How to Synchronize Large-Scale Ultra-Fine-Grained Processors in Optimum-Time

Hiroshi Umeo
School of Information Engineering
University of Osaka Electro-Communication
Neyagawa-shi, Hastu-cho, 18-8, Osaka
Email: umeo@cyt.osakac.ac.jp

***Abstract*—We introduce a new class of FSSP (firing squad synchronization problem) algorithms based on recursive-halving and construct a survey on recent developments in FSSP algorithms for one-dimensional cellular arrays. We present herein a comparison of the quantitative aspects of the optimum-time FSSP algorithms developed so far. Several state-efficient new implementations and new insights into synchronization algorithms and multi-dimensional expansions are also given.**

Keywords*—*cellular automata, synchronization

I. Introduction

Synchronization of large-scale networks is an important and fundamental computing primitive in parallel and distributed systems. The synchronization in ultra-fine grained parallel computational model of cellular automata, known as firing squad synchronization problem (FSSP), has been studied extensively for more than fifty years, and a rich variety of synchronization algorithms has been proposed. In the present paper, we introduce a new class of FSSP algorithms based on recursive-halving and construct a survey on recent developments in FSSP algorithms for one-dimensional cellular arrays. The algorithms being compared are Balzer [1], Gerken [2], Waksman [20], a number of revised versions thereof, and their generalized versions such as Moore and Langdon [8], Settle and Simon [10], Szwerinski [11], all included in the proposed new class of FSSP algorithms. We present herein a survey and a comparison of the quantitative aspects of the optimum-time synchronization algorithms developed so far. Several state-efficient new implementations, new insights into synchronization algorithms and multi-dimensional expansions are also given.

Specifically, we attempt to answer the following questions:

- First, are all previously presented transition rule sets correct?
- Do these sets contain redundant rules? If so, what is the exact rule set?
- How do the algorithms compare with each other?
- Can we expand those 1D FSSP algorithms proposed so far to 2D, 3D arrays, or more generally to multi-dimensional arrays?
- How can we synchronize multi-dimensional arrays in optimum-time?

In Section 2 we give a description of the FSSP and review some basic results on the FSSP for 1D arrays. Section 3 introduces a new class of FSSP algorithms based on recursive-halving and presents multi-dimensional generalizations of the algorithms. In the last section we give a summary of the paper.

II. Firing Squad Synchronization Problem

In this section, we define the FSSP and introduce some basic results on FSSP.

A. Firing Squad Synchronization Problem

Figure 1 illustrates a finite one-dimensional (1D) cellular array consisting of n cells. Each cell is an identical finite-state automaton. The array operates in lock-step mode in such a way that the next state of each cell is determined by both its own present state and the present states of its left and right neighbors. All cells (*soldiers*), except one *general*, are initially in the quiescent state at time $t = 0$ with the property that the next state of a quiescent cell with quiescent neighbors is the quiescent state again. At time $t = 0$, one general cell C_1 is in the *fire-when-ready* state, which is the initiation signal for the synchronization of the array. The FSSP is to determine a description (state set and next-state function) for cells that ensures all cells enter the *fire* state at exactly the same time and for the first time. The set of states and the next-state function must be independent of n.

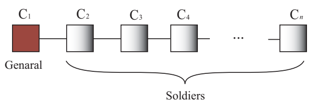

Fig. 1. One-dimensional cellular automaton

A formal definition of the FSSP is as follows: A cellular automaton $\mathcal{M}$ is a pair $\mathcal{M} = (\mathcal{Q}, \delta)$, where

1) $\mathcal{Q}$ is a finite set of states with three distinguished states $\mathtt{G}$, $\mathtt{Q}$, and $\mathtt{F}$. $\mathtt{G}$ is an initial general state, $\mathtt{Q}$ is a quiescent state, and $\mathtt{F}$ is a firing state, respectively.
2) δ is a next state function such that $\delta : \mathcal{Q} \cup \{*\} \times \mathcal{Q} \times \mathcal{Q} \cup \{*\} \rightarrow \mathcal{Q}$. The state $\mathtt{*} \notin \mathcal{Q}$ is a pseudo state of the border of the array.
3) The quiescent state $\mathtt{Q}$ must satisfy the following conditions: $\delta(\mathtt{Q}, \mathtt{Q}, \mathtt{Q}) = \delta(*, \mathtt{Q}, \mathtt{Q}) = \delta(\mathtt{Q}, \mathtt{Q}, *) = \mathtt{Q}$.

A cellular automaton of length n, $\mathcal{M}_n$ consisting of n copies of $\mathcal{M}$ is a 1D array of $\mathcal{M}$, numbered from 1 to n.

Each $\mathcal{M}$ is referred to as a cell and denoted by C_i, where $1 \le i \le n$. We denote a state of C_i at time (step) t by $\mathtt{S}_i^t$, where $t \ge 0, 1 \le i \le n$. A *configuration* of $\mathcal{M}_n$ at time t is a function $\mathcal{C}^t : [1, n] \to \mathcal{Q}$ and denoted as $\mathtt{S}_1^t \mathtt{S}_2^t$ .... $\mathtt{S}_n^t$. A *computation* of $\mathcal{M}_n$ is a sequence of configurations of $\mathcal{M}_n$, $\mathcal{C}^0$, $\mathcal{C}^1$, $\mathcal{C}^2$, ..., $\mathcal{C}^t$, ..., where $\mathcal{C}^0$ is a given initial configuration. The configuration at time $t+1$, $\mathcal{C}^{t+1}$, is computed by synchronous applications of the next transition function δ to each cell of $\mathcal{M}_n$ in $\mathcal{C}^t$ such that:

$\mathtt{S}_1^{t+1} = \delta(*, \mathtt{S}_1^t, \mathtt{S}_2^t)$, $\mathtt{S}_i^{t+1} = \delta(\mathtt{S}_{i-1}^t, \mathtt{S}_i^t, \mathtt{S}_{i+1}^t)$, and $\mathtt{S}_n^{t+1} = \delta(\mathtt{S}_{n-1}^t, \mathtt{S}_n^t, *)$.

A *synchronized configuration* of $\mathcal{M}_n$ at time t is a configuration $\mathcal{C}^t$, $\mathtt{S}_i^t = \mathtt{F}$, for any $1 \le i \le n$.

The FSSP is to obtain an $\mathcal{M}$ such that, for any $n \ge 2$,

1) A synchronized configuration at time $t = T(n)$, $\mathcal{C}^{T(n)} = \overbrace{\mathtt{F}, \cdots, \mathtt{F}}^{n}$ can be computed from an initial configuration $\mathcal{C}^0 = \mathtt{G}\overbrace{\mathtt{Q}, \cdots, \mathtt{Q}}^{n-1}$.
2) For any t, i such that $1 \le t \le T(n) - 1$, $1 \le i \le n, \mathtt{S}_i^t \ne \mathtt{F}$.

The generalized FSSP (GFSSP) is to obtain an $\mathcal{M}$ such that, for any $n \ge 2$ and for any k such that $1 \le k \le n$,

1) A synchronized configuration at time $t = T(n)$, $\mathcal{C}^{T(n)} = \overbrace{\mathtt{F}, \cdots, \mathtt{F}}^{n}$ can be computed from an initial configuration $\mathcal{C}^0 = \overbrace{\mathtt{Q}, \cdots, \mathtt{Q}}^{k-1}\mathtt{G}\overbrace{\mathtt{Q}, \cdots, \mathtt{Q}}^{n-k}$.
2) For any t, i, such that $1 \le t \le T(n) - 1$, $1 \le i \le n, \mathtt{S}_i^t \ne \mathtt{F}$.

No cells fire before time $t = T(n)$. We say that $\mathcal{M}_n$ is synchronized at time $t = T(n)$ and the function $T(n)$ is a time complexity for the synchronization.

B. A Brief History of the Developments of Optimum-Time FSSP Algorithms

The problem known as the *firing squad synchronization problem* was devised in 1957 by J. Myhill, and first appeared in print in a paper by E. F. Moore [7]. This problem has been widely circulated, and has attracted much attention. The firing squad synchronization problem first arose in connection with the need to simultaneously turn on all parts of a self-reproducing machine. The problem was first solved by J. McCarthy and M. Minsky who presented a $3n$-step algorithm. In 1962, the first optimum-time, i.e., $(2n - 2)$-step, synchronization algorithm was presented by Goto [3], with each cell having several thousands of states. Waksman [20] presented a 16-state optimum-time synchronization algorithm. Afterward, Balzer [1] and Gerken [2] developed an eight-state algorithm and a seven-state synchronization algorithm, respectively, thus decreasing the number of states required for the synchronization. In 1987, Mazoyer [5] developed a six-state synchronization algorithm which, at present, is the algorithm having the fewest states.

C. Complexity Measures for FSSP Algorithms

- **Time**
  Any solution to the original FSSP with a general at one end can be easily shown to require $(2n - 2)$ steps for synchronizing n cells, since signals on the array can propagate no faster than one cell per one step, and the time from the general's instruction until the final synchronization must be at least $2n - 2$.

  Theorem 1 The minimum time in which the firing squad synchronization could occur is $2n - 2$ steps, where the general is located on the left end.

  Theorem 2 There exists a cellular automaton that can synchronize any 1D array of length n in optimum $2n-2$ steps, where the general is located on the left end.

- **Number of States**
  The following three distinct states: the *quiescent* state, the *general* state, and the *firing* state, are required in order to define any cellular automaton that can solve the FSSP. Note that the boundary state for C_0 and C_{n+1} is not generally counted as an internal state. Balzer [1] and Sanders [9] showed that no four-state optimum-time solution exists. Umeo and Yanagihara [17], Yunès [21], and Umeo, Kamikawa, and Yunès [14] gave some 5- and 4-state *partial* solutions that can solve the synchronization problem for infinitely many sizes n, but not all, respectively. The solution is referred to as *partial* solution, which is compared with usual *full* solutions that can solve the problem for all cells. Concerning the optimum-time full solutions, Waksman [20] presented a 16-state optimum-time synchronization algorithm. Afterward, Balzer [1] and Gerken [2] developed an eight-state algorithm and a seven-state synchronization algorithm, respectively, thus decreasing the number of states required for the synchronization. Mazoyer [5] developed a six-state synchronization algorithm which, at present, is the algorithm having the fewest states for 1D arrays.

  Theorem 3 There exists a 6-state *full* solution to the FSSP.

  Theorem 4 There is no four-state *full* solution that can synchronize n cells.

  Yunès [21] and Umeo, Yunès, and Kamikawa [14] developed 4-state partial solutions based on Wolfram's rules 60 and 150. They can synchronize any array/ring of length $n = 2^k$ for any positive integer k. Details can be found in Yunès [21] and Umeo, Kamikawa, and Yunès [14].

  Theorem 5 There exist 4-state *partial* solutions to the FSSP.

- **Number of Transition Rules**
  Any k-state (excluding the boundary state) transition table for the synchronization has at most $(k-1)k^2$ entries in $(k-1)$ matrices of size $k \times k$. The number of transition rules reflects a complexity of synchronization algorithms.

- **State-Change Complexity**
  Vollmar [18,19] introduced a *state-change complexity* in order to measure the efficiency of cellular automata, motivated by energy consumption in certain SRAM-type memory systems. The state-change complexity is defined as the sum of *proper* state changes of the cellular space during the computations. A formal definition is as follows: Consider an FSSP algorithm operating on n cells. Let $T(n)$ be synchronization steps of the FSSP algorithm. We define a matrix C of size $T(n) \times n$ ($T(n)$ rows, n columns) over $\{0, 1\}$, where each element $c_{i,j}$ on ith row, jth column of the matrix is defined:

$$c_{i,j} = \begin{cases} 1 & \mathtt{S}_i^j \neq \mathtt{S}_i^{j-1} \\ 0 & otherwise. \end{cases} \tag{1}$$

The state-change complexity $SC(n)$ of the FSSP algorithm is the sum of 1's elements in C defined as:

$$SC(n) = \sum_{j=1}^{T(n)} \sum_{i=1}^{n} c_{i,j}. \tag{2}$$

Vollmar [19] showed that $\Omega(n \log n)$ state-changes are required for synchronizing n cells in $(2n - 2)$ steps.

Theorem 6 $\Omega(n \log n)$ state-change is necessary for synchronizing n cells in minimum-steps.

Gerken [2] presented a minimum-time, $\Theta(n \log n)$ minimum-state-change FSSP algorithm with a general at one end.

Theorem 7 $\Theta(n \log n)$ state-change is sufficient for synchronizing n cells in $2n - 2$ steps.

III. A Class of FSSP Algorithms Based on Recursive-Halving

Here we introduce a new class of FSSP algorithms based on recursive halving.

A. Recursive-Halving Marking

In this section, we develop a marking schema for 1D arrays referred to as *recursive-halving marking*. The marking schema prints a special mark on cells in a cellular space defined by the recursive-halving marking. It is based on a 1D FSSP synchronization algorithm. The marking will be effectively used for constructing multi-dimensional FSSP algorithms operating in optimum-time.

Let S be a 1D cellular space consisting of cells C$_i$, C$_{i+1}$, ..., C$_j$, denoted by [i...j], where $j > i$. Let $|S|$ denote the number of cells in S, that is $|S| = j - i + 1$. A center cell(s) C$_x$ of S is defined by

$$x = \begin{cases} (i+j)/2 & |S|\text{: odd} \\ (i+j-1)/2, (i+j+1)/2 & |S|\text{: even.} \end{cases} \tag{3}$$

The recursive-halving marking for a given cellular space S = [1...n] is defined as follows:

Recursive-Halving Marking: RHM

```
Algorithm RHM($S$)
begin
    if $|S| \geq 2$ then
        if $|S|$ is odd then
            mark center cell C$_x$ in $S$
            $S_L$:= [1...$x$]; $S_R$:= [$x$...$n$]
            RHM$_L$($S_L$); RHM$_R$($S_R$);
        else
            mark center cells C$_x$ and C$_{x+1}$ in $S$
            $S_L$:= [1...$x$]; $S_R$:= [$x+1$...$n$]
            RHM$_L$($S_L$); RHM$_R$($S_R$);
end
```

Left-Side Recursive-Halving Marking: RHM$_L$

```
Algorithm RHM$_L$($S$)
begin
    while $|S| > 2$ do
        if $|S|$ is odd then
            mark center cell C$_x$ in $S$
            $S_L$:= [1...$x$]; RHM$_L$($S_L$);
        else
            mark center cells C$_x$ and C$_{x+1}$ in $S$
            $S_L$:= [1...$x$]; RHM$_L$($S_L$);
end
```

The marking RHM$_R$ for the right-side space can be defined in a similar way. As an example, we consider a cellular space S = [1...15] consisting of 15 cells. The first center cell is C$_8$, then the second one is C$_4$, C$_5$ and C$_{11}$, C$_{12}$, and the last one is C$_2$, C$_3$, C$_{13}$, C$_{14}$, respectively. In case S = [1...17], we get C$_9$, C$_5$, C$_{13}$, C$_3$, C$_{15}$, and C$_2$, C$_{16}$ after four iterations.

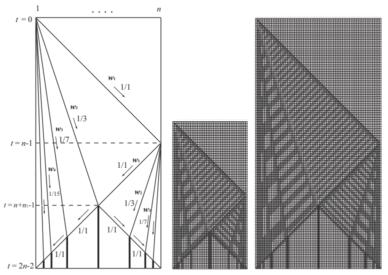

Fig. 2. Recursive-halving marking

Figure 2 (left) shows a space-time diagram for the marking. At time $t = 0$, the leftmost cell C$_1$ generates a set of signals $w_1, w_2, ..., w_k$, .., each propagating in the right direction at $1/(2^k - 1)$ speed, where $k = 1, 2, 3, ...,$. The $1/1$-speed signal w_1 arrives at C$_n$ at time $t = n - 1$. Then, the rightmost cell C$_n$ also emits an infinite set of signals $w_1, w_2, ..., w_k$, .., each propagating in the left direction at $1/(2^k - 1)$ speed, where $k = 1, 2, 3, ...,$. The readers can find that each crossing of two signals, shown in Figure 2 (left), enables the marking at middle points defined by the recursive-halving. A finite state realization for generating the infinite set of signals above is a well-known technique employed in Balzer [1], Gerken [2], and Waksman [20] for the implementations of the optimum-time synchronization algorithms on 1D arrays.

We have developed a simple implementation of the recursive-halving marking on a 13-state, 314-rule cellular automaton. In Figure 2 (middle and right) we present several snapshots for the marking on 42 and 71 cells, respectively. We have:

Lemma 8 There exists a 1D 13-state, 314-rule cellular automaton that can print the recursive-halving marking in any cellular space of length n in $2n-2$ steps.

An optimum-time complexity $2n-2$ needed for synchronizing cellular space of length n in the classical WBG-type (Waksman [20], Balzer [1], and Gerken [2]) FSSP algorithms can be interpreted as follows: Let S be a cellular space of length $n = 2n_1+1$, where $n_1 \geq 1$. The first center mark in S is printed on cell C_{n_1+1} at time $t_{\mathrm{1D-center}} = 3n_1$. Additional n_1 steps are required for the markings thereafter, yielding a final synchronization at time $t_{\mathrm{1D-opt}} = 3n_1 + n_1 = 4n_1 = 2n-2$. In the case $n = 2n_1$, where $n_1 \geq 1$, the first center mark is printed simultaneously on cells C_{n_1} and C_{n_1+1} at time $t_{\mathrm{1D-center}} = 3n_1 - 1$. Additional $n_1 - 1$ steps are required for the marking thereafter, yielding the final synchronization at time $t_{\mathrm{1D-opt}} = 3n_1 - 1 + n_1 - 1 = 4n_1 - 2 = 2n-2$.

$$t_{\mathrm{1D-center}} = \begin{cases} 3n_1 & |S| = 2n_1+1, \\ 3n_1 - 1 & |S| = 2n_1. \end{cases} \quad (4)$$

Thus, additional t_{sync} steps are required for the synchronization for a cellular space with the recursive-halving marks:

$$t_{\mathrm{sync}} = \begin{cases} n_1 & |S| = 2n_1+1, \\ n_1 - 1 & |S| = 2n_1. \end{cases} \quad (5)$$

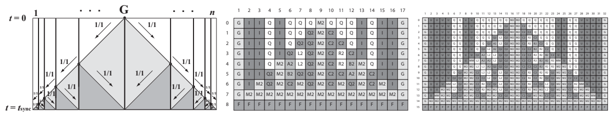

Fig. 3. Synchronization based on recursive-halving

In this way, it can be easily seen that any cellular space of length n with the recursive-halving marking initially with a general on a center cell or two generals on adjacent center cells can be synchronized in $\lceil n/2 \rceil - 1$ optimum-steps.

Lemma 9 Any 1D cellular space S of length n with the recursive-halving marking initially with a general(s) on a center cell(s) in S can be synchronized in $\lceil n/2 \rceil - 1$ optimum-steps.

In Figure 3, we illustrate a space-time diagram for synchronizing a cellular space with recursive-halving marking (left) and some snapshots for the synchronization on 17 (middle) and 32 (right) cells, respectively.

As was seen, the first marking of center cell(s) plays an important role. In order to use the marking scheme for the design of multi-dimensional FSSP algorithms, we print a special mark only on the first center cell(s) of a given cellular space, where the center cells thereafter will be marked with a different symbol from the first one.

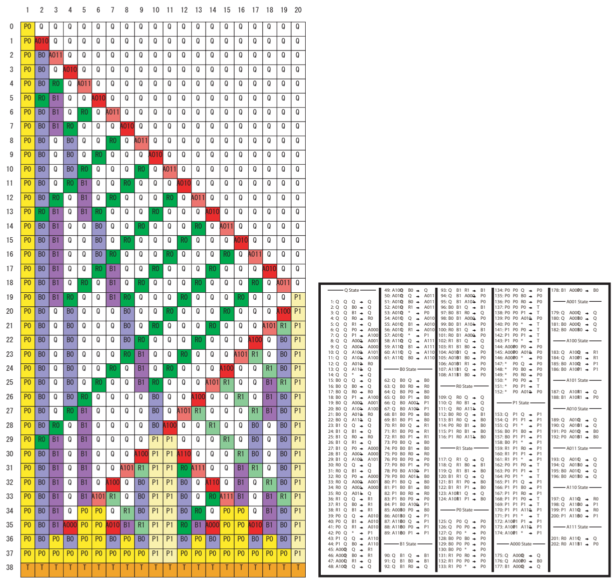

Fig. 4. Waksman's FSSP

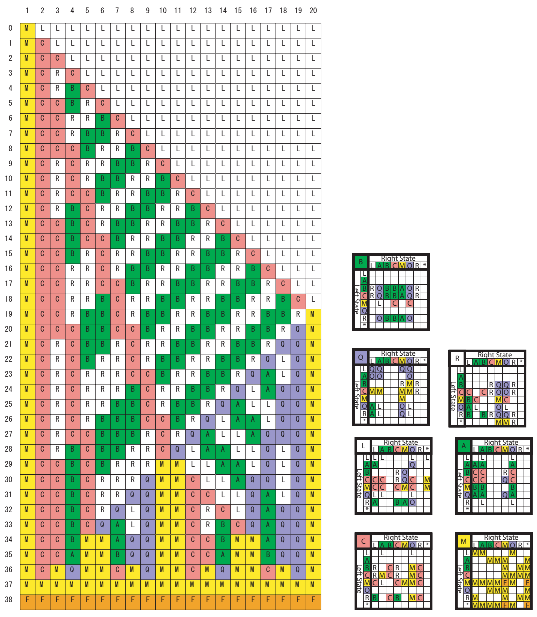

Fig. 5. Balzer's FSSP

B. *Waksman's 16-state Algorithm*

Waksman [20] proposed a 16-state firing squad synchronization algorithm. Umeo, Hisaoka, and Sogabe [13] corrected all errors in Waksman's original transition table. In Figure 4, we give a snapshot of the synchronization processes on 20 cell and a list of transition rules for Waksman's algorithm. The list is a revised version presented in Umeo, Hisaoka, and Sogabe [13]. The state-change complexity of the algorithm is $\mathrm{O}(n^2)$.

C. *Balzer's Eight-state Algorithm*

Balzer [1] constructed an eight-state, 182-rule synchronization algorithm and the structure of which is completely identical to that of Waksman [20]. In Figure 5, we give

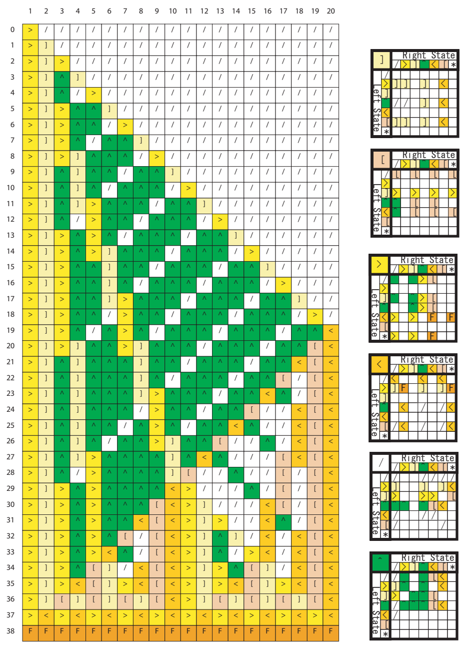

Fig. 6. Gerken's FSSP

a snapshot of the algorithm and a list of transition rules for Balzer's algorithm. The state-change complexity of the algorithm is O(n^2).

D. *Gerken's Seven-state Algorithm*

Gerken [2] reduced the number of states realizing Balzer's algorithm and constructed a seven-state, 118-rule synchronization algorithm. In Figure 6, we give a list of the transition rules for Gerken's algorithm and its snapshots. The state-change complexity of the algorithm is O(n^2).

E. *An Optimum-Time 2D FSSP Algorithm $\mathcal{A}$*

We assume that an initial general G is on the north-west corner cell C$_{1,1}$ of a given array of size $m \times n$. The algorithm consists of three phases: a marking start phase for 2D arrays, pre-synchronization phase and a final synchronization phase. An overview of the 2D synchronization algorithm $\mathcal{A}$ is as follows:

Step 1. Start the recursive-halving marking for cells on each row and column, **find** a *center cell(s)* of the 2D array, and **generate** a new general(s) on the center cell(s). Note that a crossing(s) of the center column(s) with the center row(s) is a center cell(s) of the array.

Step 2. Pre-synchronize the center column(s) using Lemma 6, which is initiated by the general in step 1. Every cell(s) on the center column(s) acts as a general at the next Step 3.

Step 3. Synchronize each row using Lemma 6, initiated by the general generated in Step 2. This yields the final synchronization of the array.

Thus, we have:

Theorem 10 The synchronization algorithm $\mathcal{A}$ can synchronize any $m \times n$ rectangular array in optimum $m + n + \max(m, n) - 3$ steps.

Theorem 11 There exists an optimum-time synchronization algorithm that can synchronize any three-dimensional array of size $m \times n \times \ell$ with a general at C$_{1,1,1}$ in optimum $m + n + \ell + \max(m, n, \ell) - 4$ steps.

Theorem 12 There exists an optimum-time synchronization algorithm that can synchronize any kD array of size $n_1 \times n_2 \times ... \times n_k$ with a general at C$_{1,1,...,1}$ in optimum $n_1 + n_2 +, ..., +n_k + \max(n_1, n_2, ..., n_k) - k - 1$ steps.

F. *A Comparison of Quantitative Aspects of Optimum-Time Synchronization Algorithms*

Here, we present a table based on a quantitative comparison of optimum-time synchronization algorithms and their transition tables discussed above with respect to the number of internal states of each finite state automaton, the number of transition rules realizing the synchronization, and the number of state-changes on the array.

TABLE I. COMPARISON OF FSSP ALGORITHMS

Algorithm	**# of states**	**# of transition rules**	**State change complexity**
Goto [3]	many thousands	—	$\Theta(n \log n)$
Waksman [20]	16	202	O(n^2)
Balzer [1]	8	165	O(n^2)
Gerken I [2]	7	105	O(n^2)
Mazoyer [5]	6	119	O(n^2)
Gerken II [2]	155	2371	$\Theta(n \log n)$

G. *O(1)-bit vs. 1-bit Communication FSSP*

In the study of cellular automata, the amount of bit-information exchanged at one step between neighboring cells has been assumed to be O(1)-bit data. An O(1)-bit CA is a conventional CA in which the number of communication bits exchanged at one step between neighboring cells is assumed to be O(1)-bit, however, such inter-cell bit-information exchange has been hidden behind the definition of conventional automata-theoretic finite state description. On the other hand, the 1-bit inter-cell communication model is a new CA in which inter-cell communication is restricted to 1-bit data, referred to as the 1-bit CA model. The number of internal states of the 1-bit CA is assumed to be finite in the usual sense. The next state of each cell is determined by the present state of that cell and two binary 1-bit inputs from its left and right neighbor cells. Thus, the 1-bit CA can be thought of as one of the most powerless and the simplest models in a variety of CA's. A precise definition of the 1-bit CA can be found in Umeo and Yanagihara [17]. Umeo and Yanagihara [17] constructed an optimum-time synchronization algorithm on a 1-bit CA model, based on Waksman's algorithm. In Figure 7, we show a configuration of the 1-bit synchronization algorithm on 15 cells. Each cell has 78 internal states and 208 transition rules. The small black triangles ▶ and ◀ indicate a 1-bit signal transfer in the right or left direction, respectively, between neighboring cells. A symbol in a cell shows internal state of the cell.

[Theorem 13] There exists a 1-bit CA that can synchronize n cells in optimum $2n - 2$ steps.

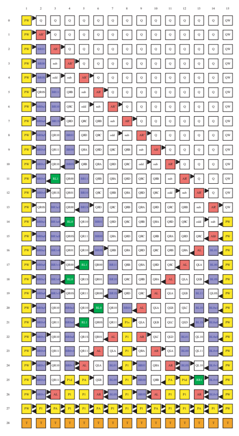
Fig. 7. FSSP on 1-bit CA

IV. CONCLUSION AND FUTURE WORK

In the present paper, we have given a survey on recent developments in FSSP algorithms for one-dimensional cellular arrays. We focus our attention on a new class of FSSP algorithms based on recursive-halving. It is shown that the recursive-halving marking has been used in the design of many optimum-time FSSP algorithms and can be generalized and expanded to multi-dimensional arrays. Several multi-dimensional generalizations of the algorithms are also given. As a future work an FSSP for growing multi-dimensional arrays would be interesting.

REFERENCES

[1] R. Balzer: An 8-state minimal time solution to the firing squad synchronization problem. *Information and Control*, vol. 10, pp. 22-42, 1967.

[2] Hans-D. Gerken: Über Synchronisations - Probleme bei Zellularautomaten. *Diplomarbeit*, Institut für Theoretische Informatik, Technische Universität Braunschweig, pp. 50, 1987.

[3] E. Goto: A minimal time solution of the firing squad problem. *Dittoed course notes for Applied Mathematics* 298, Harvard University, pp. 52-59, with an illustration in color, 1962.

[4] J. Mazoyer: An overview of the firing squad synchronization problem. *Lecture Notes on Computer Science*, Springer-Verlag, vol. 316, pp. 82-93, 1986.

[5] J. Mazoyer: A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, vol. 50, pp. 183-238, 1987.

[6] M. Minsky: *Computation: Finite and infinite machines*. Prentice Hall, pp. 28-29, 1967.

[7] E. F. Moore: The firing squad synchronization problem. in *Sequential Machines, Selected Papers* (E. F. Moore, ed.), Addison-Wesley, Reading MA., pp. 213-214, 1964.

[8] F. R. Moore and G. G. Langdon: A generalized firing squad problem. *Information and Control*, 12, pp. 212-220, 1968.

[9] P. Sanders: Massively parallel search for transition-tables of polyautomata. In *Proc. of the VI International Workshop on Parallel Processing by Cellular Automata and Arrays*, (C. Jesshope, V. Jossifov and W. Wilhelmi (editors)), Akademie, 99-108, 1994.

[10] A. Settle and J. Simon: Smaller solutions for the firing squad. *Theoretical Computer Science*, 276, 83-109, 2002.

[11] H. Szwerinski: Time-optimum solution of the firing squad synchronization problem for n-dimensional rectangles with the general at an arbitrary position. *Theoretical Computer Science*, vol. 19, pp. 305-320, 1982.

[12] H. Umeo: Firing squad synchronization problem in cellular automata. In *Encyclopedia of Complexity and System Science*, R. A. Meyers (Ed.), Springer, Vol.4, pp.3537-3574, 2009.

[13] H. Umeo, M. Hisaoka, and T. Sogabe: A survey on optimum-time firing squad synchronization algorithms for one-dimensional cellular automata. *Int. J. of Unconventional Computing*, vol. 1, pp.403-426, 2005.

[14] H. Umeo, N. Kamikawa, and J.-B. Yunès: A family of smallest symmetrical four-state firing squad synchronization protocols for ring arrays. *Parallel Processing Letters*, Vol.19, No.2, pp.299-313, 2009.

[15] H. Umeo, N. Kamikawa, K. Nishioka, and S. Akiguchi: Generalized Firing Squad Synchronization Protocols for One-Dimensional Cellular Automata - A Survey. *Acta Physica Polonica B, Proceedings Supplement*. Vol.3, pp.267-289, 2010.

[16] H. Umeo, K. Kubo, and K. Nishide: A class of time-optimum FSSP algorithms for multi-dimensional cellular arrays. *Communications in Nonlinear Science and Numerical Simulation*, 21, pp.200-209, 2015.

[17] H. Umeo and T. Yanagihara: Smallest implementations of optimum-time firing squad synchronization algorithms for one-bit-communication cellular automata. *Proc. of the 2011 International Conference on Parallel Computing and Technology, PaCT 2011*, LNCS 6873, pp. 210-223, 2011.

[18] R. Vollmar: On Cellular Automata with a Finite Number of State Change. *Computing, Supplementum*, vol. 3, pp. 181-191, 1981.

[19] R. Vollmar: Some remarks about the “Efficiency” of polyautomata. *International Journal of Theoretical Physics*, vol. 21, no. 12, pp. 1007-1015, 1982.

[20] A. Waksman: An optimum solution to the firing squad synchronization problem. *Information and Control*, vol. 9, pp. 66-78, 1966.

[21] J. B. Yunès: A 4-states algebraic solution to linear cellular automata synchronization. *Information Processing Letters*, Vol. 19, Issue 2, pp.71-75, 2008.