Sorting Using Spiking Neural P Systems with Anti-spikes and Rules on Synapses

Venkata Padmavati Metta[1](✉) and Alica Kelemenová[2]

[1] Bhilai Institute of Technology, Durg, India
vmetta@gmail.com

[2] Institute of Computer Science and Research Institute of the IT4Innovations Centre of Excellence, Silesian University in Opava, Opava, Czech Republic
alice.kelemenova@fpf.slu.cz

Abstract. This paper introduces and makes use of spiking neural P systems with anti-spikes and rules on synapses to sort integers. Here we discuss two types of sorting, bead sort and bitonic sort to sort integers.

1 Introduction

Spiking neural P systems (in short, SN P systems) introduced in [10] are parallel and distributed computing models which abstract the way neurons communicate by means of electrical impulses of identical shape, called spikes. There exist many variants of spiking neural P systems. However, in some cases, the difference in these variants are not with the actual structural features but with execution semantics like maximal, sequential, asynchronous, exhaustive etc. Some of them have special concepts like extended rules [1], astrocytes [7], anti-spikes [13], neuron division and budding [14], rules on the synapses [17] etc. We refer to the respective chapter of [15] for general information in this area, and to the membrane computing website from [18] for details.

Here we introduce the hybrid model of SN P systems combining the features of anti-spikes with rules on the synapses and name them as spiking neural P systems with anti-spikes and rules on synapses (in short, SN PA systems with rules on synapses). So these systems make use of two types of objects called spikes (a) and anti-spikes ($\overline{a}$). The use of anti-spikes not only simplify the complexity of the rules but also allow to include negative numbers in computing.

In standard SN P systems the rules reside inside neurons and upon firing the spikes emitted by the neurons are sent to all neighbouring neurons through their outgoing synapses. Instead of rules inside neurons, here we have rules on the synapses. At any step, when the number of spikes/anti-spikes present in a given neuron is satisfied by a rule on a synapse leaving from that neuron, the rule is enabled and upon firing a spike/an anti-spike is sent to the neuron at the end of the synapse. As expected, the SN PA systems with rules on synapses are able to compute all Turing computable sets of numbers.

Sorting is one of the most frequent operations in many applications, and parallel algorithms for sorting have been studied since the beginning of parallel computing. P systems are used to simulate various sorting algorithms [2,3].

DOI: 10.1007/978-3-319-28475-0_20

Batcher's bitonic sorting network [6] was one of the first methods proposed. The method is simulated to sort non-negative integers using P systems [3] and SN P systems [8]. In this paper we simulate the bitonic sorting network using SN PA systems with rules on synapses to sort integers.

Another natural parallel sorting algorithm for sorting non-negative integers is bead sort [5]. This algorithm was also simulated using P systems in [4]. Ionescu and Sburlan in [11] used SN P system to sort n non-negative integers, and the model consisted of 3 layers of n neurons each. The first layer was made up of input neurons which in the initial configuration contained the input values codified as numbers of spikes. At each time unit these neurons sent one spike each to the second layer. This layer decanted the spikes to the third layer, where the output neurons were located. After a number of steps equal to the maximum value of the n numbers, the ith output neuron received the ith smallest value, codified as number of spikes, sorting thus in ascending order. In a way, the idea of the algorithm is the same as that of bead sort. The model makes use of $3n$ neurons, $(3n^2 + n)/2$ synapses and $n^2 + n$ rules. The time complexity of the algorithm is $O(M)$ where M is the maximum of the n numbers. In this paper we use SN PA systems with rules on synapses to simulate the algorithm and observe that this model makes use of $2n + 2$ neurons, $4n$ synapses and $n^2 + 3n$ rules to sort n integers, which is comparatively less complex than the system in [11].

2 Prerequisites

We assume the reader to be familiar with formal language theory and membrane computing. The reader can find details about them in [15,16] etc.

For an alphabet V, V^* is the free monoid generated by V with respect to the concatenation operation and the identity λ (the empty string); the set of all non-empty strings over V, that is, $V^* - \{\lambda\}$, is denoted by V^+. When $V = \{a\}$ is a singleton, then we write a^* and a^+ instead of $\{a\}^*$ and $\{a\}^+$.

A regular expression over an alphabet V is defined as: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = L(E_1)^+$, for all regular expressions E_1, E_2 over V.

We now introduce spiking neural P systems with anti-spikes and rules on synapses.

2.1 Spiking Neural P Systems with Anti-spikes and Rules on Synapses

A spiking neural P system with anti-spikes and rules on synapses, of degree $m \geq 1$, is a construct

$$\Pi = (O, \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_m, syn, IN, OUT), \text{ where}$$

1. $O = \{a, \overline{a}\}$ is a binary alphabet; a is called *spike* and $\overline{a}$ is called an *anti-spike*.
2. σ_1, σ_2, σ_3,..., σ_m are neurons of the form $\sigma_i = (n_i)$ with $1 \leq i \leq m$ where n_i is the number of spikes or anti-spikes contained in the neuron σ_i and if $n_i > 0$ then the neuron is having n_i spikes and if $n_i < 0$ then the neuron is having $\mid n_i \mid$ anti-spikes;
3. syn is the set of synapses; each element in syn is a pair of the form $((i, j), R(i, j))$, where (i, j) indicates that there is a synapse connecting neurons σ_i and σ_j, with $i, j \in \{1, 2, \dots, m\}$, $i \neq j$, and $R(i, j)$ is a finite set of rules of the following two forms:
   (i) $E/b^r \to b'$ where $b, b' \in \{a, \overline{a}\}$, $r \geq 1$ and E is a regular expression over b;
   (ii) $b^s \to \lambda$ for some $s \geq 1$, with the restriction that $b^s \notin L(E)$ for any rule $E/b^r \to b'$ of type (i) from $R(i, j)$;
   There are four categories of spiking rules identified by $(b, b') \in \{(a, a), (a, \overline{a}), (\overline{a}, a), (\overline{a}, \overline{a})\}$.
4. *IN*, $OUT \subseteq \{1, 2, 3, \dots, m\}$ are the set of input and output neurons respectively.

A rule $E/b^r \to b' \in R(i, j)$ with $b, b' \in \{a, \overline{a}\}$ is applied as follows. If the neuron σ_i contains number of bs equal to c, and $b^c \in L(E)$, $c \geq r$, then the rule can *fire*, and upon application, r spikes of kind bs are consumed (thus only $c - r$ remain in σ_i) and a b' is released, which will immediately exit the neuron. The spike/anti-spike emitted by neuron σ_i will pass immediately to all neurons σ_j such that $E/b^r \to b' \in R(i, j)$. This means that the transmission of spike/anti-spike takes no waiting time (since the rules do not specify a time delay), the spike/anti-spike will be available in neuron σ_j in the next step. There is an additional restriction that a and $\overline{a}$ cannot stay together, they annihilate each other. If a neuron has either objects a or objects $\overline{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^q and $\overline{a}^s$ inside, then immediately an annihilation rule $a\overline{a} \to \lambda$ (which is implicit in each neuron), is applied in a maximal manner, so that either a^{q-s} or $(\overline{a})^{s-q}$ remain for the next step, provided that $q \geq s$ or $s \geq q$, respectively. This mutual annihilation of spikes and anti-spikes takes no waiting time and the annihilation rule has priority over spiking and forgetting rules, so each neuron always contains either only spikes or anti-spikes. If we have a rule $E/b^r \to b'$ with $L(E) = \{b^r\}$, then we write it in the simplified form as $b^r \to b'$ and call it pure. The rules of the form $b^s \to \lambda \in R(i, j)$ are called forgetting rules. If the neuron contains exactly s number of bs, then the forgetting rule $b^s \to \lambda$ can be applied removing s number of bs from the neuron immediately.

The *configuration* of the system is described by $\mathcal{C} = \langle \beta_1, \beta_2, \dots, \beta_m \rangle$, where β_i is the number of spikes/anti-spikes present in neuron σ_i. At any moment, if $\beta_i > 0$, it means that there are β_i spikes in neuron σ_i; if $\beta_i < 0$, it indicates that neuron σ_i contains $\mid \beta_i \mid$ anti-spikes. The initial configuration is $\mathcal{C}_0 = \langle n_1, n_2, \dots, n_m \rangle$.

As usual in SN P systems, a global clock is assumed, marking the time for all neurons and synapses. In each time unit, if a synapse (i, j) can use one of its rules, then a rule from $R(i, j)$ must be used. It is possible that there is more than one rule that can be used on a synapse at some moment, since two firing rules, $E_1/b^c \rightarrow b'$ and $E_2/\hat{b}^r \rightarrow \hat{b}'$ may have $L(E_1) \cap L(E_2) \neq \emptyset$ where $\{b, b', \hat{b}, \hat{b}'\} \in \{a, \overline{a}\}$. In this case, the synapse will non-deterministically choose one of the enabled rules to be used.

The system works sequentially on each synapse (at most one rule from each set $R(i, j)$ can be used), and in parallel at the level of the system (if a synapse has at least one rule enabled, then it has to use a rule).

A delicate problem appears when several synapses starting from the same neuron have rules which can be applied. We work here with the restriction that all rules which are applied consume the same number of spikes from the given neuron. Let us assume that the applied rules on the synapses leaving from σ_i are of the form $E_u/b^c \rightarrow b'$ then c number of bs are removed from σ_i (and not a multiple of c, according to the number of applied rules). Of course, this restriction can be replaced by another strategy: various rules can consume various numbers of spikes and the sum of these numbers of spikes is removed from the neuron.

Using the rules in this way, we pass from one configuration of the system to another configuration; such a step is called a transition. For two configurations $\mathcal{C}$ and $\mathcal{C}'$ of Π we denote by $\mathcal{C} \Longrightarrow \mathcal{C}'$, if there is a direct transition from $\mathcal{C}$ to $\mathcal{C}'$ in Π.

A computation of Π is a finite or infinite sequence of transitions starting from the initial configuration, and every configuration appearing in such a sequence is called reachable. A computation halts if it reaches a configuration where no rule can be used. With any halting computation, we associate a number of spikes/anti-spikes appearing in the output neurons which encode the vector of integer numbers as the output of the system. When both the input and output neurons are considered, the system can be used as a transducer. Henceforth in the paper, SN P systems with anti-spikes and rules on synapses are used as transducers and are referred to as SN PA systems.

3 Bitonic Sorting Network

This section describes a variant of a sorting network called bitonic network that has a fast sorting or ordering capability. A sorting network can be used as a multiple-input, multiple-output switching network. Other applications of sorting networks are as a switching network with buffering, a multi-access memory, a multi-access content-addressable memory, and as a multiprocessor. The advantage of bitonic networks is the flexibility (one network can accommodate input lists of various lengths) and the modularity (a large network can be split up into several identical modules).

The basic component of a bitonic sorting network is a comparator. A comparator is a device with two inputs x and y and two outputs l and h. For an increasing comparator, $l = min(x, y)$ and $h = max(x, y)$; for a decreasing comparator $l = max(x, y)$ and $h = min(x, y)$. Figure 1 gives the schematic representation of the two types of comparators. As two elements enter the input wires of the comparator, they are compared and, if necessary, exchanged before

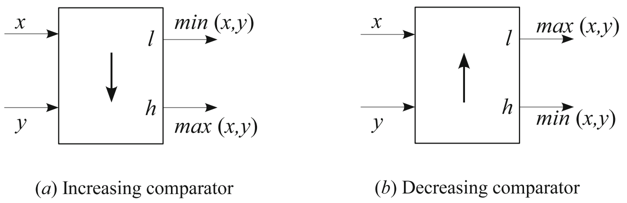

Fig. 1. A schematic representation of comparators

they go to the output wires. We denote an increasing comparator by $\downarrow$ and a decreasing comparator by $\uparrow$.

The key operation of the bitonic sorting network is the rearrangement of a bitonic sequence into a sorted sequence. A bitonic sequence is a sequence of elements $< a_0, a_1, \ldots, a_{n-1} >$ with the property that either (1) there exists an index i, $0 \le i \le n-1$, such that $< a_0, \ldots, a_i >$ is monotonically increasing and $< a_{i+1}, \ldots, a_{n-1} >$ is monotonically decreasing, or (2) there exists a cyclic shift of indices so that (1) is satisfied. For example, $< 1, 2, 4, 7, 6, 0 >$ is a bitonic sequence, because it first increases and then decreases.

We present a method to rearrange a bitonic sequence to obtain a monotonically increasing sequence. Let $S =< a_0, a_1, \ldots, a_{n-1} >$ be a bitonic sequence such that $a_0 \le a_1 \le, \ldots, \le a_{n/2-1}$ and $a_{n/2} \ge a_{n/2+1} \ge, \ldots, \ge a_{n-1}$. Consider the following subsequences of S:
$S_1 =< min(a_0, a_{n/2}), min(a_1, a_{n/2+1}), \ldots, min(a_{n/2-1}, a_{n-1}) >$
$S_2 =< max(a_0, a_{n/2}), max(a_1, a_{n/2+1}), \ldots, max(a_{n/2-1}, a_{n-1}) >$.
The sequences S_1 and S_2 are bitonic sequences. Furthermore, every element of the first sequence is smaller than every element of the second sequence. Thus, we have reduced the initial problem of rearranging a bitonic sequence of size n to that of rearranging two smaller bitonic sequences and concatenating the results. We refer to the operation of splitting a bitonic sequence S of size n into the two bitonic sequences S_1 and S_2 as a *bitonic split.* Although in obtaining S_1 and S_2 we assumed that the original sequence had increasing and decreasing sequences of the same length, the bitonic split operation also holds for any bitonic sequence.

We can recursively obtain shorter bitonic sequences using bitonic split for each of the bitonic subsequences until we obtain subsequences of size one. At that point, the output is sorted in monotonically increasing order. Since after each bitonic split operation the size of the problem is halved, the number of splits required to rearrange the bitonic sequence into a sorted sequence is $log\ n$. The procedure of sorting a bitonic sequence using a series of bitonic splits is called *bitonic merge.*

So the key components of a bitonic sorting network are the bitonic splitters and the bitonic mergers. The splitter of size n takes as input a bitonic sequence of length n and partitions it in two bitonic sequences of equal length. A bitonic merger of size n consists of a splitter of size n and of two mergers of size $n/2$, of

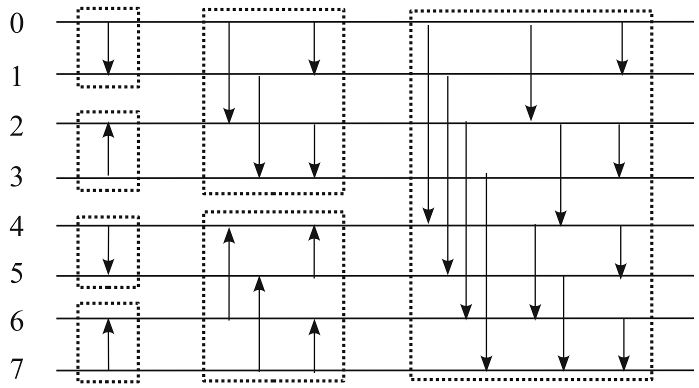

Fig. 2. A bitonic sorting network for $n = 8$. The network can be partitioned into three stages, each has bitonic mergers of size 2, 4, and 8 respectively.

opposite direction. It accepts as input a bitonic sequence and sorts it in ascending or descending order (direction).

Figure 2 illustrates a typical bitonic sorting network for sorting $n = 8$ numbers in ascending order. The input wires are numbered $0, 1, \ldots, n-1$. The network can be partitioned into three stages, each has bitonic mergers of size 2, 4, and 8 respectively. Each stage has column of comparators drawn separately. The network takes an unsorted sequence of size 8 and outputs it in ascending order.

Let us now see how this network works. The first stage groups the list into $n/2$ bitonic sequences of length two. A sequence of two elements x and y forms a bitonic sequence, since either $x \leq y$, in which case the bitonic sequence has x and y in the increasing part and no elements in the decreasing part, or $x \geq y$, in which case the bitonic sequence has x and y in the decreasing part and no elements in the increasing part. Hence, any unsorted sequence of elements is a concatenation of bitonic sequences of size two. It merges the adjacent bitonic sequences in increasing and decreasing order to get bitonic sequences of size four.

So each stage of the network shown in Fig. 2 merges adjacent bitonic sequences in increasing and decreasing order. According to the definition of a bitonic sequence, the sequence obtained by concatenating the increasing and decreasing sequences is bitonic. Hence, the output of each stage in the network in Fig. 2 is a concatenation of bitonic sequences that are twice as long as those at the input. By merging larger and larger bitonic sequences, we eventually obtain a bitonic sequence of size n. Merging this sequence sorts the input. We refer to the algorithm embodied in this method as bitonic sort and the network as a bitonic sorting network. The first three stages of the network are shown in Fig. 2. The last stage of Fig. 2 is shown explicitly in Fig. 3.

A network can also be represented as a directed acyclic graph [9].

Definition 1 (Network). *A network T of size n is a directed acyclic graph such that:*

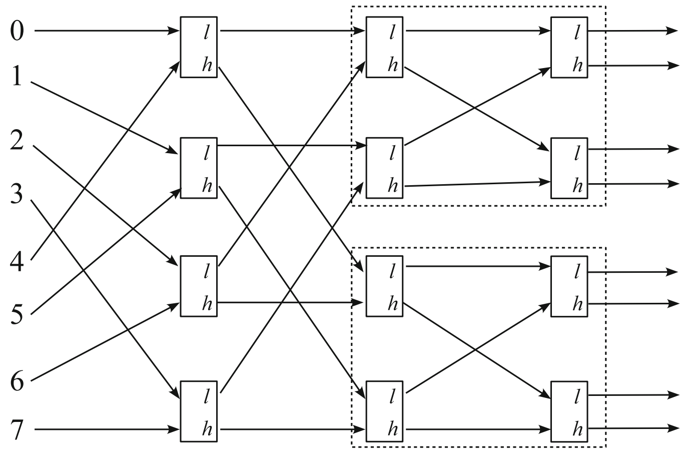

Fig. 3. Biotonic merger of size 8 represented as a graph

1. *there are* n *nodes, called input terminals, with in-degree 0 and out-degree 1, labeled from 0 to* $n-1$*;*
2. *there are* n *nodes, called output terminals, with in-degree 1 and out-degree 0, labeled from 0 to* $n-1$*;*
3. *all the remaining nodes* u*, representing comparators, have in-degree and out-degree 2.*

Figure 3 represents the bitonic merger under the above formalism. We define the depth of a node u of network T, $d(u)$, as the length of the longest path in T from an input node to u. The depth of network T, $d(T)$, is the maximum depth of a node of in-degree and out-degree 2 in T.

The last stage of an n-element bitonic sorting network contains a bitonic merging network with n inputs. This has a depth of *log n*. The other stages perform a complete sort of $n/2$ elements. Hence, the depth, $d(T)$, of the network in Fig. 2 is given by $\Theta(log^2 n)$.

The arcs of a network can be partitioned in n arc-disjoint paths, each joining an input node to an output node. Such a partition yields a line-representation of T, as in [12].

4 Bitonic Sorting of Integers Using SN PA Systems with Rules on the Synapses

We note that the above representation is a theoretical model which indicates the comparisons between input values. However, in the context of SN PA systems with rules on synapses, this model has a straightforward implementation. We encode the positive numbers as the number of spikes, negative numbers as the number of anti-spikes and zero with the symbol λ. Each wire is now represented by a synapse between two neurons, and each value x travels between two neurons as x spikes/anti-spikes, one spike/anti-spike per time unit. Comparators are implemented by a set of neurons which send the minimum and the maximum (as number of spikes/anti-spikes) through designated synapses. Once these two ingredients are at hand, we proceed to construct an SN PA system in the same way the original sorting network was constructed.

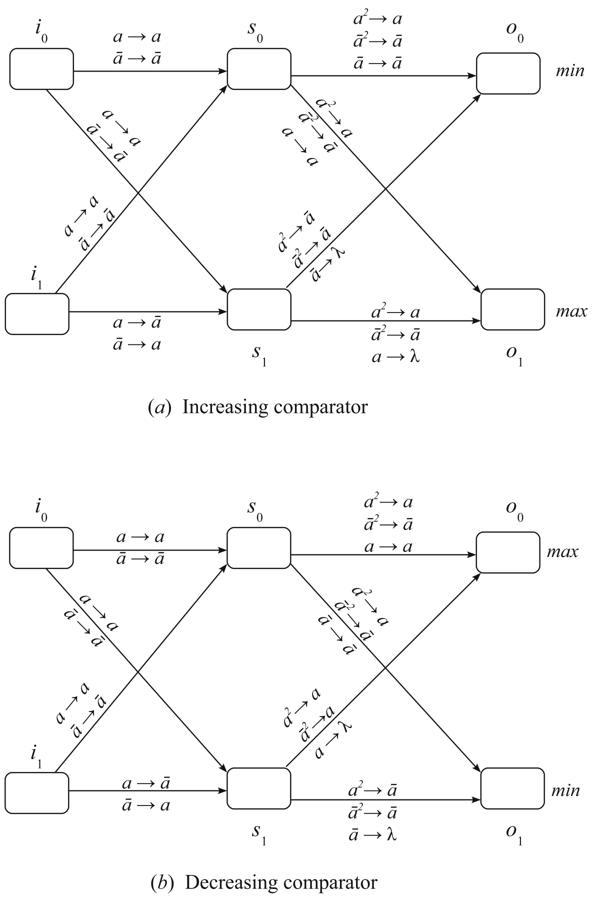

Fig. 4. SN PA with rules on synapses as comparator of integers

In this section we are concerned only with comparators of two elements, hence with SN PA systems which sort two numbers (for brevity called SN PA comparators). In Fig. 4(a) we give an ascending comparator, and in Fig. 4(b) we give a descending comparator. Consider the SN PA system modeling an ascending comparator in Fig. 4(a) and the numbers x and y to be sorted. In order to be able to use these SN PA systems with rules on synapses as building blocks of a bitonic sorting network, we assume that instead of loading the numbers

x and y as spikes/anti-spikes in i_0 and i_1 in the initial configuration, they are fed one by one to these input neurons by another neuron. At each step they instantaneously send one spike/anti-spike to both s_0 and s_1. Here there are two cases. The first case is if both the values are non-negative (negative) then as long as both the neurons i_0 and i_1 are sending their spikes (anti-spikes) in each step of the computation, only s_0 has two spikes (two anti-spikes) and thus sending a spike (an anti-spike) to both the output neurons o_0 and o_1. During these steps neuron s_1 remains empty because of the annihilation of spike and anti-spike it receives. After one input neuron has consumed all its spikes (anti-spikes), the minimum (maximum) is obtained in o_0 (o_1). There will be only one input neuron to send spikes (anti-spikes) to s_0 and s_1. In this case also, the outgoing synapse from s_1 forgets its spike (anti-spike), and s_0 forwards it to o_1 (o_0), where the maximum (minimum) is obtained.

The other case is if one value is non-negative and the other one is negative then as long as both i_0 and i_1 are sending their spikes/anti-spikes, only s_1 has two spikes or two anti-spikes and thus sending an anti-spike to neuron o_0 and a spike to neuron o_1 (since negative values are always less than non-negative values). During these steps neuron s_0 remains empty because of the annihilation of spike and anti-spike it receives. After one input neuron has consumed its spikes (anti-spikes), the maximum (minimum) value is obtained in o_1 (o_0). There will be only one input neuron to send anti-spikes (spikes) to s_0 and s_1. In this case, the outgoing synapse from s_1 forgets its spikes (anti-spikes), and s_0 sends them to o_0 (o_1), where the minimum (maximum) is obtained Now we prove the composition lemma for SN PA increasing comparators.

Lemma 1. *(Composition lemma for increasing comparator). Suppose that in each time unit from t_0 until $t_0 + (|x| - 1)$ neuron i_0 receives one spike/anti-spike and that in a rest it does not receive any spike/anti-spike. Analogously, suppose that in each time unit from t_0 to $t_0 + (|y| - 1)$ neuron i_1 receives one spike/anti-spike, and that in a rest it does not receive any spike/anti-spike. Then neurons o_0 and o_1 either or both receive spike/anti-spike only for time moments from $t_0 + 2$ until $t_0 + 2 + (max(|x|, |y|) - 1)$ and at time moment $t_0 + 2 + max(|x|, |y|)$, the minimum and maximum of x and y codified as number of spikes/anti-spikes are stored in o_0 and o_1 respectively.*

Proof. Consider the time moment t, with $t_0 \leq t \leq t_0 + (min(|x|, |y|) - 1)$. Both neurons i_0 and i_1 receive spikes/anti-spikes and in turn send them through the synapses by the rules. One of the neurons s_0 and s_1 has spikes/anti-spikes depending on the values of x and y, neuron s_0 or s_1 sends one spike/anti-spike to both of the neurons o_0 and o_1. Therefore at time moment $t + 2$ neurons o_0 and o_1 receive their first spike/anti-spike each. This continues till the step $t_0 + 2 + (min(|x|, |y|) - 1)$. From time moment $t_0 + min(|x|, |y|)$ onward, only one neuron of i_0 and i_1 sends spikes/anti-spikes, hence the rules on the outgoing synapses of s_0 and s_1 prevent one of o_0 and o_1 from receiving other spikes/anti-spikes. The first part of the claim is proved.

At each time moment t, with $t_0 + min(x, y) \leq t \leq t_0 + (max(x, y) - 1)$, one of the neurons o_0 and o_1 receives one spike/anti-spike at moment $t + 2$. After time moment $t_0 + max(|x|, |y|)$ there are no other spikes/anti-spikes enters into the system, hence from time moment $t_0 + 2 + max(|x|, |y|)$ onward there will be no other spikes entering neurons o_0 and o_1. The minimum and maximum of x and y codified as number of spikes/anti-spikes are stored in o_0 and o_1, respectively. A similar lemma is valid in the case of a SN PA decreasing comparator.

Assume that we are given a network T as a graph, and that we have a line representation of it (i.e., a set of n arc-disjoint path linking input terminals with output terminals). Hence, we extend Definition 1, by labeling edges, apart from input and output terminals. For every path that begins with input terminal labeled i, we label all its edges with i. More formally, we have the following definition.

Definition 2 (**Edge labeling**). *Given a graph T as in Definition 1 representing a sorting network, and a line-representation of T, we attach to each edge $e \in E(T)$ that belongs to a path in the line representation of T beginning with i, label $l(e) = l(i)$ (supposing that i is labeled with $l(i)$).*

For example, in Fig. 3, we have a labeled bitonic merger. A SN PA system modeling a sorting network given as a graph is obtained in the following way. For each input terminal node l we have a corresponding input neuron i_l. For each comparator (ascending / descending) we have the s- and o-neurons of a SN PA comparator (ascending / descending). For each edge of the graph between two comparators we have synapses between corresponding SN PA comparators. The output terminal nodes are the o-neurons of the last SN PA comparators.

More formally, we construct and label the SN PA system in the following recursive way.

1. for each input terminal node l we have a corresponding input neuron $i_l = i_{l,1}$, $0 \le l \le n-1$;
2. for each comparator at depth $1 \le k \le d(T)$ with incident edges labeled with l and j, $l < j$, we add the s- and o-neurons of a SN PA comparator, connected in the previously specified way. With the notations in Fig. 4, let s_0 and s_1, and o_0 and o_1 be the s-, and o-neurons, respectively, just added. We add synapses between the following pairs of neurons: $((i_{l,k}, s_0), \{a \to a, \bar{a} \to \bar{a}\})$, $((i_{l,k}, s_1), \{a \to a, \bar{a} \to \bar{a}\})$, $((i_{j,k}, s_0), \{a \to a, \bar{a} \to \bar{a}\})$, $((i_{j,k}, s_1), \{a \to \bar{a}, \bar{a} \to a\})$. Additionally, if $k < d(T)$, we label o_0 with $o_{l,k} = i_{l,k+1}$, and o_1 with $o_{j,k} = i_{j,k+1}$; else we label o_0 with $o_{l,k} = o_l$, and o_1 with $o_{j,k} = o_j$.

As an example, Fig. 5 depicts an SN PA system with rules on synapses which models the bitonic merger of size 8.

Theorem 1. *For any SN PA ascending comparator at depth k corresponding to a comparator with incident edges $l < j$ which carry values x and y, respectively, we have that*

1. *in each time moment from $2(k-1)$ until $2(k-1) + |x|$ neuron $i_{l,k}$ receives one spike;*

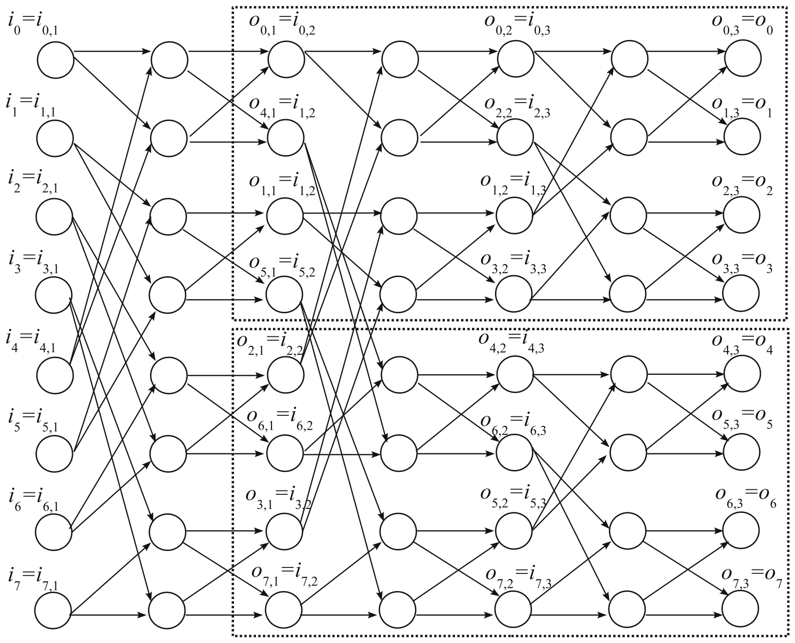

Fig. 5. SN PA system modelling the bitonic merger of size 8

2. *in each time moment from* $2(k-1)$ *until* $2(k-1)+|y|$ *neuron* $i_{j,k}$ *receives one spike.*
3. *in each time moment from* $2k$ *until* $2k+min(|x|,|y|)$ *both the neurons* $o_{l,k}$ *and* $o_{j,k}$ *receive one spike/anti-spike*
4. *in each time moment from* $2k+min(|x|,|y|)+1$ *until* $2k+max(|x|,|y|)$ *either of the neurons* $o_{l,k}$ *and* $o_{j,k}$ *receives one spike/anti-spike*

Proof. We prove the claim by induction on k. When $k=1$ we are at time moment $t=0$. We have explained previously that the behaviour of the system when the spikes/anti-spikes are loaded initially in the input neurons is identical to when they are fed one by one to these neurons. Claims 3 and 4 are true from Lemma 1 and $t_0=0$. We now suppose that the claim is true for k, with $1 \leq k < log\ n$, and prove it for $k+1$. From claims 3 and 4 of the induction hypothesis, we know that both the neurons $o_{l,k}=i_{l,k+1}$, $o_{j,k}=i_{j,k+1}$ receive one spike/anti-spike from $2k$ until $2k+min(|x|,|y|)$, where $min(|x|,|y|)$ is the number of spikes/anti-spikes carried by both the wires l and j before the comparator at depth $k+1$. After that, only one of the neurons $o_{l,k}=i_{l,k+1}$, $o_{j,k}=i_{j,k+1}$ receives one spike/spikes from $2k+min(|x|,|y|)+1$ until $2k+max(|x|,|y|)$, where $max(|x|,|y|)-min(|x|,|y|)$ is the number of spikes/anti-spikes carried by wire l or j before the comparator at depth $k+1$. After the step $2k+max(|x|,|y|)$, the minimum $u=min(x,y)$ is stored $o_{l,k}$ and $v=max(x,y)$ is stored $o_{j,k}$. This proves claims 1 and 2. If we take $t_0=2k$, $x=u$, and $y=v$ in Lemma 1, we have that claims 3 and 4 are true.

5 Bead Sorting of Integers Using SN PA Systems with Rules on the Synapses

Here we design an SN PA system Π_s with rules on synapses that can sort n integers in ascending order. This model drastically decreases the complexity in terms of the number of neurons and synapses. We encode the positive numbers as the number of spikes, negative numbers as the number of anti-spikes and zero with the symbol λ.

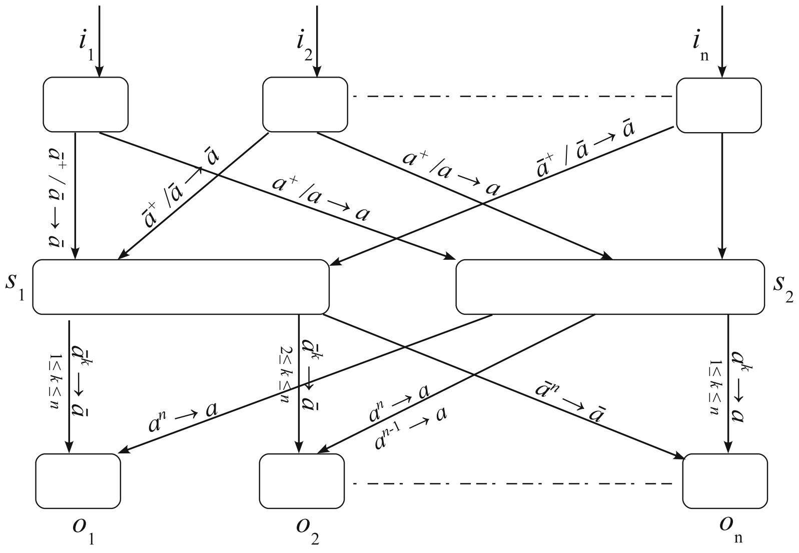

Fig. 6. SN PA system with rules on synapses Π_s for sorting integers

The SN PA system Π_s shown in Fig. 6 has n input neurons, n output neurons and two intermediate neurons (labeled s_1 and s_2). The input is stored in the first line of the system (hence in the neurons labeled i_1, i_2, ..., i_n) encoded in the form of number spikes/anti-spikes. Each input neuron has two synapses, one to neuron s_1 and the other to neuron s_2. At each step, each input neuron until not empty, sends an anti-spike to s_1 if it contains a negative number encoded in the form of number of anti-spikes. If the input neuron contains spikes (i.e., it represents a positive number), then it sends a spike to neuron s_2. So all the input neurons in the first layer of the structure Π_s are having the same type of synapses $((i_l, s_1), \{\overline{a}^+/\overline{a} \rightarrow \overline{a}\})$ and $((i_l, s_2), \{a^+/a \rightarrow a\})$ where $1 \leq l \leq n$. In the second layer the negative and non-negative integers are filtered. Let the number of negative numbers in list be m with $0 \leq m \leq n$. The number of anti-spikes neuron s_1 receives in the first step corresponds to the number of negative numbers in the original unsorted list. So, in the first step of the computation, neuron s_1 receives m anti-spikes which means that there are m number of negative integers and

$n - m$ non-negative integers in the unsorted list. The neuron s_1 sorts the m negative numbers in ascending order and stores them in the first m left most neurons of the third layer. Similarly neuron s_2 sorts the positive numbers and stores them in the rightmost neurons of the third layer.

Intermediate neurons s_1 and s_2 have outgoing synapses to all neurons in the third layer of the system. Depending upon the number of anti-spikes the neuron s_1 receives, it sends an anti-spike to one or more output neurons. At any step during computation, if neuron s_1 has p anti-spikes, then in the next step, it sends an anti-spike to all the p left most output neurons in the third layer. Similarly if the neuron s_2 receives q spikes then it sends a spike to all the q right most output neurons in the third layer. In this way, the SN PA system Π_s can sort n integers in $O(\mid M \mid)$ computational steps, where M is the absolute maximum of the n numbers. We can observe that this model makes use of $2n + 2$ neurons, $4n$ synapses and $n^2 + 3n$ rules to sort n integers, which is comparatively less complex than the system in [11].

6 Conclusion

In this paper we have simulated two parallel sorting algorithms, bitonic sort and bead sort to sort n integers using SN PA systems with rules on synapses. For bitonic sorting, the key operation is the comparison of two elements, so we have designed the SN PA comparators which can compare two integers and arrange them in ascending or descending order. Using these comparators, we have designed the SN PA bitonic sorting network that can perform sorting of an integer array. We simplified the sorting model in [11] using SN PA systems with rules on synapses and also incorporated negative numbers in the unsorted list.

Acknowledgements. Alica Kelemenová was supported by European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and Silesian University in Opava project SGS/24/2013.

References

1. Alhazov, A., Freund, R., Oswald, M., Slavkovik, M.: Extended spiking neural P systems. In: Hoogeboom, H.J., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2006. LNCS, vol. 4361, pp. 123–134. Springer, Heidelberg (2006)
2. Alhazov, A., Sburlan, D.: Static sorting P systems. In: Ciobanu, G., Păun, Gh., Pérez-Jiménez, M.J. (eds.) Applications of Membrane Computing. NCS, pp. 215–252. Springer, Heidelberg (2005)
3. Ardelean, I.I., Ceterchi, R., Tomescu, A.I.: Sorting with P systems: a biological perspective. Rom. J. Inf. Sci. Technol. **11**(3), 243–252 (2008)
4. Arulanandham, J.J.: Implementing bead-sort with P systems. In: Calude, C.S., Dinneen, M.J., Peper, F. (eds.) UMC 2002. LNCS, vol. 2509, p. 115. Springer, Heidelberg (2002)
5. Arulanandham, J.J., Calude, C.S., Dinneen, M.J.: Bead-sort: a natural sorting algorithm. Bull. Eur. Assoc. Theor. Comput. Sci. **76**, 153–162 (2002)

6. Batcher, K.E.: Sorting networks and their applications. In: AFIPS Springer Joint Computer Conference, pp. 307–314 (1968)
7. Binder, A., Freund, R., Oswald, M., Vock, L.: Extended spiking neural P systems with excitatory and inhibitory astrocytes. In: Aggarwal, A., Yager, R., Sandberg, I. W., (eds.) 8th WSEAS International Conference on Evolutionary Computing (EC 2007), pp. 320–325 (2007)
8. Ceterchi, R., Tomescu, A.I.: Implementing sorting networks with spiking neural P systems. Fundamenta Informaticae **87**(1), 35–48 (2008)
9. Dowd, M., Perl, Y., Saks, M., Rudolph, L.: The balanced sorting network. In: Second Annual ACM Symposium on Principles of Distributed Computing, pp. 161–172 (1983)
10. Ionescu, M., Păun, Gh., Yokomori, T.: Spiking neural P systems. Fundamenta Informaticae 71, 279–308 (2006)
11. Ionescu, M., Sburlan, D.: Some applications of spiking neural P systems. J. Comput. Inform. **27**, 515–528 (2008)
12. Knuth, D.E.: The Art of Computer Programming. Sorting and Searching. Addison Wesley Longman, Redwood City (1998)
13. Pan, L., Păun, Gh.: Spiking neural P systems with anti-spikes. Int. J. Comput. Commun. Control **4**(3), 273–282 (2009)
14. Pan, L., Păun, Gh., Pérez-Jiménez, M.J.: Spiking neural P systems with neuron division and budding. Sci. China Inf. Sci. **54**(8), 1596–1607 (2011)
15. Păun, Gh., Rozenberg, G., Salomaa, A. (eds.): Handbook of Membrane Computing, vol. 3, 2nd edn. Oxford University Press, New York (2010)
16. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Berlin (1998)
17. Song, T., Pan, L., Păun, Gh.: Spiking neural P systems with rules on synapses. Theor. Comput. Sci. **529**, 82–95 (2014)
18. The P System Web Page. http://ppage.psystems.eu