A Universal Flying Amorphous Computer

Petrů, L.
2010

Institute of Computer Science
Academy of Sciences of the Czech Republic

A Universal Flying Amorphous Computer

Lukáš Petrů and Jiří Wiedermann

Technical report No. 1097

December 2010

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +4202 6605 3520, fax: +420 286 585 789,
e-mail:lukas.petru@seznam.cz, jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

A Universal Flying Amorphous Computer[1]

Lukáš Petrů and Jiří Wiedermann

Technical report No. 1097

December 2010

Abstract:

Amorphous computers are systems that derive their computational capability from the operation of vast numbers of simple, identical, randomly distributed and locally communicating units. The wireless communication ability and the memory capacity of the computational units is severely restricted due to their minimal size. Moreover, the units originally have no identifiers and can only use simple communication protocols that cannot guarantee a reliable message delivery. In this work we concentrate on a so-called flying amorphous computer whose units are in a constant motion. The units are modelled by miniature RAMs communicating via radio. We design a distributed probabilistic communication protocol and an algorithm enabling a simulation of a RAM in finite time. The underlying algorithms make use of a number of original ideas having no counterpart in the classical theory of distributed computing. Our result is the first one showing computational universality of a flying amorphous computer.

Keywords:
amorphous computing, model of computation, universality

[1]This research was carried out within the institutional research plan AV0Z10300504 and partially supported by the GA ČR grant No. P202/10/1333

1 Introduction

Amorphous computing systems are a relatively recent phenomenon. Apart from the sci-fi literature where various forms of such systems have been envisaged in various futuristic scenarios (cf. an interstellar intelligent mobile cloud in [5], or a mentally controlled flying dust serving as an extension of human senses and as an interface to a very distant offspring of today's Internet in [10]), amorphous computing systems as a subject of scientific research emerged by the end of the 1990s.

In 1999 a project appeared at MIT giving the field its name: Amorphous Computing Project. This project assumed a vast number of identical computational elements randomly placed in a bounded volume. The goal was to construct these elements in such a way that they can self-organize into a prescribed shape, mimicking thus the growth of organic tissues or even of the entire organs, or to perform certain coordinated actions (cf. [1], [2], [3]).

Almost simultaneously engineering efforts for constructing such systems have materialized, e.g., in the 2001 project of a smart dust by K. S. J. Pister (University of California) (cf. [6], [7], [11], [12]). The idea of the smart dust project was to form an ad-hoc network made from micro-electronic devices of an approximatively several cubic mm size or smaller. The long-term goal was to reach the size of a dust mote (cf. the survey article [9]).

The prevailing focus of both previously mentioned projects was on engineering or technological aspects of amorphous computing systems, almost completely ignoring theoretical questions related to computational power and efficiency of such systems. Obviously, without knowing their theoretical limits, one cannot have a complete picture of the potential abilities and limitations of such systems. This was the starting point of the project of the present authors devoted to studies of theoretical issues in amorphous computing initiated in 2004. Since that time various models of amorphous systems have been investigated, roughly in the order of their increased generality.

In [13], [15] amorphous computing systems consisting of identical (with no identifiers) simple asynchronous computational processors (typically RAMs with a small number of registers), equipped by a single channel radio communicators with a limited range and a random number generator, were considered. Such processors were randomly placed in a closed area or volume, and the task was to design a randomized communication protocols leading to a self-organization of a communication network that would enable, with a high probability, to perform universal computations. For the static, non-movable processors this task was solved in [13] and [15]. Later on, in [14] it was shown that similar protocols also work on the level of molecular communication in an aqueous environment (e.g., in a bloodstream); the corresponding processors can take a form of mobile nanomachines capable of performing simple tasks, such as actuation and sensing. The computational part of nanomachines was modeled by variants of finite automata. Recently, a similar result was shown for nanomachines whose communication was inspired by so-called quorum sensing used by real bacteria [16]. The latter two results have been shown by simulating a counter automaton, probably computationally the least efficient universal computing device since it performs its computations using a unary representation of numbers.

The present paper makes a further step towards more general and computationally more efficient amorphous systems: we will consider amorphous systems with motile processors communicating via a radio. The processors take a form of miniature finite-memory RAMs. The transition from systems with a static communication pattern to systems with an unpredictably changing communication pattern brings a number of new problems that have been encountered formerly neither in the classical theory of distributed systems nor in the previous models of amorphous systems. Namely, the situation is further complicated by a continuously changing communication topology in which new communication paths keep emerging while the old ones keep vanishing. Moreover, some processors may become temporarily unaccessible leading to problems with data consistency maintenance over the entire system at times when the nodes become again accessible. These changes call for novel approaches in the design of communicating protocol resulting in unusual time complexity estimation of the simulation algorithm: without additional assumptions on the nature of processor movements one cannot come with a better than a finite upper bound on the the time complexity of this algorithm.

In its class of minimalist amorphous systems communicating over radio, our model of a flying amorphous computer presents the first model for which the computational universality in the efficient

sense has been proven. Namely, when compared with the simulations from [14] and [16], where counter automata have been simulated, the current simulation deals with a simulation of a RAM which is a far more efficient universal computer than a counter automaton. From a (future) practical viewpoint, the model opens a door for exploitation of relatively efficient amorphous computing systems communicating via radio in an airborne medium or in a vacuum.

The structure of the paper is as follows. In Section 2 the definition of the flying amorphous computer is given along with the scenario of its use. In Section 3 we sketch the main ideas of simulation of a RAM computer without yet giving the details of a so-called setup phase in which the amorphous computer is preprocessed in order to be able to perform a simulation, and under the assumption that there is a rudimentary broadcast protocol available enabling a probabilistic communication among the nearest processors. The setup phase itself is described in Section 4, while the broadcast protocol in Section 5. Conclusions are in Section 6.

2 Flying amorphous computer

Definition 1 *A flying amorphous computer is a septuple* $\mathcal{C} = [N, A, P, r, s, T, v]$. *The model has the following properties:*

- *The computer consists of* N *nodes; there is one distinguished node called the* base node.
- *Each node is modeled as a point in a square* target area A. *The initial positions of nodes in* A *are determined by a process* P *assigning a random position independently to each node.*
- *The nodes are not synchronized but they operate at nearly the same clock speed. An operation taking time* T *on the fastest node takes at most* $T(1+\varepsilon_T)$ *on the slowest node, for a fixed* $\varepsilon_T > 0$.

Node properties:

- *A node has a fixed number of* memory registers *of size* s *bits; initially, all registers contain zeros. There is a fixed subset of so-called* input and output registers *enabling message transmission among a node's neighbors. There also is a single register providing a real* random number *(truncated to* s *bits) between* 0 *and* 1 *upon request.*
- *In each node there is a* control unit *operating over all registers. The control unit operates according to a fixed program.*

Movement:

- *Initially, a non-zero random* direction vector u *is assigned to each node. Thereafter, each node moves with a constant speed* v *in the direction of its vector* u. *A node does not know and cannot influence its direction vector.*
- *If a node is about to leave the target area* A, *then its direction vector is mirrored as if a billiard ball would bounce off a wall.*

Communication properties:

- *Each node is equipped with a* radio transceiver *by which it communicates with other nodes up to the distance* r *(called* communication radius*) using a shared radio channel; such nodes are called the* neighbors *of the node at hand.*
- *One message transmission takes time* T.
- *A node receives a message if exactly one of its neighbors sends that message during transmission time* T. *While sending, a node cannot receive a message. If two or more neighbors of a node are simultaneously sending a message, a* collision *occurs and the node receives no message.*
- *A node cannot detect a collision, the radio receiver cannot distinguish the case when multiple neighbors send messages from the case when no neighbor sends a message.*

The communication properties of a node are among the weakest possible and are enforced by requirements of the maximal technological simplicity of the underlying receiver.

In order to solve a computational task a flying amorphous computer operates according to the following scenario in which we assume the existence of an external operator. The scenario consists of three subsequent phases.

First, the operator deploys the nodes in the target area and in some way powers on the computer so that the nodes start moving.

In the second phase, the operator performs a setup of the computer. During the setup a unique address is assigned to each node and the input data of the computational task are loaded into the computer.

After the setup is finished, the computer's own computation can start. At that time, the operator may disconnect from the computer and the computer carries out the computation autonomously. After a sufficient time, the operator may reconnect to check the outcome of the computation.

3 Simulation

In order to prove the versatility of our model we prove its universality by showing that it can simulate a computation of a bounded RAM machine. We shall be using the standard model of a unit cost RAM, cf. [4] with registers of the same size s as the registers of our flying computer. The program, the input and the output data are also stored in these registers.

For the sake of clarity of our exposition, we first sketch the details of the simulation algorithm postponing the details how the previous phases of the computational scenario are implemented. We will merely assume that the computer has been properly formed in the setup phase and that there is a broadcast algorithm guaranteeing a message delivery between two concrete nodes in a finite time.

In order to start a simulation of a RAM with M registers we will require that the simulating flying amorphous computer also have at least M registers loaded with the same initial data as the RAM. This will happen in the setup phase to be described in Section 4.

Definition 2 *We say that a flying amorphous computer A is* set up *with respect to a RAM R with initial configuration $i_1, i_2, \ldots, i_M$ of R if A contains at least M nodes with unique addresses in the range 1 to M and if the j-th node of A contains the same data i_j as the register j in R does, for $j = 1, 2, \ldots, M$.*

The nodes of a flying amorphous computer communicate using broadcast from one node to its neighbors within their communication radius thus forming a multi-hop communication network. A broadcast operation is started at one node and delivers the message, possibly using several hops, to all nodes in a connected communication component. However, as the nodes are moving, not all nodes may be connected (perhaps through several intermediary nodes) to the originating node at times when needed. The message cannot be delivered to such nodes. We assume that such a situation can never occur for an infinitely long time. We will suppose that the nodes are moving in a such a way that it will never happen that two nodes remain forever in different connected components of the underlying communication graph.

Definition 3 *Let A be a flying amorphous computer, B a broadcasting protocol, and n_1, n_2 two nodes of A. Assume that in A message m is being repeatedly broadcast from node n_1 until it is successfully delivered to node n_2. We say that flying amorphous computer A with protocol B is* lively flying *if m is eventually delivered to n_2 after a finite amount of broadcast attempts.*

Note that in a lively flying computer the similar process of message delivery from n_1 to n_2 works also in the reverse direction, from n_2 to n_1. This can be used for a message delivery acknowledging.

We are ready to show how the simulation proceeds.

Theorem 1 *Let R be a RAM with M registers of size s. Let $I = i_1, i_2, \ldots i_M$ be the input data in R's registers. Let C be a computation on R taking time T_1 on input data I. Let A be a lively flying amorphous computer with broadcasting protocol B. Let the nodes of A except of the base node have*

memory capacity of at least $O(\log_2 M + \log_2 N + s)$ bits. Let A be set up with initial configuration I. Then A can simulate computation C in a finite time.

Proof: Let us start with the memory requirements. Any node must hold an address in the range $[1..M]$ requiring $\lceil \log_2 M \rceil$ bits. Such a node must also hold a parameter k of a broadcasting protocol (cf. Section 5), which is of size $O(N)$ and, therefore, requires $O(\log_2 N)$ bits. The register also holds data of the same size as in R, requiring again s bits. In total, the space complexity of a node is $O(\log_2 M + \log_2 N + s)$ bits. Obviously, for a successful simulation it suffices if $N = M + 1$.

Now the amorphous computer can simulate the computation of R by simulating the individual instructions one after the other. During the simulation, the base node controls the computation and plays the role of the control unit of R. The values of registers of R are stored in other nodes of A.

For simulating an instruction the base node must be able to perform two kinds of operations. First, it must be able to simulate reading from and writing to memory registers what it does by communicating with the other nodes using broadcasts. Second, it has to simulate arithmetic computations, branching, and choosing of the next instruction to be processed. These operations are encoded in the internal fixed program of the base node.

For example, if A simulates instruction $A[1] + A[2] \to A[10]$, the base node firsts fetches the value of register 1 by communicating with a node with address 1, then the value of register 2. Then, it computes the addition and finally simulates storing of the result to register 10 by sending its contents to the node with address 10.

Obviously, a broadcast to some node may fail due to a temporary inaccessibility of that node. If, within a certain a priori given time interval, the base node does not receive an acknowledgement from the target note, the broadcast is repeated until an acknowledgement is received. Thanks to the amorphous computer being lively flying, at most a finite number of retries is required. It follows that one instruction can be simulated in finite time and, therefore, any finite computation can be simulated in finite time, too.

The computation ends with the $HALT$ instruction in which case the base node enters a special halt state and stops processing of further instructions. The results are stored in the prescribed output nodes.

□ □

If the simulated program is too long to fit into the base node's memory, one can simulate a RASP machine (cf. [4]) in which the program is a part of the input data. The simulation is then similar to the previous RAM simulation, but now there is an extra read from a register in order to fetch the next instruction, which is then simulated.

Finally, note that a given instance of an amorphous computer has but a fixed number of nodes and so all computations performed on it are necessarily space-limited. Some constants used in the broadcast protocol (cf. Section 5) also depend on the size of the target area. Therefore, an amorphous computer presents a non-uniform computing model. However, the previous result shows that an infinite family of flying amorphous computers of increasing sizes has a universal computing power.

4 Setup

The setup has two phases: address assignment and input data loading.

Address assignment. The purpose of the address assignment process is to allocate unique addresses to the nodes of amorphous computer which are initially indistinguishable, all having zeroed memory. This process is controlled by the base node using the following program.

```
for a = 1 to M
    PickSingleNode()
    AssignAddress(a, h)
```

To pick a single node, the base node makes use of the following algorithm.

```
procedure PickSingleNode()
    broadcast(Initialize, 0)
    for i = 1 to h
        broadcast(RandomGroup, i)
        broadcast(WhichGroup?)
        g = receive()
        broadcast(Choose, g)
```

In this algorithm, procedure `broadcast()` represents a call of the broadcast protocol of the flying amorphous computer, which is described in the next section. In general, a broadcast can fail, but for the moment let us assume that the broadcast messages are always successfully delivered to all nodes. Namely, under this assumption it easier to describe procedure `PickSingleNode` which makes use of the well-known method of probabilistic halving the set of candidates until a single one remains. The general case counting on the possibility of broadcast procedure failure is substantially more complicated since it must also handle the cases of temporary inaccessibility of some nodes and the problems of data consistency maintenance after the return of such nodes. This more involved case will be described later.

Procedure `PickSingleNode` initializes the nodes via the `Initialize` message. After receiving this message, all nodes still without an allocated address participate in the node selection procedure. In this procedure, the base node performs h so-called splitting rounds. During each round, all nodes randomly choose number 0 or 1 assigning themselves in this way to either of the two groups, 0 or 1. Then, all such nodes broadcast the number of the group they selected. From the received answers the base node randomly picks one group number and reports it back to all nodes. Only nodes that are in the selected group participate in the forthcoming round. It is expected that by each such split the number of candidate nodes is roughly halved. If value h is sufficiently high this algorithm selects exactly one node with high probability.

Theorem 2 *Let there be N nodes participating in the* `PickSingleNode()` *algorithm. The probability of selecting more than one node after h splitting rounds at is at most $N/2^h$.*

Proof: Consider the cardinality of the set of nodes after the last splitting round. After each round for each node the probability that an other node remained in the same group is $1/2$. The probability that after h rounds a node still participates in the algorithm is $1/2^h$; therefore for all N nodes the probability that the cardinality of the remaining set is greater than 1 is, at most, $N/2^h$.

□ □

Now we return to the rather peculiar general case in which broadcasts in the address assignment phase can fail.

Recall that our goal selection of exactly one node with high probability is now complicated by the fact that there is no guarantee that a message sent to a certain subset of nodes will always reach all its members. However, for the correctness of the `PickSingleNode` algorithm it has been necessary that a single node is picked from among the nodes that have passed all splitting rounds.

For instance, if a node participating in a splitting round would not receive the `RandomGroup` message, then answering the subsequent `WhichGroup?` message that the node might later obtain would spoil the correctness of the entire algorithm. Namely, such a node could only provide an old value of g, not the current one. Thus, should a node miss a message, which is important for the correctness of selection process and which is addressed specifically to that node, the node must no longer participate in the algorithm.

In order to enable for a node to detect whether it has missed some round we make use of parameter `i` in `RandomGroup` message that was not important in the previous non-failing communication case. This parameter gives the number of the current splitting round. A node is allowed to participate in round $i + 1$ if and only if the node has already passed successfully round i, i.e., if and only if it had first received message `(RandomGroup,i)` followed by `(Choose,g)`, with g being the same as was the number of the node's chosen splitting group. In order to implement this idea it is enough for each node to remember the number of the last round in which it has participated. Then, upon receiving

a `(RandomGroup,i)` message it can discover whether it has missed the previous message (i.e., the previous round) or not.

Upon detecting a missed message, the node at hand acts as if it was in the group that was not chosen, i.e., it stops participating in the next rounds of the splitting algorithm. This puts us on the safe side as far as the probability of selecting more than one node is concerned.

However, a premature termination of a node's further participation in the selection process could lead to an extreme case when all nodes stop participating eventually. Then, we end up with an empty set from which no node can be selected. In order to recognize this situation we introduce a mechanism of message acknowledgements. After performing a step in a computation, a node expected to perform such a step sends an acknowledging `OK` message back to the base node. The base node proceeds to the next step if and only if it hears at least one acknowledgement. Otherwise it repeats the current step. Technically, instead of performing, for example, a simple `broadcast(RandomGroup,i)`, the base node makes use of the following code:

```
repeat
    broadcast(RandomGroup,i)
    m = receive()
until m not empty
```

With these changes, the probability of selecting exactly one node holds as in Theorem 2.

Input data loading. After the addresses were assigned to nodes but before starting the RAM simulation the input data must be stored into the respective nodes. The base node stores data x to a node with address a using the following algorithm:

```
repeat
    broadcast(Write, a, x)
    m = receive()
until m not empty
```

Upon receiving this message, a node with address a stores x in its memory and responds with `OK` message.

5 Communication

The communication capabilities of the individual nodes are as simple as possible. For instance, when a node transmits a message, it cannot determine if there are none, one, or several other nodes around it which might hear its message. Neither can a node determine if the sent message was successfully received by an other node. However, if the nodes adhere to a communication protocol providing a kind of coordination of the nodes' actions, a message can be transmitted from one node to other nodes with high probability even under these restrictive conditions.

The nodes communicate using broadcasting. A message originates in one node and is later broadcast to (potentially) all other nodes of the computer. The algorithm is relatively simple. The first node sends a message which is received by the node's immediate neighbors. These neighbors again send the message so that their neighbors receive it, etc. Note that whenever several nodes share a common neighbor, they should not send the message simultaneously since in such a case the message could not be delivered due to the collisions. In order to minimize the probability of collisions each node establishes a periodic sequence of time slots in which it sends the message randomly with a fixed probability p..

The protocol used by the nodes (the base node included) is described by the following code.

```
procedure broadcast(m,k)
while k <= B
    if random() < p then
        send(m,k+1)
        wait(2T)
    k := k+2
```

In the above procedure, calling `send(m,k+1)` causes one-shot sending of both message m and parameter $k+1$ from the node at hand. Parameter k of `broadcast` procedure keeps track of how many "hops" message m have made so far, in multiples of T, until the current moment. For the node where the message originates the starting value of k is 0. For other nodes, the current value of k is derived from the corresponding value in the lastly received message. That is, the value of k is incremented by 1 upon each re-transmission of m. Hencefore, all nodes stop broadcasting when k reaches the value of B, which is a global constant known in advance to all nodes of the whole flying amorphous computer. This will happen at about the same time; some small variation due to different clock speeds are possible.

The value of B influences how far from the originating node the message will spread. If the selected value is too small, the message will not reach the nodes that are far from the originating node. If the value is too large, an excessive message sending after all nodes have already received the message means just a waste of time and energy.

We assume that the external operator is able to choose a reasonable value of constant B. Such a value should be selected, taking into account the number of nodes and their density in the target area. Asymptotically, the value rises as $O(\sqrt{N})$. In other words, it rises as the distance between the opposite corners of the target area if measured in units of r (the size of the communication radius). The operator can find a reasonable value either by using a computer simulation of the system or by trying out several values on the actual flying amorphous computer.

Theorem 3 *Let all nodes of an amorphous computer be in a quiet state (i.e. not sending anything). Let X be a node that starts broadcasting a message using procedure* `broadcast` *at time t. Then, at time $t + B(1 + \varepsilon_T)$ all nodes are again in a quiet state.*

Proof: The first node runs procedure `broadcast` with parameter k initially equal to 0. Each time through the loop the node waits for time $2T$ and the value of k is increased by 2. So, at most after time $B(1+\varepsilon_T)$ the node stops re-sending the message at hand. Transmitting a message takes time T. Therefore, whenever a neighbor node receives a message, value $k+1$ corresponds to the running time of the procedure up to now, and hence also this node ends transmitting after at most $(k-1)(1+\varepsilon_T)$ time steps. By induction, all nodes end in time $B(1+\varepsilon_T)$.

□ □

Note that in order to start a new broadcast it is necessary for all nodes to wait until the current broadcast ends. Namely, should a broadcast of two different messages have started simultaneously, then we cannot be sure which of the two messages would be delivered first. Therefore, the end of a current call to `broadcast` must be known to all nodes.

In order to guarantee the condition of non-concurrent broadcasts after the base node has started broadcasting a message, all other nodes must wait for time $B\varepsilon_T$ to be sure that in total time $B(1+\varepsilon_T)$ has passed. Only thereafter a node can broadcast another message.

It is interesting to observe that there might be specific cases where a simultaneous broadcast cannot do any harm. For instance, in the address assignment protocol, when we wanted to know to which group some nodes belong, two different messages were sent at the same time from both groups. We did not care which one of the two messages was delivered since the groups had originally been selected randomly, anyway.

6 Conclusion

We have shown a computational universality of our model of flying amorphous computer. The nature of our results is quite unusual. In principle, a computation of a flying amorphous computer consists of two phases — of the preparation phase in which the computer is setup for the simulation, and the simulation proper.

The setup phase is of a probabilistic nature. As seen from Theorem 2, there is (a vanishing) probability that address allocation procedure will not end correctly, and there might appear the nodes with the same addresses. This will spoil the subsequent simulation. However, with overwhelming probability such a case will not happen and then the subsequent simulation algorithm will, in fact, be

a deterministic algorithm: the result delivered by this algorithm will be correct with probability one. Once a flying amorphous computer is properly formed, it can be used for performing any simulation with the given number of nodes and its results will always be correct.

The next oddity of the simulation algorithm is its time complexity estimation in terms of finite time. This is due to the unpredictability of the communication paths formation. Interestingly, real computer simulations have revealed that within a flying amorphous computer a message is delivered to all nodes quite efficiently, in a wide range of velocities of node movements and parameters of the broadcast algorithm. The simulations have also confirmed intuition that there is a tradeoff between the amount of nodes' mixing in a flying amorphous computer and the frequency of message repetition sendings in the broadcast protocol.

More research is needed in order to better understand and exploit the respective phenomena. Our results present but the first steps in this direction.

Bibliography

[1] H. Abelson, et al. Amorphous Computing. MIT Artificial Intelligence Laboratory Memo No. 1665, Aug. 1999

[2] H. Abelson, D. Allen, D. Coore, Ch. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, R. Weiss. Amorphous Computing. Communications of the ACM, Volume 43, No. 5, pp. 74–82, May 2000

[3] H. Abelson, J. Beal, G. J. Sussman. Amorphous Computing. Computer Science and Artificial Intelligence Laboratory, Technical Report, MIT-CSAIL-TR-2007-030, June 2007

[4] Aho, A.V., Hopcroft J.E., Ullman J.D.: *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley; 1974

[5] F. Hoyle: The Black Cloud. Penguin Books, 1957, 219 p.

[6] J. M. Kahn, R. H. Katz, K. S. Pister. Next century challenges: mobile networking for "Smart Dust". In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom '99, ACM, pp. 271-278, Aug. 1999

[7] J. M. Kahn, R. H. Katz, K. S. J. Pister. Emerging Challenges: Mobile Networking for Smart Dust. Journal of Communications and Networks, Volume 2, pp. 188–196, 2000

[8] L. Petrů, J. Wiedermann. A Model of an Amorphous Computer and Its Communication Protocol. In: Proc. SOFSEM 2007: Theory and Practice of Computer Science. LNCS volume 4362, Springer, pp. 446–455, July 2007

[9] M. J. Sailor, J. R. Link: Smart dust: nanostructured devices in a grain of sand, Chemical Communications, vol. 11, p. 1375, 2005

[10] V. Vinge: A Deepness in the Sky. Tor Books, January 2000, 800 p.

[11] B. Warneke, M. Last, B. Liebowitz, K. S. J. Pister: Smart Dust: communicating with a cubic-millimeter computer. Computer, volume: 34, Issue: 1, pp. 44–51, Jan. 2001

[12] B. Warneke, B. Atwood, K. S. J. Pister: Smart dust mote forerunners. In Proceedings of the 14th IEEE International Conference on Micro Electro Mechanical Systems, 2001, MEMS 2001, pp. 357–360, 2001

[13] J. Wiedermann, L. Petrů: Computability in Amorphous Structures. In: Proc. CiE 2007, Computation and Logic in the Real World. LNCS volume 4497, Springer, pp. 781–790, July 2007

[14] J. Wiedermann, L. Petrů. Communicating Mobile Nano-Machines and Their Computational Power. In: Third International ICST Conference, NanoNet 2008, Boston, MA, USA, September 14-16, 2008, Revised Selected Papers, LNICST vol. 3, Part 2, Springer, pp. 123-130, 2009.

[15] J. Wiedermann, L. Petrů: On the Universal Computing Power of Amorphous Computing Systems. Theory of Computing Systems 46:4 (2009), 995-1010, www.springerlink.com/content/k2x6266k78274m05/fulltext.pdf

[16] Wiedermann, J.: Nanomachine Computing by Quorum Sensing. In: Regulation, Cooperation, Nature, and Languages, J. Kelemen, A. Kelemenová, eds. A volume in the series "Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory", World Scientific, to appear in 2010.