Convergence of Array DBMS and Cellular Automata: A Road Traffic Simulation Case

Ramon Antonio Rodriges Zalipynis
HSE University
Moscow, Russia
rodriges@gis.land

ABSTRACT

Array DBMSs manage big N-d arrays, are not yet widely known, but are experiencing an R&D surge due to the rapid growth of array volumes. Cellular automata (CA) operate on a discrete lattice of cells that can be modeled by an N-d array. CA are successfully applied to model fire spread, land cover change, road traffic, and other processes. We made traffic CA simulations possible by array DBMS due to novel components: native UDF language, proactive exec plans, convolution operator, retiling strategy, array versioning, locks, virtual axes, etc. A database approach to CA brings powerful parallelization, data fusion, array processing, and interoperability to name a few. To our best knowledge, our work is the first to run end-to-end CA simulations completely inside array DBMS: we enable array DBMS to simulate the physical world for the first time. Paper homepage: http://sigmod2021.gis.gg/

CCS CONCEPTS

• **Information systems → Parallel and distributed DBMSs**; **Geographic information systems**.

KEYWORDS

Array DBMS; cellular automata; urban traffic; physical simulation

ACM Reference Format:
Ramon Antonio Rodriges Zalipynis. 2021. Convergence of Array DBMS and Cellular Automata: A Road Traffic Simulation Case. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3458457

1 INTRODUCTION

An array DBMS manages N-d array storage, processing, and even visualization in some cases. N-d arrays are natural models for many important data types [3, 7, 36]. The first array DBMSs and add-ons appeared long ago [6, 9, 14, 40, 49]. However, only the last decade flourished with a significant body of array management R&D: array DBMSs [12, 43, 44], array stores [15, 39, 54], array engines [13, 19, 20, 22], and other array-oriented systems [5, 26, 29, 42, 53, 55].

The array DBMS R&D area is quite young and many R&D opportunities are attractive and unexplored. For example, novel indexing techniques accelerate function evaluation [46] and array joins [57]. Only recently, top-k queries [10], similarity array joins [58], view maintenance [60], distributed caching in array DBMS [59] were first introduced. Researchers explore compression potentials [25, 31, 45], interactive visualization [4], and machine learning [38, 47, 53].

This paper complements the aforementioned related work by exploring a rather unusual application of a database system for the first time: physical world simulation. This kind of workload has been traditionally implemented on diverse types of grids and meshes that can be modeled as an N-d array [3, 11, 36]. As N-d array is at the core of array DBMS, it is logical to apply array DBMSs to physical simulations to benefit from array DBMS capabilities.

Cellular automata seem to be a good starting point for integrating simulation into array DBMS due to having N-d array model for its lattice and numerous successful applications for edge detection [16, 21], modeling urban growth [17, 24, 33], fire spread [18, 23, 37], land cover change [28, 50], road traffic [27, 30, 51], and other phenomena.

To demonstrate the applicability of array DBMS to CA modeling, we take some of the most challenging CA: traffic CA (TCA). We take a complex road traffic model with multiple lanes, road intersections and traffic lights. Vehicles have different lengths, moving directions, varying speed, can change lanes, directions, and overtake each other. We show that ChronosDB [43, 44], with extensions, is capable of effectively dealing with all simulation intricacies.

Unlike ad-hoc CA implementations, array DBMSs may serve as a framework for CA modeling as they provide numerous capabilities out-of-the-box, e.g. parallel processing. The support for CA models can also be a step towards supporting even more general approaches like agent-based modeling [52]. Finally, CA models challenge array DBMSs helping them to become more robust systems in general.

At a glance, array DBMSs may seem to readily support CA simulations. However, an array DBMS requires appropriate extensions to leverage its solid codebase for CA simulations.

Our contributions in this paper are as follows. First of all, we identify a novel R&D direction and novel application of array DBMSs: physical world simulation. The very fact of the possibility of an array DBMS application to end-to-end physical simulations is valuable. We believe this may inspire the related future work.

Next we show how to extend ChronosDB [43, 44] to effectively support CA simulations and provide unique benefits inherent to array DBMSs and ChronosDB in particular. We showcase that ChronosDB array DBMS can be used for end-to-end CA simulations, from initialization to computing the resulting statistics (the goal of CA simulations). Our performance evaluation shows that ChronosDB is quite efficient for CA simulations compared to a non-DBMS approach. Finally, we discuss array DBMS benefits for CA simulations, future R&D opportunities and conclude.

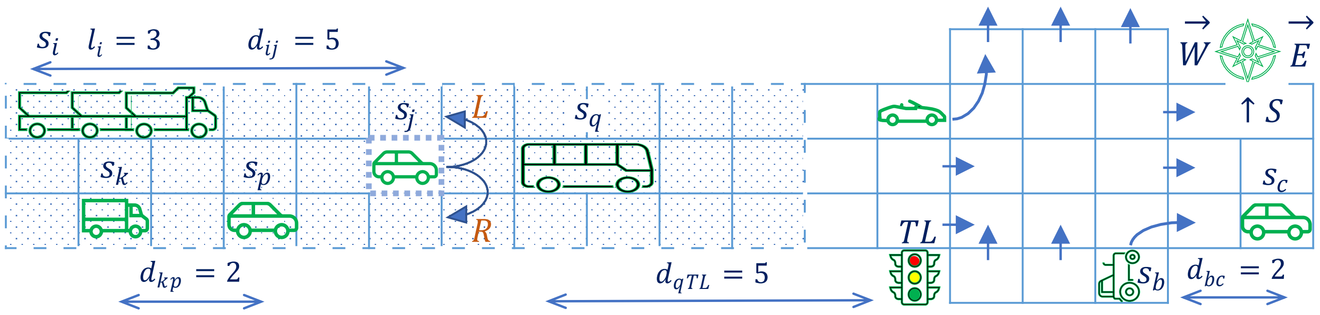


Figure 1: Traffic Cellular Automata: Physical Environment, Cells' States & Neighborhoods, and Transition Rules' Parameters

2 TRAFFIC CELLULAR AUTOMATA (TCA)

Road traffic simulation is used to plan road changes, optimize traffic lights, analyze throughput, integrate objects to name a few [2]. TCA yield realistic road traffic statistics at the macroscopic level [30].We constructed a complex automaton based on the literature [30] to challenge array DBMS principles, e.g. vehicles have several properties. Now we describe our model without referring to any array DBMS. Four main ingredients constitute CA: the physical environment, cells' states & neighborhoods, and local transition rules.

2.1 Physical Environment

We consider the entire road network as a 2-d lattice (a 2-d array). We take a traditional cell size of 7.5 m^2 [30]. A lattice cell state indicates an impassable part, traffic lights, a lane without a vehicle, or a vehicle with a non-negative speed. We account for vehicles of different lengths (l_i denotes the length of vehicle i): a vehicle occupies an integer number of maximum 3 consecutive cells, fig. 1.

In our model, we also consider road intersections (RIs). Each road consists of multiple lanes (3 in our model, without loss of generality). Vehicles are allowed to change lanes, e.g. in order to overtake a slower-moving vehicle. They can move west-east (WE) or south-north (SN) and change direction at a RI [30].

In our model, we locate Traffic Lights (TL) at the bottom left at each RI. TL are red for WE vehicles and green for SN vehicles or vice versa. In addition, TL can be yellow for all vehicles.

We use a traditional example of a TCA grid: an artificial road map resembling the Manhattan Grid, New York City, similar to [51]. Initially vehicles are scattered randomly on the grid. When a vehicle reaches a grid border, it appears at the opposite grid side.

2.2 Cells' States & Neighborhoods

Cells change their states at discrete time steps (iterations) of 1.2 seconds as in [30]. TCA evolve in time and space by applying rules to cells. A rule takes into account a cell's local neighborhood to take state transition decisions. Rules are applied to all lattice cells simultaneously (drivers make decisions independently of each other, in parallel, but obeying some generally accepted rules).

At each time step, a vehicle can stay in the same cell (no move), advance some cells forward, change its lane (the same road) or road (at a RI). The maximum vehicle speed $s_{max} = 3$ cells/step.

All cells in a window of 11×11 cells constitute the local neighborhood of a cell: the cell itself, 37.5 meters (5 cells) back and forward.

2.3 Local Transition Rules

A rule must check a set of constraints before deciding on the state change. For example, a rule for the left lane change must check for too close or fast-moving vehicles on the left to avoid collisions.

Move Forward Rules. TCA often use the Nagel-Schreckenberg one-dimensional model to advance a vehicle along the same lane [35]. The model defines four rules applied sequentially to the speed $s_k(t)$ and position $x_k(t)$ of vehicle k at time step t: (1,2) acceleration and braking: $s_k(t) = min\{s_k(t-1)+1, d_{kp}(t-1) - l_i, s_{max} - l_k + 1\}$, (3) randomization: $random(t) < const \Rightarrow s_k(t) = max\{0, s_k(t) - 1\}$, (4) movement: $x_k(t) = x_k(t-1) + s_k(t)$, where d_{kp} is the distance (number of cells) to the nearest vehicle p ahead of vehicle k.

Here a vehicle can stop directly behind a vehicle in front of it to avoid collisions. Rule 3 accounts for individual driver behavior. Note that the original model did not account for vehicle lengths.

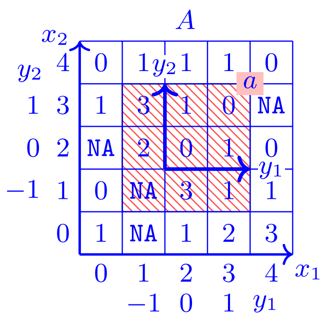
Lane Change Rules. Let us first describe lane changing rules from [30] which we will further complicate. A left/right lane change means shifting a vehicle a cell left/right. Certain constraints must be met to make the left lane change possible for vehicle j: $d_{ij} - l_i > 0$. This requires that there is no vehicle directly in the left cell and the closest vehicle i in the left lane, behind vehicle j, is at least one cell away [30]. Note that we added l_i to the original constraints.

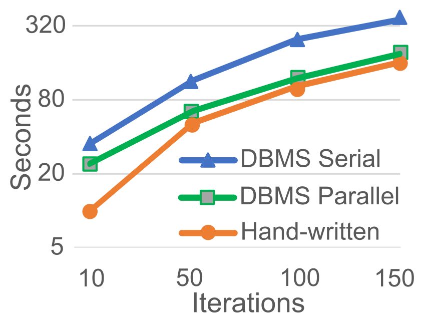
We extend the constraint to account not only for inter-vehicle distance d_{ij}, but also for vehicle speed s_i and allow left lane change if $d_{ij} - l_i - s_i > 0$, fig. 1. The right lane change rule is the same.

In our model, we allow a left lane change for vehicle j if a vehicle q in front of j is moving slower, e.g. $s_q < s_j$, fig. 1. To avoid scheduling conflicts, left and right lane changes are allowed only during odd and even time steps respectively. All lane changes are probabilistic.

Traffic light rules. TL help to avoid collisions at RIs. In our model, TL are green for $\gamma = 10$ ticks and turn yellow afterwards. TL are yellow until there are vehicles at the RI. Then TL turn green for the road for which TL were red before the yellow light. Vehicles apply an additional rule to account for TL: if TL are red or yellow, $s_q(t) = min(max(d_{qTL} - l_q, 0), s_q(t))$, where d_{qTL} is the distance to the nearest TL on the road ahead, fig. 1.

Road Crossing Rules. At a RI, in our model, a vehicle goes forward if it is not in an outer lane of a road. Otherwise, a WE/SN vehicle may randomly decide to go forward or turn left/right if it is in the left/right-most lane. Before turning, a vehicle checks for sufficient space in the lane to the left/right: $d_{bc} > l_b$, where d_{bc} is the distance to the nearest vehicle in the lane to the left/right, fig. 1.





```
cp tca.speed $speed ↵ [newline] cp tca.length $length
val {window} "-xWindowSize 11 -yWindowSize 11"
foreach {step} -from 0 -to 100 ↵ begin
  calc tca.lane:in $speed:in $length:in tca.temperature:in \
       --overwrite $speed_move:out $length_move:out    \
       -ot Int16 {window} -classfile "TCA.java"        \
       -method_name moveForwardPhase
  ... // comment: see http://gis.gg
  calc tca.lane:in $speed_lights:in $length_lights:in \
       --overwrite $speed:out $length:out           \
       -ot Int16 {window} -classfile "TCA.java"     \
       -method_name lightsPhase
  append $speed speedh ↵ append $length lenh ↵ end
```

Figure 2: Convolution Window, Speed of ChronosDB vs. Hand-Written Code, Native Array DBMS UDF for the Simulations

3 TRAFFIC SIMULATION BY ARRAY DBMS

Now we describe a holistic simulation workflow and answer the following questions. Which capabilities, relevant to CA simulations, do modern array DBMSs provide? Why can they not be readily applied to CA simulations? Which novel extensions did we develop to enable end-to-end CA simulations completely inside ChronosDB? Why is ChronosDB one of the best choices for CA simulations?

3.1 Simulation Setup

Let a 2-d array be the mapping $A : D_1 \times D_2 \mapsto \mathbb{T}$, $D_i = [0, l_i) \subset \mathbb{Z}$, $0 < l_i$ is a finite integer, $i \in \{1, 2\}$, and $\mathbb{T}$ is a numeric type, e.g. according to ISO/IEC 14882. l_i is said to be the *size* of ith dimension. We refer to a *cell* value of A of type $\mathbb{T}$ with integer indexes (x_1, x_2) as $A[x_1, x_2]$, where $x_i \in D_i$. We denote a missing value by NA [43].

The physical environment and cells' states can be modeled as 2-d arrays. Novel array DBMS extensions are driven by our new flexible convolution operator and native UDF language enabling us to apply local transition rules and code the simulation logic respectively.

Input: 2-d arrays ($lat \times lon$): `tca.lane` (road grid, cell values are −1: impassable area, 0/1: a lane with WE/SE moving direction, 2: a road intersection, 3: traffic lights), `tca.speed` and `tca.length` (initial vehicles' speeds and lengths, a vehicle of any length is modeled by a cell having its rear bumper; note that we do not store vehicle positions explicitly as they are implicitly coded by cell coordinates).

Output: history 3-d arrays ($time \times lat \times lon$): `speedh` and `lenh` (vehicles' speeds and lengths for each time step).

Goal: run the simulations for T time steps to derive statistics from history arrays (section 3.6) for decision support (section 2).

3.2 Novel Convolution Operator

A CA rule resembles a traditional convolution operator (CO) [48]. However, a CA rule is much more complex: it is a procedure accounting for constraints and updating any cells within the neighborhood. To support CA simulations, we introduce a new CO for array DBMS.

The convolution operator $\Xi : K, A_1, A_2, \ldots, A_n \mapsto B_1, B_2, \ldots, B_m$ takes n input and yields m output 2-d arrays, all shaped $l_1 \times l_2$. A kernel $K\langle k_1, k_2\rangle$ is the mapping $K : a_1^x, a_2^x, \ldots, a_n^x \mapsto b_1^x, b_2^x, \ldots, b_m^x$ of n input (windows of A_j,fig. 2) and m output 2-d arrays, all shaped $2k_1 + 1 \times 2k_2 + 1$, indexed by $(y_1, y_2) : y_i \in [-k_i, +k_i] \subset \mathbb{Z}$, $a_j^x[y_1, y_2] = A_j[x_1 + y_1, x_2 + y_2]$, $j \in [1, n]$, $a_j^x[y_1, y_2] =$ NA if $x_i + y_i \notin D_i$, $k_i \leqslant |l_i|$ div 2, $B_q[u_1, u_2] \in \{b_q^x[y_1, y_2] : x_i + y_i = u_i\}$ (choose the latest produced value), where $q \in [1, m]$, $x = (x_1, x_2)$, $x_i, u_i \in D_i$.

To compute K, users provide UDFs in Java. ChronosDB iterates over $x_i \in D_i$, forms readonly input and writable output windows, all equipped with helper functions, e.g. rotate 90/180/270° (adjusts the local coordinate system to use the same code for WE/SN vehicles).

There were efforts to develop dedicated languages for CA, but CA rules are inherently imperative and sophisticated. To date, it is more practical to use a modern language for convolution UDFs.

Unlike conventional convolution operators, our CO supplies a kernel several input windows and allows a kernel to modify an arbitrary number of cells throughout multiple output windows. However, this requires novel array retiling strategy (see below).

3.3 Native UDF Language for Array DBMS

We present the first native array DBMS language for UDFs. Although most array DBMSs can run Python/C++ UDFs, they are treated as black-boxes that cannot be optimized by DBMSs [6, 12, 43]. In addition, query languages and non-native UDFs are unable to organize iterations inherent to physical simulations. It is possible to run iterations query by query, but a query output must be completely materialized before running the next query and looking-ahead optimizations are unavailable as the "big picture" is unclear.

With strict formal definitions of array operations [43], compiler techniques (e.g., loop unrolling [1]), and our new extensions (e.g., array versioning & locks), we build execution plans for several iterations ahead (proactive exec plans). In this way we avoid redundant materialization and overheads of scheduling individual queries.

Native UDFs consist of commands that have the syntax similar to command line tools. Most users are familiar with such syntax and hopefully will find it easy to code UDFs. A part of the UDF for TCA simulations is in fig. 2. To retain the initial arrays, we start by copying `tca.speed` & `tca.length` to intermediate `$speed` & `$length` arrays which are subject to optimizations, e.g. they mostly reside in RAM. The loop will be executed 100 times.

The `calc` command runs a convolution operator that is supplied as a Java UDF: a `*.java` file that contains the specified method. The file is compiled to bytecode by ChronosDB for faster execution. `calc` accepts/produces an arbitrary number of input/output arrays, where the `-ot` parameter specifies $\mathbb{T}$. Quantiles `:in` and `:out` distinguish between the in/out arrays as their number is not fixed. The `--overwrite` flag forces `calc` to delete the output arrays if they exist and create new output arrays with the same name.

An iteration ends by appending new 2-d arrays `$speed` & `$length` to 3-d arrays `speedh` & `lenh` along the virtual *time* axis (see below).

The UDF in fig. 2 looks small, but is challenging to execute. For instance, during loop unrolling, the same array name appears several times: e.g., the last `calc` command deletes the current `$speed` array and creates a new array with the same name. As we are building a proactive exec plan, we must be able to keep and address all arrays (deleted and new) and to read/write all of them simultaneously.

3.4 Additional Improvements

Novel Retiling Strategy. Array DBMSs partition arrays into tiles to parallelize execution [32]. To support traditional COs, arrays are tiled with overlap [48]. However, our CO can update any set of cells: vehicles do not disappear and can move between the tiles. Hence, current tiling schemes are unsuitable for the new workload. To enable parallel CA simulations, we equipped ChronosDB with a new array retiling strategy to properly update array tiles and efficiently exchange cell values between the tiles at each iteration.

Array Versioning. To maintain several arrays with the same name during runtime, we introduced array versioning. ChronosDB refers to an array by its name and version. This allows us to build execution plans with deleted arrays upon which depend future, but not yet produced arrays. This is totally transparent to users, but required deep improvements to ChronosDB.

Array Locks are required for two reasons: to prevent other UDFs to operate on arrays from the common namespace and to control the state of arrays (exists, deleted, newer version created, etc.). Locks are always used in conjunction with array versioning.

Virtual axes. A traditional implementation of the `append` operator in fig. 2 coverts the input 2-d array into a 3-d form and may restructure the target 3-d array [12]. As `append` is frequent in our scenario, this may incur a large overhead. We introduced virtual axes which allow us to avoid restructuring arrays during split/merge operations, e.g. converting an N-d array to an $(N+1)$-d array. A virtual axis does not physically exist in an array and the cost of `append` is equal to copying an array. Note that we also extended ChronosDB to run operators on arrays with virtual axes.

3.5 Performance evaluation

We compared ChronosDB to a hand-written single-threaded code. Many TCA implementations are not maintained [8], not freely available [2], or do not implement the same set of rules as our model [34]. Hence, we ran the Java UDFs outside ChronosDB on input arrays shaped 4096×256 (Intel Core i5 2.5GHz, Windows 10, 256 GB SSD, 12 GB RAM). "DBMS Serial" shows no parallelization capabilities, "DBMS Parallel" runs simulations in 2 threads, fig. 2.

A considerable portion of time is spent for scheduling when the number of iterations is small. Almost 10 seconds is spent on compiling the Java UDFs. However, the performance of ChronosDB becomes comparable to a hand-written code when the number of threads and iterations increase. Please, recall that ChronosDB does not only run the simulations, but also provides many other benefits.

3.6 Array DBMS Shortcomings & Benefits

3.6.1 Shortcomings. Currently, ChronosDB introduces scheduling overhead to CA simulations and users need to write 2 types of UDFs: in Java and a new, but relatively simple language. Hence, numerous benefits outweigh a few 'cons' against using ChronosDB.

3.6.2 Benefits. Now we showcase the benefits of using ChronosDB for CA simulations. In this way, we also answer the question why ChronosDB is an excellent choice for this workload.

Data ingestion and fusion. CA rules may use other datasets, e.g. Digital Elevation Model or Surface Temperature (ST). To illustrate this benefit, we added a rule $random(t) < const \ \& \ ST < \theta \Rightarrow s_k(t) = max\{0, s_k(t) - 1\}$ (too low ST may slow down a vehicle): note the `tca.temperature` array in fig. 2. We used ChronosDB to prepare (ingest, resample, and slice) Landsat 8 satellite data for ST.

Automatic parallelization. ChronosDB partitions arrays into subarrays and runs the simulation independently on subarrays in parallel, managing all necessary data exchanges between them.

Debugging UDFs. It is easy to debug step-by-step TCA UDFs in Java used for `calc` commands, e.g. in IntelliJ IDEA. Just put `TCA.java` in a module visible for the IDE. Its powerful debugger makes it easy to code UDFs, track TCA decisions, and explore convolution windows rendered by `toString()` as ASCII 2-d maps showing nearby vehicles' speeds/lengths/locations & traffic lights.

Interactive visualization is essential for data understanding. ChronosDB provides array imagery via the open, popular protocol [56]. We extended ChronosDB to serve N-d arrays via WMTS, so its Web GUI [44] can now query ChronosDB to slice `speedh`/`lenh` and show `$speed`/`$length` for iteration step i on an interactive web map or animate `speedh`/`lenh` for all steps; see the paper homepage.

Data management: archive, query & compare the simulations.

Interoperability. ChronosDB storage layer is built on top of raw files in standard formats. Simulation arrays are full-fledged, georeferenced, GeoTIFF files readily accessible to other software. ChronosDB is also an HTTP server: users can download any array via HTTP. To showcase the benefit, at the paper homepage, we visualized simulation arrays in QGIS [41], a free and popular GIS. It is also possible to add ChronosDB as a WMTS layer in QGIS.

End-to-end simulations. ChronosDB serves all phases of the simulations within the single system: even computing statistics (the goal of simulations). For example, typical TCA simulation statistics include mean traffic density (the mean number of vehicles which passed through a cell) and space-mean speed (the mean vehicle speed for each cell) [30, 51]. A user can use an aggregation operator to compute those and interactively explore the results via WMTS.

4 CONCLUSIONS AND R&D OPPORTUNITIES

We presented a novel application of array DBMSs: physical simulations. As an example, we used road traffic cellular automata (TCA). For the first time, physical simulations run entirely inside an array DBMS. We had to introduce several extensions to the ChronosDB array DBMS to enable the simulations. It provides a wealth of benefits like automatic parallelization and interoperability.

Our work provides a number of future R&D opportunities. For example, load balancing strategies are required for irregularly distributed objects in input arrays. Efficient storage of sparse arrays would be beneficial (currently the grid array has only 5 distinct values stored uncompressed). Novel indexing techniques could allow skipping batches of missing values to avoid wasting iteration time.

Future work includes the application of array DBMSs to other fields like edge detection, fire spread, urban growth, land cover change, and other areas of cellular automata simulation.

REFERENCES

[1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. 1986. Compilers, principles, techniques. *Addison wesley* 7, 8 (1986), 9.

[2] AnyLogic. 2021. https://www.anylogic.com/road-traffic/

[3] Venkatramani Balaji, Alistair Adcroft, and Zhi Liang. 2019. Gridspec: A standard for the description of grids used in Earth System models. In *arXiv*.

[4] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*. 1363–1375.

[5] Peter Baumann et al. 2016. Big data analytics for Earth sciences: the EarthServer approach. *International Journal of Digital Earth* 9, 1 (2016), 3–29.

[6] Peter Baumann, Andreas Dehmel, Paula Furtado, et al. 1998. The multidimensional database system RasDaMan. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of Data*. 575–577.

[7] ArcGIS book. 2021. https://learn.arcgis.com/en/arcgis-imagery-book/

[8] Cellular Automaton Traffic Simulation. 2021. http://udel.edu/~mm/traffic/ca.html

[9] Chialin Chang, Bongki Moon, Anurag Acharya, Carter Shock, Alan Sussman, and Joel Saltz. 1997. Titan: a high-performance remote-sensing database. In *Proceedings 13th International Conference on Data Engineering*. IEEE, 375–384.

[10] Dalsu Choi, Chang-Sup Park, and Yon Dohn Chung. 2019. Progressive top-k subarray query processing in array databases. *PVLDB* 12, 9 (2019), 989–1001.

[11] CF Conventions. 2021. https://cfconventions.org/.

[12] Philippe Cudré-Mauroux, Hideaki Kimura, K-T Lim, Jennie Rogers, et al. 2009. A demonstration of SciDB: a science-oriented DBMS. *PVLDB* 2, 2 (2009), 1534–1537.

[13] Dask 2021. https://dask.org/.

[14] David J DeWitt, Navin Kabra, Jun Luo, Jignesh M Patel, and Jie-Bing Yu. 1994. Client-Server Paradise. In *VLDB*, Vol. 94. 558–569.

[15] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, et al. 2015. The BigDAWG Polystore System. *ACM SIGMOD Record* 44, 2 (2015), 11–16.

[16] Mohammad Farbod, Gholamreza Akbarizadeh, Abdolnabi Kosarian, and Kazem Rangzan. 2018. Optimized fuzzy cellular automata for synthetic aperture radar image edge detection. *Journal of Electronic Imaging* 27, 1 (2018), 013030.

[17] Yongjiu Feng and Xiaohua Tong. 2020. A new cellular automata framework of urban growth modeling by incorporating statistical and heuristic methods. *International Journal of Geographical Information Science* 34, 1 (2020), 74–97.

[18] Joana Gouveia Freire and Carlos Castro DaCamara. 2019. Using cellular automata to simulate wildfire propagation and to assist in fire management. *Natural hazards and earth system sciences* 19, 1 (2019), 169–179.

[19] GeoTrellis 2021. https://geotrellis.io/.

[20] Noel Gorelick et al. 2017. Google Earth Engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment* 202 (2017), 18–27.

[21] Min Han, Xue Yang, and Enda Jiang. 2016. An Extreme Learning Machine based on Cellular Automata of edge detection for remote sensing images. *Neurocomputing* 198 (2016), 27–34. https://doi.org/10.1016/j.neucom.2015.08.121 Advances in Neural Networks, Intelligent Control and Information Processing.

[22] Olha Horlova, Abdulrahman Kaitoua, and Stefano Ceri. 2020. Array-based Data Management for Genomics. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 109–120.

[23] Wenyu Jiang, Fei Wang, Linghang Fang, Xiaocui Zheng, Xiaohui Qiao, Zhanghua Li, and Qingxiang Meng. 2021. Modelling of wildland-urban interface fire spread with the heterogeneous cellular automata model. *Environmental Modelling & Software* 135 (2021), 104895.

[24] Jennifer Koch, Monica A Dorning, Derek B Van Berkel, Scott M Beck, Georgina M Sanchez, Ashwin Shashidharan, Lindsey S Smart, et al. 2019. Modeling landowner interactions and development patterns at the urban fringe. *Landscape and Urban Planning* 182 (2019), 101–113.

[25] Susana Ladra et al. 2017. Scalable and queryable compressed storage structure for raster data. *Information Systems* 72 (2017), 179–204.

[26] Adam Lewis et al. 2017. The Australian Geoscience Data Cube—Foundations and lessons learned. *Remote Sensing of Environment* (2017), 276–292.

[27] Meiyu Liu and Jing Shi. 2019. A cellular automata traffic flow model combined with a BP neural network based microscopic lane changing decision model. *Journal of Intelligent Transportation Systems* 23, 4 (2019), 309–318.

[28] Yuting Lu et al. 2019. Detection and prediction of land use/land cover change using spatiotemporal data fusion and the Cellular Automata–Markov model. *Environmental monitoring and assessment* 191, 2 (2019), 68.

[29] Hermano Lustosa, Fabio Porto, and Patrick Valduriez. 2019. SAVIME: A Database Management System for Simulation Data Analysis and Visualization.

[30] Sven Maerivoet and Bart De Moor. 2005. Cellular automata models of road traffic. *Physics reports* 419, 1 (2005), 1–64.

[31] John Mainzer et al. 2019. Sparse Data Management in HDF5. In *XLOOP*. 20–25.

[32] Parmita Mehta, Sven Dorkenwald, Dongfang Zhao, Tomer Kaftan, Alvin Cheung, Magdalena Balazinska, Ariel Rokem, Andrew Connolly, Jacob Vanderplas, and Yusra AlSayyad. 2017. Comparative evaluation of big-data systems on scientific image analytics workloads. *PVLDB* 10, 11 (2017), 1226–1237.

[33] Hossein Shafizadeh Moghadam and Marco Helbich. 2013. Spatiotemporal urbanization processes in the megacity of Mumbai, India: A Markov chains-cellular automata urban growth model. *Applied Geography* 40 (2013), 140–149.

[34] MovSim. 2021. https://github.com/movsim/movsim

[35] Kai Nagel and Michael Schreckenberg. 1992. A cellular automaton model for freeway traffic. *Journal de physique I* 2, 12 (1992), 2221–2229.

[36] Stefano Nativi et al. 2008. Unidata's Common Data Model mapping to the ISO 19123 Data Model. *Earth Sci. Inform.* 1 (2008), 59–78.

[37] Vasileios G Ntinas, Byron E Moutafis, Giuseppe A Trunfio, and Georgios Ch Sirakoulis. 2017. Parallel fuzzy cellular automata for data-driven simulation of wildfire spreading. *Journal of computational science* 21 (2017), 469–485.

[38] Carlos Ordonez, Yiqun Zhang, and S Lennart Johnsson. 2019. Scalable machine learning computing a data summarization matrix with a parallel array DBMS. *Distributed and Parallel Databases* 37, 3 (2019), 329–350.

[39] Stavros Papadopoulos, Kushal Datta, Samuel Madden, and Timothy Mattson. 2016. The TileDB Array Data Storage Manager. *PVLDB* 10, 4 (2016), 349–360.

[40] PostGIS 2021. http://postgis.net/.

[41] Quantum GIS. 2021. https://www.qgis.org/.

[42] Lucas C Villa Real, Bruno Silva, et al. 2019. Large-scale 3D geospatial processing made possible. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 199–208.

[43] Ramon Antonio Rodriges Zalipynis. 2018. ChronosDB: Distributed, File Based, Geospatial Array DBMS. *PVLDB* 11, 10 (2018), 1247–1261.

[44] Ramon Antonio Rodriges Zalipynis. 2019. ChronosDB in Action: Manage, Process, and Visualize Big Geospatial Arrays in the Cloud. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30-July 5, 2019*. ACM, 1985–1988. https://doi.org/10.1145/3299869.3320242

[45] Ramon Antonio Rodriges Zalipynis. 2019. Evaluating Array DBMS Compression Techniques for Big Environmental Datasets. In *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Vol. 2. IEEE, 859–863. https://doi.org/10.1109/IDAACS.2019.8924326

[46] Ramon Antonio Rodriges Zalipynis. 2020. BitFun: Fast Answers to Queries with Tunable Functions in Geospatial Array DBMS. *PVLDB* 13, 12 (2020), 2909–2912.

[47] Ramon Antonio Rodriges Zalipynis. 2021. Towards Machine Learning in Distributed Array DBMS: Networking Considerations. In *Machine Learning for Networking (Lecture Notes in Computer Science, Vol. 12629)*. 284–304. https://doi.org/10.1007/978-3-030-70866-5_19

[48] Ramon Antonio Rodriges Zalipynis et al. 2018. Array DBMS and Satellite Imagery: Towards Big Raster Data in the Cloud. In *Analysis of Images, Social Networks and Texts – 6th International Conference, AIST 2017, Moscow, Russia, July 27-29, 2017, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10716)*. Springer, 267–279. https://doi.org/10.1007/978-3-319-73013-4_25

[49] Oracle Spatial. 2020. oracle.com/database/technologies/spatialandgraph.html.

[50] Xiaohua Tong and Yongjiu Feng. 2020. A review of assessment methods for cellular automata models of land-use change and urban growth. *International Journal of Geographical Information Science* 34, 5 (2020), 866–898.

[51] Ozan K Tonguz et al. 2009. Modeling urban traffic: a cellular automata approach. *IEEE Communications Magazine* 47, 5 (2009), 142–150.

[52] Martin Treiber and Arne Kesting. 2013. *Traffic Flow Dynamics: Data, Models and Simulation*. Springer-Verlag Berlin Heidelberg.

[53] Sebastian Villarroya and Peter Baumann. 2020. On the Integration of Machine Learning and Array Databases. In *ICDE*. IEEE, 1786–1789.

[54] Yi Wang, Arnab Nandi, and Gagan Agrawal. 2014. SAGA: Array storage as a DB with support for structural aggregations. In *Proceedings of the 26th International Conference on Scientific and Statistical Database Management*. 1–12.

[55] Climate Wikience. 2021. http://www.wikience.org/.

[56] WMTS. 2021. https://www.opengeospatial.org/standards/wmts.

[57] Haoyuan Xing and Gagan Agrawal. 2020. Accelerating array joining with integrated value-index. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*. 145–156.

[58] Weijie Zhao et al. 2016. Similarity join over array data. In *Proceedings of the 2016 International Conference on Management of Data*. 2007–2022.

[59] Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu, Anna YQ Ho, and Peter Nugent. 2018. Distributed caching for processing raw arrays. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 1–12.

[60] Weijie Zhao, Florin Rusu, Bin Dong, Kesheng Wu, and Peter Nugent. 2017. Incremental view maintenance over array data. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 139–154.