ABSTRACT
Th software tool SAS+ (Systolic Algorithm Simulator Plus) allows to model two dimensional generalized cellular automata in the cluster-based environment. The suggested tool supports:

• spatial and temporal inhomogeneities of cells transition times;
• date-dependence of transition time of cells;
• dynamically resizing the successive configurations;
• arbitrary number of states of cells;
• use of external parameters.

The result of the SAS+ execution is a cluster-oriented programming module which imitates the cellular automaton.

Keywords
Cellular automata, systolic array, Beowulf cluster computer, software tool.

1. INTRODUCTION
A lot of software packages for cellular automata (CA) simulation has been designed. A survey on programming environment for CA is provided for example in [3, 10].

Usually, virtual CA exploit data parallelism by partitioning cells of initial configuration among processors of parallel computer. Each processor updates those cells that are assigned to it. In such a case simulation procedure involves two cyclic synchronized phases: configuration’s states computation and inter-processor communications phases. At the communication phase the system has to be synchronized to start the phase of the next configuration’s computation.

Our goal is the reduction of management overhead time. We consider two causes of growth of management overhead time.

1. The management overhead time depends on processors idle time. The cell transition time may be highly data-dependent and may vary from cell to cell and from configuration to configuration. Besides the model may run on a heterogeneous system. This implies that in the synchronized system the achievable speedup does not scale with the increase of the number of processors.

2. The management overhead time strongly depends on the volume and number of inter-processor communications. In the synchronized system inter-processor communications are repeated after each computation of new configuration, where each processor transmits data to 8 processors.

We suggest a software tool SAS+ implementing the simulation of a two-dimensional CA [5, 6]. SAS+ supposes that cells are situated on the infinite lattice. Each cell is connected to the cells of Moore neighborhood of radius 1. For reduction of management overhead time we abandon the idea of constructing the synchronized model of CA in favor of a dynamic data driven model. Instead of partitioning the 2D set of cells, the algorithm merges the 3D set of all transitions of CA into 3D tiles pairwise similar, i.e., collections of transitions to be executed as a single unit by a processor. The choice of these tiles is based on minimization of the management overhead time.

The suggested partitioning into tiles minimizes

• the number of interprocessor communications
• the volume of sent and received data per tile.

The set of transitions of all cells may be considered as the set of integral points of 3D-space with causal relations induced by information transmission among cells. Then the synchronized modeling can be viewed as simultaneous realization of transitions lying on the same plane. On the other hand, the transitions can be realized in any order that respects the partial order of causal relations. From this viewpoint the transitions of any tile may be realized at any moment after all preceding tile transitions are accomplished.

Our simulator explores the dynamic data driven mapping of tiles to processors in order to balance the total load between the processing nodes and minimizes the number of interprocessor communications [2].

Our algorithm can deal with CA, where different transitions may take different computation time depending on the state of the neighborhood. Moreover, the time of transition may depend on spatial position of the cell and the serial number of configuration. This means that spatial and temporal inhomogeneities of transitions are supported. For efficient implementation we reformulate the time driven simulation to the data driven simulation, where computations may coincide with communications.

In this paper we consider two implementations of CA modeling, based on our algorithm [5, 6]. Then we bring the results of the experiments on various types of problems. Finally, detailed description of SAS+ use is presented.
2. TWO IMPLEMENTATION WAYS FOR CA MODELING

A tile is an oblique parallelepiped whose boundaries are not parallel to the space-time axis.

They split the 3D-space to parallelepipeds (1):

\[
\begin{align*}
ak < x + y + 2z & \leq a(k + 1) \\
bl < x - y + 2z & \leq b(l + 1) \\
cm & \leq -2x + 2z < c(m + 1)
\end{align*}
\]  

(1)

A tile is a set of integral points \((x, y, z)\) where \(z \geq 0\), that satisfies conditions (1) and is defined by three parameters \(k, l, m\). The parameters \(a, b, c\) may be used for managing tiles volume. In \(\text{SAS}+\) we put

\[a = b = 2c = n\]  

(2)

thus reducing the number of controlling parameters to one parameter \(n\) that defines the tiles volume as \(O(n^2)\). Then the volume of transmitted data is \(O(n^2)\).

The management overhead time compared with transitions’ computation time of CA is \(O(n^{-1})\) if the volume of the tile is \(O(n^3)\).

Implementations of 2D CA modeling are performed such that one of processors is used as a host processor which manages the process. Other processors are slaves which carry out the host’s orders. The algorithm applies the dynamic data driven mapping of tiles to processors, that is realized by the host processor. Each slave processor fulfills transitions of all cells of the scheduled tile. The algorithm generates new tiles and annihilates the executed ones. The algorithm selects and disregards the stable tiles, consisting of stable states, even if they are surrounded by nonstable tiles. Two versions of mentioned implementations are described below.

Version 1 of \(\text{SAS}+\) : All computational resources can be held in the personal store of host processor. In this case all slaves send the data to the host and receive them from the host. This version is recommended for the modeling cellular automata when the required memory is less than the memory of host processor. In this case slaves exchange the data via host processor.

Version 2 of \(\text{SAS}+\) : Computational resources are distributed among personal stores of slave processors. In this case slaves exchange the data directly and the memory of the process is the sum of memories of slave processors.

3. EXPERIMENTS

We carry out the modeling of a series of cellular automata for the purpose of definition of optimal parameter \(n\) for our model for both Versions. Besides our experiments show the measure of overheads for both Versions.

The parallel application runs on the cluster of PCs connected by Myrinet [11]. 6 parallel processors of the cluster are used.

The experiments were carried out for the following well-known problems:

1. Forest-fire model of Drossel and Shwabl [7]. The example of constant configurations. All configurations including initial configuration are lattices \(900 \times 900\). 100 configurations were computed in both versions.

<table>
<thead>
<tr>
<th>Table 1. Fire</th>
<th>Version 1 in seconds</th>
<th>Version 2 in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n=10)</td>
<td>24003</td>
<td>1341</td>
</tr>
<tr>
<td>(n=15)</td>
<td>7553</td>
<td>791</td>
</tr>
<tr>
<td>(n=25)</td>
<td>1715</td>
<td>934</td>
</tr>
<tr>
<td>(n=30)</td>
<td>1183</td>
<td>1215</td>
</tr>
<tr>
<td>(n=50)</td>
<td>260</td>
<td>3284</td>
</tr>
</tbody>
</table>

2. A problem of Prisoner’s dilemma [9]. The example of 2\(^{th}\) states of the cells. All configurations including initial configuration are lattices \(900 \times 900\). 100 configurations were computed in both versions.

<table>
<thead>
<tr>
<th>Table 2. Prisoner’s dilemma</th>
<th>Version 1 in seconds</th>
<th>Version 2 in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n=10)</td>
<td>24842</td>
<td>2207</td>
</tr>
<tr>
<td>(n=15)</td>
<td>8025</td>
<td>1037</td>
</tr>
<tr>
<td>(n=25)</td>
<td>1917</td>
<td>1003</td>
</tr>
<tr>
<td>(n=30)</td>
<td>1384</td>
<td>1283</td>
</tr>
<tr>
<td>(n=50)</td>
<td>368</td>
<td>4517</td>
</tr>
</tbody>
</table>

3. A problem with identical transition function - the measurement of management overhead time. The computations were carried out on constant \(900 \times 900\) lattice, 100 configurations were computed in both versions.

<table>
<thead>
<tr>
<th>Table 3. Identical</th>
<th>Version 1 in seconds</th>
<th>Version 2 in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n=10)</td>
<td>23970</td>
<td>1298</td>
</tr>
<tr>
<td>(n=15)</td>
<td>7639</td>
<td>758</td>
</tr>
<tr>
<td>(n=25)</td>
<td>1712</td>
<td>916</td>
</tr>
<tr>
<td>(n=30)</td>
<td>1181</td>
<td>1204</td>
</tr>
<tr>
<td>(n=50)</td>
<td>268</td>
<td>3393</td>
</tr>
</tbody>
</table>

4. A problem with heterogenous transition function on \(900 \times 900\) constant lattice. 100 configurations were computed in both versions.

<table>
<thead>
<tr>
<th>Table 4. Heterogeneous</th>
<th>Version 1 in seconds</th>
<th>Version 2 in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n=10)</td>
<td>24655</td>
<td>1394</td>
</tr>
<tr>
<td>(n=15)</td>
<td>7913</td>
<td>828</td>
</tr>
<tr>
<td>(n=25)</td>
<td>1819</td>
<td>1009</td>
</tr>
<tr>
<td>(n=30)</td>
<td>1272</td>
<td>1319</td>
</tr>
<tr>
<td>(n=50)</td>
<td>469</td>
<td>3578</td>
</tr>
</tbody>
</table>

5. Bak-Tang-Wiesenfeld sandpile[1]. The example of growing configurations. The initial configuration consists of only one cell. 5000 grains dropped on the center of lattice. After 5000 steps the final configuration consists of \(1000 \times 1000\) cells.

<table>
<thead>
<tr>
<th>Table 5. Sandpile</th>
<th>Version 1 in seconds</th>
<th>Version 2 in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n=10)</td>
<td>35</td>
<td>24</td>
</tr>
<tr>
<td>(n=15)</td>
<td>55</td>
<td>45</td>
</tr>
<tr>
<td>(n=25)</td>
<td>135</td>
<td>125</td>
</tr>
<tr>
<td>(n=30)</td>
<td>195</td>
<td>186</td>
</tr>
<tr>
<td>(n=50)</td>
<td>539</td>
<td>524</td>
</tr>
</tbody>
</table>
6. Langton’s Ant Journey\cite{8}: The example of an irregular resized configurations. 5000 configurations were computed in both versions.

<table>
<thead>
<tr>
<th>n</th>
<th>Version 1 in seconds</th>
<th>Version 2 in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>25</td>
<td>111</td>
<td>114</td>
</tr>
<tr>
<td>30</td>
<td>165</td>
<td>170</td>
</tr>
<tr>
<td>50</td>
<td>500</td>
<td>509</td>
</tr>
</tbody>
</table>

The experiments show:

- If sizes of configurations are constant, then the Version 1 with parameter \( n = 50 \) is preferable.
- If configurations are resized at the time of computations, then the Version 2 with parameter \( n = 20 \) is preferable.
- The managements overhead time constitutes the most part of computation time.

Version 1 is chosen as a template of 2D cellular automata modeling way for SAS+ automatic system.

### 4. HOW TO USE SAS+?

The software tool SAS+ is intended for imitation both one-dimensional (1D-CA) and two-dimensional (2D-CA) on a homogeneous computational cluster (ArmCluster). The choice of 1D-CA or 2D-CA is realized by typing “a” or “b” when running “SAS+” in command line.

Detailed explanation of the simulator of the 1D-CA is given in [3, 4]. Therefore we present here only 2D-CA modeling.

To use SAS+ the user must have some knowledge of basic concepts of the programming language C, as well some skills in setting CA components.

To model 2D-CA on SAS+ user must give the description of desirable CA. In general this description consists of

- initial configuration;
- \( Computation \) function which realizes one cell’s transition;
- number of output configurations (Configuration).

Let us consider each of these items.

**Initial configuration** is a rectangular \([\text{Row} \times \text{Column}]\) table on the plane of CA, where all non-stable cells of initial configuration are situated. The rest of the plane is supposed to be completed by stable states.

Each square \((i, j)\) of the table must contain a string of \( w \) ASCII symbols – the initial states of the cells. Initial configuration should be set by user as a text file, that includes \( w \times \text{Row} \times \text{Column} \) symbols in one line. The variables \text{Row} and \text{Column} should be set by user.

The variable \text{stable} is such a state of cell, that if all cells of neighborhood of the cell have the state \text{stable} then \text{Computation} of the next state returns \text{stable} state.

In all, the user should set for description of initial configuration the following:

- values \( w \) (number of cell’s states), Configuration, Row, Column;
- the state \text{stable} (a string with length \( w \));
- the file of initial states.

**Computation** – the program of transitions of any cell. \( Computation \) function has 12 arguments (the space-temporal coordinates of the transition and \( 9 \) states of its neighborhood) and returns new state.

The body of \text{Computation} uses C or C++ languages.

\text{Computation} should have the property: there exists a state (its name is \text{stable}) such that if all neighbors of a cell have \text{stable} state, then \text{Computation} returns the state \text{stable}.

\text{Computation} function may use the external parameters. These parameters should be set by user before starting SAS+. Those parameters are:

- arguments \text{Arg1}, \text{Arg2}, ..., \text{Arg4} of types \text{int}, \text{char}, \text{string};
- files \text{DataFile1}, ..., \text{DataFile}\text{t}.

In this connection, variables \text{SizeDataFile} gives the length of \text{DataFile}, and function \text{DatumFile}(\text{j}) returns the \text{j}-st element of \text{DataFile}.

All mentioned arguments, variables and functions can be used in \text{Computation} function. The prototype of \text{Computation} is given below.

```c
string Computation( int x, int y, int z, string s, string r, string l, string d, string dl, string u, string ur, string ul ), where
```

\( x \) is the row of current cell \((0 \leq x < \text{Row})\),
\( y \) is the column of current cell \((0 \leq y < \text{Column})\),
\( z \) is the number of configuration of current cell \((0 \leq z < \text{Configuration})\),
\( s \) is the state of current cell (a string with \( w \) length),
\( r \) is the state of current cell’s right neighbor (a string with \( w \) length),
\( l \) is the state of current cell’s left neighbor (a string with \( w \) length),
\( d \) is the state of current cell’s down neighbor (a string with \( w \) length),
\( dl \) is the state of current cell’s down-right neighbor (a string with \( w \) length),
\( u \) is the state of current cell’s upper neighbor (a string with \( w \) length),
\( ur \) is the state of current cell’s upper-right neighbor (a string with \( w \) length),
\( ul \) is the state of current cell’s upper-left neighbor (a string with \( w \) length).

This function describes the transition of each cell in the cellular automaton.

This completes the entry procedure of the cellular automaton. During the entry procedure the user is allowed to edit and correct the description components of the cellular automaton. At this stage a corresponding message is sent and it is offered to reenter valid data in case if errors of various types are detected. The user can terminate the work of the program by inputting the symbol ‘q’.

The second stage of the software tools functioning is per-
formed without any participation of the user. The whole input data are classified and recorded in a temporary file. Based on the contest of this file, all variables and functions mentioned above are declared and defined in programming language C++. The programming code for cellular automaton described at the input of the program is constructed also using the same programming language.

As a result of SAS+ execution, in user’s domain of the operating system Linux, a folder having the same name as the programming module, is created, which contains the programming module and the "readme.txt" file. In the mentioned text file the user may find some useful information on the programming module, on running of cluster-based programming module, as well as on input data of the program.

An example of input information of the SAS+ for the sandpile problem [1] is presented below: the initial configuration is a text file with '1' non-stable symbol.

\[ w = 1 \]
\[ \text{Row} = 1 \]
\[ \text{Column} = 1 \]
\[ \text{Configuration} = 5000 \]
\[ \text{stable} = "0". \]

string Computation( int x, int y, int z, string s, string r, string l, string d, string dr, string dl, string u, string ur, string ul)
{
if ( strcmp ( r.c_str(), stable ) == 0 &&
   strcmp ( l.c_str(), stable ) == 0 &&
   strcmp ( d.c_str(), stable ) == 0 &&
   strcmp ( dr.c_str(), stable ) == 0 &&
   strcmp ( dl.c_str(), stable ) == 0 &&
   strcmp ( u.c_str(), stable ) == 0 &&
   strcmp ( ur.c_str(), stable ) == 0 &&
   strcmp ( ul.c_str(), stable ) == 0 )
   return string ( stable );
if ( s[ 0 ] \geq \text{‘}4\text{‘} ) s[ 0 ] -= 4;
if ( (x == 0) && (y == 0) ) s[ 0 ]++; 
if ( u[ 0 ] \geq \text{‘}4\text{‘} ) s[ 0 ]++; 
if ( d[ 0 ] \geq \text{‘}4\text{‘} ) s[ 0 ]++; 
if ( l[ 0 ] \geq \text{‘}4\text{‘} ) s[ 0 ]++; 
if ( r[ 0 ] \geq \text{‘}4\text{‘} ) s[ 0 ]++; 
return s; }

REFERENCES

Preprint N 08-03/1 IIAP NAS RA, 36 pages, Yerevan, 2008. (in Russian)