Brief Announcement: The Synergy of Finite State Machines

Yehuda Afek[1], Yuval Emek[2], and Noa Kolikant[3]

1 **Tel Aviv University, Tel Aviv, Israel**
afek@cs.tau.ac.il
2 **Technion - Israel Institute of Technology, Haifa, Israel**
yemek@technion.ac.il
3 **Tel Aviv University, Tel Aviv, Israel**
noakolikant@mail.tau.ac.il

Abstract

What can be computed by a network of n randomized finite state machines communicating under the *stone age* model [4] (a generalization of the *beeping* model's communication scheme)? The inherent linear upper bound on the total space of the network implies that its global computational power is not larger than that of a randomized linear space Turing machine, but is this tight? The reported reseach answers this question affirmatively for bounded degree networks by introducing a stone age algorithm (operating under the most restrictive form of the model) that given a designated *I/O node*, constructs a *tour* in the network that enables the simulation of the Turing machine's tape. To construct the tour, it is first shown how to 2-*hop color* the network concurrently with building a spanning tree, with high probability.

1998 ACM Subject Classification C.2.1 Distributed networks

Keywords and phrases finite state machines, stone-age model, beeping communication scheme, distributed network computability

Digital Object Identifier 10.4230/LIPIcs.DISC.2017.42

1 Introduction

Synergy, the whole is greater than the sum of its parts, is many times true, however in traditional distributed computing, each node is usually assumed to be as powerful as a Turing machine, hence its local computational power is equivalent to the global computational power of the whole network. Here, we address the computational power of a network of *randomized finite state machines* with a very weak communication scheme (similar to the communication scheme of the *beeping* model), and show that even under these harsh conditions, synergy can be achieved: *the whole is at least as powerful as the sum of its parts.*

Recently, there is a growing interest in the study of networks of sub-silicon devices, including biological networks [1, 6, 8] and networks of man-made nano-devices [7, 3, 2], through the lens of theoretical distributed computing. These are typically large networks of primitive devices that nevertheless perform complicated tasks, thus raising the following question: How do limitations on the local computation and communication capabilities of the individual nodes affect the global computational power of the whole network?

The reported research addresses this question using the *stone age (SA)* model of Emek and Wattenhofer [4] that captures a network of randomized *finite state machines (FSMs)* with very weak communication capabilities (refer to [4] for a formal definition). It has been shown in [4, Sec. 5 (full version)] that an n-node SA network with a path topology can simulate a randomized $O(n)$-space Turing machine, referred to hereafter as an RSPACE(n) machine. Little is known though about the global computational power of SA networks with more general topologies and/or more restrictive communication schemes. In th reported research, we shed some light into this unexplored research domain.

2 Sequential Stone Age Machines

We wish to use a SA network to simulate an RSPACE(n) machine $\mathcal{M}$, but before we can describe this simulation, we have to explain how the $O(n)$-bit input I of $\mathcal{M}$, that is normally stored in $\mathcal{M}$'s tape at the beginning of the execution, is provided to the implementing SA algorithm. Clearly, no node in the network can hold more than a constant number of bits of I and unlike [4], where a path topology is assumed, here the network topology is arbitrary and does not (initially) induce any sequential order on the nodes, so storing I in the network nodes before the execution commences does not seem to work. Instead, we introduce the key notion of a *sequential stone age machine (SSAM)*, where I is fed to the SA algorithm bit-by-bit in a sequential fashion.

Formally, given a network $G = (V, E)$, a SSAM is a SA algorithm operating in G that allows an external user to

(1) pick any node $v \in V$ and send to it a designated `I/O_prepare` message;
(2) wait until v sends a designated `I/O_ready` message;
(3) feed v with a sequence of input bits by means of sending a sequence of designated input messages (and receiving a corresponding sequence of acknowledgments from v);
(4) wait until the computational process terminates; and
(5) get the desired output back from v by means of receiving from it a sequence of designated output messages.

We refer to node v picked by the user in (1) as an *I/O node*. To exploit the combined computational power of all the nodes the computational process described in (4) typically involves the whole network. The SSAM is said to be a (T^p, T^{io})-*SSAM* if it is guaranteed that the external user waits at most T^p time between sending the `I/O_prepare` message and receiving the `I/O_ready` message and at most T^{io} time between feeding the input bits and getting back the output bits.

3 Our Results

We prove that any problem that can be solved with high probability (abbreviated *whp* hereafter) by an RSPACE(n) machine in time T can be solved whp on any bounded degree network of n randomized finite state machines, with a designated I/O node, using a $b = 1$ stone age communication scheme, in $O(D + T)$ time, where D denotes the diameter of the network. In the $b = 1$ stone age communication model a node cannot tell whether a received message with text M was sent to it by one neighbor or by more than one. In other words, it takes $O(D)$ time to initialize the SSAM so that it is ready to accept its input, whereas the actual simulation of the RSPACE(n) machine takes $O(T)$ time. Specifically, our main algorithmic contribution is a SA algorithm that given an n-node bounded degree graph $G = (V, E)$ and a designated *root* node $r \in V$, constructs a 2-*hop coloring* of G and a node sequence $\langle S(i) \rangle_{i=0}^{2n-1}$, referred to as a *tour*, that satisfies: (i) every node appears in S exactly twice; (ii) $S(0) = S(2n - 1) = r$; and (iii) the state of node $S(i)$ encodes enough information to route a message to $S(i + 1 \mod 2n)$ and to $S(i - 1 \mod 2n)$ that reaches its destination in $O(1)$ time for every $0 \leq i \leq 2n-1$; our algorithm terminates with a correct 2-hop coloring and a correct tour in time $O(D)$ whp.

In the SSAM context, the tour S is constructed during phase (2) (while the external user waits for the `I/O_ready` message) with the I/O node serving as the root. This tour can then be employed to simulate a randomized Turing machine $\mathcal{M}$ with a $(2n)$-cell tape in phase (4).

4 Main Technical Challenges

A 2-hop coloring is a useful construction in anonymous networks (see, e.g., [5]) that enables local point-to-point communication under broadcast communication schemes. As discussed in [4, Sec. 4.3], it is fairly easy to design a 2-hop coloring SA algorithm in bounded degree graphs with *bounding parameter* $b = 2$ (refer to [4] for the definition of a bounding parameter). However, here the bounding parameter is set to $b = 1$, thus turning the 2-hop coloring construction into a challenging task because the nodes can no longer verify (deterministically) that their neighborhood does not admit color conflicts. The setting considered in the reported research is even harder since the graph may contain self-loops (unlike the simple graphs considered in [4]).

Our algorithm resolves this issue by coloring the nodes concurrently with growing a tree $\widetilde{T}$ of depth $O(D)$, rooted at the designated root r. The nodes use a randomized test that looks for color conflicts and if a conflict is detected, the tree $\widetilde{T}$ is carefully used to reset the coloring and tree construction processes. It is interesting to point out that without a designated root, it is impossible to obtain even a 1-hop coloring in our setting.

Another source of difficulty that we had to overcome when designing our algorithm stems from the requirement that the algorithm terminates correctly whp. While whp guarantees are common in traditional distributed graph algorithms, they are more challenging to obtain with SA algorithms: the individual nodes do not (and cannot) have any notion of n; nevertheless, the algorithm should err with probability that decreases (polynomially) with n.

References

1 Y. Afek, N. Alon, O. Barad, E. Hornstein, N. Barkai, and Z. Bar-Joseph. A biological solution to a fundamental distributed computing problem. *Science*, 331(6014):183–185, 2011.

2 S. Cannon, J.J. Daymude, D. Randall, and A.W. Richa. A markov chain algorithm for compression in self-organizing particle systems. In *PODC*, pages 279–288, 2016.

3 Z. Derakhshandeh, R. Gmyr, T. Strothmann, R.A. Bazzi, A.W. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *DNA*, pages 117–132, 2015.

4 Y. Emek and R. Wattenhofer. Stone age distributed computing. In *PODC*, pages 137–146, 2013. The full version can be obtained from `https://ie.technion.ac.il/~yemek/Publications/stone-age.pdf`.

5 Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. Anonymous networks: randomization = 2-hop coloring. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 96–105, 2014.

6 O. Feinerman and A. Korman. *Theoretical distributed computing meets biology: a review*, pages 1–18. Springer Berlin Heidelberg, 2013.

7 O. Michail, I. Chatzigiannakis, and P.G. Spirakis. *New models for population protocols*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2011.

8 S. Navlakha and Z. Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, 2014.