PRALLEL SORTING ON LINEAR CELLULAR AUTOMATA

Moein Shakeri, Homa Foroughi, Hossein Deldari
Department of Computer Engineering, Ferdowsi University of Mashhad, Iran
mo_sh88@stu-mail.um.ac.ir, ho_fo46@stu-mail.um.ac.ir, hd@ferdowsi.um.ac.ir

ABSTRACT
Sorting is one of the fundamental problems in computer science and being used vastly in various domains. So different serial and parallel approaches have been proposed. One of the parallel sorting methods are algorithms that are based on computational model of cellular automata. A cellular automata machine is a structure of interconnected elementary automata evolving in a parallel and synchronous way. The most famous sorting algorithm for one dimensional cellular automata machine is Gordillo and Luna’s. This algorithm sorts n numbers in 2n-3 steps. In this paper three new sorting algorithms are proposed. In the two first proposed algorithms, despite using smaller neighborhood radius, sorting steps have not been changed and in the third algorithm sorting steps are reduced by regarding same neighborhood radius as Gordillo- Luna’s second algorithms.

Keywords: parallel sorting, linear cellular automata, cellular computation.

1 INTRODUCTION

It’s too years that scientists are investigating natural phenomena of world and trying to classify them based on human knowledge and mathematical rules. Since most of natural events are occurred concurrently in parallel manner (e.g. water stream of rivers or molecules of steam) it is actually essential to parallelize computations that are performed for classification and composing rules for these phenomena. To this aim, different methods and applicable systems like supercomputers, computer networks and distributed systems have been proposed. One of the approaches that is capable of being implemented on mentioned systems is cellular automata [2].

In this paper, we present the optimal solution for a classical problem under the cellular automata programming perspective. The problem that we are tackling is the sort of n elements in a linear cellular array, where each cell has a communication channel with its neighbors. Sorting is one of the most important problems and has a significant role in solving various problems in different domains such as image processing, graph theory, computational geometry and scheduling. To validate and illustrate the computations on the CA machine, in this paper we present the analysis and implementation of parallel sort algorithms, and then we compare results with previous models based on CA.

Generally, sorting algorithms can be categorized as:

Comparison-based algorithms
Noncomparison-based algorithms.

A comparison-based algorithm sorts an unordered sequence of elements by repeatedly comparing pairs of elements and, if they are out of order, exchanging them. This fundamental operation of comparison-based sorting is called compare-exchange. The lower bound on the sequential complexity of any comparison-based sorting algorithm is \( \Theta(n \log n) \), where \( n \) is the number of elements to be sorted. However Noncomparison-based algorithms sort by using certain known properties of the elements (such as their binary representation or their distribution). The lower bound complexity of these algorithms is \( \Theta(n) \). This assortment is legal for both serial and parallel algorithms.

Parallel algorithms can be classified based on type of problem solving, their usages and also their dependency to system architecture. So parallel algorithms are divided into two major classes:

1. Machine architecture based algorithms
   (Implementation platform dependent)
2. Mathematical model based algorithms

Table 1 demonstrates classification of sorting algorithms [5,6,7,8,9,10,12]. In this classification, Sorting Networks, Bubble Sort and Quick Sort are regarded as implementation platform dependant and comparison-based algorithms. Sample Sort have been classified as non-comparison based and implementation platform dependant algorithms.
Table 1: Classification of sorting algorithms

<table>
<thead>
<tr>
<th>Mathematical model based algorithms</th>
<th>Implementation platform dependant algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparison based sorting algorithms</td>
<td>Non-comparison based sorting algorithms</td>
</tr>
<tr>
<td>Sorting algorithms based on Systolic arrays</td>
<td>Sorting Networks</td>
</tr>
<tr>
<td>Sorting algorithms based on Cellular Automata</td>
<td></td>
</tr>
<tr>
<td>Sample Sort</td>
<td>Bubble Sort</td>
</tr>
<tr>
<td></td>
<td>(Odd-Even Transposition)</td>
</tr>
<tr>
<td></td>
<td>(Shell sort)</td>
</tr>
<tr>
<td></td>
<td>Quicksort</td>
</tr>
</tbody>
</table>

Mathematical model-based algorithms, such as algorithms based on Systolic arrays [11] and cellular automata, are free from implementation platform. In the algorithms that are based on 1-D (Linear) cellular automata, sorting is done according to cellular neighborhood and available local rules among them. Most famous proposed algorithm is Gordillo-Luna’s algorithm.

At current parallel sorting approaches on cellular automata, sorting is done by a two state cycle (q0, q1) automata. Two types of transitions exist in this automata (from q0 to q1 and vice versa). One of the algorithms that use this approach is Gordillo-Luna’s algorithm [3]. This algorithm is one of the rare ones, which uses 1-D (Linear) cellular automata for sorting; also it’s the most famous one.

Gordillo-Luna, proposed two algorithms that in both of them each cell needs three read-writeable memories, one for numeral value, and two others for controllers due to replacement of numeral value to left and right. Both of these algorithms have two steps; at first step left and right controllers are adjusted and at second step, replacement decision is made according to values of controllers. Number of neighbor cells that are been reviewed in Gordillo-Luna’s first and second algorithms, are respectively three and five.

In this paper, three parallel sorting algorithms; based on linear cellular automata, are proposed that have the following distinctive features:

Two first algorithms, use smaller neighborhood in comparison with Gordillo-Luna’s algorithms: In the first proposed algorithm, two neighbors (one at left and another at right), is used for each cell and in the second proposed algorithm, three neighbors (two at right and one at left), are considered. It’s necessary to mention that reducing neighborhood radius doesn’t change number of sorting steps (the same as Gordillo-Luna’s).

In the third proposed algorithm, we have used same neighborhood as Gordillo-Luna’s second algorithm [3], but sorting time have been reduced in comparison with Gordillo-Luna’s second algorithm.

In the first proposed algorithm, no extra memory is used for cellular automata-in contrast with Gordillo-Luna’s first algorithm- Also in the second and third proposed algorithms, only one extra cell is used; that’s less than memory used in Gordillo-Luna’s.

In the first proposed algorithm, reviewing values and replacement of them is done just in one step; unlike Gordillo-Luna’s, and this is effective in computations decrement and also speed increment.

The reminder of paper is organized as follows: Sections 2 briefly introduces cellular automata and its properties, in section 3 Gordillo-Luna’s algorithms are explained, section 4 describes our proposed algorithms and finally we draw some conclusions in section 5.

2 CELLULAR AUTOMATA

It’s too years that computational model of cellular automata has been proposed to study different fields of phenomenological argues of nature consisting communication, computation, preproduction, rivalry, growing and so on. Also CA is a suitable tool for modeling physical phenomena, via converting them to basic phenomena, by using fundamental and basic rules [2, 4].

A cellular automaton is a mathematical model, which is discrete-space & time; time is considered as specific constant intervals and space is demonstrated as one more dimensional cell networks. Dimensions of networks are related to dimensions of cellular automata. Each cell has time-variant properties, variable values of each cell in each interval, describes status of that cell and overall system state is recognized by considering status of all cell in a specific interval [1].

2.1 The Cellular Automata Features

The solution of a computational problem demands a data structure to define the input-output, and a procedure to transform the first one into the other. In spite of that, the traditional CA Machines do not contain memory space in its definition. The status and results of process are coded in the states by which the automata transit. General structure of this cellular automata is as follows:

Definition 1: A cellular automata is a 4-tuple CA = (Q,d,V,a), where:

Q is the nonempty state set that a cell can assume. A distinguished q0 ∈ Q state, determines that the state q0 of a cell remains unchanged if the directly connected cell are also in state q0. The q0 state is called the quiescent state.

The dimension d of the cellular space, building a bounded subspace on Zd (Z denoting the
integers). Each cell $x$ in the array is an element of the subspace ($x \in \mathbb{Z}^d$). Note that, when $d = 1$; it corresponds to the case of a linear array.

The neighborhood array ($V_0, V_1, \ldots, V_k, V_j$ $\in \mathbb{Z}$; $V_j \in [0, k]$). Being $x \in \mathbb{Z}^d$ an automata location, $V(x) = \{x + V_0, x + V_1, \ldots, x + V_k\}$ describes the finite list of $k + 1$ neighbors directly associated to it. In the case of a linear array, $x = i \in \mathbb{Z}$; the neighborhood array (0, -1, +1) index the central, the left and the right cells.

The transition function (or relation) $\delta$ defines the dynamical local state transformations to be applied to the cells. In standard automata without memory register the deterministic transitions are of the form $\delta: Q^{k+1} \rightarrow Q$.

### 3 GORDILLO-LUNA’S SORTING ALGORITHMS

Sorting operations in these algorithms have been performed by using cellular and mealy automata, which each cell has three memories (one for numeral value and two other for left and right controllers). These algorithms, calculate next state of each cell in two steps.

In the former step, the transition from $q_0$ to $q_1$ is applied and the value of the local register key is compared to the value of the key register of both neighbor cells. As a result, the values of the local blocking rule registers are computed. In the last step, where the transition from $q_1$ to $q_0$ is applied, the local blocking rule values are compared to the corresponding ones of the neighbors to determine, if necessary, with which of them the swap will be done.

To simplify the description, Gordillo-Luna considered $X^f, S^f_i, S^r_i$ as the local memory registers of cell $i$ coding the key, and the left and right blocking rule respectively.

Similarly, the subindex $i-1$, and $i+1$ address the precedent (left) and the subsequent (right) cells, which are directly connected to the $i$ cell. Computation of the blocking rules in the first step (the transition from $q_0$ to $q_1$) is defined as follows:

$$S^f_i = \begin{cases} 1 & \text{if } X_i > X_{i+1} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (1)

$$S^r_i = \begin{cases} 1 & \text{if } X_i < X_{i-1} \text{ And } X_i \leq X_{i+1} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (2)

Note that the left blocking rule is more constraining, given as result that the algorithm

When the blocking rules have been computed, the swapping rules are locally decided in the second automata step (the transition from $q_1$ to $q_0$), by the following evaluation:

$$X_i = \begin{cases} X_{i+1} & \text{if } S^f_i = 1 \text{ And } S^r_i = 1 \\ X_{i-1} & \text{if } S^f_i = 1 \text{ And } S^r_i = 1 \\ X_i & \text{otherwise} \end{cases}$$  \hspace{1cm} (3)

For terminating the operations, a general mechanism is used in each cycle.

Based on this algorithm, worst case of sorting $n$ elements is $O(2n-3)$.

In the first step of Gordillo-Luna’s first algorithm, computations of blocking rules for $i$ th cell, is performed by using three cells (Right neighbor $i+1$ th cell, Cell itself $i$ th cell and Left neighbor $i-1$ th cell).

Combining equation 1,2 and 3 results in:

$$X_i = \begin{cases} X_{i+1} & \text{if } (X_i > X_{i+1} \text{ And } X_i \leq X_{i+2}) \\ X_{i-1} & \text{if } (X_i < X_{i-1} \text{ And } X_i \leq X_{i+1}) \\ X_i & \text{otherwise} \end{cases}$$  \hspace{1cm} (4)

Equation 4, demonstrates that four cells ($i-1$ th, $i$ th, $i+1$ th and $i+2$ th) are used for computing next state of $i$ th cell.

### 3.1 Gordillo-Luna’s Second Algorithm

In the Gordillo-luna’s first algorithm we constraint the left blocking rule in order that most of the swaps take place with the right neighbor of the automata cell. By this fact, the algorithm starts by moving the smallest key from left to right. In the other hand, an equivalent algorithm can use a similar constraint applied to the right blocking rule, in order that most of the swaps will take place in the left side of each cell, permitting the biggest key to start moving to the right.

In the first step of Gordillo-Luna’s second algorithm, calculations for blocking rules of $i$ th cell, is done according to five cells ($i-2$ th, $i-1$ th, $i$ th, $i+1$ th and $i+2$ th). In the second step of algorithm, results of blocking rules of adjacent cells are compared with each other and then decision about replacement is made. If two steps are combined, decision for replacement of cell is done regarding to six cells (3 neighbors at right, cell itself and 2 neighbors at right). Sorting time of this algorithms is $O(\frac{3n}{2} - 2)$.

### 4 PROPOSED SORTING ALGORITHMS

First we review some tree representations that have been used in our proposed algorithms, and then three different parallel sorting algorithms are explained in details.

**Definition 2:** Any array can be displayed as a tree, like figure 1.
In this figure, “i, i+1, i+2, …, i+10” are array elements and edges represent value of each element toward adjacent elements. e.g. in figure1 value of i+3th element is more than value of i+2th & i+4th elements. Or in a similar manner, i+7th element has less value rather than i+6th one, but more than i+8th one.

Based on this definition, an array is ascending sorted only if it has a tree representation like figure2.

For descending sorted array, we have the same definition; just direction of edges will be inversed.

Obviously worst case for ascending sorting of an array, is occurred when it has been sorted descending.

4.1 First Proposed Algorithm

This algorithm uses a symmetric neighborhood of radius one and it’s sorting time is equal to Gordillo-Luna’s first algorithm. In our proposed algorithm, each cell just uses its right and left neighbor cells to sort the array. Indeed number of using cells has been decreased in comparison with with Gordillo-Luna’s first algorithm. This decrement results in less intra-cell communication and overall computation.

Another difference is that, unlike Godillo’s algorithm, our algorithm doesn’t use blocking rules, neither memory in cellular automata.

In this method, right and left neighbors of each cell are considered as follows:

\[
\begin{array}{c|c|c}
  i-1 & i & i+1 \\
\end{array}
\]

Also replacement conditions of each cell are regarded as figure4:

\[
\text{if } (X_{i-1} < X_i \text{ And } X_i > X_{i+1}) \text{ Then } X_{i-1} = X_i \text{ And } X_{i+1} = X_i + 1
\]

the above equation, \(X_{i-1}^{t+1}, X_i^{t+1}\) are values of \(i^{th}\) and \(i+1^{th}\) cells at time \(t\) and \(X_{i-1}^{t}, X_i^{t}, X_{i+1}^{t}\) are respectively values of \(i^{th}\) (Left neighbor), \(i^{th}\) and \(i+1^{th}\) (Right neighbor) cells at time \(t\).

So, in the worst case, an array can be sorted in \(O(2n-3)\), that’s equal to time of Gordillo’s first algorithm [3]. In the other cases, termination of algorithm can be announced by using an outer supervisor. Indeed if no replacement is done at one step, it means that the array is sorted. An example of ascending sorting by this approach is illustrated in table2.

<table>
<thead>
<tr>
<th>Number of states</th>
<th>Worst case for ascending sorting, using 7 numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6 7 5 4 3 2 1</td>
</tr>
<tr>
<td>6</td>
<td>5 7 4 3 2 1</td>
</tr>
<tr>
<td>5</td>
<td>6 4 3 2 1</td>
</tr>
<tr>
<td>4</td>
<td>5 3 2 1</td>
</tr>
<tr>
<td>3</td>
<td>4 2 1</td>
</tr>
<tr>
<td>2</td>
<td>3 1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

4.2 Second Proposed Algorithm

Sorting time of the second algorithm is the same as Gordillo’s second algorithm, but the dominant point of our algorithm is that, instead of using 6 cells (Gordillo-Luna’s second algorithm), we just use 4 cells. This decrement in neighborhood radius of each
cell, leads in overall computation decrement and also reduction of intra-cells communications (that has an impressive effect in declining of sorting time).

This algorithm considers three neighbor cells for each cell (two cells at right and one at left) and done in two steps.

\[
\begin{array}{c|c|c|c}
i-1 & i & i+1 & i+2 \\
\end{array}
\]

**Figure 5:** Neighbor cells for \( i \)th cell.

At first step, replacement conditions of \( i \)th cell, is regarded as figure6:

\[
x_{i-1} \quad x_i \quad OR \\
\]

**Figure 6:** Replacement conditions of \( i \)th cell

Fig.6 means that, furthermore using rule mentioned in equation 1, another rule is taken into account for reduction of sorting time. This new rule uses four cells for deciding: if \( x_i \) is greater than \( x_{i-1} \) and also \( x_{i+1} \) is smaller than \( x_{i+2} \), now it’s a suitable case for replacing. These conditions causes to memory cell setting to one:

\[
\text{if } \{X_{i-1}^t < X_i^t \text{ And } X_i^t > X_{i+1}^t\} \text{ or } \{X_i^t > X_{i+1}^t \text{ And } X_{i+1}^t < X_{i+2}^t\} \text{ Then } S_i = 1
\]

\[
\text{else } S_i = 0
\]

\( S_i \) demonstrates replacement between \( x_i \) and \( x_{i+1} \) cells. Finally in the second step replacement is done only if \( S_i=1 \) and \( S_{i-1}=0 \):

\[
\text{if } S_i = 1 \text{ And } S_{i-1} = 0 \text{ Then } \{X_{i+1}^{t+1} = X_{i+1}^t \text{ And } X_{i+2}^{t+1} = X_i^t\} \quad (7)
\]

\[ \text{else } \{\text{nothing}\} \]

### 4.3 Third Proposed Algorithm

This algorithm uses exactly the same number of cells used in Gordillo-Luna’s second algorithm, but with a substantial difference: sorting time of our proposed algorithm has a significant improvement in comparison with sorting time of Gordillo-Luna’s second algorithm \( O\left(\frac{3n}{2} - 2\right) \).

This algorithm is performed in two steps and computing the next state of each cell needs six neighbor cells (3 neighbors at left, cell itself, and 2 neighbors at right):

<table>
<thead>
<tr>
<th>Number of States</th>
<th>An example of worst case ascending sorting in an array with 15 numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14 15 12 13 11 10 9 8 7 6 5 4 3 1 2</td>
</tr>
<tr>
<td>2</td>
<td>14 12 15 11 13 9 10 8 7 6 5 4 1 3 2</td>
</tr>
<tr>
<td>3</td>
<td>12 14 11 15 9 13 8 10 6 7 5 1 4 2 3</td>
</tr>
<tr>
<td>4</td>
<td>12 11 14 9 15 8 13 6 10 5 7 1 2 4 3</td>
</tr>
<tr>
<td>5</td>
<td>11 12 9 14 8 15 6 13 5 10 1 7 2 3 4</td>
</tr>
<tr>
<td>6</td>
<td>11 9 12 8 14 6 15 5 13 1 10 2 7 3 4</td>
</tr>
<tr>
<td>7</td>
<td>9 11 8 12 6 14 5 15 1 13 2 10 3 7 4</td>
</tr>
<tr>
<td>8</td>
<td>9 8 11 6 12 5 14 1 15 2 13 3 10 4 7</td>
</tr>
<tr>
<td>9</td>
<td>8 9 6 11 5 12 1 14 2 15 3 13 4 10 7</td>
</tr>
<tr>
<td>10</td>
<td>8 6 9 5 11 1 12 2 14 3 15 4 13 7 10</td>
</tr>
<tr>
<td>11</td>
<td>6 8 5 9 1 11 2 12 3 14 4 15 7 13 10</td>
</tr>
<tr>
<td>12</td>
<td>6 5 8 1 9 2 11 3 12 4 14 7 15 10 13</td>
</tr>
<tr>
<td>13</td>
<td>5 6 1 8 2 9 3 11 4 12 7 14 10 15 13</td>
</tr>
<tr>
<td>14</td>
<td>5 1 6 2 8 3 9 4 11 7 12 10 14 13 15</td>
</tr>
<tr>
<td>15</td>
<td>1 5 2 6 3 8 4 9 7 11 10 12 13 14 15</td>
</tr>
<tr>
<td>16</td>
<td>1 2 5 3 6 4 8 7 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>17</td>
<td>1 2 3 5 4 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td>18</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
</tbody>
</table>
For parallel implementation of these algorithms, we define some extremes at the first (last) of array (lower bound for the first of array and upper bound for the last of array), where fixed values will be just copied. These algorithms programmed for the intermediate cells.

5 CONCLUSION

In this paper we proposed three novel sorting algorithms based on 1-D (Linear) cellular automata. The first and second proposed algorithms despite using smaller neighborhood radius than Gordillo-Luna’s algorithms have the same sorting time. The third proposed algorithm uses same neighborhood radius as Gordillo-Luna’s second algorithm, but less sorting time. This significant improvement shows efficiency and robustness of our proposed approach. It’s important to highlight significant of cellular automata as a computational mechanism to efficiently solve problems that use important amount of data uniformly distributed in the space. the proposed algorithms are mathematical model-based algorithms and free from implementation platform.

REFERENCES