Vol. 3 (2010) *Acta Physica Polonica B Proceedings Supplement* No 2

GCA-w: GLOBAL CELLULAR AUTOMATA WITH WRITE-ACCESS*

Rolf Hoffmann

Technische Universität Darmstadt, FG Rechnerarchitektur
Hochschulstr. 10, 64289 Darmstadt, Germany
hoffmann@ra.informatik.tu-darmstadt.de

(Received January 12, 2010)

The novel GCA-w model (**G**lobal **C**ellular **A**utomata with **W**rite access) is presented which is based on the GCA (Global Cellular Automata) model. The GCA model is a massively parallel model like the cellular automata model. In the CA model, the cells have static links to their local neighbors whereas in the GCA model, the links are dynamic according to a special local rule. In both models, the access is "read-only". Thereby no write conflict can occur and all cells can update their states independently in parallel. The GCA model is useful for many parallel problems that can be described by a non-local and changing neighborhood. A shortcoming of the GCA model is the missing write access to neighboring cells. Although a write access can be emulated in $O(\log n)$ time this slowdown may not be acceptable in some practical applications. Therefore, the GCA-w model was developed. The GCA-w model allows to change the states of the neighboring cells as well as the state of the own cell. Thereby certain parallel algorithms can be described more appropriately and the number of active cells can be controlled by the cells themselves in a decentralized way. Activity control also enables dynamic resource sharing and the reduction of power consumption. The usefulness of the GCA-w model is demonstrated by some fine-grain parallel applications: one-to-all communication, synchronization and moving particles.

PACS numbers: 87.17.Aa, 92.60.hk, 87.18.Hf

1. Introduction

We propose a novel massively parallel computing model [1,19], called "GCA-w" (GCA with write-access) that is an extension of the GCA (Global Cellular Automata) model [11,12,14–16] which is in turn an extension of the cellular automata (CA) model. The cells of a GCA can dynamically

---

* Presented at the Summer Solstice 2009 International Conference on Discrete Models of Complex Systems, Gdańsk, Poland, June 22–24, 2009.

establish links to their global neighbors, whereas the cells of a CA use fixed links to their local neighbors. The GCA and CA models have in common that they allow only read access to their neighbors and therefore no write conflicts can occur. Thereby the complexity of these models is kept low and implementations in software or parallel hardware can easily be accomplished.

It was already shown that the GCA model is suited for a large number of parallel problems (Jacobi iteration to solve a system of linear equations [3,9], finding the connected components of a graph [4,10], random distribution of particles with non local dynamic neighbors [5], N-body force calculation [2], sorting numbers [8], and graph algorithms [12]). Also efficient parallel hardware architectures [2,3,5,8,13] have been designed. The language GCA-L [9] was developed to simulate GCA algorithms and to use the language as an input for an automatic design process generating an application specific data parallel hardware to be configured on an FPGA.

The GCA model can also be mapped onto the PRAM-CROW model [6]. Therefore, PRAM-CROW algorithms can be executed on the GCA model with the same time complexity using the same amount of processors respectively cells.

The GCA model has two significant restrictions:

1. *No write access to the neighbors:* Although a write access can be simulated in $O(\log n)$ time [6], this slowdown might be too high for practical applications. In addition, a certain class of algorithms can be described more conveniently if the neighbor's state can be modified.
2. *No dynamic activation:* In the GCA model cells can deactivate themselves. Thereby the number of active cells can be reduced generation by generation. An inactive cell cannot change its state but its state can be read by another cell. Enlarging the number of active cells dynamically is only achievable by an additional mechanism like a central control. In order to control the number of active cells in a decentralized way, write access to the neighbors is mandatory. The reason is that an inactive cell cannot activate itself; it has to be activated by another active cell. A dynamic varying activity is very often an inherent property of parallel algorithms which should be exploited in order to use the computational resources of inactive cells for other computations or to reduce the power consumption.

Related work. The PSA (Parallel Substitution Algorithm) model [7] of computation is a very general and powerful model based on *substitution rules*. It allows also modifying the states of arbitrary target cells (*right side* of the substitution) using a "*base*" and a "*context*". In relation to the GCA-w the base corresponds to the cell under consideration, the context corresponds to the read neighbors and the right side corresponds to the cells

which are modified. There is also a relation to the CRCW-PRAM [17,18] model. The PRAM model is based on a physical view onto p processors that have global memory access to physical data words whereas the GCA-w is based on logical computing cells tailored to the application. Another difference of the GCA-w model compared to PRAM is the direct support of dynamic links and the rule based approach similar to the CA model.

2. The GCA-w model

The GCA-w model overcomes the restrictions of the GCA model by allowing write access to the neighbors. A cell can operate in two modes:

1. *Normal GCA mode:* A cell reads information from the dynamically linked neighbors and then updates its own state (data and link information) only.
2. *Write-mode:* A cell reads information from the dynamically linked neighbors and then updates its neighbors' states and optionally its own state (Fig. 1).

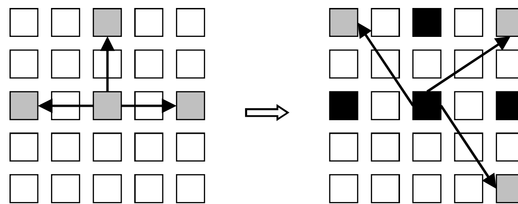

Fig. 1. Each cell is dynamically connected to global (or locally restricted) neighbors (grey). The state of the center cell including the links and the states of its neighbors can be changed (grey to black) by a local rule.

The write-mode can be used to activate or to inactivate a neighbor, *e.g.*, by sending a certain control code to the neighbor. An inactive cell serves as a storage-only cell that can be accessed (read and write) by another active cell. With the availability of the write-mode, many parallel algorithms can be described with a lower time-complexity and furthermore the computing resources of inactive cells could be utilized for other computations.

Now an inherent problem has appeared that is complicating an implementation: write conflicts may occur. They can either be avoided by using the write-mode in an "exclusive" way meaning that the algorithm ensures that no write conflict can occur. Otherwise, the conflicts have to be resolved in a defined way. Well-known conflict resolution strategies among others are *Priority*, *Arbitrary*, *Common*, or *Reduction*.

GCA-w with unstructured state. A GCA-w consists of an array of processing cells (Fig. 2). Each cell k contains a state q, an address function h, and a rule function f. The cells' states are updated as follows:

1. The effective address p_{eff} of the global neighbor (in the general case multiple neighbors are permitted) is computed.
2. The dynamic link to the neighbor is established in order to read state q^*.
3. The local rule f is applied yielding the results *f1* and *f2*.
4. The result *f1* is optionally written to update the cell's state q, and the result *f2* is optionally written to update the state q^* of the neighbor cell.

Optionally, the functions h and f may take into account central control information, like the generation counter t, common parameters, control codes or address offsets. Note that the model does not require central control information because the computation of such information can be replicated in each cell. The reason for using a central control is to minimize the cell's complexity.

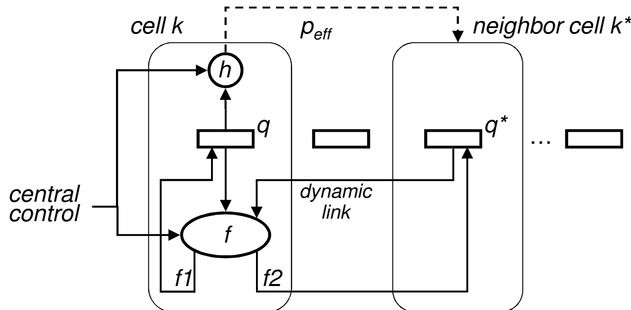

Fig. 2. The global neighbor is accessed using the effective address p_{eff} computed by the cell. The next states *f1* and *f2* are computed and are stored in q and q^*.

GCA-w with structured state. Each cell (Fig. 3) contains a *data field d* and one or more *link information fields p*. The link information field p is also denoted as pointer field because it can directly act as a pointer if h is the identity function. The GCA-w model is called *one-handed* if only one neighbor can be addressed, *two-handed* if two neighbors can be addressed and so on. The one-handed model seems to be sufficient for most practical applications, as it is the case for most of the GCA algorithms investigated so far. In addition, the multi-handed model can be simulated on the one-handed model. Therefore, the following considerations are restricted to the one-handed model.

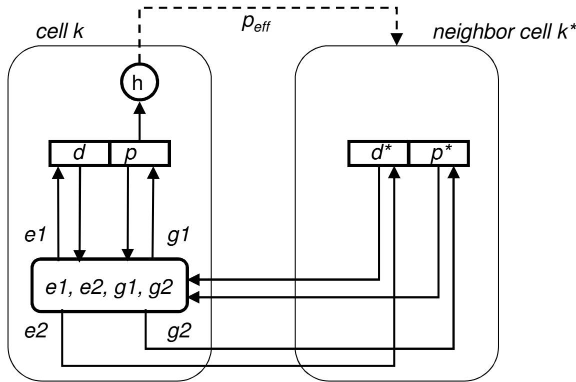

Fig. 3. The state q of a cell can be partitioned into a data field d and one or more link information fields p.

The local information (d, p) and the global information (d^*, p^*) are inputs of the functions e and g which compute the next data and the next link respectively (Fig. 3). All cells are updated in parallel in a synchronous way generation by generation. The functions e and g may further depend on the local space index k of the cell and central control information. An optional function h is used to compute the effective address p_{eff} of the global cell in the current generation.

A structured GCA-w can be transformed into an unstructured GCA-w by unifying the two fields d and p into one single word q, such that q is partly or alternatively interpreted as data or pointer information. The unstructured model has the advantage that it is simpler and can be implemented with fewer hardware resources. The unstructured model can also be seen as "untyped" because the types "data" and "pointer" are not distinguished whereas the structured model can be seen as "typed".

The operation principle of a GCA-w can be defined in two forms: *basic* GCA-w model, or *general* GCA-w model that includes the basic model.

Basic model. The basic GCA-w model does not use the address calculation function p_{eff} (p_{eff} is the identity function $h = p$), meaning that the effective address is $p_{\mathrm{eff}} = p$. The next pointer and the next data fields are computed by the following rules:

$$\begin{aligned} d &<= e1(d, p, d^*, p^*, k, \mathrm{control})\,, \\ p &<= g1(d, p, d^*, p^*, k, \mathrm{control})\,, \\ d^* &<= e2(d, p, d^*, p^*, k, \mathrm{control})\,, \\ p^* &<= g2(d, p, d^*, p^*, k, \mathrm{control})\,. \end{aligned}$$

The assignment symbol “<=” is used to indicate the synchronous updating. If the arrays $D = d_0 d_1 ... d_{n-1}$ and $P = p_0 p_1 ... p_{n-1}$ are used to denote the data fields respectively the pointer fields of the cells then d, d^*, p, p^* are equivalent to

$$d = D[k]\,, \qquad p = P[k]\,, \qquad d^* = D[p]\,, \qquad p^* = P[p]\,.$$

Note that the uniform functions used in the cells (not dependent on the local space index k) can be specialized resulting in non-uniform functions: $f(k, \ldots) \to f_k(\ldots)$.

The basic model has the advantage that it is simple because it does not require the function h, thereby also reducing the implementation effort. The user of this model has to be aware that the effective address in the current generation has to be computed in the previous generation. The following *general* GCA-w model is a more convenient choice from the programmer's point of view when an algorithm can be described more naturally by computing the effective address in the current generation.

General model. The general GCA-w model uses the additional rule h in order to compute the effective address:

$$p_{\text{eff}} = h(d, p, k, \text{control})\,, \qquad d^* = D[p_{\text{eff}}]\,, \qquad p^* = P[p_{\text{eff}}]\,.$$

The assignment symbol “=” indicates that a value is assigned to the temporary pointer variable p_{eff} that may be a wire or a temporary register in a hardware implementation.

A hardware implementation of the GCA-w model can be simplified in all cases where the pointer p follows an address pattern which is known in advance and which is not data dependent: $p_{\text{eff}} = h(k, \text{control})$. Such a case with “known pointers” appears in many applications (*e.g.*, hypercube algorithms). Then the link fields in the cells are not necessary. In a sequential implementation of the model a central address generator can easily generate these addresses.

Write conflicts. As mentioned before write conflicts may occur. We will restrict the discussion to the one-handed model with n cells. A conflict occurs if there is more than one write request on a cell, for different reasons:

- A cell may use itself as a neighbor if the effective address is equal to the cell's index. If this case is not forbidden or given another semantic then the two values *f1* and *f2* (both produced by the cell itself) to be written cause a conflict.

- The probably most frequent conflict might occur when more than one cell $k1, k2, \ldots$) tries to update a common neighboring cell k^* which does not update itself. In this case, at most $n-1$ write requests may occur on a cell.

- The maximum number $n+1$ of write requests occurs when a cell k tries to update itself with $f1$ and $f2$ as described above and in addition all $n-1$ other cells try to update this cell, too.

If the absence of write conflicts cannot be guaranteed, which is especially the case if the neighborhood is data dependent or random, the model and a concordant implementation have to offer a defined conflict handling strategy. It is obvious that the conflict handling increases the complexity of the model, slows down the computation and will complicate a hardware or software implementation.

The conflict handler works in principle as follows: The own write request (from the target cell) and the other write requests (from the source cells) are send to the arbiter of the conflict handler (Fig. 4). The arbiter detects and resolves the conflicts, and then sends back acknowledge signals to those requesters which shall be used for reduction. The selected subset of write information is then transferred to the reducer which reduces the subset to a scalar that will fit into the target cell.

In case of conflicts an extra computation phase (selection phase) is necessary. In this phase the acknowledge values and the write information subset are computed. The implementation of the conflict handlers in hardware or software requires temporary variables, *e.g.*, to store the requests and the acknowledge information.

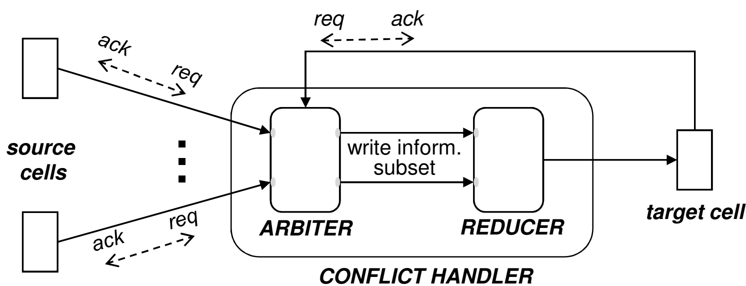

Fig. 4. The state q of a cell can be partitioned into a data field d and one or more link information fields p.

Activation of cells. Cells are able to control their activity in a decentralized way. By this property, the overall computational effort and the power consumption can be reduced, and the computing facilities of sleeping cells

may be utilized for other computations in the system. The activity can be controlled in a GCA-w algorithm (algorithmic description of an application according to the GCA-w model) in one of the following ways

- The set of states that can be stored in a cell is separated into an *active* set and a *passive* set. When assigning an active value (a value from the active set) then the cell is activated at the same time. When assigning a passive value (a value from the passive set) then the cell is deactivated at the same time
- The cell's state is extended by an additional *activity bit* that can be set/reset in order to activate/deactivate a cell.

In a hardware or software implementation the active cells could be stored in activity lists defining the set of active cells to be updated. The principle of a sequential operating hardware architecture without conflict handler was presented in [1,19].

Compared to a CA hardware implementation, the GCA requires hardware support to access a neighbor randomly out of the range of all possible neighbors that are used in an application; this means that for each cell a network has to be supplied which can read a global neighbor. In special cases, when the accessed neighbors are known in advance or the accesses are restricted to special patterns (like the hypercube neighborhood), the access network can significantly be reduced. The GCA-w is even more complex in terms of hardware. It needs in addition a network for each cell that allows also modifying a global neighbor, out of the range of neighbors that are modified by the specific application. In case that conflicts cannot be avoided, conflict handlers (with connections to all possible sources) have to implemented in each cell at worst. There are three ways to reduce the hardware effort: (1) use a locally restricted neighborhood, at least for the cells to be modified in order to reduce the write network and the conflict handling hardware, (2) restrict the networks to the accesses that really occur in an application (use application specific networks), and (3) use applications that ensure the "exclusive-write" property in order not to need conflict handlers. In addition, hardware architectures (*e.g.* multiple pipelines) can be used that do not work totally in parallel (extreme case: optimized sequential emulation of the model) in order to yield a better hardware utilization.

Modifications or future extensions of the GCA-w model. The model can be modified or supplemented by further features in order to meet practical, dedicated or more general requirements. Such modifications or features are

- The cell array may contain storage-only cells that cannot compute and thus cannot be activated. They are distinguished into "constant" cells (with read-only access) and "variable" cells (with read and write access).
- Another updating scheme is used such as asynchronous updating.
- The output of one generation is written into a new cell field in a dataflow manner.
- Several GCA-w cell arrays (interacting or not interacting) are computed in parallel.
- Non-uniform (space-dependent) cells are used.
- The neighborhood is locally restricted and/or partially static (non-dynamic). If the neighborhood is static and local then the GCA-w model may be called "CA-w".
- The cell's state is separated into a public and private (hidden) part. Only the public part can be accessed by another cell.
- Cells are considered as objects offering methods which can be invoked by the cell itself or by another cell.
- Cells are dynamically created or deleted.

3. Applications

It was already shown that the GCA model is applicable to many parallel applications and that it can efficiently be supported by hardware [11,13]. Recently it was shown that the classical PRAM models can be simulated on the GCA [6]. The write access available in the PRAM models can be simulated with a slowdown of $O(\log n)$ using a tree of cells. If a GCA-w algorithm guarantees the exclusive-write property this slowdown can be eliminated through the direct write-access capability. In the following, the usefulness and expressiveness of the GCA-w model is demonstrated by selected applications (one-to-all communication, synchronization, moving of particles). Other applications (pointer inversion, sorting with pointers, Pascal's triangle) were presented in [1,19].

3.1. One-to-all communication

The information stored in cell *0* shall be replicated in each other cell. This can be accomplished using a tree with cell *0* as the root. In the case of the normal GCA model, all $n-1$ receiver cells have to be active over all

log n generations, copying the information from their parent cells. Using the GCA-w model, the receiver cells are activated by the parent cells when the information shall be transferred (Fig. 5). An active parent cell activates a sleeping child (receiver cell) and at the same time writes the information to it. Therefore a child can sleep until it receives a message. The GCA-w algorithm for the one-to-all communication is the following:

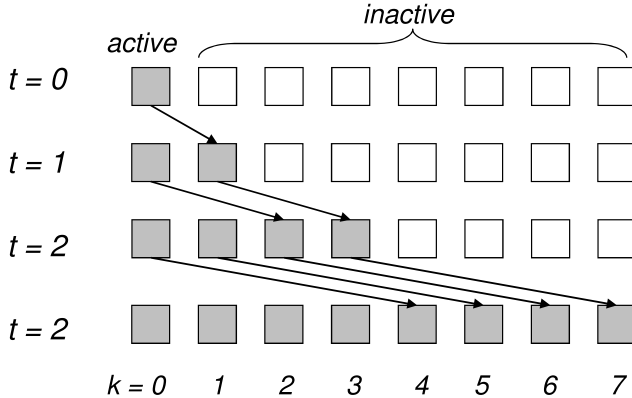

Fig. 5. The state q of a cell can be partitioned into a data field d and one or more link information fields p.

```
    type cell = (data: integer, active: activity)
    C: array [0 .. n-1] of cell

    // for all cells C[k] in the array
    parallel C[k = 0 .. n-1]
      if (k=0) then active <= TRUE
                else active <= FALSE endif
    endparallel

    for t = 0 to ceiling(log2 n) - 1 do
      // only do for active cells
      parallel C[k where active]
        // compute eff. address peff, neighbor is a temp. var.
R0.     neighbor = k + 2t
        // activate neighbor cell
R1.     neighbor.active <= TRUE
        // cell(k) writes its data to neighbor cell
R2.     neighbor.data <= data
      endparallel
    endfor
```

Note that the cell rule is given by (R0, R1, R2). R0 defines the effective address of the neighbor, and (R1, R2) define the updates of this neighbor. Compared to a CA cell rule, the neighbor is dynamically selected, and the neighbor is modified, in this case without conflict.

3.2. Synchronization

All cells shall change simultaneously from the ZERO state into the FIRE state. The problem is related to the well-known Firing–Squad problem. The number n of cells is not known in advance. The "general" is located on the left end (at $k = 0$, Fig. 6) and starts the process being the only active cell. Using the global neighborhood from the beginning the problem could be solved trivially if all the soldiers (the other cells) would directly observe the general. But it is assumed that at the beginning only local neighborhoods are allowed. Initially each cell is connected to its right neighbor except the right end soldier who is pointing to himself thereby marking the end of the chain (Fig. 6(A)). The purpose of this version with $N + 1$ generations is to show how $N = n + 1$ cells can be activated in principle one after the other forming a wave of activity. Note that a more efficient algorithm with $2 + \log_2 N$ generations can be designed using the well-known pointer-jumping technique (Fig. 6(B)).

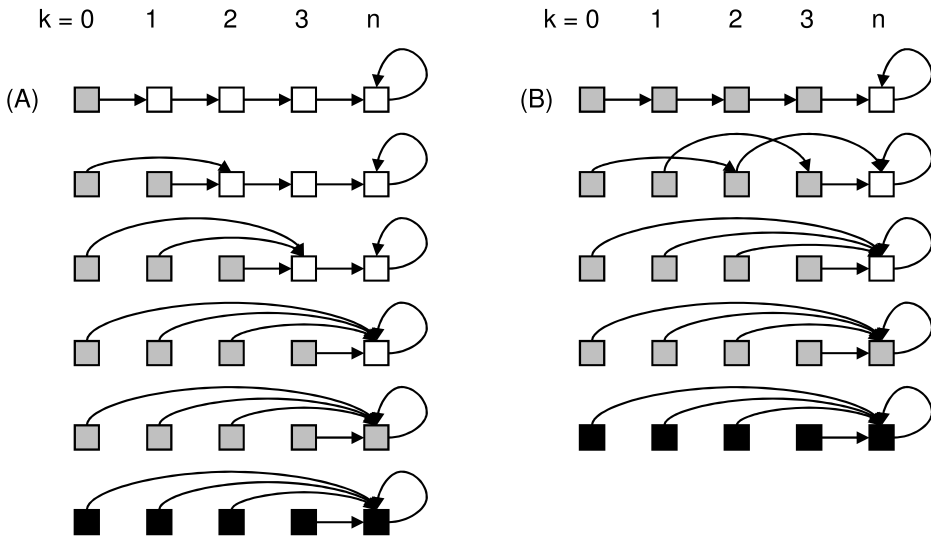

Fig. 6. (A) The activity is gradually propagated (grey) until after $n+1$ generations all cells change into the FIRE state (black). (B) The last soldier can be detected faster using pointer jumping.

```
type cell = (data: (ZERO, FIRE); p: 0 .. n;  active: activity)
C: array [0 .. n] of cell

parallel C[k = 0 .. n-1]  // initialize
 data <= ZERO
 if (k=0) then active <= TRUE else active <= FALSE endif
 if (k=n) then p <= n else p <= k+1 endif
endparallel

repeat n+1 times
  parallel C[k where active]
    // activate right neighbor, write mode
    if (p = k+1) then p.active <= TRUE endif
    // if not right border reached increment
    // pointer for active cells only
    if (p.p != p) then p <= p+1 endif
    // wait one step until right border cell was activated
    if (p.p = p) and (p.active = TRUE) then data <= FIRE endif
  endparallel
endrepeat
```

3.3. Modeling moving agents

Modeling as CA. In order to describe the movement of an agent from a source cell to a target cell, a pair of two CA rules (the source and the target rule) has to be applied simultaneously (Fig. 7, top). It is assumed that the agent has a direction (computed in the previous generation or computed in the current generation before usage) that can be observed by the target. If the agent is allowed to move then (a) the source rule is: *delete (consume) agent*, and (2) the target rule is: *copy (generate) agent*. Thus the movement is a joint application of these two rules. Both cells have to be active. Before an agent can move, moving conflicts have to be detected. If the agent wants to move one cell ahead, it has to check (1) if there is no agent or obstacle directly in front (situated on the *front cell*) and (2) no other agent (to the left, right or two cells ahead) wants to move to the same front cell. The conflict handling can be performed in each agent, or the front cell may undertake this task [22,23]. In the case that an agent wants to jump (more than one step ahead) the CA rules become much more complicated: (a) each agent has to check all the other agents which might move to the same target and at the same time (b) each possible target has to check the environment for agents that want to move to it. Thus conflict resolution has to be implemented in the entire source and target cells for each possible conflict situation in a consistent way such that one determined movement can take place. If the conflict resolution is random, then (1) the decision has to be computed by the target, and (2) read from the target and obeyed

by the source. Normally this solution requires an extra phase or generation in order to resolve the conflict before the movement can take place. We can summarize, that the modeling of moving agents in CA is possible but becomes very difficult because complex conflict handling procedures have to be implemented in the source and the target cells. Therefore, if a moving rule shall be modified, the source rule, the target rule, and the conflict handling in the source and in the target have to be modified. Modeling agents in the GCA model is similar as in the CA model, but the GCA model offers more flexibility because the neighbors can be selected dynamically.

Modeling as GCA-w. There are different ways to model moving agents as a GCA-w. Compared to the CA modeling the movement of an agent can be executed by one cell only. There are two solutions: (A) *push* principle and (B) *pull* principle.

(A) The agent situated on the source cell copies itself to the target cell and deletes itself (Fig. 7, middle).

(B) The target cell copies the agent from the source cell and deletes the agent (Fig. 7, bottom).

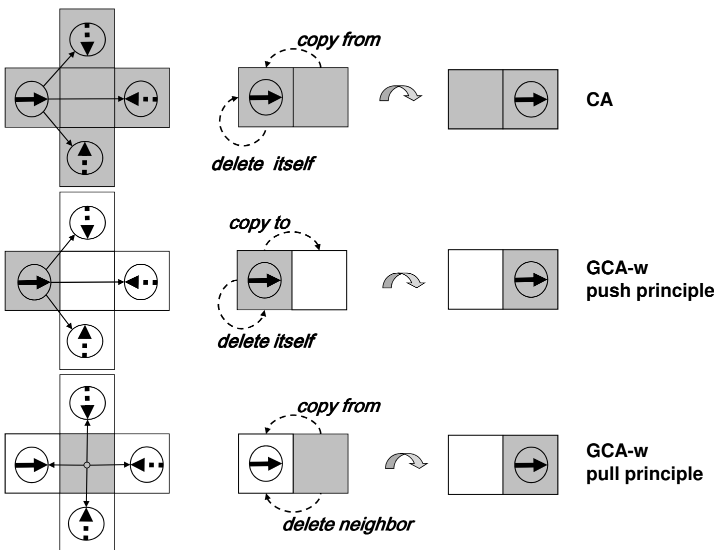

Fig. 7. Modeling the movement of an agent as CA and as GCA-w. Active cells are shaded.

3.4. Random walks of particles

The GCA-w push principle was used to model the random distribution of particles. The intention is to show that different variants of random walks can be modeled and implemented without much effort. The intention was not to show how realistic these variants model physical phenomena, like in [20,21]. Four random walk variants were modeled:

(Rsimple) *Simple Random Walk.* Each particle chooses randomly one of the nearest NESW neighbors as its moving direction. If there is particle on the target or it is not selected to move in case of a conflict, it will not move. (Interchanging of particles is not allowed.)

(Rsmart) *Smart Random Walk.* The particle first checks which of the NESW neighbors are free. Then it chooses randomly one direction from this subset. In the case that no nearest neighbor is free, it cannot move at all.

(Rsmart+) *Smart+ Random Walk.* The particle checks all neighbors within Manhattan distance of two. Then the four directions are ordered with regard to the amount of free neighbors within the distance of two. Then one direction is selected randomly from the "free" directions with the highest scores.

(R12) *Random Walk with 12 Neighbors.* A particular set of 12 non-local neighbors was defined in order to accelerate the distribution (Fig. 8). Each particle chooses randomly one neighbor out of the given set as its target position.

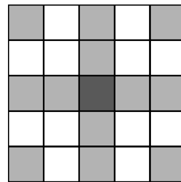

Fig. 8. Special neighborhood with 12 neighbors used for the random walk *R12*.

In the case of write conflicts (more than one source particle wants to move to the same target position) one of the source particles is selected randomly to move.

As can be seen from Fig. 9, the distribution evolves faster in the following order of the variants: *Rsmart+, Rsmart, Rsimple.* The variant *R12* distributes the particles faster at the beginning (due to the extended neighborhood) but does not perform so well when time is proceeding (*e.g.*, at $t = 240$ the distribution seems qualitatively to be better than for *Rsimple* but worse in comparison to *Rsmart* and *Rsmart+*).

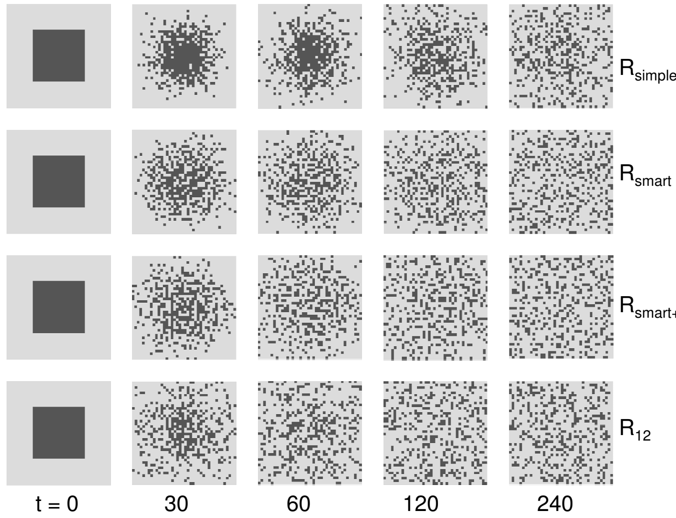

Fig. 9. The evolution of the four variants of random walks. Field size is 40×40 with wrap around. At the beginning 20×20 particles are placed in the middle.

In order to understand the different effectiveness (how fast the initial distribution converges to a random distribution) of the variants a simple evaluation was conducted for $t = 240$ when the distribution is already random or nearly random (Table I). The results are averaged over 10 simulations. The mobility can be defined as

$$\text{Mobility} = (N_{\text{Particles}} - N_{\text{NotMovedParticles}})/N_{\text{Particles}}\,. \tag{1}$$

TABLE I

Mobility and conflicts for the different random walk variants averaged over 10 runs.

	$N_{\text{NotMovedParticles}}$ particles not moved out of 400	Mobility	Total no. of write conflicts	No. of write conflicts with 2 requests	No. of write conflicts with 3 requests	No. of write conflicts with 4 requests
Rsimple	128.0	68.0%	37.6	35.1	2.5	0
Rsmart	40.4	89.9%	38.1	36.1	2.0	0
Rsmart+	13.7	96.9%	13.6	13.3	0.3	0
R12	123.3	69.2%	38.8	36.5	2.3	0

It can be seen that the mobility correlates very well with the effectiveness of the variants. Note that in these models a particle cannot move for two reasons: either the target is occupied by another particle or because of a

denied request. The total number of write conflicts is about the same for *Rsimple*, *Rsmart* and *R12*, but much lower for *Rsmart+*. Write conflicts with three requests are very rare, and with four requests did not occur. Note that in this example the percentage of particles was only 25% of the whole number of cells, *e.g.*, the total number of conflicts increases to around 175 and the mobility decreases to 33% for *Rsmart+* when the percentage of particles is 56%.

We like to reveal some details of the software implementation to those who like to implement such GCA-w algorithms. The data set is:

F: cell field holding the current generation of cells.

F_{next}: cell field holding the next generation in order to allow synchronous updating.

P_{eff}: array holding the effective addresses for each cell (relative x, y coordinates of the targets). Temporary array.

Conf: array holding the number of write requests. Temporary array.

One computation cycle consists of the following steps:

1. *Address calculation.* The effective address $p_{\text{eff}} = \textit{target}$ of the neighbor is computed for all cells and stored in p_{eff}. This address can be computed for *Rsimple* and *R12* in the previous generation and stored in the cell to be used directly in the current generation (Basic Model). The effective address is calculated in the current generation for *Rsmart* and *Rsmart+* (General Model).
2. *Store requests.* The write requests to each cell (x, y) are accumulated and stored in *Conf*, *e.g.*, if the entry at position (x, y) is 3 than there are 3 requests on this target.
3. *Handle conflicts and write.* If (no agent on target and $\textit{Conf}[\textit{target}]=1$) then move (write to target and delete self). If (*no agent on target* and $\textit{Conf}[\textit{target}] > 1$) then select one of the sources randomly and move.
4. *Synchronous update.* Copy F_{next} to F.

In general, the number and the form of these steps may vary depending on the specific application and the required conflict handling.

4. Conclusion

A novel parallel computing model called GCA-w, Global Cellular Automata with write access to the neighbors, was presented. GCA-w is an extension of the GCA model that is related to the Cellular Automata (CA) model. In the GCA and GCA-w model the neighbors are linked dynamically to the cell under consideration and the data and the link information are

modified by a local rule. Thereby a cell can decide by itself which shall be its neighbors in the next generation. The novel GCA-w model overcomes the restriction that a cell can only modify its own state. Thereby any global cell in the whole cell array can be the target of an information transfer. Furthermore "sleeping" cells can be turned dynamically into active cells in a decentralized way. Sleeping resources might be assigned dynamically to other active cells leading to better resource utilization or lower power consumption. Conflicts can be solved systematically using a conflict handler in the target cell. Classical PRAM algorithms as well as other practical parallel applications can be mapped onto this model without much effort as shown for one-to-all communication, synchronization, and different variants of random walks. We expect that the model is practical for a wide range of applications with dynamic activities, like diffusion limited aggregation, crystal growth, pattern formation, forest fire, traffic simulation or agent systems.

REFERENCES

[1] R. Hoffmann, Das massiv-parallele Berechnungsmodell GCA-w (Global cellular automata with write-access), Fachgebiet Rechnerarchitektur, Technische Universität Darmstadt, Internal Report (1/2009), `http://www.ra.informatik.tu-darmstadt.de/forschung/publikationen/`

[2] J. Jendrsczok, R. Hoffmann, Th. Lenck, Generated horizontal and vertical data parallel GCA machines for the N-body force calculation, 22nd ARCS Conference, LNCS 5455/2009.

[3] J. Jendrsczok, R. Hoffmann, P. Ediger, A generated data parallel GCA machine for the Jacobi method, 3. HiPEAC Workshop on Reconfigurable Computing, HiPEAC Conf. Cyprus 2009.

[4] J. Jendrsczok, R. Hoffmann, J. Keller, Implementing Hirschberg's PRAM-algorithm for connected components on a global cellular automaton, International Journal of Foundations of Computer Science (IJFCS) Vol. 19, No. 6, 2008.

[5] J. Jendrsczok, P. Ediger, R. Hoffmann, A scalable configurable architecture for the massively parallel GCA model, In: IEEE International Symposium on Parallel and Distributed Processing (IPDPS), Workshop on Advances in Parallel and Distributed Computational Models (APDCM), April 2008.

[6] A. Osterloh, J. Keller, Das GCA-Modell im Vergleich zum PRAM-Modell, Informatik-Bericht 350 — 3/2009, FernUniversität in Hagen, `http://pv.fernuni-hagen.de/publikationen.php`

[7] S. Achasova, O. Bandman, V. Markova, S. Piskunov, *Parallel substitution algorithms, theory and applications*, World Scientific, 1994.

[8] W. Heenes, Entwurf und Realisierung von massivparallelen Architekturen für Globale Zellulare Automaten, Dissertation, Technische Universität Darmstadt (2007), `http://www.ra.informatik.tu-darmstadt.de/forschung/publikationen/`

[9] J. Jendrsczok, P. Ediger, R. Hoffmann, The global cellular automata experimental language GCA-L, Technischer Bericht, RA-1-2007, Technische Universität Darmstadt, FB Informatik (2007), `http://www.ra.informatik.tu-darmstadt.de/forschung/publikationen/`

[10] J. Jendrsczok, R. Hoffmann, J. Keller, Hirschberg's algorithm on a GCA and its parallel hardware implementation, 13th International European Conference on Parallel and Distributed Computing (Euro-Par 2007).

[11] W. Heenes, R. Hoffmann, J. Jendrsczok, A multiprocessor rrchitecture for the massively parallel model GCA, IPDPS/SMTPS 2006, IEEE Proceedings: 20th International Parallel and Distributed Processing Symposium.

[12] Chr. Ehrt, Globaler Zellularautomat: Parallele Algorithmen, Diplomarbeit, Technische Universität Darmstadt, 2005, `http://www.ra.informatik.tu-darmstadt.de/forschung/publikationen/`

[13] R. Hoffmann, W. Heenes, M. Halbach, Implementation of the massively parallel model GCA, In PARELEC, IEEE Computer Society (2004) 135–139.

[14] R. Hoffmann, K.-P. Völkmann, W. Heenes, GCA: A massively parallel model, IPDPS 2003, IEEE Comp. Soc.

[15] R. Hoffmann, K.-P. Völkmann, S. Waldschmidt, W. Heenes, GCA: Global cellular automata, a flexible parallel model, In Proceedings of: 6th International Conference on Parallel Computing Technologies PaCT 2001, Lecture Notes in Computer Science (LNCS 2127), Springer (2001).

[16] R. Hoffmann, K.-P. Völkmann, S. Waldschmidt, Global cellular automata GCA: An universal extension of the CA model, In: T. Worsch (Ed.): ACRI Conference (2000).

[17] J. Keller, Chr. Kessler, J. Träff, *Practical PRAM Programming*, Wiley, 2001.

[18] J. JaJa, *An Introduction to Parallel Algorithms*, Addison–Wesley, 1992.

[19] R. Hoffmann, The GCA-w massively parallel model, In Proc. of 10th International Conference on Parallel Computing Technologies PaCT2009, Lecture Notes in Computer Science, Volume 5698/2009.

[20] O. Bandman, Comparative study of cellular-automata diffusion models, Lecture Notes in Computer Science, Vol. 1662, p. 756, Springer, 1999.

[21] O. Bandman, Mapping physical phenomena onto CA-models, In Automata-2008 Theory and Application of Cellular Automata, Eds.: A. Adamatzky *et al.*, p. 381–395, Luniver Press, 2008.

[22] M. Halbach, R. Hoffmann, Parallel hardware architecture to simulate movable creatures in the CA model, In Proceedings of: 6th International Conference on Parallel Computing Technologies PaCT2007, Lecture Notes in Computer Science 4671, Springer, 2007.

[23] P. Ediger, R. Hoffmann, Optimizing the creature's rule for all-to-all communication, In Automata-2008 Theory and Application of Cellular Automata, Eds.: A. Adamatzky *et al.*, p. 398–410, Luniver Press, 2008.