Refining cellular automata with routing constraints

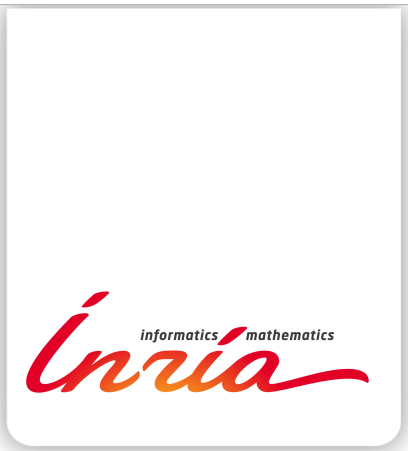
Jean-Vivien Millo, Robert de Simone

Refining cellular automata with routing constraints

Jean-Vivien Millo, Robert de Simone

RESEARCH REPORT
N° 8051
August 2012
Project-Teams AOSTE

ISSN 0249-6399 ISRN INRIA/RR--8051--FR+ENG



Refining cellular automata with routing constraints

Jean-Vivien Millo, Robert de Simone

Project-Teams AOSTE

Research Report n° 8051 — August 2012 — 12 pages

Abstract: A cellular automaton (CA) is an infinite array of cells, each containing the same automaton. The dynamics of a CA is distributed over the cells where each computes its next state as a function of the previous states of its neighborhood. Thus, the transmission of such states between neighbors is considered as feasible directly, in no time.
When considering the implementation of a cellular automaton on a many-cores System-on-Chip (SoC), this state transmission is no longer abstract and instantaneous, but has to follow the interconnection medium of the SoC. It is usually a grid or a mesh matching the underlying topology of the CA but finite. In order to consider such constraints at a higher level, we propose a refinement of the classical model of CA where the topology is considered as the communication medium.
If the state of a cell depends on its neighbors up to a certain distance, then a given state must be broadcasted to all its neighbors at the same distance, as they all require it to compute their next state. It means routing and duplicating the state in the topology. We study the routing patterns needed to efficiently implement such state broadcasting algorithm. We provide a solution by which each router can locally predict where to redirect the states to correctly and efficiently implement this broadcasting algorithm.

Key-words: Cellular Automata, Synchronous, Refinement, Routing

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Refining cellular automata with routing constraints

Résumé : Un automate cellulaire (AC) est un tableau infini de cellules, chacune contenant le même automate. La dynamique d'un AC est distribuée entre les cellules, chacune calcule son prochain état comme une fonction des états de ses voisins. Donc, la transmission d'un état entre deux cellules est donc considérée comme faisable directement et instantanément.

Quand on s'intéresse à l'implémentation d'un AC sur un système sur puce à plusieurs cores, on ne peut plus considérer la transmission d'un état comme une action abstraite et instantanée. Cette transmission doit suivre le medium d'interconnexion du système sur puce. Ce dernier est habituellement une grille ou un mesh (grille dans laquelle les extrémités opposées sont connectées) correspondant à la topologie logique de l'AC mais finie. Afin de prendre en compte la notion de medium d'interconnexion à un niveau d'abstraction supérieur, par rapport à l'implémentation, nous proposons un raffinement du modèle classique des AC dans lequel la topologie est considérée comme le medium d'interconnexion.

Si l'état d'une cellule dépend de son entourage jusqu'à une certaine distance, alors cet état doit être diffusé à tous les voisins jusqu'à cette même distance puis ce que chacun d'eux en a besoin pour calculer son nouvel état. Cela signifie router et diffuser l'état en question dans la topologie. Nous étudions le schéma de routage nécessaire pour mettre en oeuvre efficacement cet algorithme de diffusion d'état. Dans cette solution, chaque router peut localement prédire où envoyer les états en transit afin de garantir la justesse de l'algorithme.

Mots-clés : Automate Cellulaire, Synchrone, Raffinement, Routage

1 Introduction

A cellular automaton (CA) is a multidimensional infinite array of cells, each containing an identical automaton. The dynamics of a CA is based on the fact that each automaton synchronously computes its next state depending on its current one as well as the current states of a selected set of neighbors.

Communications (and communication feasibility) is not a real concern in the modeling framework. One supposes that instantaneous pointwise communication is available between any two cells in such neighborhood on demand. We shall challenge this assumption here and consider a refinement where communications have to be routed according to an interconnection topology. In the case of CA, a natural interconnection topology consists of the underlying topology of the CA itself. As far as we know, there is no existing work in that direction.

The coupling of a model of computation with communication constraints is commonplace in modern computer science. In the case of CA, it allows to consider efficient implementations on multi-processors Systems-on-Chip (MP-SoC) equipped with aspecific mesh interconnection structure such as a Network-on-Chip (NoC). We consider here CA that can be seen as the infinite unfolding of a spatial periodic states structure. Such CA can be simulated/implemented on a finite-size torus.

A given cell of the CA is thus subdivised into a router component and a computation cell that contains the original automaton. While the computation cell is tightly linked to its local router (by dedicated input/output links), the router is used to 1) inject a datum (which represent the current state) from the computation cell into the network, 2) route the passing data, or 3) duplicate a datum from the network to the computation cell. For example, it pushes a datum further north from a south port.

We shall here consider only the 2-dimensional case for sake of simplicity. We shall also consider that the router has a multicasting capacity. For example, a datum from the computation cell can be sent in many directions if it is relevant to several opposite locations.

Our present goal shall be to consider how static routing schemes can be defined and made up for frequently encountered communication patterns in CA dynamics. We shall in particular use the example of cells requiring all previous states from their Moore neighborhood of radius n or less ($||\vec{d}||_\infty \leq n$ where d is the offset vector of the neighbor). Many applications such as the game of life [4], the class of stencil applications [3], or the Jacobi numerical method [7] match with this neighborhood dependency scheme. One can then hope to discover regular patterns through which global communication behavior can be exposed, while the various data flows combine harmoniously into global traffic.

We will study the case where at some point in time, every cell sends synchronously its current local state (as a datum) on the network. This case can be seen as the worst communication case where the network is saturated. We propose the Neighborhood Broadcasting Algorithm (NBA) to resolve the situation and we study its behavior. Our analysis focuses on two aspects: 1) Performance analysis: the timing estimation of the NBA execution and the size of the interconnection channels (between routers) required to ensure conservation of data. 2) the discovery of regular routing patterns toward the goal of expressing the *propagation rule* (routing directives in the routers) statically once for all the routers and once for all iterations.

2 Cellular automata model

The basic definitions related to CA are imported from Jarkko Kari survey [6]. For instance, we consider synchronous CA and the underlying topology is an infinite rectangular grid of dimension d. Each cell has the same finite set of state S. The configuration of the automaton is a mapping

$c : \mathbb{Z}^d \rightarrow S$ where each cell is identified by d coordinates. The evolution of a CA occurs synchronously on discrete time steps. All cells are updated simultaneously.

Let c_a and c_b be two cells of the CA identified by their vector of coordinates respectively $\vec{c_a}$ and $\vec{c_b}$ (in the sequel, when it is clear from context, $\vec{c}$ will be used instead of c). Let $\vec{x} = (x_1, x_2, ..., x_d)$ be the difference between $\vec{c_a}$ and $\vec{c_b}$ ($\vec{x} = \vec{c_a} - \vec{c_b}$).

The distance relative to the i^{th} dimension (or coordinate) is $|x_i|$ which is the value of the vector $\vec{x}$ at the index i.

The Manhattan distance (or norm) between $\vec{c_a}$ and $\vec{c_b}$ is denoted $||\vec{x}||_1$ and is equals to

$$||\vec{x}||_1 = |x_1| + |x_2| + \cdots + |x_d|$$

The Moore distance (or norm) between $\vec{c_a}$ and $\vec{c_b}$ is denoted $||\vec{x}||_\infty$ and is equals to

$$||\vec{x}||_\infty = max(|x_1|, |x_2|, \cdots, |x_d|)$$

We define the neighborhood of a cell based on the Moore distance. $N^n(c)$ is the neighborhood of the cell c up to a radius n (c is in its own neighborhood).

$$N^n(c) = \{c_a \texttt{ such that } ||\vec{c} - \vec{c_a}||_\infty \leq n\}$$

2.1 Cellular automata with routing management

We shall now consider that the transmission of a datum between cells is not instantaneous anymore. When a cell sends a datum (that represents its current state) to one of its neighbors, the datum has to travel through the underlying topology and is subject to travel durations, channel capacity limitations, and routing decisions.

In the classical model of CA, when the global clock ticks, 1) every cell sends simultaneously a datum on the network, and so every cell receives data from its neighborhood. 2) every cell computes its new state according to the update rule. Step 1 is not instantaneous, it requires many micro-steps into which every datum is propagated to all its neighbors in $N^n(c)$. It takes at least n instants to go at a Manhattan distance n. When many data travel on the same branch of the network, they are queued in FIFO channels and processed one-by-one. Thus the global clock is refined in a *communication clock* where every step of the global clock corresponds to a finite sequence of micro-steps.

In order to manage the communication, the cell is augmented with routing capacities. The original finite state machine become the computation cell. It is complemented with a communication cell called “router" which interfaces the computation cell with the network. The router and the computation cell are strongly connected through dedicated channels.

Each router has four connections per dimensions. It is connected through an input and an output channels to the left and right (or up and down) neighbors. The generic behavior of a router is presented in Algorithm 1. At each micro-step, the router consumes a datum on one of its input channels and propagates it to one or many of its output channels (including the local channel to the computation cell) according to an internal *propagation rule*. One could also define a *feeding rule* to determine which input channels are considered micro-step after micro-step. It is allowed to considered many input channels simultaneously but only if the propagation rule guaranties that two data (or more) cannot be propagated on the same output channel at the same micro-step. Lastly, when a datum is propagated by a router, it can be considered by the direct neighbor only at the next micro-step. Propagation and feeding rules are data independent similarly to a cell of a Kahn process network [5].

Figure 1 presents a 2-dimensional CA where each cell is divided in a computation cell plus a router.

Algorithm 1 describes the generic behavior of a router as a reactive process

```
while true do
  in_ports = feeding_rule(){decide which ports to read}
  for all in_port ∈ in_ports do
    data = in_port.read()
    destinations = propagation_rule(data){decide where to propagate data}
    for all out_port ∈ destinations do
      out_port.write(data)
    end for
  end for
  pause{wait for the next micro-step}
end while
```

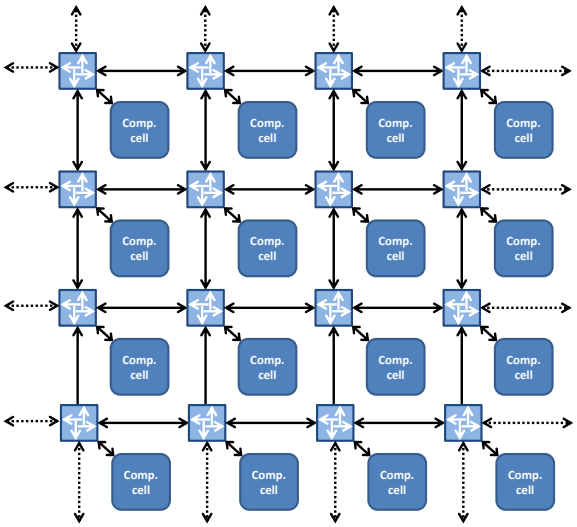


Figure 1: A 2-dimensional CA with bidirectional channels.

3 Neighborhood Broadcasting Algorithm (NBA)

In the game of life [4], the state of a cell depends upon the previous states of its direct neighborhood. However there is some applications such as the class of *stencil computation* [2] where not only the direct neighborhood are concerned but also some indirect neighbours matching a *stencil*. In the worst case, every cell needs to know the state of all its neighborhoods at a radius n. This operation requires a consequent exchange of data which would generate congestions on the network.

NBA is a 2-dimensional propagation algorithm to simultaneously send a datum from every single cell to all their neighborhoods (up to a given radius). Thanks to the multicasting ability of the routers, a single datum is emitted once by its source cell and then broadcasted to all the other cells through the network. Informally, at the first micro-step, a router receives a datum on its local port. The datum is cloned and propagated on every port. Then, at the next micro-steps, the neighbours multicast the data straight, left, and to the local port so that after i micro-steps, all the routers at a Manhattan distance i from the source receive the data. The propagation edge forms a diamond shape as one can see on the first three pictures of Figure 2.

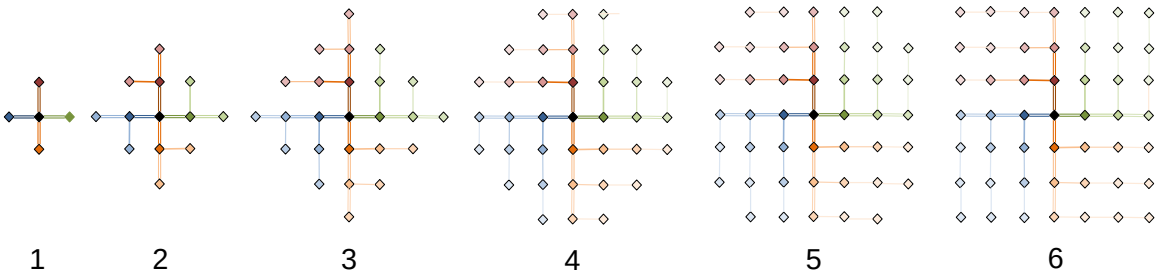


Figure 2: The broadcasting pattern from a single node.

The path from a given source to a given destination always follows one coordinate direction fully, then the other one. The data always turn left at this change of directions, i.e. counter-clockwise. When a datum reaches a Manhattan distance n, the router that receives it sends it to its local port and left but does not propagate the data straight further.

On the third picture of Figure 2, the top-most, bottom-most, left-most, and right-most routers are at a Manhattan distance $n(= 3)$ from the source. So on the fourth picture, the data are not propagated straight but left.

As a result, the northbound data spawns duplicated data that travel West (left of North) to reach all destination cells in the North-West quadrant. Similarly, the westbound data spawns duplicated data to the South-West quadrant, and so on.

The last thing to notice is that data traffics originating from North and South (resp. from East and West) never interfere in the same router. Northbound data only use up and leftward channels, southbound ones the down and rightward channels instead. So data processing along vertical directions (resp. horizontal) can be parallelized in the same micro-step. So the feeding rule is to alternate vertical and horizontal input channels.

3.1 Router

NBA is distributed amongst routers. It consists in a set of routing directives (the propagation rule) that every router follows. The resulting global behavior will be the one presented in Figure 2 but for every cell simultaneously.

Figure 3 presents the only four propagation patterns among all (modulo rotation) that are used to realize the NBA. In these four patterns, the datum is sent to the local port. In B and D, it is sent straight, lastly, in C and D it is multicasted on the left branch.

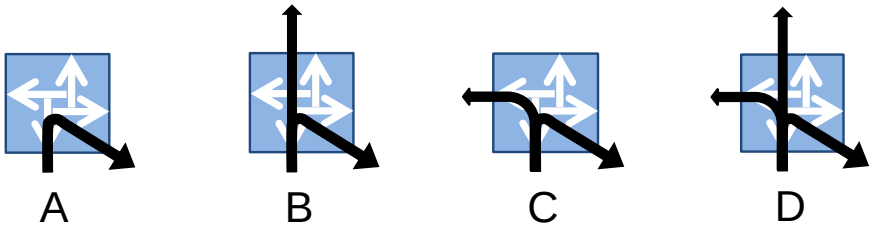


Figure 3: The three data propagation patterns for the all-to-all propagation algorithm.

As we have seen, a datum first travels on a direction fully and then turn left and travel on the secondary direction until the destination. When a datum has been emitted on vertical

ports (resp. horizontal ports), the main direction/coordinate is x (resp. y) and the secondary direction/coordinate is y (resp. x). The radius is denoted n.

The routing directives are the following:

1. At the first micro-step, a router broadcasts the datum from its local port in every direction,
2. At the even micro-steps, the North and South input ports are processed.
   At the odd micro-steps, the East and West input ports are processed.
   (a) If the data comes from a straight neighbour (the secondary coordinate equals 0) :
      i. If the data comes from a distance n relative to the main direction, it follows Pattern C of Figure 3.
      ii. If the data comes from less than a distance n relative to the main direction, it follows the Pattern D of Figure 3.
   (b) If the data does not come from a straight neighbour :
      i. If the data comes from a distance n relative to the secondary direction, it follows the Pattern A of Figure 3.
      ii. If the data comes from less than a distance n relative to the secondary direction, it follows the Pattern B of Figure 3.

4 Running the NBA

4.1 Performance analysis

NBA has been designed to be as fast as possible. For a radius n, NBA execution takes $2*n*(n+1)$ micro-steps to complete. Every port of every router is in charge of a quadrant of size $n \times n + 1$. In Figure 2 the radius is 3 and the quadrant contains 12 cells. A port has to deal with one datum from each cell of its quadrant, one at each micro-step. Since horizontal (East, West) ports are sequential with vertical (North, South) ports, the completion time for a router is twice the time than for a single port. Finally, every router acts and terminates simultaneously so the completion time of a router is the one of the NBA.

The capacity of the interconnection channels are derived from the following behavioral analysis. A capacity n for every channel is required. At the i^th stage of the execution, data from a Manhattan distance i are in the input channel of the router (see below for details). When $i = n$, the maximum is reached.

4.2 Step-by-step execution

Let us focus on the traffic generated at the input port of the routers while running NBA. First, it appears that the data arrive in the order of their Manhattan distance from their source. However, all the data with the same Manhattan distance does not come in the same order if we consider the vertical ports or the horizontal ports.

The evolution of the traffic can be divided in stages. The i^{st} stage starts when the input channel of the studied port contains the data from a Manhattan distance i. The stage ends when all these data have been routed. As a indirect consequence, the channel contains at the end of the i^{st} stage, the data from a Manhattan distance $i + 1$ but nothing else. Figure 4 shows which data are present in the channel at the beginning of each stage. Figure 4 focuses on the right top corner of a neighborhood with a raduis 5 ($N^5(r)$).

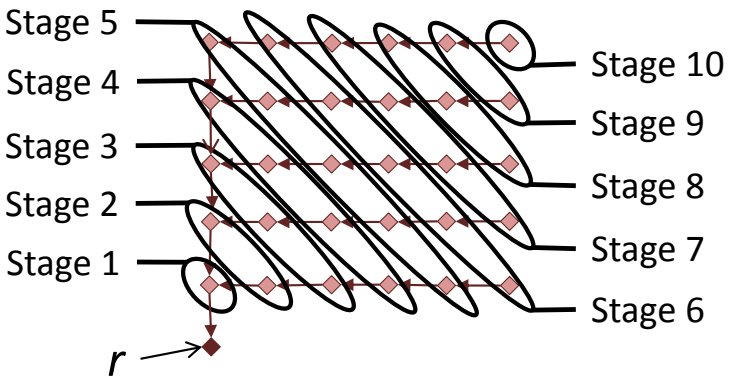


Figure 4: The origin of the data in input of the North port of a router r stages after stages.

At the first stage, every router broadcasts the datum from their local port in every direction. Consequently, every router gets the data from its direct neighbours in each of its input port.

At the second stage, every router multicasts these data straight, left and to the local port because they come from a straight neighbour. At the end of the second stage, every router gets two data in each of its input port.

At the third stage, the first data in the channel has to be multicasted because it comes from a straight neighbour but the second should not. So it generates three data in each input port.

At stage n (Step 5 in Figure 4), each router receives a data from a straight neighbour at a Manhattan distance n. This data is propagated left but not straight. It correspond to the case (2.-(a)-i.) of the routing directives (Section 3).

At stage $n+1$ onward, each router receives some data coming from the neighbours at a distance n relative to the main direction. These data are not propagated but only sent to the local port. However, the routers also receive some data that have to be forwarded straight but none of them is multicasted on the left any more.

In Figure 4, the six routers on the upper horizontal line are at a distance n from r relative to the main direction. The left most one is a straight neighbour.

$$North(r) = \left[\begin{pmatrix}\mathbf{0}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}\mathbf{0}\\\mathbf{2}\end{pmatrix} \begin{pmatrix}1\\1\end{pmatrix} \begin{pmatrix}\mathbf{0}\\\mathbf{3}\end{pmatrix} \begin{pmatrix}1\\2\end{pmatrix} \begin{pmatrix}2\\1\end{pmatrix} \begin{pmatrix}\mathbf{0}\\\mathbf{4}\end{pmatrix} \begin{pmatrix}1\\3\end{pmatrix} \begin{pmatrix}2\\2\end{pmatrix} \begin{pmatrix}3\\1\end{pmatrix} \begin{pmatrix}\mathbf{0}\\\mathbf{5}\end{pmatrix} \begin{pmatrix}1\\4\end{pmatrix} \begin{pmatrix}2\\3\end{pmatrix} \begin{pmatrix}3\\2\end{pmatrix} \begin{pmatrix}4\\1\end{pmatrix} \right.$$
$$\left. \begin{pmatrix}\mathbf{1}\\\mathbf{5}\end{pmatrix} \begin{pmatrix}2\\4\end{pmatrix} \begin{pmatrix}3\\3\end{pmatrix} \begin{pmatrix}4\\2\end{pmatrix} \begin{pmatrix}5\\1\end{pmatrix} \begin{pmatrix}\mathbf{2}\\\mathbf{5}\end{pmatrix} \begin{pmatrix}3\\4\end{pmatrix} \begin{pmatrix}4\\3\end{pmatrix} \begin{pmatrix}5\\2\end{pmatrix} \begin{pmatrix}\mathbf{3}\\\mathbf{5}\end{pmatrix} \begin{pmatrix}4\\4\end{pmatrix} \begin{pmatrix}5\\3\end{pmatrix} \begin{pmatrix}\mathbf{4}\\\mathbf{5}\end{pmatrix} \begin{pmatrix}5\\4\end{pmatrix} \begin{pmatrix}\mathbf{5}\\\mathbf{5}\end{pmatrix} \right]$$

$$West(r) = \left[\begin{pmatrix}\mathbf{0}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}\mathbf{1}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}0\\2\end{pmatrix} \begin{pmatrix}\mathbf{2}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}1\\2\end{pmatrix} \begin{pmatrix}0\\3\end{pmatrix} \begin{pmatrix}\mathbf{3}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}2\\2\end{pmatrix} \begin{pmatrix}1\\3\end{pmatrix} \begin{pmatrix}0\\4\end{pmatrix} \begin{pmatrix}\mathbf{4}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}3\\2\end{pmatrix} \begin{pmatrix}2\\3\end{pmatrix} \begin{pmatrix}1\\4\end{pmatrix} \begin{pmatrix}0\\5\end{pmatrix} \right.$$
$$\left. \begin{pmatrix}\mathbf{5}\\\mathbf{1}\end{pmatrix} \begin{pmatrix}4\\2\end{pmatrix} \begin{pmatrix}3\\3\end{pmatrix} \begin{pmatrix}2\\4\end{pmatrix} \begin{pmatrix}1\\5\end{pmatrix} \begin{pmatrix}\mathbf{5}\\\mathbf{2}\end{pmatrix} \begin{pmatrix}4\\3\end{pmatrix} \begin{pmatrix}3\\4\end{pmatrix} \begin{pmatrix}2\\5\end{pmatrix} \begin{pmatrix}\mathbf{5}\\\mathbf{3}\end{pmatrix} \begin{pmatrix}4\\4\end{pmatrix} \begin{pmatrix}3\\5\end{pmatrix} \begin{pmatrix}\mathbf{5}\\\mathbf{4}\end{pmatrix} \begin{pmatrix}4\\5\end{pmatrix} \begin{pmatrix}\mathbf{5}\\\mathbf{5}\end{pmatrix} \right]$$

Table 1: $North(r)$ and $West(r)$ for a neighborhood of radius 5.

Formally, let n be the radius of the neighborhood and r be a router. We give to r the coordinate $\begin{pmatrix}0\\0\end{pmatrix}$. The coordinates iterate positively toward resp. the North and East. Let

$North(r)$ be the ordered list of data going through the FIFO channel in input of the North port of r. $North(r) = \left[\begin{pmatrix} x \\ y \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} ... \right]$ where $\begin{pmatrix} x \\ y \end{pmatrix}$ are the coordinates of the source of the first data entered in the channel and $\begin{pmatrix} x' \\ y' \end{pmatrix}$ are the coordinates of the second. Initially, $North(r) = [\emptyset]$. $South(r)$, $East(r)$, and $West(r)$ are defined similarly to $North(r)$. $North(r)$ is given by Table 1.

$South(r)$ is the same as $North(r)$ except that the coordinates of the origins are the exact opposite of the ones in $North(r)$. Formally $\forall \begin{pmatrix} x \\ y \end{pmatrix} \in North(r)$ at position i, $\exists \begin{pmatrix} -x \\ -y \end{pmatrix} \in South(r)$ at position i. $East(r)$ can be computed from $West(r)$ with the same transformation. Table 1 also presents $West(r)$. Here we choose to put the main coordinate (i.e. y) on top to enhance the similitude with $North(r)$. For the same reason, the second value is $-x$ instead of x.

The order of arrival of the data in $West(r)$ inside a stage is the exact opposite of $North(r)$. For instance, $\begin{pmatrix} 0 \\ 2 \end{pmatrix}$ followed by $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ in the second and third position of $North(r)$ but $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ followed by $\begin{pmatrix} 0 \\ 2 \end{pmatrix}$ in the second and third position of $West(r)$. This order inversion occurs because the feeding rule of the routers gives priority to the vertical ports at the expense of the horizontal ones.

4.3 Static scheduling of router

Table 1 gives the status of the channel in input of a router step by step. Therefore, the behavior of the routers is predictable and can be computed statically. The following algorithm generates the sequence of Table 1 but for any radius n. A similar algorithm can be written for the other ports of the router.

Algorithm 2 computes $North(r)$ for any given router r and radius n

Input : a radius n.
Output : $North(r)$ for any router r.
$North(r) =$
for sum **from** 1 **to** $2n$ **do**
  for x **from** $(sum <= n)?0 : sum - n$ **to** $j < sum$ and $j \leq n$ **do**
    $North(r).append(\begin{pmatrix} x \\ sum - x \end{pmatrix})$
  end for
end for
return $North(r)$

From the sequence generated with Algorithm 2, the propagation rule can be symbolically applied on any given ordered list of data such as $North(r)$ and one can generate the sequence of decisions that every router shall take to correctly route such data. Figure 5 gives the internal crossbar of a router for a radius 5. The crossbar connects the inputs on the left side of the figure to the outputs on the right side. The connections to the computation cell are at the bottom.

From left to right, the first demultiplexer says when a data has to be forwarded straight. The sequence of decisions is given as a binary word. Concerning the data from the North, the first tenth data are forwarded, the eleventh is not, and so on. The second demultiplexer says when

a data should be forwarded left. Concerning the data from the West, the first one comes from a straight neighbour and thus is forwarded. The second is not and the third is. See Table 1 to match the complete sequence.

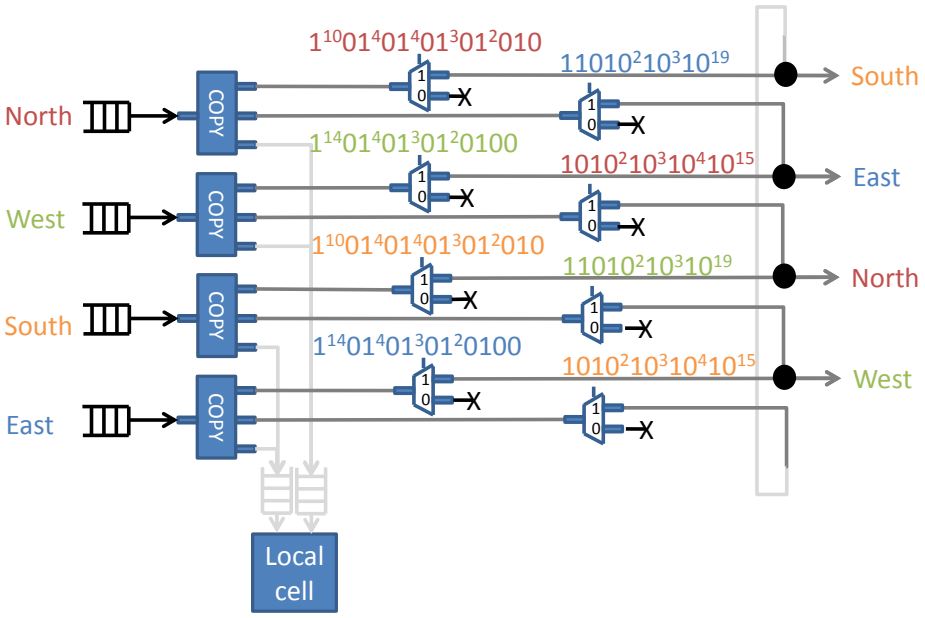


Figure 5: The internal crossbar of the router with the switching conditions.

5 Experimental results

We have implemented the NBA in SystemC [1]. Our implementation is composed of:

- a module *Cell* which implements the structure and the behavior of a computation cell. Initially, it sends its state and then collects the states from the other cells in its neighborhood. When it has got all the data, it computes its new state and, at the next micro-step, re-sends its state.

- a module *Router* which implements the structure and the behavior of a router, namely the NBA.

- a module *CA* which is a torus with a parametric size composed of cells (router + computation cell).

The propagation rule has been implemented in two ways. First, we have implemented the dynamic routing directives given in Section 3. Secondly, these routing directives has been replaced by the static patterns given in Figure 5. The overall behavior of the system is the same whatever option has been selected.

This experiment consolidates the fact that the propagation rule for a given application can be computed analytically upfront as static decisions and then inserted into the routers.

Stencil applications We have also extended the experiments to the more general case of stencil applications [2, 3]. Stencil applications are a class of distributed applications where the new state of a cell depends of the current state of some of its neighbours. The map which says which neighbours are concerned and which others are not is called a stencil. Stencil applications are massively used in scientific computation.

In our implemetation, the stencil is given as a odd square matrix of booleans where the middle entry represents the concerned node. The other entries represent its neighborhood. The boolean says whether the state of the corresponding neighbor is required or not. Every router is aware of the stencil and routes the data with respect to their origin.

We ran many case studies where each stencil is characterized by the number of neighbors it involves and the moore distance to the farest neighbor. Table 2 relates the number of micro-steps required to route all the data with respect to the different cases (The number of neighbors in each stencil is noted N).

$\backslash$	$N=4$	$N=8$	$N=16$	$N=24$	$N=32$	$N=48$
$\|\|\vec{x}\|\|_\infty \leq 1$	2	4	-	-	-	-
$\|\|\vec{x}\|\|_\infty \leq 2$	4	4	12	16	-	-
$\|\|\vec{x}\|\|_\infty \leq 3$	6	8	18	18	18	24

Table 2: Experimental results of routing duration for stencil applications

Again, whatever is the shape of the stencil, the dynamic routing conditions can be replaced by static pattern which are the same in every router. Similarly to the NBA, a pattern represents the sequence of decisions to route the data required for the current computation step. The next computation step of the stencil application repeats the same behavior.

6 Conclusion

In the classical model of Cellular Automata (CA), the communications between cells are assumed to be instantaneous. However if we consider the possible implementation of a 2-dimension CA on a many-cores system-on-chip, the communications take time. In order to prepare such implementation, this paper has presented a refinement of the classical model of CA where the exchance of data between cells is time-consuming. The focus is given on routing the data through the communication topology.

Every step of the global clock (the one of the classical synchronous CA) is refined as a sequence of micro-steps during which the data from each cell travels through the underlying topology toward its destination. To ensure the routing of data, each cell is augmented with routing capabilities and so is divided in the original computation cell plus a router.

In order to compute the next state of the cells when it depends upon all the neighboors up to a Moore distance n, we have proposed the Neighborhood Broadcasting Algorithm (NBA) to simultaneously broadcast the state of every cell to their respective neighborhoods. We have studied the NBA and deducted that the propagation rules in every router can be expressed as either dynamic on-line or static off-line decisions.

To go further in this scope, we would like to challenge some of the assumptions we took initially. First, our approach is limited to CA that can be seen as infinite unfolding of a spatial periodic state structure. Our approach considers that the size of a tile (a spatial periodic state structure) is less or equals to the size of the the Network-on-Chip (NoC). However, NoC size are often limited to 64 up to 128 nodes but tiles can be much bigger. By extention, we consider that the physical topology (of the NoC) matches with the topology of the CA. One could consider

the mapping of a n-dimension CA on a m dimension NoC. Lastly, we consider only synchronous CA but one can consider the same challenge, routing data through the communication topology in the asynchronous CA.

References

[1] Accellera. System C, March 2007. http://www.accellera.org/downloads/standards/systemc.

[2] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 4:1–4:12, Piscataway, NJ, USA, 2008. IEEE Press.

[3] Kaushik Datta, Samuel Williams, Vasily Volkov, Jonathan Carter, Leonid Oliker, John Shalf, and Katherine Yelick. Auto-tuning the 27-point stencil for multicore. In *In Proc. iWAPT2009: The Fourth International Workshop on Automatic Performance Tuning*, pages –, 2009.

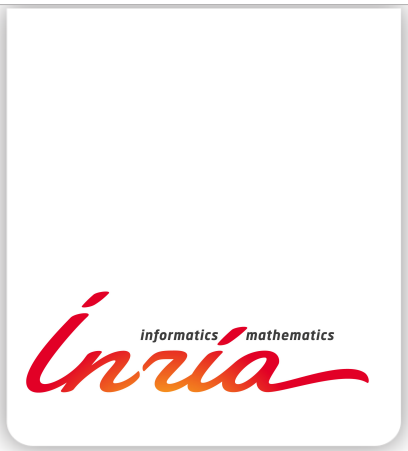
[4] Martin Gardner. The fantastic combinations of john conway's new solitaire game «life». *Scientific American*, 223:120–123, 1970.

[5] Gilles Kahn. The semantics of a simple language for parallel programming. In *Proceedings of IFIP CONGRESS 74*, pages 471–475. North-Holland publishing company, 1974.

[6] Jarkko Kari. Theory of cellular automata: A survey. *Theoretical Computer Science*, 334(1-3):3 – 33, 2005.

[7] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, third edition, 2007.

Contents



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399