Complex Systems **10** (1996) 239–255

Synchronization of One-way Connected Processors*

Salvatore La Torre†
Margherita Napoli
Mimmo Parente
Dipartimento di Informatica ed Applicazioni,
Università degli Studi di Salerno,
84081 Baronissi, Italy

Abstract. A network of identical processors that work synchronously at discrete steps is given. At each step every processor sends messages only to a given subset of its neighboring processors and receives only from the remaining neighbors. The computation starts with one distinguished processor in a particular starting state and all other processors in a quiescent state. The problem is the following: to set all the processors in a given state for the first time and at the very same instant.

This problem is known as the firing squad synchronization problem and was introduced in [9]. In this paper solutions are presented that synchronize processors communicating on one-way links arranged in a ring or in a square with rows and columns that are rings. In particular, we provide optimal algorithms to synchronize both of the networks.

In addition, compositions of solutions are shown and solutions which synchronize at a time $f(n)$ are given for $f(n)$ equal to n^2, $n \log n$, and 2^n.

1. Introduction

We are given a network of identical processors that work synchronously at discrete steps. At each step every processor sends messages only to a given subset of its neighbors and receives messages only from the remaining neighbors. The computation starts with one distinguished processor in a particular starting state and all other processors in a quiescent state. The problem is to set, for the first time and at the very same instant, all the processors in a particular state.

This problem is known as the firing squad synchronization problem (FSSP) as the processors can be viewed as soldiers that have to fire simultaneously.

*A preliminary version of this paper was presented to FCT'97.

†Electronic mail addresses: {sallat,mn,parente}@dia.unisa.it.

The problem was introduced in [9]. In that version each processor in a line of processors could transmit its current state, at each step, to its two adjacent processors. Since then many solutions to the problem and to its variations have been given (e.g., [2, 3, 10, 11]). In [8] it is shown that a solution to the FSSP requires at least $2n - 1$ time, where n is the number of processors in the line. In [12] the first minimal time solution is given and in [5] a minimal time solution is given with the least number of states being six (let us observe that in [1] it was shown that five states are always necessary).

The FSSP was originally introduced to synchronize a line of n processors connected to each other through two-way links. In [7] a constrained version of the problem is investigated: each processor can transmit just one bit of information to the neighboring processors (if any) to the left and to the right. A minimal time solution is provided. Further, nonminimal time solutions to this constrained problem have been proposed [4].

In this paper we consider a different constraint. We focus on the synchronization of processors that communicate on one-way links and are arranged in a ring or in a square with rows and columns that are rings. We present for the first time solutions that synchronize in minimal time such a ring and such a square of processors. In particular, we provide an algorithm to synchronize the ring of n processors in time $2n$ and an algorithm to synchronize the square of $n \times n$ processors in time $3n-1$ and we prove that both of these algorithms are optimal in time. As a matter of fact, the optimality of the algorithm for the ring contradicts a result of [2] claiming to synchronize the ring in time $2n - 1$. In section 3.1 we prove that a synchronization of such a ring requires at least time $2n$.

We also show different ways to combine solutions to obtain a new solution. In particular, if we have two solutions in time $t_1(n)$ and $t_2(n)$, we provide solutions in time $t_1(n) + t_2(n) + d$, for a given integer d, and $t_1(n)t_2(n)$. Finally, given a condition $P(n)$ and two solutions in time $t_1(n)$ and $t_2(n)$, a solution is provided with time $t_1(n)$, if $P(n)$ holds, and $t_2(n)$ otherwise.

In [6] the problem of the composition of different cellular automata (CA) was posed. However, there the composition was reduced to *space-time constructibility* of CA in the following sense: a pair of functions $(g(n), f(n))$ is space-time constructible if there exists a CA that synchronizes $g(n)$ cells at time $f(n)$, for all n. In this paper we consider $g(n) = n$ and give algorithms for synchronizing at the times $f(n)$ of the following types: $n \log n$, n^2, and 2^n.

The rest of this paper is organized as follows. In section 2 we give the definitions and recall some known results. In section 3 we give the minimal time solutions for the ring and for the square of one-way connected processors and prove that they are optimal in time. The composition of two solutions, as a method of obtaining new solutions, is treated in section 4 while in section 5 the solutions in time n^2, $n \log n$, and 2^n are given. Section 6 contains some conclusions.

2. Preliminaries

Let A be an array of identical connected finite state processors that are all in a quiescent state, called the *latent* state. At time $t = 1$ some processors are awakened by an external input and thus enter a particular state, called the *general* state, while all the others remain in the latent state. The problem is to program the processors in such a way that they simultaneously enter for the first time a *firing* state. In this definition many parameters may be fixed: the dimension of the array (one-dimensional, two-dimensional, *etc.*), the type of communication among the processors (one-way or two-way), and the choice of the processors which are awakened at the initial time. Here we consider a ring-shaped network (circular array) and a square-shaped network (two-dimensional circular array) of one-way connected processors, see Figure 1.

All the processors are indistinguishable, but for descriptive reasons, we will number them. Thus when a processor is awakened, entering the general state, we say that it is the processor number 0 (or $(0, 0)$ in the two-dimensional case) and all the others are consecutively numbered from left to right and from up to down. The numbers denoting the processors are all taken modulo the length of the array (or the length of the rows in the case of the square). We now give the definitions of the two models.

Ring of one-way connected processors

A ring of n one-way connected processors (one-way ring) can be formally seen as a one-way circular CA (one-way CA) consisting of n identical finite-state processors (also called *cells*) connected through one-way channels. The one-way ring is denoted by the triple (Q, δ, n) where Q is a finite set of states and $\delta : Q \times Q \to Q$ is the transition function. In a one-way ring the ith cell is connected to the $(i - 1)$ and $(i + 1)$ cells, for all $i = 0, \ldots, n - 1$. Each cell exchanges bits with its adjacent cells: it receives bits from the left and sends bits to the right. Then the cell modifies its state depending on its current state and the state of the adjacent cell on the left. We consider the time-unrolling of the one-way CA $A = (Q, \delta, n)$, that is, we discuss a

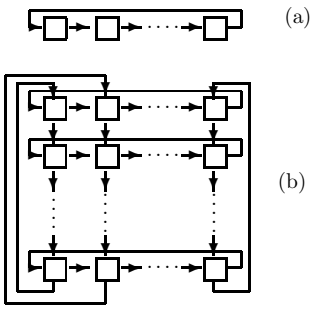

Figure 1: A one-way ring of n processors (a) and a one-way square of $n \times n$ processors (b).

space-time two-dimensional array. A pair (k,t) of this array, with $0 \leq k \leq n-1$ and $t \geq 1$, denotes the cell k at time t. The state of the cell k at time t is denoted by $\text{state}(k,t)$, thus we have that $\text{state}(k,t) = \delta(\text{state}(k-1,t-1), \text{state}(k,t-1))$, for $0 \leq k \leq n-1$, $t > 1$. A *configuration* of A is an n-tuple of states of Q. A configuration at time t is the n-tuple $(\text{state}(0,t), \ldots, \text{state}(n-1,t))$. A solution to the FSSP on a one-way ring in time $t(n)$, with starting configuration C_1, is a pair $S = (Q, \delta)$. The set of states Q contains three particular states: G (general), L (latent), and F (firing). The transition function is such that $\delta(L,L) = L$ and, for every n, the configurations C_t at time t of the one-way CA (S,n), for $1 \leq t \leq t(n)$, have the following conditions.

1. C_1 contains only the states G and L.
2. C_t does not contain the state F, for $1 \leq t < t(n)$.
3. $C_{t(n)}$ is the n-tuple $(F, \ldots, F)$.

The usual starting configuration for a solution to the FSSP on a one-way ring is $C_1 = (G, L, \ldots, L)$, and we assume it as the starting configuration by default, if not specified otherwise.

Square of one-way connected processors

A square of one-way connected processors (one-way square) can be formally seen as a two-dimensional one-way circular CA (two-dimensional one-way CA) which consists of $n \times n$ identical cells (finite-state processors) and is denoted by a triple (Q, δ, n) where Q is a finite set of states and $\delta : Q \times Q \times Q \to Q$ is the transition function.

The rows and the columns of the array are one-way rings of n cells, thus each cell (i,j) modifies its state depending on its current state and on the states of the cells $(i-1,j)$ and $(i,j-1)$.

As in the case of a one-way ring, we consider the three-dimensional array representing the time-unrolling of the one-way square. A triple (i,j,t) of this array, with $0 \leq i,\ j \leq n-1$, and $t \geq 1$, denotes cell (i,j) at time t. The state of cell (i,j) at time t is denoted by $\text{state}(i,j,t)$ and we have that $\text{state}(i,j,t) = \delta(\text{state}(i-1,j,t-1), \text{state}(i,j-1,t-1), \text{state}(i,j,t-1))$, for $0 \leq i,\ j \leq n-1$, and $t > 1$. A *configuration* of A is an $n \times n$-tuple of states from Q. A configuration at time t and a solution (Q, δ) to the FSSP on a one-way square are defined in a way analogous to the one-way ring.

As for the one-dimensional case, the usual starting configuration for a solution to the FSSP on a one-way square is that having the general in the cell $(0,0)$ and all the other cells are latent; we assume this as the starting configuration by default, if not specified otherwise.

Note that the time taken by a solution is sometimes expressed in terms of the number of steps, (e.g., [2, 3]), and sometimes with the number of configurations of the solution, (e.g., [4, 7]). Obviously, given a solution A, if $t(n)$ is the number of its steps then $t(n)+1$ is the number of its configurations. In this paper the time is expressed by the number of configurations.

2.1 Previous results

The FSSP was introduced in [9] as the problem of synchronizing a line of n identical processors, where each processor can exchange information only with its adjacent processors. To date, some variants of this problem have been introduced. In this section we briefly recall the FSSP on one-, two-, and three-dimensional arrays of two-way connected processors. The FSSP on a one-dimensional array is usually referred to as the FSSP on a line.

Let the processors be arranged in a k-dimensional array, we assume that the distance between two processors, respectively at positions $(i_1, \ldots, i_k)$ and $(j_1, \ldots, j_k)$, is given by $\sum_{l=1}^{k} |i_l - j_l|$. In the variants of the FSSP considered in this section, every processor can exchange information only with the processors at distance 1. Note that these topologies are not circular, and this implies that the processors are not identically connected since the processors on the border have a missing link. This makes a clear difference with respect to the case of one-way rings and one-way squares. The definitions of configuration and solution for each of these variations of the FSSP are analogous to the case of the one-way ring and one-way square, thus we omit them.

Let us recall a well-known result of the FSSP on a line of n processors.

Theorem 1. *Given a line of n two-way connected processors, there is a solution to the FSSP in time $2n-1$ and every other solution has time greater than or equal to $2n-1$. Moreover, if the leftmost and rightmost processors of the line are in the general state in the starting configuration, then there is a solution to the FSSP in time n.*

In [11] the following results are shown, optimal in time, for the FSSP on two-dimensional $n \times n$ arrays and three-dimensional $n \times n \times n$ arrays.

Theorem 2. *There is a solution to the FSSP on a two-dimensional array of $n \times n$ two-way connected processors in time $2n-1$ and every other solution has time greater than or equal to $2n-1$.*

Theorem 3. *There is a solution to the FSSP on a three-dimensional array of $n \times n \times n$ two-way connected processors in time $3n-2$ and every other solution has time greater than or equal to $3n-2$.*

In [3] the FSSP is dealt with on connected nonempty subsets of a two-dimensional array of processors and minimal time solutions are given for some classes of such subsets.

In [2] the problem of synchronizing the one-way ring is considered: in Corollary 2 an algorithm for a minimal time solution to synchronize a n-cell ring in time $2n-1$ is given. We *contradict* this result in Lemma 1 by giving a lower bound of time $2n$ for the synchronization of one-way rings.

3. Minimal time solutions

In this section we first give the lower bounds on the time of the solutions to the FSSP on one-way rings and one-way squares and then present the algorithms for the synchronization in minimal time.

3.1 Lower bounds on the time of the solutions

In Lemma 1 we show that time $2n$ is necessary to synchronize a one-way ring of n processors. In Lemma 2 we show that the minimal time is $3n-1$ for a one-way square of $n \times n$ processors.

Lemma 1. *The time of every solution to the FSSP on a one-way ring of n processors is greater than or equal to $2n$.*

Proof. Assume by contradiction that there exists a solution $S = (Q, \delta)$ within time $\bar{t}(n) < 2n$ on a one-way ring and let $A = (Q, \delta, n)$ and $B = (Q, \delta, 2n)$ be two one-way rings. Since for all $t < n$, $\text{state}_A(n-1, t) = L$ and $\text{state}_B(2n-1, t) = L$, then $\bar{t}(n) \geq n$ and $\text{state}_A(i, t) = \text{state}_B(i, t)$ for all $0 \leq i \leq n-1$ and $1 \leq t \leq n$. The following, simple, observation for both A and B is crucial for the rest of the proof. The state of the cell $n-1$ at time $n+t$, for $0 \leq t \leq \bar{t}(n) - n$, depends on the states at time n of the following cells: $n-1$ and $n-2$, when $t = 1$; $n-1$, $n-2$, and $n-3$, when $t = 2$; and in general on the states of the cells $n-1, \ldots, n-t-1$ for $2 < t \leq \bar{t}(n) - n$. As a consequence, at time $\bar{t}(n)$ the cell $n-1$ of both A and B will enter the state F. Anyway, the cell $2n-1$ of B at time $\bar{t}(n)$ is still in the state L, thus we have a contradiction. ■

Lemma 2. *The time of every solution to the FSSP on a one-way square of $n \times n$ processors is greater than or equal to $3n-1$.*

Proof. Assume by contradiction that there exists a solution $S = (Q, \delta)$ in time $\bar{t}(n) < 3n-1$ on a one-way square and let $A = (Q, \delta, n)$ and $B = (Q, \delta, 2n)$ be two one-way squares. Since for all $t < n$ and $0 \leq i \leq n-1$, $\text{state}_A(i, n-1, t) = \text{state}_A(n-1, i, t) = L$ and $\text{state}_B(i, 2n-1, t) = \text{state}_B(2n-1, i, t) = L$, then $\text{state}_A(i, j, t) = \text{state}_B(i, j, t)$ for all $0 \leq i, j \leq n-1$, and $1 \leq t \leq n$. Furthermore, for both A and B the state of cell (i, j) at time n is L for all cells (i, j) such that $i + j > n-1$. The state of the cell $(n-1, n-1)$ at time $n+t$, for $0 \leq t \leq \bar{t}(n) - n$, depends on the states at time n of the cells $(n-1-u, n-1-v)$, for $u + v \leq t$. As a consequence, at time $\bar{t}(n)$ the cell $(n-1, n-1)$ of both A and B will enter the state F. Anyway, since the cell $(2n-1, 2n-1)$ of B at time $\bar{t}(n)$ is still in the state L, we have a contradiction. ■

3.2 Synchronization in minimal time

In this section we present the minimal time algorithms for the synchronization of a one-way ring and a one-way square.

Lemma 3. *There is a solution to the FSSP on a one-way ring of n processors in time $2n$.*

Proof. Using standard techniques, a computation of a two-way CA A of n processors in time $t(n)$ can be executed by a one-way CA B in time $2t(n)$, provided that the initial configuration of A can be reached in one step from the initial configuration of B. We informally use an induction on the number of steps. Let the state of the cell $i+1$ of B after the first step be equal to the state of the cell i of A at the starting configuration and assume that the cell $i+j$ of B at time $2j$ has the state that cell i of A has at time j. (To be more precise, since the cell $i+j$ of B has to simulate the cell i of A, then when $i=0$ or when $i=n-1$ the state of the cell $i+j$ of B encodes a state of A and the information that the simulated cell is the leftmost or the rightmost in the line.) Now the cell i of A at the jth step needs the states of cells $i-1$ and $i+1$ at time j. Cell $(i-1)+j$ of B at step $2j$ passes its own state p to the cell $(i+j)$ and this in turn forwards p along with its state to the right neighboring cell, the cell $(i+1)+j$, that at step $2j+1$ can simulate cell i of A at step $j+1$. So the cell $i+(j+1)$ at time $2(j+1)$ contains the state of the cell i of A at time $j+1$. The overall simulation thus takes a multiplicative delay factor of two.

Let us consider now the solution S of Theorem 1 on a two-way connected line starting with the leftmost and rightmost cells in the general state. This solution takes time n and a solution S' to the FSSP on a one-way ring in time $2n$ can be obtained with the above simulation. More precisely, in the first step S' lets the second cell enter a general state, so that the state of the cell $i+1$ after the first step of S' is equal to the state of the cell i in the starting configuration of S. ■

Lemma 4. *There is a solution to the FSSP on a one-way square of $n \times n$ processors in time $3n-1$.*

Proof. We first give a solution that is easier to describe which takes time $3n$ and then show how to save one time unit.

Consider a square of $n \times n$ two-way connected processors. For our purpose it is convenient if we look at the square organized in n concentric frames, where the $(i+1)$ inner frame is constituted by the four lines $(i,i)\ldots(i,n-i-1)$, $(i,n-i-1)\ldots(n-i-1,n-i-1)$, $(i,i)\ldots(n-i-1,i)$, and $(n-i-1,i)\ldots(n-i-1,n-i-1)$, see Figure 2. Suppose now that the cells $(0,0),(0,n-1),(n-1,0)$, and $(n-1,n-1)$ are all in the same general state, then the following statement holds: a solution to the FSSP on a square of $n \times n$ two-way connected processors can be executed in time n. In fact, the four lines of the first frame can all synchronize in time n using Theorem 1; during such synchronizations, after the first two steps, the four cells $(1,1)$, $(1,n-2)$, $(n-2,1)$, and $(n-2,n-2)$ all enter a general state and thus the four lines of the second frame can synchronize in time $n-2$. Iterating this argument, the ith frame synchronizes in time $n-2(i-1)$, $1 \leq i \leq \lceil n/2 \rceil$. As this synchronization starts at time $2(i-1)+1$, then the overall time to synchronize the processors is still n. Let us call this solution S.

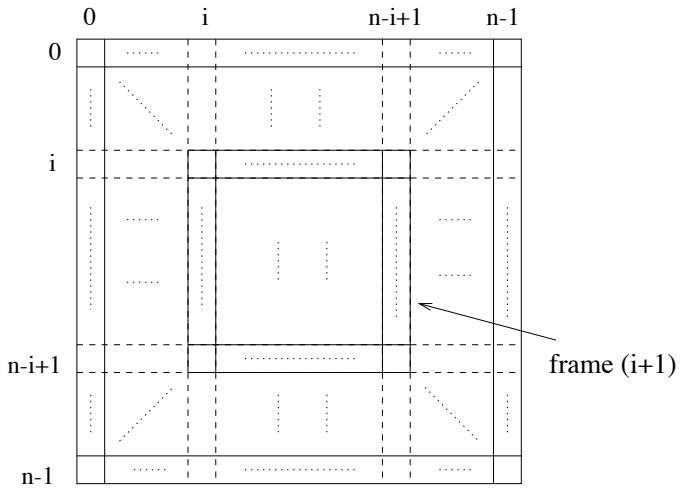

Figure 2: The frames in a square of $n \times n$ processors.

Using standard techniques (as in the previous proof), any computation of a two-dimensional two-way CA A in time $t(n)$ can be executed by a two-dimensional one-way CA B in time $3t(n)$ in the following way. We informally use an induction on the number of steps. Assume that the cell $(i+1, k+1)$ in the third configuration of B contains the state that the cell (i, k) has in the first configuration of A and that cell $(i+j, k+j)$ of B at time $3j$ has the state that cell (i, k) of A has at time j. Actually, when the cell (i, k) is a border cell, that is, when either $i \in \{0, n-1\}$ or $k \in \{0, n-1\}$, this information is also stored in the state of the cell $(i+j, k+j)$ of B. Now the cell (i, k) of A at the jth step computes the new state from its own state and the states of cells $(i-1, k)$, $(i, k-1)$, $(i+1, k)$, and $(i, k+1)$ at time j. Within three steps the cell $(i+(j+1), k+(j+1))$ of B can collect the states that at time $3j$ are in the cells $(i+j, k+j)$, $((i-1)+j, k+j)$, $(i+j, (k-1)+j)$, $((i+1)+j, k+j)$, and $(i+j, (k+1)+j)$. These three steps can be described as follows.

1. At step $3j$, cell $(i+j, k+j)$ of B stores the two states p, q of cells $((i-1)+j, k+j)$ and $(i+j, (k-1)+j)$.
2. At step $3j+1$ the states p, q are passed to cells $((i+1)+j, k+j)$ and $(i+j, (k+1)+j)$ (note that in the previous step the state of cell $(i+j, k+j)$ at time $3j$ has been passed to these cells).
3. At step $3j+2$, cell $((i+1)+j, (k+1)+j)$ simulates cell (i, k) of A at step j.

So the state of the cell $(i+(j+1), k+(j+1))$ of B at time $3j+3$ contains the state that the cell (i, k) of B has at time $j+1$. The overall simulation thus takes a multiplicative delay factor of three.

Consider now a solution S' which in the first two steps reaches a configuration such that the states of all the cells $(0,0),(0,1),(1,0)$, and $(1,1)$ contain the general state (recall that the states of the cells $(0,0),(0,n-1),(n-1,0)$, and $(n-1,n-1)$ in the starting configuration of the solution S are all in the general state). Then S' simulates the solution S within time $3n$.

Now let us briefly explain how S' can be modified to save one step, thus reaching time $3n-1$. The first $3n-3$ steps (and thus the first $3n-2$ configurations) remain unmodified. Now let $A=(S,n)$ and $A'=(S',n)$, for a given n, and let us observe what follows.

1. Each cell of A in the configuration j participates for the synchronization of the frame which it belongs to; actually each cell participates either only for a row line or only for a column line of the frame except for the four corner cells of the frame which participate for both of the lines. The same holds also for A' in the configurations $3j$ (due to the mapping between the cells of the configuration j of A and those of configuration $3j$ of A').

2. At time $3j+2$ in A', $1 \leq j < n$, a cell $(i+(j+1),k+(j+1))$ is aware of the states at time $3j$ of the following cells.

   (a) $((i-1)+(j+1),(k-1)+(j+1))$, $(i+(j+1),(k-2)+(j+1))$, $(i+(j+1),(k-1)+(j+1))$, and $(i+(j+1),k+(j+1))$.

   (b) $((i-1)+(j+1),(k-1)+(j+1))$, $((i-2)+(j+1),k+(j+1))$, $((i-1)+(j+1),k+(j+1))$, and $(i+(j+1),k+(j+1))$.

Thus at step $3n-2$, the cell $(i+n,k+n)$ can correctly simulate either cell $(i,k-1)$ or cell $(i-1,k)$ of S at step $n-1$, hence entering the firing state. In particular, the cell $(i+n,k+n)$ simulates the former if $(i,k-1)$ participates in the synchronization for a row line, or simulates the latter, if $(i-1,k)$ participates in the simulation for a column line (note that at least one of these conditions must hold). Then, there is a solution to the FSSP on a one-way square of $n \times n$ processors in time $3n-1$. We stress that one step can be saved only at the configuration $3n-1$ and not at some earlier time, because in the last step all cells must enter the same state (the firing state) and the information about the frames can be lost. ■

Now we can give the main results of this section.

Theorem 4. *Given a one-way ring of n processors, there is a solution to the FSSP in time $2n$ and every other solution has time greater than or equal to $2n$.*

Theorem 5. *Given a one-way square of $n \times n$ processors, there is a solution to the FSSP in time $3n-1$ and every other solution has time greater than or equal to $3n-1$.*

4. Composition of solutions

In general it may be useful to design solutions that synchronize the processors in times which are not minimal. Anyway, the design of such solutions may not be obvious and tools allowing various combinations of two or more solutions may be useful.

In this section we show how to compose more solutions to obtain a new solution. In particular, if we have two solutions in time $t_1(n)$ and $t_2(n)$, in Lemma 5 and in Lemma 6 we show how to obtain solutions in time $t_1(n) + t_2(n) + d$, for a given constant d, and in time $t_1(n)t_2(n)$, respectively. We conclude the section with Lemma 7 in which, given a condition $P(n)$, a solution is provided having time $t_1(n)$, if $P(n)$ holds, and time $t_2(n)$, otherwise.

In the following, we use the product of automata as a method for combining one-way CA, in fact, given two one-way CA A_1 and A_2, the product automaton $A_1 \times A_2$, defined in a standard way, behaves simultaneously as A_1 and A_2. Furthermore, if S_i is a solution to the FSSP then G_i, L_i, and F_i denote the general, latent, and firing states of S_i, respectively, and Q_i, δ_i denote the set of states and the transition function, respectively.

Lemma 5. *If S_i, $i = 1, 2$, are two solutions to the FSSP on a one-way ring of n processors (resp. one-way square of $n \times n$ processors) in time $t_i(n)$, then there is a solution to the FSSP on a one-way ring of n processors (resp. one-way square of $n \times n$ processors) in time $t_1(n) + t_2(n) + d$ for all $d \geq -\min\{t_1(n), t_2(n)\}$.*

Proof. The proof is very similar in the one-dimensional and two-dimensional cases. We now consider only the one-dimensional case. Let $d \geq 0$. It is easy to give a solution S to the FSSP on a one-way ring such that S behaves as S_1 from time 1 up to time $t_1(n)$, then at time $t_1(n) + 1$ it switches to S_2. Thus S is a solution in time $t_1(n) + t_2(n)$. Furthermore, given a solution to the FSSP $S' = (Q', \delta')$ in time $t(n)$ and with firing state F'_0, a solution in time $t(n) + d$ can be obtained from S' by adding the states $F'_1, \ldots, F'_d$ and the transition rules $\delta'(F'_i, F'_i) = F'_{i+1}$ for $i = 0, \ldots, d-1$. Obviously, F'_d is the firing state of the resulting solution.

Now, let us consider $0 > d \geq -\min\{t_1(n), t_2(n)\}$. As $|d| \leq t_1(n)$, it is possible to modify the solution S_1 to mark in time $|d|$ exactly $|d|$ cells with $|d|$ different states (actually a marker is a component of the state of a processor). Clearly, if $n < |d|$ then a cell is marked twice or more. Thus, the solution S in time $t_1(n) + t_2(n) - |d|$ behaves as S_1 up to time $t_1(n) - 1$, then it switches to the $|d|$th configuration of (S_2, n) and behaves as S_2. ■

Lemma 6. *If S_i, $i = 1, 2$, are two solutions to the FSSP on a one-way ring of n processors (resp. one-way square of $n \times n$ processors) in time $t_i(n)$, then there is a solution to the FSSP on a one-way ring (resp. one-way square of $n \times n$ processors) in time $t_1(n)t_2(n)$.*

Proof. There are no substantial differences if we consider one-dimensional or two-dimensional one-way CA. Thus, we informally describe how to obtain the solution in time $t_1(n)t_2(n)$ in both cases. We define a solution S consisting of a phase of length $t_1(n)$, which is iterated $t_2(n)$ times. The set of states of S is $Q_1 \times Q_2$, the general, latent, and firing states are the pairs (G_1, G_2), (L_1, L_2), and (F_1, F_2), respectively. In each iteration S modifies the first component of its state according to the transition function of S_1, until this component becomes F_1. At the end of this phase S executes a transition modifying the second component of the state according to the transition function of S_2. Moreover in this same step, S replaces F_1 with either G_1 or L_1 in the first component (depending on whether the cell is the first or not). Now the iterative phase starts again until the firing state (F_1, F_2) is entered by the cells. Thus the solution S_1 is iterated exactly $t_2(n)$ times. ■

A construction of automata slightly different from the product of automata can be used to design a solution that synchronizes by selecting among different solutions. The selection is made according to a given condition, for example, the parity of the number of processors or the fastest (or slowest) solution in the set. The following notion of *selecting* CA is our formalization of the condition to be tested. Let Q be a finite set, $\delta : Q \times Q \to Q$ and let O_1 and O_2 be disjoint subsets of Q. We say that (Q, δ, O_1, O_2) is a selecting CA in time $t(n)$ if for all n the configurations C_t of (Q, δ, n) are such that: C_1 is the usual starting configuration for a solution to the FSSP and $C_t \in O_1^n$, for all $t \geq t(n)$ or $C_t \in O_2^n$, for all $t \geq t(n)$. A selecting CA for the two-dimensional case is analogous. Lemma 7 shows how to design a one-way CA that selects a solution between two given solutions, according to a condition on the number of cells. Clearly, by iterating this construction, a selection among more than two solutions can be realized.

Lemma 7. *Let S_i be a solution to the FSSP on a one-way ring of n processors (resp. one-way square of $n \times n$ processors) in time $t_i(n)$ for $i = 1, 2$, let $K = (Q, \delta, O_1, O_2)$ be a selecting CA in time $t(n) \leq t_i(n)$, and let C_t be the configurations of K. Then there is a solution to the FSSP on a one-way ring of n processors (resp. one-way square of $n \times n$ processors) in time $s(n)$ such that if $C_{t(n)} \in O_1^n$ then $s(n) = t_1(n)$, otherwise $s(n) = t_2(n)$.*

Proof. For simplicity we consider only the one-dimensional case, the two-dimensional case is similar.

Let G_K and L_K be the general and latent states of K. Assume without loss of generality that the transition rules δ_i are such that $\delta_i(F_i, F_i) = F_i$, $i = 1, 2$. The solution S can be defined as $S = ((Q_1 \times Q_2 \times Q) \cup \{F\}, \delta)$ where δ is the transition function:

$$\delta((q_1, q_2, q_K), (q_1', q_2', q_K')) = F \text{ if } (\exists i : \delta_i(q_i, q_i') = F_i) \text{ and } \delta_K(q_K, q_K') \in O_i$$

and

$$\delta((q_1, q_2, q_K), (q_1', q_2', q_K')) = (\delta_1(q_1, q_1'), \delta_2(q_2, q_2'), \delta_K(q_K, q_K')), \text{ otherwise.}$$

Trivially, it is a solution to the FSSP with (G_1, G_2, G_K), (L_1, L_2, L_K), and F as the general, latent, and firing states, respectively. Thus, the synchronization is obtained in time $t_i(n)$ if at this time the selecting CA K is in a configuration with all the states belonging to the set O_i. ■

We show two examples as applications of Lemma 7. In the first example we are faced with the problem of obtaining a solution synchronizing at the maximum or minimum time between two solutions. We first define a selecting CA performing the test $t_1(n) \leq t_2(n)$, then we show that this selecting CA can be used to obtain a solution synchronizing at the maximum or minimum time between two solutions in time $t_1(n)$ and $t_2(n)$. In the second example a particular behavior is selected depending on the result of a comparison between the number of processors n and a constant h.

Example 1. Let R_i and $i = 1, 2$ be two solutions to the FSSP on a one-way ring in time $t_i(n)$. We define a selecting CA for the condition $t_1(n) \leq t_2(n)$ in time $t(n) = \min\{t_1(n), t_2(n)\}$. Let s_i and $i = 1, 2$ be two states not belonging to $Q_1 \cup Q_2$. We extend the transition rules δ_i by including the rules $\delta_i(F_i, F_i) = F_i$. Let δ' be defined as $\delta'(s_i, s_i) = s_i$:

$$\delta'((q_1, q_2), (q_1', q_2')) = s_1 \text{ if } \delta_1(q_1, q_1') = F_1$$
$$\delta'((q_1, q_2), (q_1', q_2')) = s_2 \text{ if } \delta_2(q_2, q_2') = F_2 \text{ and } \delta_1(q_1, q_1') \neq F_1, \text{ and}$$
$$\delta'((q_1, q_2), (q_1', q_2')) = (\delta_1(q_1, q_1'), \delta_2(q_2, q_2')), \text{ otherwise.}$$

Thus the selecting CA is $K = ((Q_1 \times Q_2) \cup \{s_1, s_2\}, \delta', \{s_1\}, \{s_2\})$. The general and latent states are obviously (G_1, G_2) and (L_1, L_2). The selecting automaton for the same condition in the two-dimensional case can be obtained analogously. The selecting CA K can be used to obtain a solution synchronizing at the maximum or minimum time between two solutions. In fact, let us consider again two solutions R_1 and R_2 in time $t_1(n)$ and $t_2(n)$ respectively. By Lemma 7 with $S_i = R_i$, we can get a solution in time $s(n)$, with $s(n) = t_1(n)$ if $t_1(n) \leq t_2(n)$ and $s(n) = t_2(n)$ otherwise. Thus a solution synchronizing at the minimum time is obtained. If we apply Lemma 7 with $S_1 = R_2$ and $S_2 = R_1$, then a solution synchronizing at the maximum time is obtained.

Example 2. In this example we describe a selecting CA K performing the test $n \leq h$, for a given positive integer h. The selecting automaton for the same condition in the two-dimensional case can be obtained analogously. Let $Q = \{G, L, p_1, \ldots, p_h, p_{\leq h}, p_{>h}\}$ with G and L as the general and latent states respectively, then K is defined as $(Q, \delta, \{p_{\leq h}\}, \{p_{>h}\})$ where δ can be informally described as follows. In the first step the processors 0 and 1 enter the states p_1 and p_2 respectively; next, each processor in the latent state enters state p_{i+1} if its adjacent processor on the left is in state p_i for $i < h$, while it enters the state $p_{>h}$ if this neighbor is in state p_h; the state $p_{\leq h}$ is entered by processor 0 if its adjacent processor on the left (i.e., the processor $n-1$) is in state p_i for $i \leq h$. When a processor enters the state $p_{\leq h}$ or $p_{>h}$ all the other processors are forced to enter the same state within a time n. Obviously, K is a selecting CA in time $t(n) = n + \min\{h, n\}$.

Note that the selecting CA of Example 2 can be used for any pair of solutions, as the time of the selecting CA is less than the time of any solution.

5. Particular time solutions

In this section we show the existence of solutions to the FSSP in time n^2, $n\lceil \log n \rceil$, and 2^n. In particular, the first two solutions are described directly for the one-way CA and the solution in time 2^n is obtained quite easily by using the usual construction to convert a solution for a two-way CA into a solution for a one-way CA.

Theorem 6 gives the solution in time n^2.

Theorem 6. *There is a solution to the FSSP on a one-way ring of n processors and on a one-way square of $n \times n$ processors in time n^2.*

Proof. First we consider the one-dimensional case. Assume $n \geq 3$, the case $n < 3$ can be dealt with using a simple *ad hoc* strategy and is omitted (Lemma 7 can be used with the test $n \geq 3$ to select the behavior, see Example 2).

The solution is divided into two phases: *counting* and *synchronization*. The counting phase has length $(n-2)n+1$ and can be seen as constituted by $n-2$ iterations of a subphase of n steps. At the beginning of each iteration a token T is in the first cell and at each step it is passed from cell j to cell $j+1$. Thus, at the last step of the subphase, T is moved again to the first cell. Moreover, in the first iteration the cell 3 enters a marker M (actually a token and a marker are components of the states of processors). At each successive iteration M is moved one cell to the right, so M is moved to the first cell when $n-2$ iterations have been executed, that is, at time $(n-2)n+1$.

The synchronization phase consists of a minimal time solution (in time $2n$) to the FSSP on a one-way ring. Now we can suppose that during the counting phase all the cells, except for the first cell and the cells in the states containing M and T, are in the latent state. Thus at step $(n-2)n+1$ all the cells, except cell 0, are in the latent state. As a consequence, the synchronization phase can start exactly at time $(n-2)n+1$ and the total solution has thus length n^2.

Now let us consider the two-dimensional case. Here assume $n \geq 5$ and, as before, Lemma 7 is used to select the behavior. The solution in time n^2 is easily obtained through the following two steps.

- The first row is synchronized in time $2n$ with a minimal time solution on a one-way ring.
- A solution in time $n^2 - 2n$ is applied to each column.

The solution in time $n^2 - 2n$ is easily obtained from a solution in time n^2 on a one-way ring and modifying the first iteration of the counting phase in order to mark the cell 5 (instead of 3). In this way the counting phase is constituted by $n-4$ iterations of the subphase, thus saving $2n$ steps. ■

Now we show how to obtain a solution in time $n \log n$. First let us recall that given a line of n two-way connected processors, if the leftmost and rightmost processors are in the general state in the starting configuration, then by Theorem 1 there is a solution to the FSSP on a line of n processors in time n. Note that it is easy to modify the solution in such a way that in the $(n-1)$ configuration the processor $\lceil n/2^i \rceil - 1$, for some i, is in a particular state which is different from all the states entered by the other processors (actually this "marking" of a processor is a standard technique widely used, e.g., [2, 4, 5, 7]). A similar result can be easily obtained for the one-way ring.

Lemma 8. *There is a solution to the FSSP on a one-way ring of n processors in time $2n$ such that in the configuration $2n-1$ the processor $\lceil n/2^i \rceil - 1$, for a given $i \geq 0$ is in a particular state which is different from the state of any other processor.*

Now we are ready for the solution in time $n \log n$.

Theorem 7. *There is a solution to the FSSP on a one-way ring of n processors and on a one-way square of $n \times n$ processors in time $n\lceil \log n \rceil$.*

Proof. First we consider the one-dimensional case. Let us assume for the moment $n > 8$. The solution is divided into three phases: *initialization*, *iterative*, and *synchronization*. The iterative phase is executed if $n > 16$, otherwise it is skipped. Informally speaking, the whole solution is described as follows.

In the initialization phase the cell $\lceil n/16 \rceil - 1$ is marked with a particular state *marker*. Then the cell 0 is marked if and only if $n \leq 16$. Using Lemma 8 this phase can be realized in time $2n$.

In the iterative phase, at the ith iteration the marker is moved from the cell $\lceil n/2^{i+3} \rceil - 1$ to cell $\lceil n/2^{i+4} \rceil - 1$ for $i = 1, \ldots, \lceil \log n \rceil - 4$ and again the cell 0 is marked if $n \leq 2^{i+4}$. The ith iteration starts at time $(i+1)n + 1$ and ends at time $(i+2)n + 1$. Note that the first step of the ith iteration coincides with the last step of the $(i-1)$ iteration. Thus the total time taken by this phase is $n(\lceil \log n \rceil - 4) + 1$. The third phase is actually a minimal time solution. Thus, the total time is $2n + n(\lceil \log n \rceil - 4) + 1 + 2n - 1 = n\lceil \log n \rceil$.

The case $n \leq 8$ can be easily solved with a particular strategy and the appropriate behavior can be selected by using Lemma 7.

Now let us consider the two-dimensional case. Here assume $n > 32$ and, as before, Lemma 7 is used to choose the behavior. The solution in time $n\lceil \log n \rceil$ is easily obtained through the following two steps.

- The first row is synchronized in time $2n$ with a minimal time solution on a one-way ring.
- A solution in time $n\lceil \log n \rceil - 2n$ is applied to each column.

The solution in time $\lceil \log n \rceil - 2n$ is easily obtained from the solution in time $n\lceil \log n \rceil$ on a one-way ring by modifying the initialization phase in order to mark the cell $\lceil n/64 \rceil - 1$ (instead of cell $\lceil n/16 \rceil - 1$) thus saving $2n$ steps. ■

Theorem 8. *There is a solution to the FSSP on a one-way ring of n processors and on a one-way square of $n \times n$ processors in time 2^n.*

Proof. In [4] a solution to the FSSP in time 2^n on a line of n two-way connected processors is shown. Loosely speaking, a solution in time 2^{n-1} can be obtained from this latter result by putting the second cell in a general state and then starting a solution on $n-1$ cells. In an analogous way it is possible to obtain a solution in time 2^{n-2}. Using very standard techniques as in Lemma 1, any computation of a one-dimensional two-way CA A in time $t(n)$ can be executed by a one-dimensional one-way CA B in time $2t(n)-1$. In fact, assume that cell $i+j-1$ of B at time $2j-1$ has the state cell i of A at time j. Now the cell i of A at step j needs the states of cells $i-1$ and $i+1$ at time j. Cell $(i-1)+(j-1)$ of B at step $2j-1$ passes its own state p to the cell $(i+(j-1))$ and this forwards p along with its state to the right neighboring cell, the cell $(i+1)+(j-1)$, that at step $2j$ can simulate cell i of A at step j. Now by this simulation and the above two solutions, solutions to the FSSP on a one-way ring of n processors in time 2^n and 2^{n-1}, respectively, are achieved. Moreover, a solution to the FSSP on a one-way square of $n \times n$ processors in time 2^n can be obtained by first synchronizing the first row in time 2^{n-1} and then all the columns, always with the same solution. ■

6. Conclusions

In this paper networks of processors connected via one-way links are considered. In particular, the focus is on processors arranged in a one-way ring or in a square having rows and columns that are one-way rings. For the first time an algorithm to synchronize the one-way ring of n processors in time $2n$ and an algorithm to synchronize the one-way square of $n \times n$ processors in time $3n-1$ are presented. It is proved that both of these algorithms are optimal in time. The optimality of the algorithm for the ring contradicts a result of [2] which claims to synchronize the ring in time $2n-1$.

Moreover, different ways to combine more solutions in order to obtain new ones are shown. In particular, given two solutions in time $t_1(n)$ and $t_2(n)$, solutions are provided in time $t_1(n)+t_2(n)+d$, for an integer number d, and $t_1(n)t_2(n)$ and given a condition $P(n)$, a solution is provided having time $t_1(n)$, if $P(n)$ holds, and $t_2(n)$ otherwise.

A final contribution of this paper is represented by the algorithms to synchronize the one-way ring of n processors and the one-way square of $n \times n$ processors in time $n \log n$, n^2, and 2^n. These algorithms give the solution to

the *space-time constructibility* of CA [6] for the pairs $(n, f(n))$ where $f(n)$ is one of the above times.

A development of this research is the realization of new solutions in meaningful time and the study of new topologies of networks. The results on the one-way rings and one-way squares can be extended to k-dimensional one-way hypercubes. Concerning the k-dimensional one-way hypercubes of n^k processors we conjecture that the lower bound to the time of a solution to the FSSP on such networks is greater than or equal to $(k+1)n-(k-1)$.

In this paper we have not considered faulty processors and communication channels without noise. We might ask how the algorithms would change to obtain the synchronization with a reasonable likelihood in a certain time if these aspects are considered. In this case we have to consider in our model the probabilities with which the transitions happen and some results of the Code Theory could be used.

Aknowledgements

Work partially supported by M.U.R.S.T. in the framework of the project "*Modelli di Sistemi Concorrenti*." We thank the referee for their careful reading of this paper.

References

[1] R. Balzer, "An 8-states Minimal Time Solution to the Firing Squad Synchronization Problem," *Information and Control*, **10** (1967) 22–42.

[2] K. Culik, "Variations of the Firing Squad Problem and Applications," *Information Processing Letters*, **30** (1989) 153–157.

[3] K. Kobayashy, "The Firing Squad Synchronization Problem for Two-Dimensional Arrays," *Information and Control*, **34** (1977) 177–197.

[4] S. La Torre, M. Napoli, and M. Parente, "Synchronization of a Line of Identical Processors at a Given Time," in *Proceedings of the Colloquium on Trees in Algebra and Programming CAAP '97 (TAPSOFT'97)*, Lille (France), 1997. *Lecture Notes in Computer Science*, **1214** (1997) 405–416.

[5] J. Mazoyer, "A Six States Minimal Time Solution to the Firing Squad Synchronization Problem," *Theoretical Computer Science*, **50** (1987) 183–238.

[6] J. Mazoyer and N. Reimen, "A Linear Speed-up Theorem for Cellular Automata," *Theoretical Computer Science*, **101** (1992) 59–98.

[7] J. Mazoyer, "On Optimal Solutions to the Firing Squad Synchronization Problem," *Theoretical Computer Science*, **168** (1996) 367–404.

[8] M. Minsky, *Computation: Finite and Infinite Machines* (Prentice-Hall, London, 1972).

[9] E. F. Moore, *Sequential Machines, Selected Papers* (Addison Wesley, Reading, 1964).

[10] Z. Roka, "The Firing Squad Synchronization Problem on Caley Graphs," in *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science MFCS'95*, Prague, Czech Republic, 1995. *Lecture Notes in Computer Science*, **969** (1995) 402–411.

[11] I. Shinahr, "Two- and Three-Dimensional Firing-Squad Synchronization Problems," *Information and Control*, **24** (1974) 163–180.

[12] A. Waksman, "An Optimum Solution to the Firing Squad Synchronization Problem," *Information and Control*, **9** (1966) 66–78.