RZ 3810 (# Z1111-01) 11/07/2011
Computer Science 16 pages

Research Report

Determination of One-Way Bandwidth of Cellular Automata using Binary Decision Diagrams

Andreas C. Doering

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research
Almaden • Austin • Brazil • Cambridge • China • Haifa • India • Tokyo • Watson • Zurich

Determination of One-Way Bandwidth of Cellular Automata using Binary Decision Diagrams

Andreas C. Doering
IBM Research – Zurich,
Rueschlikon, Switzerland,
ado@zurich.ibm.com

Abstract

Cellular Automata are an inherently parallel computing architecture and can be scaled close to the physical limits due to the local-only data exchange. For economical and technical reasons application of cellular automata as computer architecture requires the use of partitions that are assembled into larger units, similar to memory in current systems (memory chips, Dual-In-line Memory Modules, etc.). This requires the exchange of the cell state at the chip boundaries. In this paper we consider the analysis of the required bandwidth over a chip boundary when a one-way protocol is used. The analysis algorithm counts the number of equivalence classes with respect to the cell states on the send side. When the states along the chip boundary are combined, a closed form solution for the number of equivalence classes is derived. The algorithm can be implemented using Binary Decision Diagrams, and we give results for several example cellular automata.

1 Introduction

In addition to their role as models of physical phenomena and models of computation, cellular automata have always been regarded as potential computer architecture. If future computers are to be built as cellular automata, it is very likely that an integration method will be used, that is similar to memory, large-scale FPGA systems, or the BlueGene^{TM}supercomputer [4]: individual chips are mounted on modules and connected with each other to form a large system. There are several reasons why Cellular Automata become more attractive as building block of computers with progress of device technology, including limited design capabilities of heterogeneous systems, and reduction of device parameters. For instance, CMOS FETs can be adapted by channel length and width, doping and gate isolation. In contrast, carbon nanotube transistors are

expected to have only one or two individual device parameters[12], while all other properties need to be fixed for a given chip.

The operation principle of cellular automata requires the evaluation of the states of neighbor cells to determine the next state of a given cell. The partitioning of a cellular automaton into multiple chips implies that there are cells (border cells) with one or more neighbors on a different chip. Dependent on the transition function of the cellular automaton, not all state combinations need be transferred over a chip-to-chip, because different configurations on the remote side can have the same contribution to the next state on the affected side. There are two dimensions to these redundancies, spatial and temporal. The spatial redundancies exploit the fact that the states of neighboring cells are not independent of each other, at least after several computation steps. For instance, many cellular automata have so called "Garden of Eden" configurations that cannot result from a previous computation step. An implementation that exploits the spatial redundancies does not transfer the cell state for each border cell separately, but combines the state of several cells to compress the combined state before transmission. Temporal redundancies are found when the possible sequence of cell states is restricted, e.g. if a state A cannot be followed by a state B in one cell for any given neighbor configurations. Exploiting temporal redundancies implies the combination of the states of cells that will impact the cells in the remote chip only after several time steps. Both dimensions, spatial and temporal redundancies will be considered in this paper.

Communication in Cellular Automata is investigated from different angles: Kutrib and Malcher assume an upper limit on the communication bandwidth on a single-step level and analyze the capabilities for problems such as arithmetics or parsing of formal languages. With this approach they obtain results for crucial problems resulting in automata with a considerable number of states. In [6] the class of two-state one-dimensional automata is investigated regarding a single reception cell.

Furthermore, there are different protocols that can be used. In the one-way protocol, the sending side does not use information about the state of the receiving side. Of course, data is typically transmitted in both directions and the send side will have some information from the receive side. To reduce the impact of transmission latency, one can overlap the cells at the chip boundary, which implies that the send side cells for one direction are different from the receive side cells in the opposite direction. As far as I know this paper is the first detailed analysis of one-way bandwidth of general cellular automata. For results and algorithms for the two-way protocol, where the state of the receive side is used to reduce the bandwidth, please refer to [5].

In the course of investigating the bandwidth of cellular automata, it transpired that Binary Decision Diagrams (BDDs) are very useful for the analysis of cellular automata properties. For instance, D. Knuth's "The Art of Computer Programming', Volume 4a illustrates the power of BDD algorithms using cellular automata. There exist several well tested BDD libraries [10, 8], and basing an algorithm on them allows a fast path to experiments. Hence, even if the BDD-based algorithms are not the best ones, they still allow a fast way

to understanding the problem better. A good introduction to BDDs and their algorithms is the book by Meinel and Theobald [9].

This paper is structured as follows: The definition of bandwidth is presented in Section 2. A short overview of Binary Decision Diagrams and some algorithms using them is provided in Section 3. In Section 4 the basic algorithm for determining the one-way bandwidth of cellular automata is described. In Section 5 it is charted, how a formal solution for scaling along the chip boundary can be derived. In Section 6 results for several cellular automata are given. In the final section, a summary and outlook on further work is found.

2 Definitions

A cellular automaton (CA) is a tuple (T, S, f), where $T = \{t_0, \ldots, t_{n-1}\}$ is a finite generator set of a group $G = \langle T \rangle$, S is a set of states, and $f : S^{|T|} \to S$ is the local transition function. The state (or configuration) of a CA is a mapping $s : G \to S$. Similarly the configuration of a set of cells assigns each cell a state from S. A computation step of a CA converts a CA state s into s' by $s'(g) = f(s(gt_0), \ldots, s(gt_{n-1})$. The set $\{gt_0, \ldots, gt_{n-1}\}$ are the neighbors of cell g. Typically, the identity element 1_G of the group G is an element of T implying that the state of a cell depends on its previous state. Furthermore, in most cases, for each $t_i \in T$ the inverse is part of the generator set, which means, that the neighborhood of cells is bidirectional. The algorithms presented in this paper apply to the general case including unidirectional neighborhoods.

Examples: The two-state one-dimensional CA $TS_k = (\{-1, 0, 1\}, \{0, 1\}, (l, c, r) \mapsto (k/2^{4*r+2*c+l})\%2)$ are identified by an index k. The state of a cell and its neighbors are combined into an index, the corresponding digit in the binary representation of k gives the next state. TS_{110} has been proven to be computational universal.

Conway's Game of Life uses a two-dimensional neighborhood (Moore with Radius 1): the next state of a cell depends on the number of neighbor cells in state 1. $T_{M2} = \{(0,0), (-1,0), (1,0), (0,1), (-1,1), (1,1), (0,-1), (-1,-1), (1,-1)\}$,

$$\begin{aligned} f(x_{(0,0)}, \ldots, x_{(1,-1)}) &= 1 \text{ if } x_{(0,0)} = 1 \text{ and } c \in \{2, 3\} \\ &= 1 \text{ if } x_{(0,0)} = 0 \text{ and } c = 3 \\ &= 0 \text{ otherwise} \end{aligned}$$

$$\text{where } c = \sum_{t=(-1,0)}^{(1,-1)} s(gt).$$

When a CA is split into several chips, the topology G is partitioned into several non-empty sets. In this paper we consider only two sets, because one can expect each set to contain millions of cells, and the behavior at the corners to be negligible to the data transmission over the long borders. Assume $G = G_s \dot{\cup} G_r$ is split into a send set G_s and a receive set G_r of cells, for instance $G_s = \{x | x < 0\}$ for $G = \mathbb{Z}$. Then there will be cells in the receive set G_r that

have a neighbor in the send set. Figure 1 illustrates the definition of bandwidth in a split CA. Formally, we define a set of receive result cells $B \subset G_r$ and a number of computation steps d. We can define, in the graph-theoretic sense, a ball of cells around B of radius d as $B_d(B) = \{bt_1 \cdots t_{h-1} | b \in B, t_i \in T, h \leq d\}$, i.e. the set of cells that can reach any cell in B in, at most, d steps. Deriving from this we get the two sets $A = B_d(B) \cap G_r$ and $C = B_d(B) \cap G_s$ of those cells on the send and receive side, whose configuration determine the state of the receive result cells B (see Figure 2). The data transmission takes place between the C cells and the B cells in d computation steps by the use of the states of cells in the set C (remote cells). For the one-way protocol the transmitted information has to be independent of the state of the A cells. The bandwidth in units of bits/step results by taking the binary logarithm of the number of different configurations divided by the spatial width and the number of computation steps.

Consider for example a set B consisting of one cell with coordinates $(0, 0)$ in the Game-of-Life CA where $G_r = \{(x, y) | x \geq 0\}$, then the set C is $\{(-1, -1), (-1, 0), (-1, 1)\}$, and A consists of the other 6 cells $A = \{(x, y) | x \in \{0, 1\}, y \in \{-1, 0, 1\}\}$. As said before, the next state for Game-of-Life depends on the number of neighbors in state 1, hence if the cell is in state 0 and the other cells in set A have zero, one, two or three cells in state 1, the information needed from the send side, is whether there are three, two, one, or zero cells in state 1, respectively. Since for the one-way protocol the transmitted information has to be sufficient independently of the state of the cell set A, it is necessary to transmit the number of ones in the three send cells, which can be encoded in 2 bits. When considering k adjacent cells at the border without taking the transition function into account, $k + 2$ bits are needed per computation step, which is somewhat more as well be seen later.

3 Binary Decision Diagrams

BDDs were first proposed in the 1950s, but they only became more popular after the paper by Bryant [3]. Since then they have become a standard tool in electronic design automation tools such as circuit synthesis. Also, general purpose tools for topics such as relations or Petri nets make use of BDDs [1]. A BDD is a data type that represents a Boolean Function $\{0, 1\}^k \rightarrow \{0, 1\}$. Several BDDs can be handled in a common framework (library instance, manager, or similar terms are in use). Therefore, BDDs can be used to handle functions between finite sets by binary encoding. Several versions (for instance ZDD = Zero-suppressed Decision Diagrams, ROBDD Reduced Ordered Binary Decision Diagrams) and improvements (addition of inverted edges) have been developed over time, that improve the efficiency for some application domains. The basic operations of BDDs are

- Basic functions for constant one, zero, and identity of a single variable $f(x) = x$
- Basic logic operations of two inputs, including XOR, AND, OR

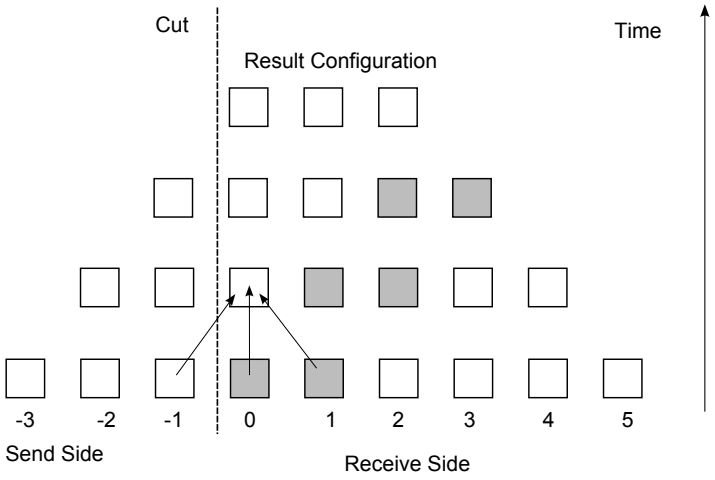

Figure 1: A Calculation of a Cut CA: One Part is Considered Receiving Data for its Computation.

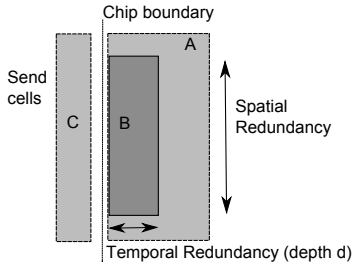

Figure 2: The Three Sets of Cells Relevant for Data Transmission: Receive Cells A, Result Cells B, and Send Cells C

- Composition of two functions, by replacing an input variable in one function by a second function
- Testing, whether two given functions are equal, including test for constant one or false
- Decomposition of a function in a disjoint set of product terms (AND-combination of variables or inverses of variables)
- FORALL, EXIST quantization of a function for one or several of its input variables.

Internally, a BDD is an acyclic graph, of nodes, which are associated with an input variable and (except for two leaves) have two outgoing edges (`THEN` and `ELSE`). In the case of ROBDDs the sequence of variables is the same for all paths, but variables can be skipped. Semantically, a node corresponds to a Shannon function $f(v,T,E) = (v \wedge T) \vee (\overline{v} \wedge E)$. That is where the name Binary Decision Diagram comes from, each node corresponds to the distinction, whether the input variable is true or not, and for both cases, the edge to the subcase leads to the next node. The two lowest nodes correspond to `true` and `false`. In the reduced case each node is unique, which is established bottom-up after each operation using a hash table.

4 Bandwidth Analysis Algorithm

The algorithm described in this section can be regarded as a Boolean function division method [2]. The algorithm handles a generalization of the CA bandwidth problem. Later in this paper refinements for the special case are discussed. Given the aggregated transition function $F : S^{|A|} \times S^{|C|} \rightarrow S^{|B|}$ from the states of the cell sets A and C into the states of the cell set B after d computation steps, the dependency of the states in cell set C is extracted. This means, F is to be decomposed into two functions Z and U, such that $\forall s_a, s_c : F(s_a, s_c) = U(s_a, Z(s_c))$. The function Z should have a minimal image size.

The size of the image of Z is the provides the one-way bandwidth, after applying logarithm and scaling.

To obtain such a function Z an equivalence relation $\cong_F$: $s_c \cong s'_c \Leftrightarrow \forall s_a : F(s_a, s_c) = F(s_a, s'_c)$ is defined. The number of equivalence classes of $\cong$ is the image size of Z, as the canonical mapping of each element of $S^{|C|}$ into its equivalence can be used as Z. The necessary steps, creating the equivalence relation and counting its equivalence classes can be carried out using BDDs with good performance. In fact, for several examples the construction of the BDD representation of the aggregated transition function requires much more running time than the creation of the equivalence relation and the counting of its classes. There are several BDD algorithms known which determine the size of a set. To count the classes, a single representative needs to isolated from each class. This is done by isolating the (arithmetical) maximum.

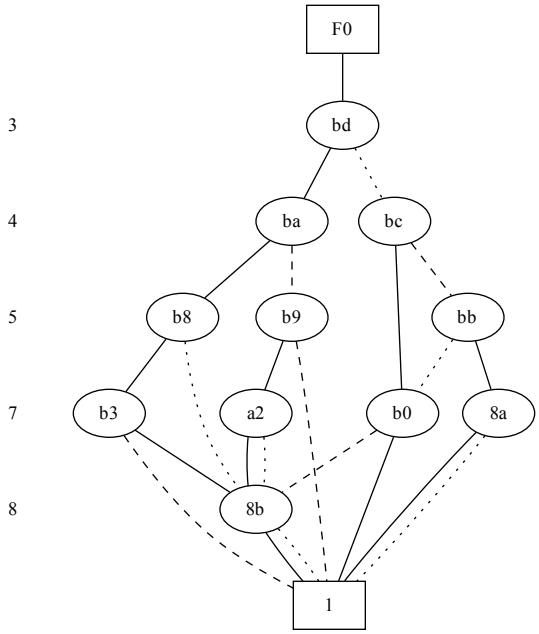

Figure 3: Example of a Binary Decision Diagram

1. Convert the function F into a relation format $r_F(s_a, s_c, s_b) = 1 \Leftrightarrow F(s_a, s_c) = s_b$
2. Create a copy of the relation with a different set of input variables for c: $r'_F(s_a, s'_c, s_b)$
3. Combine r_F and r'_F forming a relation $v(s_a, s_b, s_c, s'_c) = r_F \wedge r'_F$.
4. Remove the inputs for s_a and s_b by forall-quantorization of variables s_a and s_b: $e(s_c, s'_c) =$ FORALL $s_a, s_b v(s_a, s_b, s_c, s'_c)$
5. Create the greater-than-or-equal function for inputs s_c, s'_c.
6. Form $M(s_c, s'_c) = (s_c \geq s'_c) \wedge e(c, c2)$
7. Remove the inputs s'_c from M by FORALL- quantorization, resulting in N
8. Determine the number of fulfilling inputs of N.

Step 8 can use the decomposition into disjoint cubes (product terms). The set size corresponding to one cube is obtained by counting the don't-care variables for the relevant variables (s_c in this case). Since the decomposition is disjoint, the contributions from the cubes can be added. Since the cube decomposition is obtained by recursively traversing the BDD, the method can be simplified if the internal BDD representation is exploited.

For CAs with more than two states, the representation of each state (s_a, s_b, etc.) requires several bits, which are either BDDs or BDD variables depending on the step in the algorithm. In the algorithm description above, first one big equivalence relation is created in the steps 2 to 4. It was found that the algorithm runs much faster if these steps are executed separately for each individual bit of the cell set B state representation. The results for the individual equivalence relations are AND-combined after step 4. This is because the temporarily created BDDs will be much smaller, though more BDD operations are needed.

5 Formal Solution for Spatial Scaling

In the previous section the algorithm for the determination of the one-way bandwidth was described. It requires the construction of a BDD that represents the equivalence relation for the C cell configurations. For spatial scaling this BDD can grow significantly. Furthermore, this method only gives numeric results but not better understanding of the underlying mechanisms. In this section we describe a method that determines a closed formal solution for scaling the result set along a regular chip boundary (as for instance in the Game of Life). It makes use of the situation of CA to have a regular structure and is therefore not as general as the previous algorithm. Regular means in formal terms, that there exists a subgroup $B < G$, that acts transitively on the boundary cells. For the example of Game of Life and the given send and receive sets, this action is the addition of y-coordinates. When starting from an arbitrary border cell, any other border cell can be reached this way. It will become evident that the method can also be used for irregular borders. The only difference is, that why the presented method uses the power of a single matrix, in this case several matrices have to be determined and multiplied.

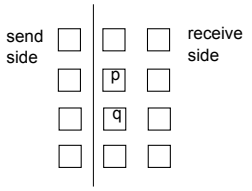

Figure 4: Situation for Scaling along the Border

When considering two different receive side cells A_1, A_2, that each depend after d time steps on at least one remote cell, both cells induce their equivalence relation $\cong_1$ and $\cong_2$ (as one element receive cell sets). When combining the two cells into one cell set $\{A_1, A_2\}$, a combined equivalence relation $\cong_r$ results. At least from the algorithm in the previous section it can be seen, that the resulting relation is the logical AND of the two relations, i.e. $s_x \cong_r s_y \Leftrightarrow s_x \cong_1 s_y \wedge s_x \cong_2 s_y$. Formally, we have to extend the C sets in both cases to make the previous formula meaningful. We are interested in the number of equivalence classes of the resulting equivalence relation. Two configurations are in the same resulting equivalence class $\cong_r$ if they are in the same equivalence class in both original equivalence relations $\cong_1$ and $\cong_2$. Reversely, considering a pair of equivalence classes from $\cong_1$ and $\cong_2$, they can be combined into a new equivalence class, if their set-theoretic intersection is non-empty. In this case the equivalence classes are called compatible. Two equivalence classes are compatible if the restriction of the elements (which are state vectors) to the overlapping cells results in the same set. This non-empty set of configurations of the overlapping cells can be viewed as a "continuation type", i.e. an indicator how the next equivalence class has to look like. For $|S|$ states and l overlapping cells, there are $2^{|S|^l} - 1$ different continuation types. For a given target set, the number of equivalence classes per continuation type can be counted and arranged in a vector.

Coming back to the example of Game of Life, Figure 4 shows two adjacent cells p and q for $d = 1$ with the corresponding send side cells, which overlap on two cells. As observed before, the four equivalence classes for each cell (which are shifted copies of each other) consist of those configurations, that have the same number of ones $(0, \ldots, 3)$. As one example consider the equivalence class of $\{(0,0,0)\}$ for both cells. The overlapping cells for the first class are the second two vector elements, for the second class these are the first to vector elements. We obtain one equivalence class with one element $\{(0,0,0,0)\}$ Another example is the class with one 1 for the first relation (for receive cell p): $\{(1,0,0),(0,1,0),(0,0,1)\}$. It is compatible with two equivalence classes for receive cell q, namely $\{(0,0,0)\}$ (resulting in $\{(1,0,0,0)\}$) and $\{(1,0,0),(0,1,0),(0,0,1)\}$, resulting in $\{(0,1,0,0),(0,0,1,0)\}$.

As a result, we get the following equivalence classes for the combined relation (for simplicity, the commas have been omitted):

Classes	Continuation Type
$\{(0000)\}$	00
$\{(1000)\}$	00
$\{(0001)\}$	01
$\{0100, 0010, 1001\}$	00,10,01
$\{1100, 1010\}$	*0
$\{0011, 0101\}$	*1
$\{1011, 1101, 0110\}$	11,01,10
$\{(1110)\}$	10
$\{(0111)\}$	11
$\{(1111)\}$	11

Since there is only a finite number of send side cells there can only be a finite number of conditions on the overlapping cells, which apply to the equivalence classes. Such a condition is a subset of the set of all possible configurations on the overlapping cells. As we are interested in the number of equivalence classes, we can associate with each subset of configurations the number of corresponding equivalence classes. This can be represented as a vector.

For the example, by counting the equivalence classes per continuation type in the table, a vector $(2, 1, 0, 1, 1, 0, 1, 2, 0, 1, 0, 0, 0, 1, 0)$ is obtained.

For k overlapping cells with s states the length of the vector giving the number of equivalence classes for each continuation condition is $|S|^k - 1$, as the set of conditions cannot be empty. In a similar way, the matrix is obtained that represents the process of appending one cell (or slot of cells when considering a larger depth d). Each entry of the matrix reflects whether a continuation condition on the one side can be connected with the continuation condition on the other side under the given equivalence relation. In the Game-of-Life example, consider the equivalence class with one 1. If the continuation condition on the one side is the set 00, only one element of the equivalence class, $(0, 0, 1)$ matches, and hence the continuation condition to the other side is 01. If the continuation condition was however $*0$ on one end, there would be two matching elements, $(1, 0, 0)$ and $(0, 0, 1)$, resulting in the new continuation condition $0*$. Note that there are many entries in the matrix that can never be 1, independent of the equivalence relation. The situation is similar to De-Bruijn-Graphs, which is not a surprise, as these graphs have been widely used for the analysis of CA properties [11]. The resulting matrix for the Game of Life with $d = 1$ is

$$\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

The fact that the last column is zero, is explained by the observation that there is no equivalence class in Game of Life that there is no single equivalence class that contains elements that end on 11 and 00. Note that the size of the matrix grows very rapidly, for game of life depth 2, the a $2^{256}-1 \times 2^{256}-1$-matrix results.

6 Results

The experimental implementation of the presented algorithm was done in C++ using Microsoft Visual C++ for debugging, while the runs with performance measurement and those for the more complex problems were compiled using gcc version 4.1.2 on 64-bit Linux. The machine used is an IBM xSeries server based on 4 Intel Xeon processor cores at 3.6 GHz populated with 8GB of Memory. This machine was used because it was also used in previous experiments and was available in the same configuration.

The entire program code (including previous methods, and new methods for the two-way protocol) has approximately 6000 lines of code. For BDD, the already mentioned library CUDD was used. Is C++ layer allows a comfortable programming, abstracting from internals of the library. Furthermore, Xerces-C from the Apache project is used to parse XML files which are used to provide parameters and options for the program.

Since the 256 one-dimensional, two-state cellular automata have found a lot of interest in the past, their bandwidth for depth 15 is given in the table at the end of the section. The columns are named R for the rule, BW for the bandwidth (not normalized), the total run time and the maximum number of BDD-nodes during the computation. It is worth noting the very differing running times for the different rules. A depth of 15 means that 2^{45} configurations are considered, but in most cases the BDD algorithm is so efficient that it runs the entire program in less than a second. Of course, this incorporates cases where the aggregate transition function is very simple, and so is the equivalence relation.

For $k = 254$ the CA does a OR-combination of its inputs. Therefore, the maximum amount of information is received when the receive side has only zeroes, the one-way and the two-way bandwidth is the same for this CA. For the resulting state only the distance to the first 1 on the send side matters, all further 1s will not change the result. Therefore, in k steps a value between 0 and k is communicated, which can be encoded in $\log_2(k+1)$ bits, or $\log_2(k+1)/k$ bits per step. For $k = 15$ the bandwidth is 4/15. Of course, because these CA are only one-dimensional the scaling method cannot be used for them.

For one example, rule 107, the total run time was 404 seconds, of which 66 s were needed for the construction of the aggregate transition function. The creation of the 15 equivalence relations and the AND-abstraction did cost between 5 s and 20 s per digit. The last steps, creating the comparison function, applying it to the equivalence relation and image size determination takes another 50 s. The algorithm tries to set unused BDDs from intermediate results to 0, so that the total number of nodes in the BDD environment (in particular the unique hash table) is kept as small as possible. For this example, the construction of aggregate transition function requires around 11 million BDD nodes, while the resulting equivalence relation consists of only 18484 nodes, and the final predicate identifying the resulting 1872 equivalence classes has only 804 nodes.

R	BW	run time	nodes	R	BW	run time	nodes
0	1	184msec	112	1	2	243msec	271
2	2	240msec	146	3	2	253msec	142
4	2	253msec	146	5	2	239msec	143
6	57	649msec	50k	7	16	353msec	11k
8	1	197msec	145	9	237	5sec	408k
10	2	239msec	129	11	5	307msec	456
12	2	241msec	129	13	16	302msec	4k
14	4	286msec	412	15	2	240msec	112
16	2	241msec	149	17	1	249msec	142
18	971	1min	3M	19	3	256msec	277
20	60	695msec	52k	21	16	385msec	13k
22	2023	13min	21M	23	30	347msec	7k
24	6	300msec	417	25	288	6sec	407k
26	1615	9min	16M	27	92	871msec	66k
28	252	1sec	136k	29	1597	531msec	24k
30	1753	5min	10M	31	16	351msec	11k
32	16	334msec	2k	33	29	449msec	13k
34	1	183msec	129	35	2	241msec	245
36	4	260msec	487	37	770	46sec	2M
38	55	739msec	45k	39	92	857msec	66k
40	24	398msec	12k	41	1872	6min	10M
42	2	242msec	150	43	2	259msec	416
44	4	287msec	618	45	579	19sec	930k
46	2	243msec	363	47	5	304msec	456
48	2	239msec	129	49	2	242msec	260

R	BW	run time	nodes	R	BW	run time	nodes
50	29	340msec	7k	51	1	192msec	112
52	79	688msec	44k	53	108	746msec	62k
54	619	35sec	1M	55	3	252msec	277
56	4	305msec	594	57	30	337msec	10k
58	30	346msec	9k	59	2	247msec	245
60	256	305msec	1k	61	350	5sec	402k
62	102	789msec	95k	63	2	241msec	142
64	1	182msec	145	65	261	5sec	411k
66	2	243msec	404	67	350	5sec	402k
68	1	187msec	129	69	16	311msec	4k
70	162	1sec	135k	71	1597	523msec	24k
72	3	263msec	357	73	615	41sec	1M
74	170	1sec	166k	75	579	19sec	930k
76	2	253msec	149	77	30	361msec	7k
78	28	407msec	9k	79	16	324msec	4k
80	2	258msec	129	81	4	327msec	478
82	1795	9min	16M	83	108	757msec	62k
84	2	246msec	424	85	1	183msec	112
86	1176	5min	10M	87	16	397msec	13k
88	153	1sec	167k	89	386	19sec	925k
90	32768	413msec	147k	91	770	46sec	2M
92	28	423msec	9k	93	16	321msec	4k
94	1021	41sec	1M	95	2	242msec	143
96	23	399msec	11k	97	1817	6min	10M
98	3	253msec	596	99	30	347msec	10k
100	4	308msec	661	101	386	19sec	925k
102	128	303msec	1k	103	288	6sec	407k
104	108	607msec	54k	105	32768	397msec	147k
106	1149	6min	11M	107	1872	6min	10M
108	6	300msec	811	109	615	41sec	1M
110	610	23sec	1M	111	237	5sec	408k
112	2	240msec	148	113	2	241msec	416
114	29	352msec	10k	115	2	241msec	260
116	4	308msec	359	117	4	308msec	478
118	86	859msec	95k	119	1	183msec	142
120	1700	5min	11M	121	1817	6min	10M
122	1569	4min	9M	123	29	411msec	13k
124	631	22sec	1M	125	261	5sec	411k
126	1278	2min	5M	127	2	242msec	271
128	16	295msec	2k	129	1278	2min	5M
130	16	311msec	6k	131	102	791msec	95k
132	16	310msec	5k	133	1021	41sec	1M
134	159	1sec	189k	135	1753	5min	10M
136	8	297msec	1k	137	610	23sec	1M

R	BW	run time	nodes	R	BW	run time	nodes
138	2	240msec	149	139	2	243msec	363
140	9	297msec	2k	141	28	413msec	9k
142	2	249msec	416	143	4	292msec	412
144	16	310msec	6k	145	86	886msec	95k
146	1376	4min	8M	147	619	35sec	1M
148	145	2sec	183k	149	1176	5min	10M
150	32768	396msec	147k	151	2023	14min	21M
152	11	321msec	4k	153	128	304msec	1k
154	807	1min	4M	155	55	746msec	45k
156	179	1sec	116k	157	162	1sec	135k
158	159	2sec	189k	159	57	648msec	50k
160	72	341msec	1k	161	1569	4min	9M
162	16	309msec	4k	163	30	344msec	9k
164	877	583msec	43k	165	32768	400msec	147k
166	807	1min	4M	167	1615	9min	16M
168	16	388msec	9k	169	1149	6min	11M
170	1	185msec	112	171	2	242msec	150
172	18	359msec	8k	173	170	1sec	166k
174	2	241msec	149	175	2	241msec	129
176	16	302msec	4k	177	29	361msec	10k
178	30	346msec	7k	179	29	340msec	7k
180	1245	1min	4M	181	1795	9min	16M
182	1376	4min	8M	183	971	1min	3M
184	2	255msec	398	185	3	256msec	596
186	16	316msec	4k	187	1	183msec	129
188	10	327msec	4k	189	2	241msec	404
190	16	318msec	6k	191	2	281msec	146
192	9	295msec	1k	193	631	22sec	1M
194	10	320msec	4k	195	256	333msec	1k
196	9	299msec	2k	197	28	402msec	9k
198	179	1sec	116k	199	252	1sec	136k
200	2	240msec	147	201	6	300msec	811
202	18	357msec	8k	203	4	293msec	618
204	1	183msec	112	205	2	250msec	149
206	9	307msec	2k	207	2	239msec	129
208	2	241msec	150	209	4	302msec	359
210	1245	1min	4M	211	79	695msec	44k
212	2	259msec	416	213	2	242msec	424
214	145	2sec	183k	215	60	693msec	52k
216	20	392msec	9k	217	4	303msec	661
218	877	587msec	43k	219	4	257msec	487
220	9	305msec	2k	221	1	183msec	129
222	16	315msec	5k	223	2	251msec	146
224	16	399msec	9k	225	1700	5min	11M

R	BW	run time	nodes	R	BW	run time	nodes
226	2	253msec	398	227	4	359msec	594
228	20	393msec	9k	229	153	1sec	167k
230	11	326msec	4k	231	6	312msec	417
232	30	349msec	7k	233	108	603msec	54k
234	16	409msec	9k	235	24	398msec	12k
236	2	242msec	147	237	3	258msec	357
238	8	301msec	1k	239	1	184msec	145
240	2	240msec	112	241	2	242msec	148
242	16	320msec	4k	243	2	241msec	129
244	2	241msec	150	245	2	242msec	129
246	16	320msec	6k	247	2	241msec	149
248	16	393msec	9k	249	23	402msec	11k
250	72	339msec	1k	251	16	328msec	2k
252	9	300msec	1k	253	1	184msec	145
254	16	317msec	2k	255	1	184msec	112

7 Outlook

The main limitation of the presented approach is the need to construct the BDD for the aggregated transition function. As this can quickly hit memory and computation time limitations, a method is needed that circumvents this problem. I am currently working on such a method, but it is too early to present results here. The algorithm uses two phases, in the first phase it characterizes the set of configurations on the "slope" in the space-time structure (these are the shaded cells in Figure 1). In the second phase the equivalence relation is refined top-down (backward in time). It determines the same equivalence relation as in the method described here, hence the scaling method can also be applied with the new algorithm. How much larger temporal depth is possible with this new method will need to be seen. Even though, the algorithm can be implemented in a general way allowing the representation of the function to be factored in an arbitrary composed way, it makes use of the special situation of multiple time step computation of CA.

For the two-way protocol better methods compared to [5] have been developed. However, they do not allow such a formal way of scaling as in the one-way case. These newer two-way protocol methods are in the same way limited by the construction of the aggregated transition function, and they can be improved in a similar way; it is somewhat harder, because the two-way protocol takes the receive side configuration into account in a more detailed way. With the progress for bandwidth analysis presented in this paper, it would be also interesting to investigate other two-dimensional CAs with more states per cell, including WireWorld or even a universal constructor [7]. Considering the large memory and computation requirements of more complex CAs and investigation depth and width, a parallel implementation of the analysis algorithms would be highly interesting. There are a few BDD libraries which use internal parallelism. But as long as there are options for strong improvements of the sequential algorithm available, a parallelization is too early.

References

[1] R. Berghammer, B. Leoniuk, and U. Milanese. Implementation of relational algebra using binary decision diagrams. In H.C.M. de Swart, editor, *6th International Conference RelMiCS 2001 and 1st Workshop of COST Action 274 TARSKI, Oisterwijk, The Netherlands, October 16-21*, volume 2561 of *Lecture Notes in Computer Science*, pages 241–257, Berlin-Heidelberg-New York, December 2002. Springer.

[2] R. K. Brayton and C. McMullen. The Decomposition and Factorization of Boolean Expressions. In *International Symposium on Circuits and Systems*, pages 49–54, May 1982.

[3] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[4] P. Coteus, H. R. Bickford, and et. al. Cipolla, T. M. Packaging the blue gene/l supercomputer. *IBM Journal of Research and Development*, 49:213–248, March 2005.

[5] Andreas Döring. Bandwidth of firing squad algorithms. Number 24 in PARS Mitteilungen, pages 176 – 184, 2007.

[6] Christoph Dürr, Ivan Rapaport, and Guillaume Theyssier. Cellular automata and communication complexity. *Theor. Comput. Sci.*, 322(2):355–368, 2004.

[7] Tim J. Hutton. Codd's self-replicating computer. *Artificial Life*, 16(2):99–117, 2010.

[8] Geert Janssen. Design of a pointerless bdd package. In *International Workshop Logic and Synthesis*, 2001.

[9] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., 1998.

[10] F. Somenzi. CUDD: CU decision diagram package release, 1998. available at http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html.

[11] Klaus Sutner. Linear cellular automata and de bruijn automata. In *Volume 460 of Mathematics and Its Applications [4*. Kluwer, 1991.

[12] H.-S. Philip Wong, Jie Deng, Arash Hazeghi, Tejas Krishnamohan, and Gordon C. Wan. Carbon nanotube transistor circuits: models and tools for design and performance optimization. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, ICCAD '06, pages 651–654, New York, NY, USA, 2006. ACM.