GCA: A Massively Parallel Model

Rolf Hoffmann, Klaus-Peter Völkmann, Wolfgang Heenes

1. Motivation

2. The GCA model

3. Classification

4. GCA Algorithms

5. Hardware-Architectures

6. Conclusion
CA: Cellular Automata

optimal model for applications with inherent local neighborhood


**Restriction of the model**

- only local access to fixed neighbors
- global communication between remote cells is sequential in space
Features of the New Model

- Direct Dynamic Read Access to Global Neighbors
- Massively Parallel
- suited for a wide class of parallel algorithms
2. GCA: Global Cellular Automata Model

Cell Field: \( C = \text{array } [0..n - 1] \text{ of State} \)

State of each cell:

\[
\text{State} = \text{record} \\
\text{Data: Datatype} \quad \text{// the data} \\
\text{L1: 0..n-1} \quad \text{// points to the first global cell} \\
\text{L2: 0..n-1} \quad \text{// points to the second global cell} \\
\text{endrecord}
\]

Local Rule: \( \text{function } f(\text{Self:State, Neighbor1:State, Neighbor2:State}):\text{State} \)

Next Generation:

\[
\text{for } i:=0..n-1 \text{ do in parallel} \\
C[i] \leftarrow f(C[i], C[C[i].L1], C[C[i].L2]) \\
\text{endfor}
\]
Links to global neighbors

$k = 2$ arms/links : L1, L2
# Features of CA and GCA

<table>
<thead>
<tr>
<th></th>
<th>CA</th>
<th>GCA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ordering in Space</strong></td>
<td>n-dim. grid</td>
<td>ordered set (array)</td>
</tr>
<tr>
<td><strong>Neighbors</strong></td>
<td>local</td>
<td>global</td>
</tr>
<tr>
<td><strong>Links</strong></td>
<td>fixed</td>
<td>fixed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– time dependent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– data dependent</td>
</tr>
<tr>
<td><strong>Access to NeighborsCStates</strong></td>
<td>read-only</td>
<td></td>
</tr>
<tr>
<td><strong>Local Rule</strong></td>
<td>nextCState:=Rule(CState, NeighborsCStates)</td>
<td>Cell changes it's own state</td>
</tr>
<tr>
<td><strong>Updating</strong></td>
<td>synchronous</td>
<td></td>
</tr>
<tr>
<td><strong>Massively-Parallel</strong></td>
<td>yes (no write conflicts)</td>
<td></td>
</tr>
</tbody>
</table>
3. Classification of different types of GCA

**Access Pattern:**
*Links to global cells in generation t (at time step t)*

**space dependent:** access pattern varies from cell to cell. (regular/irregular)

**time dependent:** access pattern varies from time to time. (static/dynamic)

**data dependent:** access pattern depends on the actual state of the cell and optionally by the state of the global neighbors.

Subclasses can be defined depending on
- which part of the cell state (data part, address part) influences the access pattern
- which parts of the global neighbor states influence the access pattern
# Types of GCA depending on space, time or data

<table>
<thead>
<tr>
<th>Type</th>
<th>space dependent</th>
<th>time dependent</th>
<th>data dependent</th>
<th>Access pattern Neighborhood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>regular static</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
<td></td>
<td>irregular static</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>x</td>
<td></td>
<td>regular dynamic (time)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>x</td>
<td>regular dynamic (data)</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>e</td>
<td>x</td>
<td>regular dynamic (data and time)</td>
</tr>
<tr>
<td>4</td>
<td>x</td>
<td>x</td>
<td></td>
<td>irregular dynamic (time)</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td></td>
<td>x</td>
<td>irregular dynamic (data)</td>
</tr>
<tr>
<td>7</td>
<td>e</td>
<td>e</td>
<td>x</td>
<td>irregular dynamic (data and time)</td>
</tr>
</tbody>
</table>

*e* : emulated by parts of the data field of the state
Static GCA, 2 Links

Regular static GCA (Type 0)

Irregular static GCA (Type 1)

<table>
<thead>
<tr>
<th>cell index</th>
<th>access pattern d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,3</td>
<td>-1, +2</td>
</tr>
<tr>
<td>1,4</td>
<td>-1, +3</td>
</tr>
<tr>
<td>2,5</td>
<td>-1, -2</td>
</tr>
</tbody>
</table>
3. Classification

Dynamic GCA, time dependent, 1 Link

<table>
<thead>
<tr>
<th>Cell Index</th>
<th>$d(t = 0)$</th>
<th>$d(t = 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1,2,3,4,5</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

Regular (Type 2)

<table>
<thead>
<tr>
<th>Cell Index</th>
<th>$d(t = 0)$</th>
<th>$d(t = 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,2,3,5</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>1,4</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

Irregular (Type 4)
4. GCA Algorithms
4.1 Dynamic GCA, time dependent, 1 Link

- **Bitonic Merge Algorithm** sorts a bitonic sequence
- A sequence of numbers is called *bitonic*, if the first part of the sequence is ascending and the second part is descending, or if the sequence is cyclically shifted
- Bitonic merge is the second part of a complete sorting algorithm

Total complexity for Sorting is $O((\log N)^2)$
4.2 Dynamic GCA, data and space dependent, 1 Link Sort Algorithm

(Binary search position) Start with the initial access pattern (only one pointer) at the position \( N - N/4 \) for the search in RIGHT and at the position \( N/4 \) for the search in LEFT. The pointer is incremented/decremented by \( \delta \) if the data of the cell is greater/smaller the data of the cell in access, with \( \delta = N/8, N/4, \ldots 1, 1 \). If the access pointer has already reached the bounds of RIGHT/LEFT, then it may not be changed any more to prevent the access pointer to point outside the bounds of the subarrays.

(Calculate the target position) The target position is computed by adding the own position to the searched position minus \( N/2 \). The last step of (1) was in fact a test which locates the pointer on the searched position or one position left from the searched position. So if the data is greater the data of the cell in the found position, one has to be added to the result of the above calculation.

(Move to the target position) The GCA model does not allow to write data to another cell. Only the cell may change its own state. In this example the computed pointers need to be inverted. The inversion of the pointers can be accomplished again by a binary search algorithm on the pointers instead of the data.

Total complexity for Sorting is \( O((\log N)^2) \)
5. Hardware-Architectures

Sequential Architecture

**PHASE 1**

*(Fetch) a=Cell[z]*

*(Get) global Cells b=Cell[L1] and c=Cell[L2]*

*(Execute) y=f(a,b,c)*

*(Write) y (buffered)*
5. Hardware-Architectures

Parallel Architecture, $p=2$, $k=1$ Link

- read two cells (from separate banks)
- read global cells
- local computations
- write results into buffer memories

Memory Bits

$$s = 2 \cdot (1 + kp) \cdot 2^m \cdot (1 + k) \cdot m \cdot O(p)$$
6. Conclusion

- GCA-Architectures have been classified into types
  - regular/irregular (space dependent)
  - static/dynamic (data, time dependent)

- Most algorithms can be implemented on an GCA which are not data dependent → Hardware implementation cost can be reduced

- An algorithm making use of data dependency was presented, no speed-up → Open question: for what type of algorithms data dependency can be used for speed-up

- GCA can be implemented in parallel → Memory bits $O(p)$

- Future research → Hardware and software support