Passively Mobile Communicating Machines that Use Restricted Space*

Ioannis Chatzigiannakis
R. A. Computer Technology Institute (CTI), Patras, Greece
ichatz@cti.gr

Othon Michail
R. A. Computer Technology Institute (CTI), Patras, Greece
michailo@cti.gr

Stavros Nikolaou
R. A. Computer Technology Institute (CTI), Patras, Greece
nikolaou@cti.gr

Andreas Pavlogiannis
Department of Computer Science, UCDavis, Davis
apavlogiannis@ucdavis.edu

Paul G. Spirakis
R. A. Computer Technology Institute (CTI), Patras, Greece
spirakis@cti.gr

ABSTRACT

We propose a new theoretical model for passively mobile Wireless Sensor Networks, called PM, standing for *Passively mobile Machines*. The main modification w.r.t. the Population Protocol model [Angluin *et al.* 2006] is that the agents now, instead of being automata, are Turing Machines. We provide general definitions for unbounded memories, but we are mainly interested in computations upper-bounded by plausible space limitations. However, we prove that our results hold for more general cases. We focus on *complete interaction graphs* and define the complexity classes **PMSPACE**($f(n)$) parametrically, consisting of all predicates that are stably computable by some PM protocol that uses $O(f(n))$ memory in each agent. We provide a protocol that generates unique identifiers from scratch only by using $O(\log n)$ memory, and use it to provide an exact characterization of the classes **PMSPACE**($f(n)$) when $f(n) = \Omega(\log n)$: *they are precisely the classes of all symmetric predicates in* **NSPACE**($nf(n)$). As a consequence, we obtain a space hierarchy of the PM model when the memory bounds are $\Omega(\log n)$. Finally, we establish that the minimal space requirement for the computation of non-semilinear predicates is $O(\log \log n)$.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]; F.1.2 [**Modes of Computation**]: Alternation and nondeterminism; F.1.3 [**Complexity Measures and Classes**]: Complexity hierarchies; F.1.3 [**Complexity Measures and Classes**]: Relations among complexity classes

*Supported in part by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
FOMC'11, June 9, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0779-6/11/06 ...$10.00.

General Terms

Theory

Keywords

population protocols, communicating machines, diffuse computation, passively mobility, sensor network

1. INTRODUCTION

Theoretical models for Wireless Sensor Networks (WSNs) have received great attention over the past few years. Recently, Angluin *et al.* [3] proposed the *Population Protocol* (PP) model. Their aim was to model sensor networks consisting of tiny computational devices (called *agents*) with sensing capabilities that follow some unpredictable and uncontrollable mobility pattern. Due to the minimalistic nature of their model, the class of computable predicates was proven to be fairly small: it is the class of *semilinear predicates* [12], which does not e.g. support multiplication of input variables.

The work of Angluin *et al.* shed light and opened the way towards a brand new and very promising direction. The lack of control over the interaction pattern, as well as its inherent nondeterminism, gave rise to a variety of new theoretical models for WSNs. These models draw most of their beauty precisely from their inability to organize interactions in a convenient and predetermined way. In fact, the PP model was the minimalistic starting-point of this area of research. Most efforts are now towards strengthening the model of Angluin *et al.* with extra realistic and implementable assumptions, in order to gain more computational power and/or speed-up the time to convergence and/or improve fault-tolerance [9, 13].

In this work, we want to allow the agents to use $f(n)$ space for various f, where n is the population size (i.e. the number of agents), while preserving the *uniformity* property of PPs. We think of each agent as being a Turing Machine (TM). In particular, we propose a new theoretical model for passively mobile sensor networks, called the PM model. It is a model of Passively mobile Machines (that we keep calling agents) with sensing capabilities, equipped with two-way communication. We initially focus on PM protocols that use $O(\log n)$ memory, which is an interesting space bound since (as we shall prove) it allows the assignment of unique identi-

fiers (uids) to the agents of the population and plays a major role on establishing the computational power of the model. In addition, we explore the computability of the PM model on different space bounds in order to get an insight of the trade-off between computational power and resource (memory) availability. How are the computational capabilities affected under modifications of the available memory? For example, does more available memory to the agents imply increased computational power? We arrive at exact characterizations for the input symmetric computations performed by communicating TMs using some natural space bounds. As we shall see, for all $f(n) = \Omega(\log n)$ there is a PM protocol using $O(f(n))$ space that can organize the agents into a distributed Nondeterministic TM (NTM) that makes use of all the available space. In the case where $f(n) = o(\log \log n)$, however, we establish that PM protocols are computationally equal to PPs.

1.1 Other Previous Work

Several extensions of the basic model have been proposed in order to more accurately reflect the requirements of practical and more powerful systems. The *Mediated Population Protocol* (*MPP*) model of [9] was based on the additional assumption that each edge of the interaction graph is a finite storage. It has been recently proved [8] that in the case of complete graphs the corresponding class of stably computable predicates is the symmetric subclass of $\mathbf{NSPACE}(n^2)$, rendering the MPP model extremely powerful. Guerraoui and Ruppert [13] made another natural assumption: each agent has its own unique id and can store up to a constant number of other agents' ids. In this model, which they named *Community Protocol* model, the only permitted operation on ids is comparison. It was proven that the corresponding class consists of all symmetric predicates in $\mathbf{NSPACE}(n \log n)$. Some other works incorporated agent failures [10] and gave to some agents slightly increased computational power [6] (heterogeneous systems). For some introductory texts to the subject of PPs see [5, 14, 2].

2. OUR RESULTS - ROADMAP

In Section 3, we begin with a formal definition of the PM model. The section proceeds with a thorough description of the functionality of the systems under consideration and then provides definitions of *configurations* and *fair executions*. In Section 4, first *stable computation* and the family of classes $\mathbf{PMSPACE}(f(n))$ (stably computable predicates by the PM model using $O(f(n))$ space in each agent) are defined and then an illustrating example of a PM protocol using $O(\log n)$ space in each agent is presented. In Section 5, we show that the PM model using $O(f(n))$ space can simulate a NTM (Theorem 2) of space $O(nf(n))$, for any $f(n) = \Omega(\log n)$. This along with Theorem 3, establishing that $\mathbf{PMSPACE}(f(n)) \subseteq \mathbf{SNSPACE}(nf(n))$ (the symmetric subclass of $\mathbf{NSPACE}(nf(n))$), provide the following exact characterizations: $\mathbf{PMSPACE}(f(n)) = \mathbf{SNSPACE}(nf(n))$ for all $f(n) = \Omega(\log n)$. Based on the results of this section, we establish a space hierarchy theorem for the PM model, when the corresponding protocols use $\Omega(\log n)$ space (Theorem 7). In Section 6, we examine the interesting case of the $o(\log logn)$ space bounded protocols, showing that this particular bound acts as a computability threshold. In fact, we show that $\mathbf{PMSPACE}(f(n))$ is equal to the class of semilinear predicates when $f(n) = o(\log \log n)$ and a proper superset of the semilinear predicates when $f(n) = \Omega(\log \log n)$. Finally, in Section 7 we conclude and discuss some interesting open problems. Due to space restrictions, some of our proofs are intuitive and incomplete. The reader is refered to the full version of this paper ([7]) for rigorous proofs of our arguments.

3. THE MODEL

In this section, we formally define the PM model and describe its functionality. In what follows, we denote by $G = (V, E)$ the (directed) interaction graph: V is the set of agents, or *population*, and E is the set of permissible ordered pairwise interactions between these agents. We provide definitions for general interaction graphs and unbounded memories, although in this work we deal with complete interaction graphs only and we are mainly interested in computations that are space-bounded by a logarithm of the population size. We generally denote by n the population size (i.e. $n \equiv |V|$).

DEFINITION 1. *A* PM *protocol is a 6-tuple* $(X, \Gamma, Q, \delta, \gamma, q_0)$ *where* X, Γ *and* Q *are all finite sets and*

1. X *is the* input alphabet*, where* $\sqcup \notin X$,
2. Γ *is the* tape alphabet*, where* $\sqcup \in \Gamma$ *and* $X \subset \Gamma$,
3. Q *is the set of* states,
4. $\delta : Q \times \Gamma^4 \to Q \times \Gamma^4 \times \{L, R, S\}^4 \times \{0, 1\}$ *is the* internal transition function,
5. $\gamma : Q \times Q \to Q \times Q$ *is the* external transition function *(or* interaction transition function*), and*
6. $q_0 \in Q$ *is the* initial state.

Each agent is equipped with: (i) a *sensor* in order to sense its environment and receive a piece of the input, (ii) four read/write *tapes*: the *working tape*, the *output tape*, the *incoming message tape* and the *outgoing message tape*. We assume that all tapes are bounded to the left and unbounded to the right, (iii) a *control unit* that contains the state of the agent and applies the transition functions, (iv) four *heads* (one for each tape) that read from and write to the cells of the corresponding tapes and can move one step at a time, either to the left or to the right, or remain stationary, and (v) a binary *working flag* either set to 1 meaning that the agent is *working* internally or to 0 meaning that the agent is *ready* for interaction.

Initially, all agents are in state q_0, their working flag is set to 1, and all their cells contain the *blank symbol* $\sqcup$. We assume that all agents concurrently receive their sensed input (different agents may sense different data) as a response to a global start signal. The input to each agent is a symbol from X and is written on the leftmost cell of its working tape. We call an *input assignment* to the population, any string $x = \sigma_1\sigma_2 \ldots \sigma_n \in X^*$, with n being the size of the population. In particular, by assuming an ordering on V, the input to agent i is the symbol σ_i, $1 \leq i \leq n$.

When its working flag is set to 1 we can think of an agent working as a usual multitape TM (with the additional step of writing the working flag). In particular, while the working flag is set to 1 the internal transition function δ is applied, the control unit reads the symbols under the heads and its

own state, updates all of them, moves each head one step to the left or to the right or keeps it stationary, and sets the working flag to 0 or 1, according to δ.

As it is common in the PP literature, an *adversary scheduler* selects ordered pairs of agents (edges from E) to interact. Assume now that two agents u and v are about to interact with u being the *initiator* of the interaction and v being the *responder*, i.e. the interacting pair is (u, v). Let $f : V \to \{0, 1\}$ be a function returning the current value of each agent's working flag. If at least one of $f(u)$ and $f(v)$ is equal to 1, then nothing happens, because at least one agent is still working internally. Otherwise, both agents are ready and an *interaction* is established. In the latter case, the external transition function γ is applied, the states of the agents are updated accordingly, the outgoing message of the initiator is copied to the leftmost cells of the incoming message tape of the responder (replacing its contents and writting $\sqcup$ to all other previously non-blank cells) and vice versa (we call this the *message swap*), and finally the working flags of both agents are again set to 1. [1] These operations are also considered as atomic, which intuitively means that the interacting agents cannot take part in another interaction before the completion of these operations.

Since each agent is a TM, we use the notion of a configuration to capture its "state". An *agent configuration* is a tuple $(q, l_w, r_w, l_o, r_o, l_{im}, r_{im}, l_{om}, r_{om}, f)$, where $q \in Q$, $l_j, r_j \in \Gamma^*$ for $j \in \{w, o, im, om\}$, and $f \in \{0, 1\}$. q is the state of the control unit, l_w (l_o, l_{im}, l_{om}) is the string of the working (output, incoming message, outgoing message) tape to the left of the head (including the symbol scanned), r_w (r_o, r_{im}, r_{om}) is the string of the working (output, incoming message, outgoing message) tape to the right of the head (excluding the blank cells), and f is the working flag indicating whether the agent is ready to interact ($f = 0$) or carrying out some internal computation ($f = 1$). We call an agent configuration *initial* if the agent is in state q_0, all its tape cells contain the blank symbol except from the leftmost cell of the working tape that contains its input symbol, and the flag bit is 0. Let $\mathcal{B}$ be the set of all agent configurations. Given two agent configurations $A, A' \in \mathcal{B}$, we say that A *yields* A' if A' follows A by a single application of δ.

A *population configuration* is a mapping $C : V \to \mathcal{B}$, specifying the agent configuration of each agent in the population. A population configuration specifying the initial agent configuration of each of the population's agents is called *initial population configuration*. Note that every input assignment corresponds to an initial configuration of the population, in which each agent is in state q_0 and has a symbol of the input assignment written in its working tape. Let C, C' be population configurations and let $u \in V$. We say that C *yields* C' via *agent transition* u, denoted $C \xrightarrow{u} C'$, if $C(u)$ yields $C'(u)$ and $C'(w) = C(w)$, $\forall w \in V - \{u\}$.

Denote by A_q the state component of an agent configuration A. For any external transition $\gamma(q_1, q_2) = (q_1', q_2')$ define $\gamma_1(q_1, q_2) = q_1'$ and $\gamma_2(q_1, q_2) = q_2'$. We say that C *yields* C' via *encounter* $e = (u, v) \in E$, denoted $C \xrightarrow{e} C'$, if one of the following two cases holds: (i) both agents are ready for an interaction under population configuration C ($f(C(u)) = f(C(v)) = 0$) and therefore the message swap takes place and the state components of the participating agents are updated according to the external transition function ($C_q'(u) = \gamma_1(C_q(u), C_q(v))$, $C_q'(v) = \gamma_2(C_q(u), C_q(v))$ and $C_q'(w) = C_q(w)$ $\forall w \in V - \{u, v\}$), (ii) at least one agent between u and v is working internally under the population configuration C ($f(C(u)) = 1$ or $f(C(v)) = 1$) and no effective interaction takes place ($C'(w) = C(w)$, $\forall w \in V$).

Generally, we say that C *yields* (or *can go in one step to*) C', and write $C \to C'$, if $C \xrightarrow{e} C'$ for some $e \in E$ (via encounter) or $C \xrightarrow{u} C'$ for some $u \in V$ (via agent transition). We say that C' is *reachable* from C, and write $C \xrightarrow{*} C'$ if there is a sequence of population configurations $C = C_0, C_1, \ldots, C_t = C'$ such that $C_i \to C_{i+1}$ holds for all $i \in \{0, 1, \ldots, t-1\}$. An *execution* is a finite or infinite sequence of population configurations $C_0, C_1 \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$. An infinite execution is *fair* if for all population configurations C, C' such that $C \to C'$, if C appears infinitely often then so does C'. This global *fairness condition* is a restriction imposed on the adversary to ensure that the protocol makes progress. A *computation* is an infinite fair execution.

The *space used by an agent running any protocol* $\mathcal{A}$ is the number of tape cells used to store its configuration, that is the sum of the number of tape cells for its finite-state control, for the contents of its four tapes and for its flag. In addition, we say that a *PM protocol* $\mathcal{A}$ *uses* $f(n)$ *space* if the maximum space used by any agent for storing any configuration over all computations is $f(n)$. A (N)TM is called $f(n)$ space bounded if for every input of size n (and in any of its computation paths in the case of a NTM) it scans at most $f(n)$ tape cells on any of its tapes. Similarly we call a *protocol* $\mathcal{A}$, $f(n)$ *space bounded* if it uses $f(n)$ space.

4. STABLY COMPUTABLE PREDICATES

Any mapping $p : X^* \to \{0, 1\}$ is a *predicate on input assignments*. Such a predicate p is called *symmetric* if for every $x \in X^*$ and any x' which is a permutation of x's symbols, it holds that $p(x) = p(x')$. In words, permuting the input symbols does not affect the symmetric predicate's outcome. From each predicate p the language $L_p = \{x \in X^* \mid p(x) = 1\}$ is derived and L_p is symmetric (a.k.a. *commutative*) iff p is symmetric.

A population configuration C is called *output stable* if for every configuration C' that is reachable from C it holds that $\omega(C') = \omega(C)$, where $\omega(C) \in \{0, 1\}$ according to the output value that all agents agree to. In other words, the system does not change its overall output in any subsequent step and no matter how the computation proceeds. A predicate on input assignments p is said to be *stably computable* by a PM protocol $\mathcal{A}$ in a graph family $\mathcal{U}$ if, for any input assignment $x \in X^*$, any computation of $\mathcal{A}$, on any interaction graph from $\mathcal{U}$ of order $|x|$, contains an output stable configuration in which all agents have $p(x)$ written on their output tape. In what follows, we always assume that the graph family under consideration contains only complete interaction graphs.

We say that a predicate p over X^* belongs to $\mathbf{SPACE}(f(n))$ ($\mathbf{NSPACE}(f(n))$) if there exists some deterministic (nondeterministic, resp.) TM that decides L_p using $O(f(n))$ space. A TM is called a *decider* if it accepts all $x \in L_p$ and rejects all $x \notin L_p$ (a decider halts on every input). A deterministic (nondeterministic, resp.) TM decides a lan-

[1] These operations could be handled by the protocols themselves, but then protocol descriptions would become awkward. So, we simply think of them as automatic operations performed by the hardware.

guage L_p using $f(n)$ space if it halts on every input x of size n, accepting (or, in the case of NTMs, there is at least one computation path that accepts) if $x \in L_p$ and rejecting (for all computation paths of an NTM) otherwise, and the maximum number of tape cells scanned/used (in any branch of its computation for an NTM) is $f(n)$. Throughout this work, we use **SSPACE**$(f(n))$ and **SNSPACE**$(f(n))$ to denote the **SPACE**$(f(n))$'s and **NSPACE**$(f(n))$'s restrictions to symmetric predicates, respectively. In addition, we denote by **SEM**, the class of the semilinear predicates, consisting of all predicates definable by first-order logical formulas of Presburger arithmetic (see, e.g., [12]).

DEFINITION 2. *Let* **PMSPACE**$(f(n))$ *be the class of all predicates that are stably computable by some PM protocol that uses* $O(f(n))$ *space.*

All agents are initially identical (they do not have unique ids) and since the interaction graph is complete and the executions are fair, all predicates in **PMSPACE**$(f(n))$ are symmetric for any function $f(n)$.

As a simple illustration of the new model, we present a PM protocol that, using $O(\log n)$ memory, stably computes the non-semilinear predicate $(N_1 = 2^t)$, where $t \in \mathbb{Z}_{\geq 0}$, on the complete interaction graph of n nodes, that is, all agents eventually decide whether the number of 1s in the input assignment is a power of 2.

The protocol counts in binary the number of 1s in the input. The sum of 1s is eventually aggregated in one *awake* agent and all other *sleeping* agents copy the former's output value (see e.g. the parity protocol in [3]). The awake agent can easily recognize whether its counter holds a power of 2 and performs this check every time the counter is incremented. Eventually, the awake agent will know the correct answer to the predicate and the rest of population will obtain it.

Note that the counter of 1s can be at most n. Thus, it requires at most $\lceil \log n \rceil$ bits of memory. In addition, the check of whether the counter is a power of 2 can be easily computed by an agent in $O(\log n)$ space.

COROLLARY 1. **SEM** $\subsetneq$ **PMSPACE**$(\log n)$.

PROOF. PM protocols using $O(\log n)$ space can simulate PPs and $(N_1 = 2^t) \in$ **PMSPACE**$(\log n)$, which is non-semilinear. □

5. A SPACE HIERARCHY

In this section, we study the behaviour of the PM model for various space bounds. Such a study is of particular interest since it is always important to know what computations is a model capable of dispatching according to the capabilities of the available hardware.

5.1 Lower Bounds

We prove here that, for space functions $f(n) = \Omega(\log n)$, the PM model can simulate a NTM of space $O(nf(n))$ using $O(f(n))$ space in each agent.

We begin by proving that PM protocols can assume the existence of unique consecutive ids and knowledge of the population size at the space cost of $O(\log n)$ (Theorem 1). In particular, we present a PM protocol that correctly assigns unique consecutive ids to the agents and informs them of the correct population size using only $O(\log n)$ memory, without assuming any initial knowledge of none of them. We show that this protocol can simulate any PM protocol that assumes the existence of these ids and knows the population size.

Protocol 1 $\mathcal{I}$

```
1: if rid == id then // two agents with the same ids interact
2:     if initiator == 1 then // the initiator
3:         id ← id+1, sid ← id // increases its id by one and stores it in the outgoing message
4:         ps ← id + 1, sps ← ps // sets the population size equal to its updated id + 1
5:     else // the responder
6:         ps ← id + 2, sps ← ps
7:     end if
8:     // both clear their working block and copy their input symbol into it
9:     // they also clear their output tape
10:    working ← binput, output ← ∅
11: else // two agents whose ids differ interact
12:    if rps > ps then // the one who knows an outdated population size
13:        working ← binput, output ← ∅ // is reinitialized
14:        ps ← rps, sps ← ps // and updates its population size to the greater value
15:    else if rps == ps then // they know the same population size
16:        // so they are both already reinitialized and can proceed executing A
17:        execute A for 1 step
18:    end if
19: end if
```

DEFINITION 3. *Let* IPM *('I' standing for "Ids") be the extension of the PM model in which the agents have additionally the unique ids* $\{0, 1, \ldots, n-1\}$ *and in which each agent knows the population size (these are read-only information stored in a separate read-only tape).*

DEFINITION 4. *Let* **IPMSPACE**$(f(n))$ *be the class of all predicates that are stably computable by some IPM protocol that uses* $O(f(n))$ *space in every agent (and in all of its tapes, excluding the space used for the read-only tape) and denote by* **SIPMSPACE**$(f(n))$ *its symmetric subclass.*

Pick any $p \in$ **SIPMSPACE**$(\log n)$. Let $\mathcal{A}$ be the IPM protocol that stably computes it in $O(\log n)$ space. We now present a PM protocol $\mathcal{I}$, containing protocol $\mathcal{A}$ as a subroutine (see Protocol 1), that stably computes p, by also using $O(\log n)$ space. $\mathcal{I}$ is always executed and its job is to assign unique ids to the agents, to inform them of the correct population size and to control $\mathcal{A}$'s execution (e.g. restarts its execution if needed). $\mathcal{A}$, when $\mathcal{I}$ allows its execution, simply reads the unique ids and the population size provided by $\mathcal{I}$ and executes itself normally. We first present $\mathcal{I}$ and then prove that it eventually correctly assigns unique ids and correctly informs the agents of the population size, and that when this process comes to a successful end, it restarts $\mathcal{A}$'s execution in all agents without allowing non-reinitialized agents to communicate with the reinitialized ones. Therefore, at some point, $\mathcal{A}$ will begin its execution reading the

correct unique ids and the correct population size (provided by $\mathcal{I}$), thus, it will get correctly executed and will stably compute p.

We begin by describing $\mathcal{I}$'s variables. *id* is the variable storing the id of the agent (from which $\mathcal{A}$ reads the agents' ids), *sid* the variable storing the *id* that an agent writes in its outgoing message tape in order to send it, and *rid* the variable storing the *id* that an agent receives via interaction. The model's definition implies that all variables used for sending information, like *sid*, preserve their value in future interactions unless altered by the agent. Initially, $id = sid = 0$ for all agents. All agents have an input backup variable *binput* which they initially set to their input symbol and make it read-only. Thus, each agent has always available its input via *binput* even if the computation has proceeded. *working* represents the block of the working tape that $\mathcal{A}$ uses for its computation and *output* represents the contents of the output tape. *initiator* is a binary flag that after every interaction becomes true if the agent was the initiator of the interaction and false otherwise (this is easily implemented by exploiting the external transition function). *ps* is the variable storing the population size, *sps* the one used to put it in a outgoing message, and *rps* the received one. Initially, $ps = sps = 1$.

We now describe $\mathcal{I}$'s functionality. Whenever a pair of agents with the same id interact, the initiator increases its id by one and both update their population size value to the greater id plus one. Whenever two agents with different ids and population size values interact, they update their population size variables to the greater size. Thus the correct size (greatest id plus one) is propagated to all agents. Both interactions described above reinitialize the participating agents (restore their input and erase all data produced by the subroutine $\mathcal{A}$, without altering their ids and population sizes). $\mathcal{A}$, runs as a subroutine whenever two agents of different ids and same population sizes interact, using those data provided by $\mathcal{I}$.

The following lemmas provide some important properties of Protocol 1. Lemma 1 shows that $\mathcal{I}$ correctly assigns unique consecutive ids and propagates the correct population size to the agents of the population in a finite number of steps, whereas Lemma 2 guarantees the fairness of subroutine's $\mathcal{A}$ execution.

LEMMA 1. *(i) No agent id becomes greater than $n-1$, and no ps variable becomes greater than n. (ii) $\mathcal{I}$ assigns the ids $\{0, 1, \ldots, n-1\}$ in a finite number of interactions. (iii) $\mathcal{I}$ sets the ps variable of each agent to the correct population size in a finite number of interactions.*

PROOF. (i) By an easy induction, in order for an *id* to reach the value v, there have to be at least $v + 1$ agents present in the population. Thus, whenever an *id* becomes greater than $n - 1$, there have to be more than n agents present, which creates a contradiction. Similar arguments hold for the *ps* variables
(ii) Assume on the contrary that it does not. Because of (i), at each point of the computation there will exist at least two agents, u, v such that $id_u = id_v$. Due to fairness, an interaction between such agents shall take place infinitely many times, creating an arbitrarily large *id* which contradicts (i).
(iii) The correctness of the id assignment ((i),(ii)) guarantees that after a finite number of steps two agents, u, v will set their *ps* variables to the correct population size (upon interaction in which $id_u = id_v = n - 2$). It follows from (i) that no agent will have its *ps* variable greater than n. Fairness guarantees that each other agent will interact with u or v, updating its *ps* to n. □

LEMMA 2. *Given that $\mathcal{I}$'s execution is fair, $\mathcal{A}$'s execution is fair as well.*

PROOF. The state of each agent may be thought of as containing an $\mathcal{I}$-subcomponent and an $\mathcal{A}$-subcomponent, with obvious contents. Denote by $C_{\mathcal{A}}$ the unique subconfiguration of C consisting only of the $\mathcal{A}$-subcomponents of all agents and note that some $C_{\mathcal{A}}$ may correspond to many superconfigurations C. Assume that $C_{\mathcal{A}} \to C'_{\mathcal{A}}$ and that $C_{\mathcal{A}}$ appears infinitely often. $C_{\mathcal{A}} \to C'_{\mathcal{A}}$ implies that there exist superconfigurations C, C' of $C_{\mathcal{A}}$, $C'_{\mathcal{A}}$, respectively, such that $C \to C'$. Due to $\mathcal{I}$'s fairness, if C appears infinitely often, then so does C' and so does $C'_{\mathcal{A}}$ since it is a subconfiguration of C'. Thus, it remains to show that C appears infinitely often. Since $C_{\mathcal{A}}$ appears infinitely often, then the same must hold for all of its superconfigurations. The reasoning is as follows. All those superconfigurations differ only in the $\mathcal{I}$-subcomponents, that is, they only differ in some variable checks performed by $\mathcal{I}$ (after the id-assignment process and the population size propagation have come to an end, nothing else is performed by $\mathcal{I}$). But all of them are reachable from and can reach a common superconfiguration of $C_{\mathcal{A}}$ in which no variable checking is performed by $\mathcal{I}$, thus, they only depend on which pair of agents is selected for interaction and they are all reachable from one another. Since at least one of them appears infinitely often then, due to the fairness of $\mathcal{I}$'s execution, all of them must also appear infinitely often and this completes the proof. □

By combining the above lemmas we can prove the following:

THEOREM 1. $\mathbf{PMSPACE}(\log n) = \mathbf{SIPMSPACE}(\log n)$.

PROOF. As $\mathbf{PMSPACE}(\log n) \subseteq \mathbf{SIPMSPACE}(\log n)$ hodsl trivially, it suffices to show that $\mathbf{SIPMSPACE}(\log n) \subseteq \mathbf{PMSPACE}(\log n)$. We have already presented a protocol in $\mathbf{PMSPACE}(\log n)$ (Protocol 1) that assigns the agents unique consecutive ids after a finite number of interactions and informs them of the population size (Lemma 1). It follows directly from the protocol that after that point, further fair execution of $\mathcal{I}$ will result in execution of protocol $\mathcal{A}$ which can take into account the existence of unique ids. Moreover, execution of $\mathcal{A}$ is guaranteed to be fair (Lemma 2). □

We now show that for space functions $f(n) = \Omega(\log n)$, the PM model can simulate a NTM of space $O(nf(n))$ using $O(f(n))$ space in each agent. We first prove that this holds for Deterministic TMs, and then generalize to NTMs.

LEMMA 3. $\mathbf{SSPACE}(nf(n)) \subseteq \mathbf{PMSPACE}(f(n))$, *for any* $f(n) = \Omega(\log n)$.

PROOF. Let $p : X^* \to \{0, 1\}$ be any predicate computable in $\mathbf{SSPACE}(nf(n))$ and $\mathcal{M}$ be the deterministic TM that decides p by using $O(nf(n))$ space. We can construct a PM protocol $\mathcal{A}$ that uses $f(n) = \Omega(\log n)$ space on each agent and that stably computes p by exploiting its knowledge of unique ids and the population size. Such knowledge can be obtained by the protocol $\mathcal{I}$ of Theorem 1 (see Subsection

5.1). Note that *protocol $\mathcal{I}$ can be executed by any PM protocol whose agents use $\Omega(\log n)$ space*. Let x be any input assignment in X^*. Each agent receives its input symbol according to x (e.g. u receives symbol $x(u)$). We assume for the sake of simplicity that the agents are equipped with an extra tape, the *simulation tape* that is used during the simulation. The agent that has obtained the unique id 0 starts simulating $\mathcal{M}$.

In the general case, assume that currently the simulation is carried out by an agent u having the id i_u. Agent u uses its simulation tape to write symbols according to the transition function of $\mathcal{M}$. Any time the head of $\mathcal{M}$ moves to the right, u moves the head of the simulation tape to the right, pauses the simulation, writes the current state of $\mathcal{M}$ to its outgoing message tape, and passes the simulation to the agent v having id $i_v = (i_u + 1) \mod n$. Any time the head of $\mathcal{M}$ moves to the left, u pauses the simulation, writes the current state of $\mathcal{M}$ to its outgoing message tape, and passes the simulation to the agent v having id $i_v = (i_u - 1) \mod n$. From agent v's perspective, in the first case it just receives the state of $\mathcal{M}$, copies it to its working tape and starts the simulation, while in the second case it additionally moves the head of the simulation tape one cell to the left before it starts the simulation.

It remains to cover the boundary case in which the head of the simulation tape is over the special symbol that indicates the beginning of the tape. In that case, the agent moves the head to the right and continues the simulation himself (notice that this can only happen to the agent that begins the simulation, that is, the one having the id 0).

Whenever, during the simulation, $\mathcal{M}$ accepts, then $\mathcal{A}$ also accepts; that is, the agent that detects $\mathcal{M}$'s acceptance, writes 1 to its output tape and informs all agents to accept. If $\mathcal{M}$ rejects, it also rejects. Finally, note that $\mathcal{A}$ simulates $\mathcal{M}$ not necessarily on input $x = (\sigma_0, \sigma_1, \ldots, \sigma_{n-1})$ but on some x' which is a permutation of x. The reason is that agent with id i does not necessarily obtain σ_i as its input. The crucial remark that completes the proof is that $\mathcal{M}$ accepts x if and only if it accepts x', because p is symmetric.

Because of the above process, it is easy to verify that the $k-$th cell of the simulation tape of any agent u having the id i_u corresponds to the $(n(k-1)+i_u+1)-$th cell of $\mathcal{M}$. Thus, whenever $\mathcal{M}$ alters $l = O(nf(n))$ tape cells, any agent u will alter $l' = \frac{l-i_u-1}{n} + 1 = O(f(n))$ cells of its simulation tape. □

The next Theorem shows how the above approach can be generalized to include NTMs.

THEOREM 2. **SNSPACE**$(nf(n)) \subseteq$ **PMSPACE**$(f(n))$, *for any* $f(n) = \Omega(\log n)$.

PROOF. We have already shown that the PM model can simulate a deterministic TM $\mathcal{M}$ of $O(nf(n))$ space, where $f(n) = \Omega(\log n)$, by using $O(f(n))$ space (Lemma 3). We now present some modifications that will allow us to simulate a NTM $\mathcal{N}$ of the same memory size. Keep in mind that $\mathcal{N}$ is a decider for some predicate in **SNSPACE**$(nf(n))$, that is, it halts for every input. Upon initialization, each agent enters a reject state (writes 0 to its output tape) and the simulation is carried out as in the case of $\mathcal{M}$.

Whenever a nondeterministic choice has to be made, the corresponding agent gets ready and waits for participating in an interaction. The id of the other participant will provide the nondeterministic choice to be made. One possible implementation of this idea is the following. Since there is a fixed upper bound on the number of nondeterministic choices (independent of the population size), the agents can store them in their memories. Any time a nondeterministic choice has to be made between k candidates the agent assigns the numbers $0, 1, \ldots, k-1$ to those candidates and becomes ready for interaction. Assume that the next interaction is with an agent whose id is i. Then the nondeterministic choice selected by the agent is the one that has been assigned the number $i \mod k$. It follows directly from the fairness constraint that if the computation reaches any state S infinitely many times, all the possible "nondeterministic" choices from S will be followed. In what follows, we will see that this is sufficient for the population to simulate the behaviour of $\mathcal{N}$.

Any time the simulation reaches an accept state, all agents change their output to 1 and the simulation halts. Moreover, any time the simulation reaches a reject state, it is being reinitialized. The correctness of the above procedure is captured by the following two cases.

1. *If $\mathcal{N}$ rejects then every agent's output stabilizes to* 0. Upon initialization, each agent's output is 0 and can only change if $\mathcal{N}$ reaches an accept state. But all branches of $\mathcal{N}$'s computation reject, thus, no accept state is ever reached, and every agent's output forever remains to 0.

2. *If $\mathcal{N}$ accepts then every agent's output stabilizes to* 1. Since $\mathcal{N}$ accepts, there is a sequence of configurations S, starting from the initial configuration C that leads to a configuration C' in which each agent's output is set to 1 (by simulating directly the branch of $\mathcal{N}$ that accepts). Notice that when an agent sets its output to 1 it never alters its output tape again, so it suffices to show that the simulation will eventually reach C'. Assume on the contrary that it does not. Since $\mathcal{N}$ always halts the simulation will be at the initial configuration C infinitely many times. Due to fairness, by an easy induction on the configurations of S, C' will also appear infinitely many times, which leads to a contradiction. Thus the simulation will eventually reach C' and the output will stabilize to 1.

□

5.2 Exact Characterizations

We first prove that **PMSPACE**$(f(n)) \subseteq$ **SNSPACE**$(nf(n))$.

THEOREM 3. *For any function $f(n)$ it holds that* **PMSPACE**$(f(n)) \subseteq$ **SNSPACE**$(nf(n))$.

PROOF. The proof is similar to the one of Theorem 8 in [9]. We construct a TM that, starting from every initial configuration, it nondeterministically guesses all reachable ones storing at most one configuration. □

Using a different representation of population configurations, in the cases that $f(n) = o(\log n)$ the above upper bound can be improved to **SNSPACE**$(2^{f(n)}(f(n)+\log n))$.

THEOREM 4. *For any function $f : \mathbb{N} \to \mathbb{N}$, any predicate in* **PMSPACE**$(f(n))$ *is also in* **SNSPACE**$(2^{f(n)}\ (f(n) + \log n))$.

PROOF. The TM works like the machine of Theorem 3 but uses a different representation of population configurations: it stores for each agent configuration the number of agents that have it instead of storing an agent configuration per agent of the population. □

We have now arrived at the following exact characterizations of **PMSPACE**($f(n)$), when $f(n) = \Omega(\log n)$:

THEOREM 5. *For $f(n) = \Omega(\log n)$,* **PMSPACE**($f(n)$) = **SNSPACE**($nf(n)$).

PROOF. Follows from Theorems 2 and 3. □

5.3 A Space Hierarchy

In this section, we prove a *Space Hierarchy Theorem of the PM model*. We begin with a space hierarchy of NTMs computing symmetric languages, which we then use for establishing a similar hierarchy of PM protocols.

THEOREM 6 (SYMMETRIC SPACE HIERARCHY). *For each $h(n)$ and each recursive $l(n)$, separated by a nondeterministically fully space constructible function $g(n)$, with $h(n) \in \Omega(g(n))$ but $l(n) \notin \Omega(g(n))$, $\exists$ a language L in* **SNSPACE**($h(n)$) − **SNSPACE**($l(n)$).

PROOF. Follows immediately from the unary (tally) separation language presented in [11] and the fact that any unary language is symmetric. □

THEOREM 7 (PM SPACE HIERARCHY). *For each $h(n) \in \Omega(\log n)$ and each recursive $l(n)$, separated by a nondeterministically fully space constructible function $g(n)$, with $h(n) \in \Omega(g(n))$ but $l(n) \notin \Omega(g(n))$, $\exists$ a language in* **PMSPACE**($h(n)$) − **PMSPACE**($l(n)$).

PROOF. Since $l(n)$ is recursive so is $nl(n)$ and since $g(n)$ is nondeterministically fully space constructible so is $ng(n)$. Moreover, $h(n) \in \Omega(g(n))$ and $l(n) \notin \Omega(g(n))$ imply $nh(n) \in \Omega(ng(n))$ and $nl(n) \notin \Omega(ng(n))$, respectively. Now we apply Theorem 6 to the functions $nh(n)$, $nl(n)$, and $ng(n)$ to get a language $L \in$ **SNSPACE**($nh(n)$) − **SNSPACE**($nl(n)$). Note that because of Theorem 5, $h(n) \in \Omega(\log n)$ implies **SNSPACE**($nh(n)$) = **PMSPACE**($h(n)$). Thus, $L \in$ **PMSPACE**($h(n)$). Moreover, $L \notin$ **SNSPACE**($nl(n)$) implies $L \notin$ **PMSPACE**($l(n)$), otherwise we could apply Theorem 3 to obtain a contradiction. Thus, we can conclude that L is in **PMSPACE**($h(n)$) −**PMSPACE**($l(n)$). □

In simple words, Theorem 7 says that for the space bounds discussed in this section, protocols using more memory can compute more things.

6. A COMPUTATIONAL THRESHOLD

In this section, we explore the computability of the PM model when the protocols use $o(\log\log n)$ space. We show that $\log\log n$ acts as a threshold under which PM protocols become computationally equivalent to PPs. In particular, we prove that **PMSPACE**($f(n)$) = **SEM** when $f(n) = o(\log\log n)$ but **SEM** $\subsetneq$ **PMSPACE**($f(n)$) when $f(n) = \Omega(\log\log n)$. The latter is established by showing that $O(\log\log n)$ space suffices for computing a non-semilinear predicate.

6.1 $\log\log n$ Threshold

Here, we prove an interesting limitation on the computability of the PM model when the memory bounds are too restrictive.

THEOREM 8. *PM protocols using $f(n) = o(\log\log n)$ space can only compute semilinear predicates.*

Proof Idea. The result lies on the fact that *populations of different size share common executions at the beginning of their computation.* Indeed, the set of initial states is identical for any two populations A, B, $|A| = n < |B|$. In the first step of the execution, any non-initial state can occur by an interaction of agents being in the initial states. Thus, the sets of states that occur by such interactions are also the same in the two populations (for non-trivial values of n). Proceeding inductively this way, we can see that in order for a new state w to occur in B, but not in A, there has to be an interaction between two states u, v, which can be present in both populations. But since w appears only in B, *u and v cannot exist in A at the same time*, otherwise w would occur in A too. Based on this observation, one can establish that protocols that use $o(\log\log n)$ memory restrict the state space so much, that any two states u, v can occur concurrently in any population A, so that any state that appears in such a B has to be present in A too. The following formalize these arguments.

DEFINITION 5. *Let $\mathcal{A}$ be a PM protocol executed in a population V of size n. An* agent configuration graph, *$R_{\mathcal{A},V} = \{U, W, F\}$, is a graph such that:*

- *U is the set of the agent configurations that can occur in any execution of $\mathcal{A}$ such that the working flag is set to 0.*
- *W is the set of edges (u, v), $u, v \in U$ so that there exists an edge (u, v) when there exists an agent configuration w so that an interaction between two agents with configurations u, w will lead the first one to configuration v.*
- *$F : W \to \{u_1, u_2, \dots\}, u_i \in U \times \{i, r\}$ is an edge labeling function so that when an agent k being in configuration u enters configuration v via a single interaction with an agent being in configuration w, and k acts as $x \in \{i, r\}$ (initiator-responder) in the interaction, then $\{w, x\} \in F((u, v))$.*

In other words, U contains the configurations that an agent may enter in any possible execution, when we do not take into consideration the ones that correspond to internal computation, while W defines the transitions between those configurations through interactions defined by F. We call a $u \in U$ *initial node* iff it corresponds to an initial agent configuration.

Because of the uniformity property, we can deduce the following theorem:

LEMMA 4. *Let $R_{\mathcal{A},V}$, $R_{\mathcal{A},V'}$ be two agent configuration graphs corresponding to a protocol $\mathcal{A}$ for any two different populations V, V' of size n and n' respectively, where $n < n'$. Then, there exists a subgraph R^* of $R_{\mathcal{A},V'}$ such that $R^* = R_{\mathcal{A},V}$, and whose initial nodes contains all the initial nodes of $R_{\mathcal{A},V'}$.*

PROOF. Indeed, let V_1', V_2' be a partitioning of V' such that $V_1' = V$, and observe the agent configuration graph that is yielded by the execution of $\mathcal{A}$ in V_1'. Since both populations execute the same protocol A the transitions are the same, thus all edges in $R_{\mathcal{A},V}$ will be present in $R_{\mathcal{A},V_1'}$ between the common pairs of nodes and their F labels will be equal as well since $V_1' = V$. Therefore $R_{\mathcal{A},V} = R_{\mathcal{A},V_1'}$. Moreover, since the initial nodes are the same for both populations, they must be in $R_{\mathcal{A},V_1'}$. Finally, $R_{\mathcal{A},V_1'}$ is a subgraph of $R_{\mathcal{A},V'}$, as $V_1' \subset V'$, and the proof is complete. □

The above theorem states that while we explore populations of greater size, the corresponding agent configuration graphs are only enhanced with new nodes and edges, while the old ones are preserved.

Given an agent configuration graph, we associate each node a with a value $r(a)$ inductively, as follows:

Base Case For any initial node a, $r(a) = r_{init} = 1$.

Inductive Step For any other node a, $r(a) = min(r(b) + r(c))$ such that a is reachable from b through an edge that contains c in its label, and b, c have already been assigned an r value.

LEMMA 5. *Let $R_{\mathcal{A},V} = \{U, W, F\}$ be an agent configuration graph. Every node in $R_{\mathcal{A},V}$ get associated with an r value.*

PROOF. Assume for the sake of the contradiction that there is a maximum, non-empty set of nodes $U' \subset U$ such that $\forall v \in U'$, v does not get associated with an r value. Then $B = U - U'$, and $C = (B, U')$ defines a cut, with all the initial nodes being in B. We examine any edge (u, v) with label L that crosses the cut, having an arbitrary $(w, x) \in L$. Obviously $u \in B$ and $v \in U'$, and u is associated with a value $r(u)$. Since v is not associated with any r value, the same must hold for node w (otherwise $r(v) = r(u) + r(w)$). We now examine the first agent c that enters in some execution a configuration corresponding to some $v \in U'$. Because of the above observation, this could only happen through an interaction with an agent being in a configuration that is also in U' which creates the contradiction. □

Note that for any given protocol and population size, the r values are *unique* since the agent configuration graph is unique. The following lemma captures a bound in the r values when the corresponding protocol uses $f(n) = o(\log\log n)$ space.

LEMMA 6. *Let r_{max-i} be the $i-th$ greatest r value associated with any node in an agent configuration graph. For any protocol $\mathcal{A}$ that uses $f(n) = o(\log\log n)$, there exists a n_0 such that for any population of size $n > n_0$, $r_{max} < \frac{n}{2}$.*

PROOF. Since $f(n) = o(\log\log n)$, $\lim_{n\to\infty} \frac{f(n)}{\log\log n} = 0$, so $\lim_{n\to\infty} \frac{\log\log n}{f(n)} = \infty$ and $\lim_{n\to\infty} \frac{\log n}{2^{f(n)}} = \infty$. It follows from the last equation that there exists a fixed n_0 such that $\frac{\log n}{2^{f(n)}} > 2$ for any $n > n_0$.

Fix any such n and let $k = |U| \leq 2^{f(n)}$ in the corresponding agent configuration graph. Since any node is associated with an r value, there can be at most k different such values. Now observe that $r_{max} \leq 2 \cdot r_{max-1} \leq \cdots \leq 2^k \cdot r_{init} \leq 2^{2^{f(n)}} < 2^{\frac{\log n}{2}} \leq \sqrt[2]{n} \leq \frac{n}{2}$ for $n > max(n_0, 2)$. □

LEMMA 7. *Let a be a node in the agent configuration graph RAV. Then for every subpopulation of V of size $r(a)$ there is an input and an execution of the protocol A that leads to the configuration a.*

PROOF. We prove the above lemma by generalized induction in the r values.

Base Case The lemma holds for any initial node u, since $r_{init} = 1$.

Inductive Step We examine any non-initial node u that has been associated with a value $r(u) = r(a) + r(b)$, for some a, b. The inductive hypothesis guarantees that a and b can be reached in two separate subpopulations of size $r(a)$ and $r(b)$. Then an interaction between those agents will take one of them to the configuration u, so the lemma holds for u too.

□

Lemmas 6 and 7 lead to the following:

LEMMA 8. *For any protocol $\mathcal{A}$ that uses $f(n) = o(\log\log n)$ there exists a fixed n_0 such that for any population of size $n > n_0$ and any pair of agent configurations u, v, there exists an execution in which the interaction (u, v) takes place.*

PROOF. Indeed, because of the Lemma 6, there exists a n_0 such that for any $n > n_0$, $r(a) < \frac{n}{2}$ for any a. With that in mind, Lemma 7 guarantees that in any such population, any interaction (u, v) can occur since any of the agent configurations u, v can occur independently, by partitioning the population in two subpopulations of size $\frac{n}{2}$ each. □

We can now complete our proof of Theorem 8:

PROOF. Because of the uniformity constraint, A can be executed in any population of arbitrary size. We choose a fixed n_0 as defined in Lemma 6 and examine the population L of size $n = n_0$. Let $R_{\mathcal{A},L}$ be the corresponding agent configuration graph. Let L' be any population of size $n' > n$ and $R_{\mathcal{A},L'}$ the corresponding agent configuration graph. Because of Theorem 4, $R_{\mathcal{A},L'}$ contains a subgraph K, such that $K = R_{\mathcal{A},L}$, and the initial nodes of $R_{\mathcal{A},L'}$ are in K. Let $U^* = U' - U$, and k the first agent configuration that appears in L' such that $k \in U^*$ through an interaction $(u, v)(k$ can't be an initial configuration, thus it occurs through some interaction). Then $u, v \in U$, and the interaction (u, v) can occur in the population L too (Lemma 8), so that $k \in U$, which refutes our choice of k creating a contradiction. So, $U^* = \emptyset$, and the set of agent configurations does not change as we examine populations of greater size. Since the set of agent configurations remains the same as described by the *fixed* $R_{\mathcal{A},L}$, the corresponding predicate can be computed by the PP model, thus it is semilinear. □

Theorem 8 practically states that when the memories available are strictly smaller than $\log\log n$ (asymptotically) then these PM protocols are nothing more than PPs, and although their memory is still dependent on the population size, they cannot exploit it as such; instead they have to use it as a constant memory much like PPs do.

6.2 The Power of 2 Predicate

We will now present the non-semilinear power of 2 predicate, and devise a PM protocol that computes it using $O(\log\log n)$ space in each agent.

The predicate's definition is slightly different to the one described in the example of Section 4. We here define the power of 2 as follows: During the initialization, each agent receives an input symbol from $X = \{a, 0\}$, and let N_a denote the number of agents that have received the symbol a. We want to compute whether $\log N_a = t$ for some arbitrary t. We give a high level protocol that computes this predicate, and prove that it can be correctly executed using $O(\log\log n)$ space.

Each agent u maintains a variable x_u, and let out_u be the variable that u uses to write its output. Initially, any agent u that receives a as his input symbol sets $x_u = 1$ and $out_u = 1$, while any other agent v sets $x_v = 0$ and $out_v = 1$.

The main protocol consists of two subprotocols, $\mathcal{A}$ and $\mathcal{B}$, that are executed concurrently. Protocol $\mathcal{A}$ does the following: whenever an interaction occurs between two agents, u, v, with u being the initiator, if $x_u = x_v > 0$, then $x_u = x_u + 1$ and $x_v = 0$. Otherwise, nothing happens. Protocol $\mathcal{B}$ runs in parallel, and computes the semilinear predicate of determining whether there exist 0, two or more agents having $x > 0$. If so, it outputs 0, otherwise it outputs 1. Observe that $\mathcal{B}$ is executed on *stabilizing inputs*, as the x-variables fluctuate before they stabilize to their final value. However, it is well known that the semilinear predicates are also computable under this constraint [4].

Thus, we have presented a non-semilinear predicate that can be computed by a PM protocol using $O(\log\log n)$ space. Combining this result with Theorem 8, and the fact that $\mathbf{SEM} \subseteq \mathbf{PMSPACE}((f(n)), \forall f$, we obtain the following threshold theorem:

THEOREM 9. $\mathbf{PMSPACE}(f(n)) = \mathbf{SEM}$ *when* $f(n) = o(\log\log n)$ *and* $\mathbf{SEM} \subsetneq \mathbf{PMSPACE}(f(n))$ *when* $f(n) = \Omega(\log\log n)$.

Theorem 9 resembles a similar well-known result of Computational Complexity for the class of regular languages **REG**, according to which $\mathbf{REG} = \mathbf{SPACE}(o(\log\log n)) \subsetneq \mathbf{SPACE}(\Omega(\log\log n))$ (see [15, 1] and Theorem 5.1.3, pages 29-30, of [16]). However, the model under consideration here and, consequently, the proof that we provide are quite different.

7. CONCLUSIONS - OPEN PROBLEMS

We proposed the PM model, an extension of the PP model, in which the agents are communicating TMs. Throughout our work, we studied the computational power of the new model when the space used by each agent is bounded by a function $f(n)$ of the population size. To do so, we presented protocols in which the number of states used by any execution on n agents is bounded by $O(c^{f(n)})$ (so that each state can be represented by $O(f(n))$ tape cells), where c constant, and the new states of the interacting agents are computable in $f(n)$ space by a TM. Although the model preserves uniformity and anonymity, interestingly, we have been able to prove that the agents can *organize themselves into a NTM* that makes full use of the agents' total memory (i.e. of $O(nf(n))$ space) when $f(n) = \Omega(\log n)$. The agents are initially identical and have no global knowledge of the system, but by executing an *iterative reinitialization process* they are able to get assigned *unique consecutive ids* and get informed of the population size. In this manner, we showed that $\mathbf{PMSPACE}(f(n))$, the class of predicates stably computable by the PM model using $O(f(n))$ memory, contains all symmetric predicates in $\mathbf{NSPACE}(nf(n))$. Moreover, by proving that $\mathbf{PMSPACE}(f(n)) \subseteq \mathbf{SNSPACE}(nf(n))$, we concluded that for $f(n) = \Omega(\log n)$, $\mathbf{PMSPACE}(f(n))$ is precisely equal to the class consisting of all symmetric predicates in $\mathbf{NSPACE}(nf(n))$. We also explored the behavior of the PM model for space bounds $f(n) = o(\log n)$ and proved that $\mathbf{PMSPACE}(f(n)) = \mathbf{SEM}$ when $f(n) = o(\log\log n)$. Finally, we showed that this bound acts as a threshold, that is, $\mathbf{SEM} \subsetneq \mathbf{PMSPACE}(f(n))$ when $f(n) = \Omega(\log\log n)$.

Many interesting questions remain open. Is the PM model *fault-tolerant*? What preconditions are needed in order to achieve satisfactory fault-tolerance? What is the behavior of the model when the agents use $O(f(n))$ memory, where $f(n) = o(\log n)$ and $f(n) = \Omega(\log\log n)$? Does a space hierarchy similar to the one presented in Section 5.3, hold for functions $o(\log n)$?

8. ACKNOWLEDGEMENTS

We wish to thank some anonymous reviewers for their very useful comments on a previous version of this work.

9. REFERENCES

[1] M. Alberts. Space complexity of alternating turing machines. In L. Budach, editor, *Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 1–7. Springer Berlin / Heidelberg, 1985. 10.1007/BFb0028785.

[2] C. Àlvarez, I. Chatzigiannakis, A. Duch, J. Gabarró, O. Michail, S. Maria, and P. G. Spirakis. Computational models for networks of tiny artifacts: A survey. *Computer Science Review*, 5(1), January 2011.

[3] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006.

[4] D. Angluin, J. Aspnes, and D. Eisenstat. Stably computable predicates are semilinear. In *25th annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, New York, NY, USA, 2006. ACM Press.

[5] J. Aspnes and E. Ruppert. An introduction to population protocols. *Bulletin of the European Association for Theoretical Computer Science*, 93:98–117, October 2007.

[6] J. Beauquier, J. Clement, S. Messika, L. Rosaz, and B. Rozoy. Self-stabilizing counting in mobile sensor networks. In *26th annual ACM symposium on Principles of distributed computing (PODC)*, pages 396–397, New York, NY, USA, 2007. ACM.

[7] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. Spirakis. Passively mobile communicating machines that use restricted space. Technical Report FRONTS-TR-2011-28, Apr. 2011.

[8] I. Chatzigiannakis, O. Michail, S. Nikolaou, A. Pavlogiannis, and P. G. Spirakis. All symmetric

predicates in $NSPACE(n^2)$ are stably computable by the mediated population protocol model. In *35th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6281 of *Lecture Notes in Computer Science*, pages 270–281. Springer-Verlag, August 23–27 2010.

[9] I. Chatzigiannakis, O. Michail, and P. G. Spirakis. Mediated population protocols. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5556 of *Lecture Notes in Computer Science*, pages 363–374. Springer-Verlag, July 2009.

[10] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, and E. Ruppert. When birds die: Making population protocols fault-tolerant. In *IEEE 2nd Intl Conference on Distributed Computing in Sensor Systems (DCOSS)*, volume 4026 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, June 2006.

[11] V. Geffert. Space hierarchy theorem revised. *Theor. Comput. Sci.*, 295:171–187, 2003.

[12] S. Ginsburg and E. H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[13] R. Guerraoui and E. Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5556 of *Lecture Notes in Computer Science*, pages 484–495. Springer-Verlag, 2009.

[14] O. Michail, I. Chatzigiannakis, and P. G. Spirakis. *New Models for Population Protocols*. N. A. Lynch (Ed), Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool, 2011.

[15] R. E. Stearns, J. Hartmanis, and P. M. Lewis. Hierarchies of memory limited computations. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, FOCS '65, pages 179–190, Washington, DC, USA, 1965.

[16] A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Springer-Verlag New York, Inc., 1994.