RZ 3772 (# 99782) 04/08/2010
Computer Science 9 pages

Research Report

Bandwidth of Firing Squad Algorithms

A. Döring

IBM Research – Zurich
8803 Rüschlikon
Switzerland

Research
Almaden • Austin • Beijing • Delhi • Haifa • T.J. Watson • Tokyo • Zurich

Bandwidth of Firing Squad Algorithms

Andreas C. Döring

ado @ zurich.ibm.com
IBM Research GmbH, Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon, Switzerland

Abstract. The Firing Squad Synchronization Problem (FSSP) is one of the best-studied problems for Cellular Automata (CA) and is part of many more sophisticated algorithms. Solutions for it such as the one from Mazoyer or Gerken use travelling signals, a basic technique of CA algorithms. Hence in addition to their importance, they represent typical CA algorithms. Beside the interest in their bandwidth following the definition of [1] they also serve well to study the methods for determining the bandwidth in the first place. As was demonstrated, bandwidth calculation for CA is a complex problem and requires the development of sophisticated algorithms. New approaches in this direction are given and compared using selected FSSP algorithms.

1 Introduction

Cellular automata (CA) combine three aspects in one model: inherently parallel computation, a model for physical processes, and a possible structure for future computing devices. For this reason the bandwidth necessary for proper operation of a given CA is of interest when the CA is cut into parts. From a technical standpoint, the use of CA as a basic technology requires the solution of this problem because inter-chip communication will have lower bandwidth and higher latency than intra-chip communication for future technologies. Furthermore, understanding communication requirements of CA combines storage, communication and computation capabilities in one model, which is accepted as being close to most physical phenomena used for computation. In [1] and [2] bandwidth for CA is defined and the complexity of basic enumeration pointed out. To find results for automata with higher dimensions or more states better algorithms are needed. Most of the algorithmic options rely on typical behavior rather than improving the worst case. For example, the incremental enumeration method described in Section 3 will be slower than the basic enumeration algorithm when applied to a CA with linear transition function. This particular CA type can easily be analyzed as described in [1]. The linearity of the transition function can be detected with acceptable effort. Therefore, the bandwidth determination algorithms need to be evaluated using typical CA algorithms. For one of the oldest problems of CA, namely the synchronization of a continuous set of cells – called Firing Squad Synchronization Problems (FSSP) – a considerable number of algorithms has been proposed in the past. The fundamental bandwidth requirements of the problem is easily derived. Hence, evaluating various algorithms for this problem reveals both the portion of bandwidth associated with the problem and that of the algorithm considered.

Even though bandwidth is a well-known concept it is not straight-forward how to define it in the domain of CA. A CA consists of a (potentially infinite) set of cells, which are connected in a regular fashion, called topology. In the case of the FSSP the automata are indexed by integers and each cell is connected with the two cells according to the successor and predecessor of its own location. The cells are finite-state machines that switch synchronously. Each cell uses its own and its neighbors' states as an input to determine its new state. For the definition of bandwidth, the CA is divided into two sets of cells. For the

problem considered here, the sets are those with negative and non-negative indexes. To carry out the computation of a CA, information needs to be exchanged between the parts. For instance the cell with index 0 has the two neighbors with indexes 1 and -1, where the latter is in a different set. Hence at every step, the state of the cell -1 needs to be transmitted to the other CA part and vice versa. However because the state of a cell depends on its own and its neighbors' preceding states, redundancy between the data transmitted in adjacent time steps is expected. Furthermore, depending on the transition function and the state of the receiving side, only partial information is needed. In effect, a compression of the information transmitted is possible. As it is pointed out in [2], one can consider protocols with different numbers of data exchanges. In this paper a two-way protocol is assumed: in a first step the receiving side tells the send side, which kind of information is needed, in a second step the send side compresses the information accordingly. The bandwidth a connection between the two chips needs to provide depends therefore on the maximum number of different images in the cells denoted *Result Configuration*. This set of images is generated for any fixed configuration of the receive side cells, while varying the send side configurations. If compression is done over more time steps a lower maximal bandwidth can result. To obtain bandwidth numbers in bit/step units, the logarithm of the image size divided by the number of steps would be used. For better discussion of the algorithmic options, mainly image-size arguments are used in this paper.

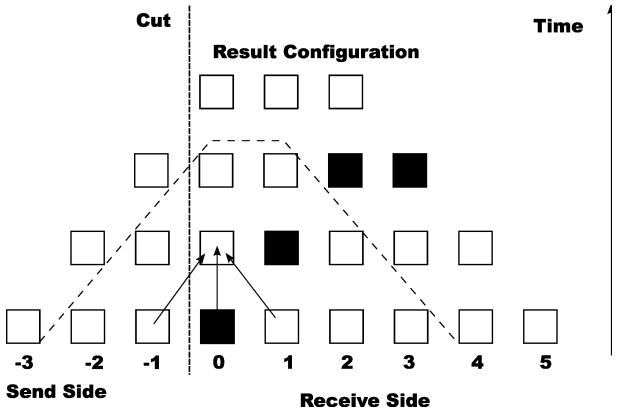

Fig. 1. A calculation of a cut CA: one part is considered receiving data for its computation.

In the next section, the FSSP is introduced in more detail and its intrinsic bandwidth is discussed. Then Sections 3 and 4 introduce two new algorithmic approaches. In Section 5 background information on the implementation of the algorithms and FSSP examples is provided. Section 6 gives the results, and in the final section an outlook on further options for widening the approach and designing even more powerful algorithms is given.

2 Firing Squad Synchronization Problem and Algorithms

As pointed out, the FSSP is based on a one-dimensional, direct neighborhood topology. Its input is based on three states, called border, quiet and general. A finite section of cells is marked by border cells on each end. On the right end (larger position coordinate), the cell next to the border within the section turns into the general state to start the algorithm. All other cells within the section start with the quiet state. The FSSP requires that all states within the section be turned into another defined state, called fire, at the same time. Before that no cell may take on this fire state. In short, a configuration of the form "$qqq...qg$" has to be translated into "$fff...f$", without hitting 'f' before. In [3] a survey of the history and

solutions of the FSSP is given. Historic solutions are investigated by extensive simulations and some minor flaws found in them are corrected. These corrected versions have been used for this paper.

Further solutions (7 states) are found in [4], which are also used. They are interesting because they use a different basic principle than the best solution by Mazoyer [5] with 6 states. In total, 5 FSSP solutions are used in this paper, namely those by Mazoyer, by Balzer, by Gerken and the two by Yunès. For comparison, in [6] a FSSP solution is presented that can operate with only 1 bit of data transmission per step, but requires 78 states.

Considering the problem from an abstract point of view, every deterministic algorithm will translate the distance between the two borders into a time span after which the cells fire. Hence, this distance is the essential input. Note that all algorithms only use cells between the two borders. We only need to consider the case when the two borders are in different parts of the divided CA. In this case, exactly the distance of the corresponding border symbol to the cut between the two parts is needed by the other part. If both parts have this information from the other part, they can translate the total distance into the time the algorithm would need in the undivided case and fire after that time. In this way, both parts will fire synchronously, as required. Therefore, for larger distances of the border, the bandwidth required is $\log k/k$ if redundancies that are more than k cells deep are exploited. However, for short distances, the values need to be transferred faster to allow the algorithm to react in time. As several published algorithms assume anyway a minimum distance for correct operation, this detail is not discussed further here.

3 Incremental Enumeration Algorithm

The basic enumeration algorithm exhaustively searches a configuration of the receiving side, such that the number of resulting images is maximum when the send side assumes all possible configurations. Both aspects, the search for the maximum and the determination of the image set size are done by enumerating all possible configurations. This results in a high computation effort: for a one-dimensional CA with direct neighborhood such as those in the FSSP problem, it is $(2t-1)ts^{3t}$ cell transitions because for the configuration of t cells after t computation steps the configuration of $3t$ cells is needed, where each cell can take s states.

Because the bandwidth is defined as a maximum it is not necessary to enumerate all configurations on the receiving side, as long as two subtasks can be solved:

1. Find the receive-side configuration that leads to the maximum data transmission, and
2. prove that there are no other configurations with higher bandwidth.

The basic idea behind the incremental algorithm is estimating the bandwidth for a higher temporal depth from the results of a lower one. As computing the bandwidth for the lower temporal depth is much faster, at relatively low cost the receive side configuration resulting in the maximum bandwidth is found much faster. The correlation of the bandwidth for a lower temporal depth and a higher one is intuitively clear from several points of views. It is illustrated by the dashed lines in Figure 1. Considering the data transmission in several steps, it is clear that in three steps not more data can be transmitted than the sum of the maximum achievable data transmission of 1 and 2 steps. Considering the size of images of a function f, we deal with a decomposition of f into $f(x_1, x_2, x_3) = f_1(f_2(x_1, x_2), f_3(x_3))$. By knowing the image space size of f_2 and f_3, an upper bound for the number of images of f is given. In terms of the CA-bandwidth definition, the bandwidth for temporal depth 1 and n provides information on the achievable bandwidth after $n+1$ steps. In particular, when the maximum bandwidth b for a receive side configuration of size $2*n$ is known, the bandwidth

after extending it by 2 cells cannot be higher than $b * b_1$, with b_1 the maximum bandwidth for temporal depth one (based on two receive side cells). One could consider keeping also the set of resulting configurations from the previous configurations, which would save some cone computations but would require much more storage. The algorithm maintains a data structure that contains the receive side configurations for a given configuration sorted by the bandwidth they allow.

The entire algorithm is sketched in Figure 2.

```
incremental(CA,td)
{
(bestconf[1],sorted_configurations) := enum_imagesize(CA,1);// base enumeration algorithm
FOREACH depth ∈ {2,...,td}
            bestconf[depth] := 0;
            FOREACH conf ∈ sorted_configurations // high-to-low bandwidth order
                        IF conf.bw < bestconf[depth]/bestconf[1] THEN break;
                        FOREACH conf_extension ∈ ReceiveSide(2*depth,2*depth+1)
                                    h := enum_imagesize_conf(CA,conf & conf_extension,depth);
                                    new_sorted_configurations[h].append(conf & conf_extension);
                                    bestconf[depth] := max(bestconf[depth],h);
```

Fig. 2. Incremental bandwidth enumeration algorithm

Consider for example the FSSP algorithm by Mazoyer [5]. When investigating all configurations for a temporal depth of 1, the following image sizes result:

Image set size	5	4	3	2
Number of Configurations	2	11	7	15

The incremental algorithm will first determine this distribution by full enumeration. This needs the handling of $s^3 = 256$ configurations. Note that Mazoyer's algorithm is given in the literature with 6 states; however, without including the border state; an illegal state was needed as well (see Section 5). Following this, the algorithm starts with the two configurations resulting in 5 images (`aa` and `bb`) and extends them by two more states. In the best case, 25 images could result, but the actual maximum found has 11 images for `babb`. Because of this result, all configurations that result in 1 or 2 images need to be considered because they cannot result in more than 10 images. The history of a computation with 4 levels can be found in Figure 3. One can see how the algorithm at generation 3 stopped the computation of successors for the five lower sets.

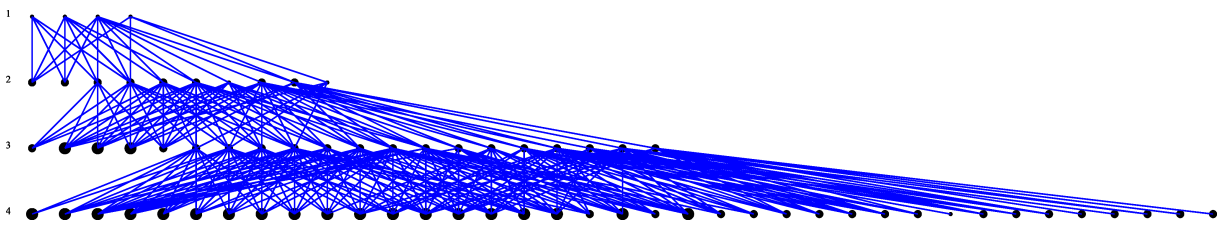

Fig. 3. Four generations of the incremental algorithm. The area of the circles is proportional to the logarithm of the number of configurations with the same number of images. The lines mark from which configurations in the preceding generation the new ones are derived

4 BDD-Based Image Size Determination

Whereas the incremental enumeration method can typically reduce the number of receive-side configuration evaluations tremendously, using enumeration of all send-side configurations contributes a factor s^{t_d} for a one-dimensional topology to the running time of the bandwidth determination. This can be considered a special case of the analysis of a finite function's image space size. Surprisingly, I was not able to find any advanced algorithmic results on this topic. Therefore, several approaches for representing a finite function resulting from composition and pairing of simpler functions were investigated. The result, a BDD-based algorithm, is introduced here. As it turns out, it provides a substantial acceleration of the bandwidth determination. Because BDDs were not primarily designed for this purpose, it can be expected that there are other more efficient methods than the one described in the following.

A Binary Decision Diagram is a data type based on the so-called Shannon operator $s(v, a, b) = va + \bar{v}b$. Although it was known much earlier, the paper by Bryant [7] triggered intensive research and application of this data type. A BDD represents a Boolean function with several input and output variables as an acyclic graph. For one particular type, the reduced ordered BDD, the representation is canonical, i.e. the BDDs of two functions are identical iff the functions are the same. Therefore, already the creation of a BDD solves the satisfyability problem, a known hard problem. In the worst case this takes exponential time and space, but BDDs behave quite well for typical functions. BDDs can be constructed incrementally by starting with variables or the constants One and Zero by using elementary operations such as AND, XOR, and function concatenation. As the considered FSSP algorithms were both published and represented by tables in the analysis program, the translation of an investigated transition function f into the BDD representation was done as shown in Figure 4. Note that the configuration consisting of several state fields is handled like a bit vector, same for the result.

```
int logs := log2 |states|;
int confsize := logs*3; // for our 1-dimensional topology
BDD p;
BDD[logs] res;
res := (false,...,false);
FOR conf ∈ states
      p := true;
      FOR i ∈ {0,...,confsize − 1}
            IF conf[i] THEN p := p & v[i]; ELSE p := p & !v[i];
      FOR i ∈ {0,...,logs − 1}
            IF f(conf)[i] THEN res[i] := res[i] + p;
return res;
```

Fig. 4. Algorithm for converting a CA's transition function into a BDD

For the FSSP algorithms considered, the BDD representation is considerable small. The generation itself took less than 1 s for all examples. To determine the bandwidth, the concatenation of several instances of the transition function is needed; for example, when considering two generations, the resulting function for the cell with index 0 is $(a_2, a_1, a_0, c_1, c_2 \mapsto f(f(a_2, a_1, a_0), f(a_1, a_0, c_1), f(a_0, c_1, c_2))$. The calculation time of the BDDs for the entire computation cone depending on the number of generations is given in Table 1.

Given an m-bit function f in BDD representation, the recursive algorithm in Figure 5 (invoked as `imagesize(f,true,0)` determines the size of f's image space. Note that the

algorithm does not use any BDD-specific operations. It requires the ability to combine functions with Boolean operations and the test whether the function is constant-false.

```
imagesize(f,cond,ix)
{
IF ix ≥ m THEN return 1;
h1:=0;h2:=0;
IF (c * f[i] != 0)  THEN h1=imagesize(f,c*f[i],i+1);
IF (c * !f[i] != 0) THEN h2=imagesize(f,c*!f[i],i+1);
return h1+h2;
}
```

Fig. 5. Algorithm for determining the image space of a function

Algorithm	2	3	4	5
Balzer	0.08	1.0	7.8	52
Gerken	0.05	0.6	4.1	32
Mazoyer	0.01	0.1	0.89	4.5
Yunès 1	0.06	1.0	11.5	124
Yunès 2	0.08	1.5	13.3	116

Table 1. Running time in seconds to create the BDD for the calculation cone with varying temporal depth

As the function result is obtained by adding the constant 1 returned in the first case, the running time of the algorithm is at least linear with the number of images of the function. For the FSSP with a comparably low bandwidth, which corresponds to a small image size, the algorithm is quite fast. For other applications or with other CA, it might be quite inefficient. For those cases another BDD-based algorithm is in work.

5 Implementation

The algorithms for bandwidth determination (base enumeration, incremental enumeration, and BDD-based image size) were implemented in C++ using the Standard Template Libraries, the Xerxes-C Library (Apache Project) and the BDD-library CUDD [8]. The program was developed using mainly Microsoft Visual C++ on Windows XP, but the measurements were carried out on a Intel-processor-based server running Linux. During the implementation several compromises had to be made; hereby the following priorities were applied:
detailed reporting > generality > speed > memory.
XML is used for the configuration that covers the CA, the analysis algorithm, and the reporting details. Also the results are created in XML format, including the illustration in Scalable Vector Graphics. The measurement of the runtime is finished before the report files are generated; the report data collected during the execution of the algorithm is done in data structures such as arrays and hash tables. Of course the data collection will have an impact of the running time of the algorithms investigated. For the various features, program options allow selectively activation of the recording to avoid memory overflow of these structures with high temporal depth computations.

To allow arbitrary topologies in a fast algorithm, an indexing structure is used as intermediate data type. This allows an abstract representation in program code or the configuration file, and the overhead for finding the neighbor is only one level of indirection. The indexing structure is based on a breadth-first-search enumeration starting at the cell positions

that hold the destination image configuration. For the index of cells in the receiving partition positive indexes are used; the sender-side positions are indexed with negative integers. Therefore, the same indexing structure can be used for all generations in the calculation cone, even though they differ in size. For each index an array with the indexes of the neighbors is also created. Two tables provide translation between a display view and the indexing structure. This allows the resulting configurations to be displayed in a format that corresponds to the original topology representation. As simplest processing method, the CA can run from a given starting configuration. This was used to ensure that the implementation of the examples corresponds to its publication. For the states from the definition of the FSSP, the same characters were used to allow test runs with all example algorithms from the same starting string.

Several example FSSP algorithms are incompletely given. However, the analysis algorithms have to enumerate the entire domain. Therefore, an additional state "invalid" was added, and any undefined places in the transition function were filled with that state. Also, all transitions that include the invalid state as an input result in the invalid state again, except for the border states which were considered to be stable. The existence of these don't-care places raises the questions which assignment using the existing states would result in the same bandwidth as the one that uses the illegal state. For the BDD-based algorithm even more completion is necessary. As the restriction of a multi-bit input representing one state value would be difficult to combine with the image set determination algorithm, the values between the number of states and the next power-of-two also have to be filled accordingly.

Otherwise the implementation is straight forward, and will serve to investigate more algorithms and problems.

6 Bandwidth Results and Performance

The results were obtained by batch runs on a xSeries server with dual Xeon 3.6 GHz processors and 8 GByte main memory running a 32-bit Linux kernel. The code was compiled with "-g -O6" settings. The BDD library was built with the standard settings.

The Table 2 shows the bandwidth values and running times for various depth values and, for comparison, the running time for the basic enumeration algorithm for depths 2 and 3.

Algorithm	BDD-based algorithm runtime, image size, normalized bandwidth					runtime of basic enumeration	
	1	2	3	4	5	2	3
Balzer	0 6 2.6	0.7 18 2.1	18.9 42 1.8	414 91 1.6	14940 183 1.5	4	
Gerken	0 7 2.8	0.4 27 2.4	11 67 2.0	69 164 1.6	433 404 1.7	2	1122
Mazoyer	0 5 2.3	0.1 11 1.7	4.2 21 1.5	109 39 1.3	2918 70 1.2	0.6	
Yunès I	0 6 2.6	0.5 14 1.9	23 28 1.6	1004 46 1.4	29087 79 1.3	1.8	1128
Yunès II	0 8 3.0	0.5 35 2.6	12 114 2.3	149 321 2.1	1724 843 1.9	2.6	1128

Table 2. Bandwidth and runtime results using the BDD-image combined with incremental enumeration. For comparison, the run time for the basic enumeration algorithm.

7 Summary and Outlook

The results of this paper are twofold. First, results for the bandwidth requirements of FSSP algorithms from the literature are determined. As it turns out, several algorithms with a higher number of states require also a higher bandwidth, independently of the previously published number on the state change complexity or the number of rules. Therefore, these algorithms do not use their higher number of states to do more work locally.

The second contribution is the new algorithms for bandwidth determination. They allow much more complex instances to be analyzed in a realistic time. They still can be strongly improved:

A The incremental algorithm still analyzes many configurations that later on are not used. Furthermore, it could happen that it has cut off the calculation in a low generation too early. If in higher generations the number of images got smaller even though a wider send side configuration is used, configurations from previously "too bad" configurations would have to be considered. The algorithm can easily detect this if the break-off point is stored with the set of sorted configurations. For FSSP this does not happen, but one can construct CA that exhibit such a behavior. Both effects are a result of the generation-by-generation construction of the algorithm. This approach was chosen, because it has low intermediate memory requirements. For reference, the BDD-based incremental algorithm needs approximately 1 GByte of memory for the algorithm Yunès I at temporal depth 5 when the logging of information is minimal. Both deficiencies can be overcome with an inter-generation search front. However, it might well be that this requires more memory, and the algorithm would certainly be more complex.

B Consider the shaded cell-timestep fields in Figure 1. It turns out that the resulting images only depend on those states. These are fewer cells than the space of the original configuration. Therefore, an algorithm that first determines the set of possible configurations on these cells, and bases the incremental enumeration on them will have a tremendously smaller search space. This is also applicable in higher dimensions. However, the determination of which cells form these particular set at which time step is more complex, in particular an automatic determination with the breadth-first-search algorithm.

C As mentioned, another BDD-based algorithm for determining the image space size is in work.

D To handle CA with a two-dimensional topology (including Conway's Game of Life), a two-step approach is under development. It first handles slices of configurations orthogonal to the cutting edge. Then, based on limitations to the neighboring slices (match on the receiving side, orthogonality of the configuration partitions on the sending side), a graph is formed. The bandwidth is a maximum-weight path in this graph. It needs to be investigated whether it is preferable to ignore the send-side condition first and add it after path determination, which would result in a smaller graph but a higher number of pathes, or to include it, which would have the opposite effect.

E Determine the bandwidth for a one-way and for three-way protocols.

Furthermore, several questions stated in [1] are still open.

Acknowledgement

I thank Geert Janssen from IBM Watson Research Laboratory for the inspiring discussions and suggestions on the use of BDDs and BDD libraries.

References

1. Döring, A.: Bandwidth in cellular automata. PARS-Mitteilungen (2005) 118–125

2. Dürr, C., Rapaport, I., Theyssier, G.: Cellular automata and communication complexity. Theor. Comput. Sci. **322**(2) (2004) 355–368
3. Umeo, H., Hisaoka, M., Sogabe, T.: A comparative study of optimum-time synchronization algorithms for one-dimensional cellular automata - a survey. In Sloot, P.M.A., Chopard, B., Hoekstra, A.G., eds.: ACRI. Volume 3305 of Lecture Notes in Computer Science., Springer (2004) 50–60
4. Yunès, J.B.: Seven-state solutions to the firing squad synchronization problem. Theor. Comput. Sci. **127**(2) (1994) 313–332
5. Mazoyer, J.: A six-state minimal time solution to the firing squad synchronization problem. Theor. Comput. Sci. **50**(2) (1987) 183–238
6. Umeo, H., Michisaka, K., Kamikawa, N., Nishimura, J.: Efficient 1-bit-communication synchronization protocols for cellular automata. In: Proc. of International Conference on Engineering of Intelligent Systems, EIS. (2004)
7. Bryant, R.E.: Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys **24**(3) (1992) 293–318
8. Somenzi, F.: CUDD: Cu decision diagram package release (1998) available at http://vlsi.colorado.edu/ fabio/CUDD/cuddIntro.html.