hig] (P4 Xeon 3GHz, Linux)
  - 100-node PlanetLab [PACR03] (global, varying speeds and resources)
Experimental Setup

- **Mahjong:** tile style implementation
  - Java, 3K LoC
  - Leverages Prism-MW [MMRM05]
  - Download: [http://csse.usc.edu/~ybrun/Mahjong](http://csse.usc.edu/~ybrun/Mahjong)

- **Networks**
  - 11-node private cluster (P4 1.5GHz, 512MiB, WinXP/2000)
  - 186-node USC HPCC cluster [Hig] (P4 Xeon 3GHz, Linux)
  - 100-node PlanetLab [PACR03] (global, varying speeds and resources)

- **Sample problems:**
  - \( \mathcal{A} \): 5-number 21-bit *SubsetSum*
  - \( \mathcal{B} \): 11-number 28-bit *SubsetSum*
  - \( \mathcal{C} \): 20-variable 20-clause 3-*SAT*
  - \( \mathcal{D} \): 33-variable 100-clause 3-*SAT*
Scalability: Speed $\propto$ Network Size

<table>
<thead>
<tr>
<th>Network &amp; Problem</th>
<th># of Nodes</th>
<th>Execution Time</th>
<th>Speed-up Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Cluster</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>5</td>
<td>43 sec.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>23 sec.</td>
<td>1.9</td>
</tr>
<tr>
<td>HPCC</td>
<td>93</td>
<td>220 min.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>186</td>
<td>116 min.</td>
<td>1.9</td>
</tr>
<tr>
<td>PlanetLab</td>
<td>50</td>
<td>9.2 min.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>4.8 min.</td>
<td>1.9</td>
</tr>
<tr>
<td>Simjong</td>
<td>125,000</td>
<td>8.7 hours</td>
<td></td>
</tr>
<tr>
<td></td>
<td>250,000</td>
<td>4.5 hours</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>2.1 hours</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>64 min.</td>
<td>2.0</td>
</tr>
</tbody>
</table>
Scalability: Speed $\propto$ Network Size

<table>
<thead>
<tr>
<th>Network &amp; Problem</th>
<th># of Nodes</th>
<th>Execution Time</th>
<th>Speed-up Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Cluster</td>
<td>5</td>
<td>43 sec.</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>23 sec.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simjong</td>
<td>125,000</td>
<td>8.7 hours</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>250,000</td>
<td>4.5 hours</td>
<td></td>
</tr>
<tr>
<td></td>
<td>500,000</td>
<td>2.1 hours</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>1,000,000</td>
<td>64 min.</td>
<td>2.0</td>
</tr>
</tbody>
</table>

System speed scales almost linearly with network size
## Robustness to Network Delay

<table>
<thead>
<tr>
<th>Problem</th>
<th># of Nodes</th>
<th>Network Delay</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mahjong</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>11</td>
<td>Private Cluster</td>
<td>20.1 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPCC</td>
<td>19.3 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlanetLab</td>
<td>18.5 sec.</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>Private Cluster</td>
<td>41.6 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HPCC</td>
<td>41.2 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlanetLab</td>
<td>43.9 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simjong</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1,000,000</td>
<td>0 ms</td>
<td>65 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 ms</td>
<td>57 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100 ms</td>
<td>64 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>500 ms</td>
<td>60 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gaussian</td>
<td>68 min.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distance-based</td>
<td>59 min.</td>
</tr>
</tbody>
</table>
### Robustness to Network Delay

<table>
<thead>
<tr>
<th>Problem</th>
<th># of Nodes</th>
<th>Network Delay</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mahjong</td>
<td></td>
<td>Private Cluster</td>
<td>20.1 sec.</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>HPCC</td>
<td>19.3 sec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PlanetLab</td>
<td>18.5 sec.</td>
</tr>
</tbody>
</table>

Network latency does not affect system throughput

| D | 1,000,000 | 0ms | 65 min. |
|   |           | 10ms| 57 min. |
|   |           | 100ms| 64 min. |
|   |           | 500ms| 60 min. |
|   | Gaussian  |     | 68 min. |
|   | Distance-based |     | 59 min. |
Efficiency: Solving Real-World-Sized Problems

Graph showing network nodes vs. time (seconds, minutes, hours, days, months, years, centuries, millenia) for 30, 40, and 50 variables.
Efficiency: Solving Real-World-Sized Problems

Aimed at large untrusted networks
Related Work

- Private computation in quantum computing through entanglement [Chi05]
- Homomorphic encryption for private computation [Gen09]
- Plethora of non-private distributed computation work [BOI09, KWA⁺96, LSSP02, Ros07, DG04, CB04]
- ...and fault-tolerant computation work [Sar02, BGX93, BCGX02, FS01, KK07, HK03]
Contributions

- Developed self-assembling systems to solve complex computational problems

- Designed the tile architectural style for deploying tile systems on large networks
Leonard Adleman, Qi Cheng, Ahish Goel, Ming-Deh Huang, and Hal Wasserman.
Linear self-assemblies: Equilibria, entropy, and convergence rates.

Leonard Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund.
Combinatorial optimization problems in self-assembly.

Leonard Adleman, Ashish Goel, Ming-Deh Huang, and Pablo Moisset de Espanés.
Running time and program size for self-assembled squares.

Andrea Bondavalli, Silvano Chiaradonna, Felicita Di Giandomenico, and Jie Xu.
An adaptive approach to achieving hardware and software fault tolerance in a distributed computing environment.

Andrea Bondavalli, Felicita Di Giandomenico, and Jie Xu.
A cost-effective and flexible scheme for software fault tolerance.

Yuriy Brun and Nenad Medvidovic.
An architectural style for solving computationally intensive problems on large networks.

BOINC.
The Berkeley open infrastructure for network computing.
Yuriy Brun.
Arithmetic computation in the tile assembly model: Addition and multiplication.

Yuriy Brun.
Nondeterministic polynomial time factoring in the tile assembly model.
A previous version appeared as a Center for Software Engineering, University of Southern California technical report USC-CSSE-2007-707.

Yuriy Brun.
Solving NP-complete problems in the tile assembly model.
A previous version appeared as a Center for Software Engineering, University of Southern California technical report USC-CSSE-2007-703.

Yuriy Brun.
Solving satisfiability in the tile assembly model with a constant-size tileset.
A previous version appeared as a Center for Software Engineering, University of Southern California technical report USC-CSSE-2008-801.

Yuriy Brun.
Improving efficiency of 3-sat-solving tile systems.
*In Submission*, 2009.

Robert Barish, Paul W. K. Rothemund, and Erik Winfree.
Two computational primitives for algorithmic self-assembly: Copying and counting.

Arjav J. Chakravarti and Gerald Baumgartner.
The organic grid: Self-organizing computation on a peer-to-peer network.
Andrew M. Childs.
Secure assisted quantum computation.

Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: Simplified data processing on large clusters.

Pascal Felber and Andre Schiper.
Optimistic active replication.

Craig Gentry.
Fully homomorphic encryption using ideal lattices.

High performance computing and communications.
[http://www.usc.edu/hpcc](http://www.usc.edu/hpcc).

Soonwook Hwang and Carl Kesselman.
A flexible framework for fault tolerance in the grid.

Israel Koren and C. Mani Krishna.
*Fault-Tolerant Systems*.

Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky.
SETI@home — massively distributed computing for SETI.
Stefan M. Larson, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande.
"Folding@Home and Genome@Home: Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology."

Sam Malek, Marija Mikic-Rakic, and Nenad Medvidovic.
"A style-aware architectural middleware for resource-constrained, distributed systems." 

Pablo Moisset de Espanés.
"Computerized exhaustive search for optimal self-assembly counters."

Rajeev Motwani and Prabhakar Raghavan.
"Randomized Algorithms."

Marija Mikic-Rakic, Nikunj R. Mehta, and Nenad Medvidovic.
"Architectural style requirements for self-healing systems."

Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe.
"A blueprint for introducing disruptive technology into the Internet."

Rosetta@home.

Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree.
"Algorithmic self-assembly of DNA Sierpinski triangles."
Paul W. K. Rothemund and Erik Winfree.
The program-size complexity of self-assembled squares.

Luis F. G. Sarmenta.
Sabotage-tolerance mechanisms for volunteer computing systems.

David Soloveichik and Erik Winfree.
Complexity of self-assembled shapes.

Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy.
*Software Architecture: Foundations, Theory, and Practice.*

Erik Winfree.
*Algorithmic Self-Assembly of DNA.*

Erik Winfree.
Simulations of computing by self-assembly of DNA.