International Journal of Geographical Information Science, 2016
Vol. 30, No. 3, 494–514, http://dx.doi.org/10.1080/13658816.2015.1039538

A hybrid parallel cellular automata model for urban growth simulation over GPU/CPU heterogeneous architectures

Qingfeng Guan[a,b], Xuan Shi[c]*, Miaoqing Huang[d] and Chenggang Lai[d]

[a]*National Engineering Research Center of GIS, China University of Geosciences (Wuhan), Wuhan, Hubei 430074, China;* [b]*Faculty of Information Engineering, China University of Geosciences (Wuhan), Wuhan, Hubei 430074, China;* [c]*Department of Geosciences, University of Arkansas, Fayetteville, AR 72701, USA;* [d]*Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA*

(*Received 13 August 2014; final version received 29 March 2015*)

As an important spatiotemporal simulation approach and an effective tool for developing and examining spatial optimization strategies (e.g., land allocation and planning), geospatial cellular automata (CA) models often require multiple data layers and consist of complicated algorithms in order to deal with the complex dynamic processes of interest and the intricate relationships and interactions between the processes and their driving factors. Also, massive amount of data may be used in CA simulations as high-resolution geospatial and non-spatial data are widely available. Thus, geospatial CA models can be both computationally intensive and data intensive, demanding extensive length of computing time and vast memory space. Based on a hybrid parallelism that combines processes with discrete memory and threads with global memory, we developed a parallel geospatial CA model for urban growth simulation over the heterogeneous computer architecture composed of multiple central processing units (CPUs) and graphics processing units (GPUs). Experiments with the datasets of California showed that the overall computing time for a 50-year simulation dropped from 13,647 seconds on a single CPU to 32 seconds using 64 GPU/CPU nodes. We conclude that the hybrid parallelism of geospatial CA over the emerging heterogeneous computer architectures provides scalable solutions to enabling complex simulations and optimizations with massive amount of data that were previously infeasible, sometimes impossible, using individual computing approaches.

Keywords: parallel computing; cellular automata; GPU; heterogeneous architecture

1. Introduction

1.1. Geospatial cellular automata for spatial optimization and decision-making

A classical cellular automata (CA) model is established based on a set of cells with identical shape and size. Each cell is located in a regular, discrete space (i.e., cellspace) and is associated with a state from a finite set, indicating its attribute or condition. The model evolves in discrete time steps (or iterations), changing the states of its cells according to a collection of transition rules, homogeneously and synchronously applied over the cellspace at every step. The new state of a cell depends on the previous states of a set of cells, which may include the cell of interest, and its neighbors.

With the naturally embedded space and time properties, CA provide a straightforward approach for spatiotemporal dynamic simulations, and have been widely adopted in

*Corresponding author. Email: xuanshi@uark.edu geospatial studies and spatial decision-making applications, such as land-use and land-cover change (LULCC) (Clarke *et al.* 1997, Couclelis 1997, White *et al.* 1997, Clarke and Gaydos 1998, Wu and Webster 1998, Li and Yeh 2000, 2001, Yeh and Li 2002, Silva and Clarke 2002), wildfire propagation (Clarke *et al.* 1995), and freeway traffic (Nagel and Schreckenberg 1992, Benjamin *et al.* 1996).

The core of geospatial CA models is to determine the status changes of cells through the application of the transition rules. When used in spatial decision-making applications (e.g., LULCC simulations and planning), the model essentially identifies the most possible and/or suitable locations (i.e., cells within the cellspace) for certain types of spatiotemporal dynamics to occur. Such location identification process entails a natural connection with optimization problems, which focus on selecting the optimal option to achieve certain objectives while subjecting to a set of constraints and criteria. The integration of geospatial CA and optimization have been seen in the following three forms.

(1) Using an optimization model to generate the macro-scale quantitative constraints for CA. CA are bottom-up models, meaning the evolvements of regional dynamics are presented through the local changes of cells. Macro-scale constraints that are derived through top-down approaches are often used to limit the number of cells to change. For example, Ward *et al.* (2003) used a linear programming optimization model to determine the optimal fraction of residential area (i.e., number of cells) of each planning unit to accommodate the projected population while maintaining environmental targets, and a geospatial CA model to spatially allocate the residential lands within the study area.

(2) Using an optimization method to determine the parameters of CA's transition rules. Many geospatial CA models involve preset transition rules and a set of parameters that control the behaviors of the transition rules. To generate realistic or optimal simulation results, the parameters must be carefully set, often through calibrations using historical datasets. A variety of optimization methods have been used in the calibration of CA models to search for the optimal parameters within a large multidimensional parameter space, such as self-organizing map (SOM) (Dietzel and Clarke 2007), genetic algorithm (GA) (Li *et al.* 2013), and particle swarms optimization (PSO) (Blecic *et al.* 2014).

(3) Using an optimization model as CA's transition rule. Utilizing spatial optimization algorithms and models to construct CA's transition rules is able to locate the optimal (i.e., most possible or suitable) cells to change in order to meet certain requirements, thus largely reduce the subjectivity in the transition rules. For example, Fotakis and Sidiropoulos (2012) used a multi-objective self-organizing algorithm (MOSOA) in the transition rules of a CA to optimize both land use and water allocation, and to achieve minimal environmental pollution and maximal economic profit. Other optimization models used as CA transition rules include artificial neural networks (ANNs) (Li and Yeh 2002), ant colony optimization (ACO) (Liu *et al.* 2008), artificial immune systems (AISs) (Liu *et al.* 2010), and bee colony optimization (BCO) (Yang *et al.* 2013).

1.2. Computational intensity of geospatial CA

The computational intensity of a CA model depends on both the computational complexity of its transition rules and the size of the datasets to be used. Many geospatial CA models include complex algorithms in the transition rules and/or the calibration process, which demand a large amount of computing power and usually lead to an extensive length of computing time. For example, a Brute-Force calibration of SLEUTH (a CA model for urban LULCC simulation) with a medium-sized dataset and minimal data layers requires about 1200 CPU hours on a workstation (Clarke 2003). Even though some machinelearning-based optimization methods are able to identify the optimal parameters or locations quicker than the Brute-Force approach, the computational complexity can still be quite high. For example, the ANN's computational complexity largely depends on the structure of the network (including the number of layers, the number of neurons on each layer, and the activation transfer functions connecting layers) and the learning algorithm. The more complex are the ANN's structure and learning algorithm, the more time the computation consumes (Long and Gupta 2008).

Data is another source of computational intensity for geospatial CA models. Besides the data layer that represents the geospatial phenomenon of interest, many geospatial CA models require several other data layers representing the spatial distributions of the driving factors that pose impacts on the phenomenon. In the Big Data era, high-resolution data have been widely available and used in geospatial studies. When massive amount of high-resolution data are used, not only the computational intensity of a CA model will increase extensively, but also the computational performance may be further degraded. The random access memory (RAM) on a single workstation may not be big enough to accommodate all the data. Consequently, a large proportion of the data has to temporarily reside in the virtual memory (i.e., hard disk) whose accessing speed is much slower, which greatly increases the overall computing time, and in worse cases, leads to run-time failure (Guan and Clarke 2010).

Therefore, geospatial CA models can be both computationally intensive and data intensive, requiring extensive, sometimes intractable, length of computing time and amount of memory space, which largely degrade the applicability, feasibility, and scalability of geospatial CA in real-world problem-solving and spatial optimization.

1.3. Parallel cellular automata

Parallel computing, in contrast to serial computing, which performs the computation on a single computing unit (e.g., a CPU or CPU core) in sequence, divides and distributes the computation (data and/or task) to multiple computing units to be carried out concurrently and synchronously. Powered by high computational capability (both computing power and memory space), distributed and parallel computing infrastructures and technologies have been increasingly adopted to perform complex geospatial computation over massive amount of data within a feasible length of time, which can hardly be handled by traditional serial computing approaches (for example, Tang and Wang 2009, Zhang and You 2013, Shi and Ye 2013, Qin *et al.* 2014, Shi *et al.* 2014a, 2014b; Laura *et al.*). With the advancement of cyberinfrastructure, large-scale parallel computing systems and platforms are becoming widely available for geospatial studies and applications (Wang 2010).

The classical CA model has been recognized to be a natural parallel computing system, as the transition rules are applied to cells homogeneously and synchronously (Bandini *et al.* 2001). The commonly used approach for parallelizing CA models is data parallelism, which decomposes (or divides) the cellspace into multiple sub-cellspaces and assigns them onto multiple computing units. Each computing unit is responsible for applying the transition rules on the cells within one or more sub-cellspace(s), thus the memory requirement per computing unit is reduced, and more significantly, the computational speed and scalability can be greatly enhanced. The advantage of data parallelism is that the majority of CA models can be easily parallelized, regardless of the algorithms used in the transition rules (including the optimization methods mentioned above).

Two issues must be handled appropriately in the data parallelism of geospatial CA computation. First, multiple layers of data are often used in geospatial CA models.

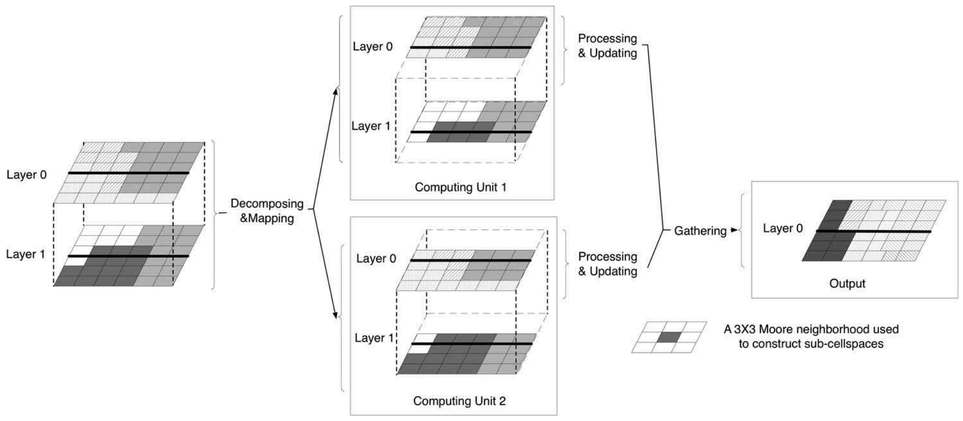


Figure 1. Decomposing multiple layers for parallel computing.

Therefore, the decomposition process must assure all layers to be divided in exactly the same manner, such that the sub-cellspaces across layers will match in locations and dimensions (Figure 1).

Second, the transition rule of a CA model is essentially a focal (also termed moving-window or neighborhood-scope) operation in raster data processing. When evaluating a certain cell, the transition rule requires the attributes of the cell itself, as well as those of its neighbors. Consequently, after decomposition, each sub-cellspace should contain not only the block of cells to be processed locally, but also a ring of 'halo' cells as the neighbors of the edge cells (Figure 2). The halo cells are actually the replica of the edge cells in the neighboring sub-cellspaces. At the end of each time step, the status of halo cells must be updated according to their origins, leading to communications between computing units if the discrete memory architecture[1] is used.

Some efforts have been made in the last few years to parallelize geospatial CA models. Guan and Clarke (2010) developed a parallel geospatial CA model, pSLEUTH, for the simulation of urban growth, using the parallel Raster Processing Library (pRPL). Li *et al.* (2010) developed a parallel geospatial CA model for urban growth simulation that includes a line-scanning method for load balancing. Cheng *et al.* (2012) developed a CA-based model to parallelize terrain analysis processes.

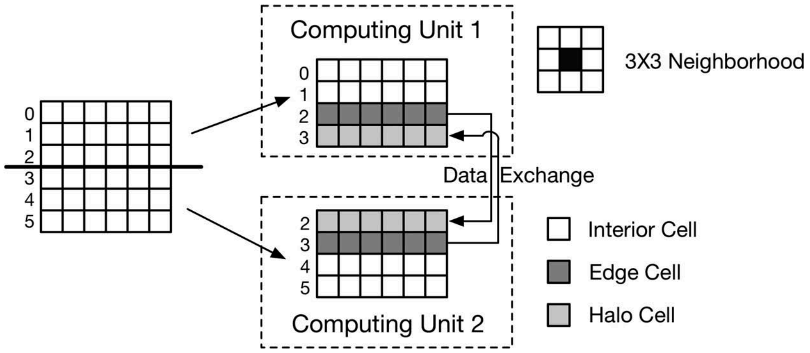


Figure 2. Halo, edge, and interior cells of sub-cellspaces.

1.4. GPUS and GPU/CPU heterogeneous architectures

All of the above-mentioned efforts in parallelizing CA computation are based on traditional CPU-only architectures. The last few years have seen the emergence of some new computer architectures and computing technologies, such as graphics processing units (GPUs) and Many Integrated Core (MIC) CPUs. Compared with a multi-core CPU (excluding MIC CPUs), a GPU is able to integrate hundreds, even thousands, of cores on a single chip, thereby providing much higher computing power.

Even though GPUs were originally developed for graphics processing, modern GPUs have the capability of general-purpose computation (i.e., GPGPU). Within NVIDIA's Compute Unified Device Architecture (CUDA), for example, a GPU works as the co-processor of a CPU (Sanders and Kandrot 2011). The CPU works as the host that reads the data into the RAM and copies it to the GPU. The GPU allocates a number of concurrent threads to apply the algorithm on the data. After the computation, the result is copied from the GPU back to the CPU's RAM for output or further processing.

With their roots in graphics domain, GPUs are very suitable for matrix manipulation and processing, which is similar to CA computation. Some efforts have been made to implement parallel CA models on single GPUs (Thor 2008, Li *et al.* 2012, Blecic *et al.* 2014). In GPU-based CA models, each GPU thread is responsible for processing a group of cells (ideally one cell) in the cellspace. With tens of thousands of concurrent threads, the computing time can be significantly reduced using a GPU compared with a CPU.

However, parallel computing on a single GPU still has certain limitations: the memory and number of threads on a GPU are often limited and insufficient when handling large volume of data. For example, the datasets used in this study include a total of seven input layers, four intermediate layers, and one output layer, which require over 20GB of memory, while NVIDIA's latest K20 GPU only has 6GB of memory. When the amount of data exceeds the capacity of the GPU's memory, the data has to be divided into smaller subsets that can fit in the GPU's memory and be iteratively processed by the GPU, which causes multiple data transfers between the GPU and CPU, and hence degrades the performance. Even though the data streaming technique can be used to overlap the computation and communication (Wang *et al.* 2011), the performance can be further improved if all data reside in the memory.

The GPU/CPU heterogeneous computer architecture, which integrates multiple GPUs and CPUs, has been increasingly adopted in various cyberinfrastructure environments to build HPC computing platforms for super large-scale scientific and engineering computations. Examples include some of the most powerful computer systems in the world, such as Titan (no. 2 on the Top500 list as of June 2014), Piz Daint (no.6), HPC2 (no. 11), and Tianhe-1A (no. 14).[2] There are two advantages of the GPU/CPU heterogeneous architecture:

(1) Multiple GPUs are able to accommodate a much larger number of concurrent threads than a single GPU, leading to higher computing power.

(2) With multiple GPUs, comes much larger combined GPU memory space. The dataset can be partitioned into multiple subsets and assigned to multiple GPUs. Ideally, the subset is small enough to fit in the memory of each GPU such that only one round of GPU–CPU data transfers are needed.

This study aims to utilize the GPU/CPU heterogeneous architecture to enable complex CA simulations for urban growth simulation over massive datasets that might be previously infeasible, or intractable, using traditional individual computing approaches, to provide quick, and more importantly, scalable support for land allocation decision-making. With the rapid adoption of cyberinfrastructure in geospatial studies and applications, this work is expected to shed light on utilizing the latest HPC technologies available over the cyberinfrastructure to enable complex GIS processing, including various optimization methods.

2. Urban growth simulation using geospatial cellular automata

A large number of geospatial CA models were proposed for urban growth simulation in the past decades (Liu and Phinn 2003, Clarke *et al.* 2007, Li *et al.* 2008, 2013, Feng and Liu 2013). Among them, one typical model was developed by Wu (2002). It integrated a logistic regression model to identify the relationships between urban growth and a variety of spatial factors (termed site attributes), and a set of CA transition rules that take into account the neighborhood properties, exclusionary preferences, distance-decay functions, macro-scale constraints, and random selections during the growth of urban lands. Users are allowed to control the selection of spatial factors (i.e., data layers of site attributes, including elevation, slope, distance to transportations and major facilities) that are assumed to affect the urbanization process, the exclusion layer defining the areas where urbanization cannot occur (e.g., parks, natural reserves, water bodies, and other types of restricted areas) or defining the development preferences of lands, the distance-decay parameter that defines the dispersion rate, and the macro-scale constraint that defines the maximal amount of urbanization at each time step.

Such a geospatial CA model provides an effective means to derive and test optimal urban land allocation strategies and to generate what-if scenarios by combining various data layers and parameters. For example, the exclusion layer can be generated using a spatial optimization model that gives higher values to preferred lands for urbanization such that these lands will be more likely to be urbanized during the simulation, and lower values to less preferred lands. However, when large volume of data is used, the model can hardly be accomplished due to the limited computing power and memory on a desktop computer equipped with a single CPU. For this reason, this model was chosen in our study to demonstrate how GPU/CPU heterogeneous computer architectures and systems can overcome the computational barrier and improve the applicability, feasibility, and scalability of geospatial CA models.

To help understand the details of the model, the rest of this section elaborates on the four transition rules of this geospatial CA model.

(1) The global conversion probability rule is defined as follows:

$$p_{\mathrm{g}} = \frac{\exp\left(a + \sum_k b_k x_k\right)}{1 + \exp\left(a + \sum_k b_k x_k\right)} \tag{1}$$

where p_{g} is the global probability for a cell to convert to urban, ranging within [0, 1]; a is a constant; b_k is the weight associated with a particular site attribute x_k, representing the direction and magnitude of x_k's influence on urbanization; and x_k represents a site attribute. Parameters a and b_k can be determined using a logistic regression process with a collection of samples out of historical datasets. Each sample includes a set of site attribute values and the corresponding urbanization conversion (i.e., 1 for urbanization, and 0 for non-urbanization) at a particular location (i.e., a cell within the cellspace). Once a and all b_k values are obtained, the global conversion probabilities for all cells can be calculated using a set of data layers representing the current or future statuses of site attributes. Such a global probability surface represents the suitability of urbanization for all cells within the study area, and serves as the basis for the following iterative evolution of the CA.

(2) The joint conversion probability rule is defined as follows:

$$p_{\mathrm{c}}^{t} = p_{\mathrm{g}} \times excl \times \frac{\sum_{3\times 3} \mathrm{con}\left(s_{ij}^{t-1} = \mathrm{urban}\right)}{3\times 3 - 1} \tag{2}$$

where p_c^t is the joint probability for a particular cell to convert to urban at time t, ranging within [0, 1]; *excl* is the exclusion status of the cell that ranges within [0, 1] (i.e., 0 means the cell is absolutely excluded from urbanization, and 1 means the cell has no restriction or preferential obstacle for urbanization); and $\frac{\sum_{3\times 3} \mathrm{con}\left(s_{ij}^{t-1}=\mathrm{urban}\right)}{3\times 3-1}$ calculates the urban density of the cell's 3 × 3 neighborhood at time $t - 1$. The joint probability combines the global probability calculated above with the exclusionary preference and neighborhood properties at a certain location. As mentioned above, the exclusion layer is provided by the user, which can be used as a planning tool to implement some spatial optimization (e.g., land allocation) strategies that determine the development preferences of cells. The higher is the *excl* value (meaning more preferred for urbanization), the higher the joint probability. Likewise, a larger urban density of the cell's neighborhood leads to a higher joint probability. The rationale is quite straightforward. A piece of land is more likely to be converted to urban when a large proportion of its neighboring lands have already been urbanized.

(3) The distance-decay conversion probability rule is defined as follows:

$$p_{\mathrm{d}}^{t} = p_{\mathrm{c}}^{t} \times \exp\left(-\delta \times \frac{1 - p_{\mathrm{c}}^{t}}{\max\left(p_{\mathrm{c}}^{t}\right)}\right) \tag{3}$$

where p_{d}^t is the distance-decay conversion probability for a cell to convert to urban at time t, ranging within [0, 1]; δ is the dispersion parameter, and $\max\left(p_{\mathrm{c}}^{t}\right)$ returns the highest joint probability of all cells at time t, i.e., the best cell's evaluation score. The function $\exp\left(-\delta \times \frac{1-p_{\mathrm{c}}^{t}}{\max\left(p_{\mathrm{c}}^{t}\right)}\right)$ determines the 'distance' between the score of the best cell and that of the cell to be evaluated. δ is a user-defined parameter ranging within [1, 10], which controls the shape of the skewed probability curve. The higher is the value of δ, the steeper is the distance-decay gradient. This transition rule uses the best cell as the benchmark and evaluates all cells by comparing their scores with the best cell's score. The 'closer' is a cell's score to the best score, the more likely it is to be urbanized.

(4) The final conversion rule contains two steps. First, the final conversion probability is defined as follows:

$$p_{\mathrm{s}}^{t} = q \times p_{\mathrm{d}}^{t} \times \sum_{ij} p_{\mathrm{d}}^{t} \tag{4}$$

where p_{s}^t is the final probability for a cell to convert to urban at time t, ranging within [0, 1]; q is the quantitative constraint representing the maximal number of cells to be converted at each time step; and $\sum_{ij} p_{\mathrm{d}}^t$ calculates the sum of distance-decay probabilities of all cells at time t. The parameter q is provided by the user, which can be obtained using a macro-scale optimization model that determines the optimal number of cells for urbanization (for example, see Ward *et al.* 2003). A higher value of q allows for more cells to be converted to urban, thus a particular cell is more likely to be urbanized, as indicated in Equation (4).

Eventually, the final conversion of a cell is determined as follows:

$$s^t = \begin{cases} \text{urban}, & p_{\mathrm{s}}^t > \text{rand}() \\ \text{non-urban}, & p_{\mathrm{s}}^t \leq \text{rand}() \end{cases} \tag{5}$$

where s^t is the urbanization status of a cell at time t, and rand() generates a random number with uniform distribution within the range of [0, 1]. The random number generator embedded in the final conversion enables a random selection procedure that is often seen in real-world urbanization processes, allowing for some degree of uncertainty in the simulation.

Transition rule (1), i.e., the global probability calculation, is executed once at the beginning, before the iterative evolution, to generate the global conversion probability surface. Then, transition rules (2), (3), and (4) are iteratively executed for multiple time steps (each representing a year) during the CA simulation.

3. Hybrid parallelism of geospatial CA

A hybrid parallelism strategy was used to parallelize the above geospatial CA model over the GPU/CPU heterogeneous architecture (Figure 3). On the computer cluster level, the Message Passing Interface (MPI), a commonly used parallel computing model for distributed memory systems (Gropp *et al.* 1998), was used to handle the domain decomposition and data I/O, and to coordinate the computation of multiple GPU/CPU nodes and the data exchange between nodes. On the GPU/CPU node level, the CUDA (see Sanders and Kandrot 2011) was used to coordinate the data transfers between the CPU and GPU, and to allocate multiple concurrent GPU threads to apply the transition rules on the sub-domain that is assigned to the node.

3.1. MPI-based computing

MPI is used to coordinate multiple parallel processes, each of which forms a logical computing node that includes a CPU (or CPU core) and a GPU. Specifically, an MPI process performs the following tasks (Figure 4):

(1) initializing the MPI protocol to form a logical computing node (including a CPU or CPU core as the 'host', and a GPU as the 'device') for parallel computing;
(2) retrieving the spatial dimensions of all input layers (i.e., the numbers of rows and columns) from the input files;
(3) decomposing the whole domain (i.e., cellspace) into multiple sub-domains (i.e., sub-cellspaces) and assigning (also termed mapping) them onto computing nodes (one sub-domain per node);

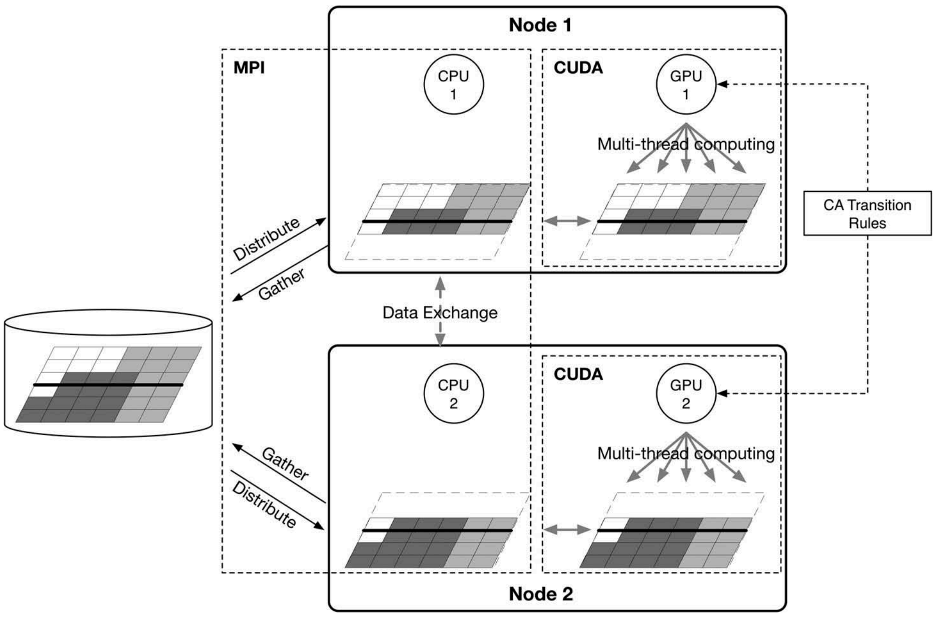


Figure 3. Hybrid parallelism of the geospatial CA model.

(4) reading the subset of input data according to the assigned sub-domain;
(5) executing a CUDA-based computing procedure on the GPU to apply transition rule (1) of CA to generate the global conversion probability layer (including copying the input layers from CPU to GPU, i.e., CPU→GPU transfers, before the computation);
(6) executing a CUDA-based computing procedure on the GPU to apply transition rule (2) of CA to update the joint conversion probability layer (including finding the highest joint probability in the sub-domain and copying it from GPU to CPU, i.e., GPU→CPU transfer);
(7) finding the highest joint conversion probability in the whole domain through communications with other processes (i.e., CPU↔CPU transfers);
(8) executing a CUDA-based computing procedure on the GPU to apply transition rule (3) of CA to update the distance-decay conversion probability layer (including calculating the sum of distance-decay probability in the sub-domain and copying it from GPU to CPU, i.e., GPU→CPU transfer);
(9) calculating the sum of distance-decay conversion probability in the whole domain through communications with other processes (i.e., CPU↔CPU transfers);
(10) executing a CUDA-based computing procedure on the GPU to apply transition rule (4) of CA to update the final conversion probability layer and update the output layer (including summarizing the number of urbanized cells in the sub-domain, and copying the edge cells or all cells from GPU to CPU, i.e., GPU→CPU transfers, after the computation);

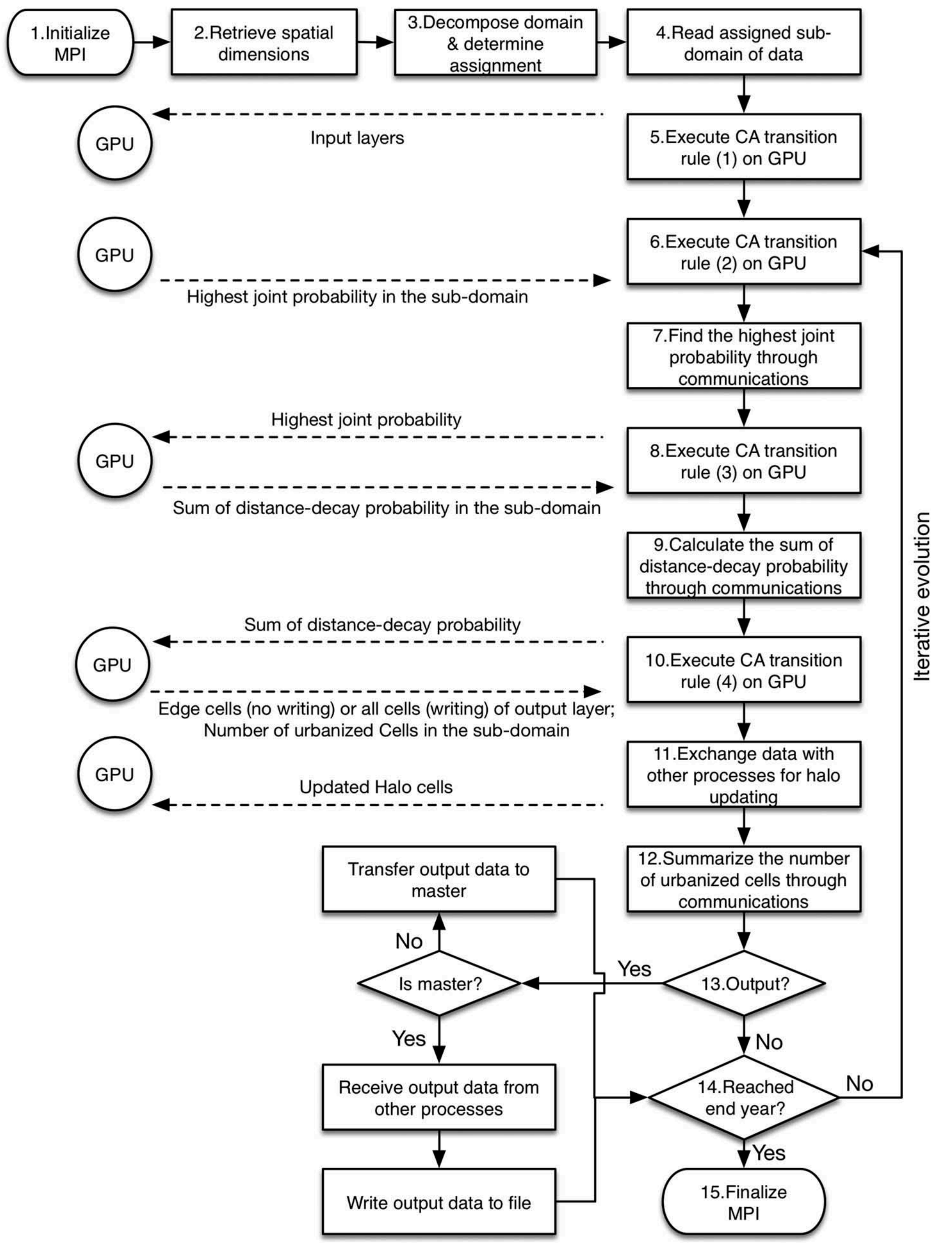

1.Initialize MPI
2.Retrieve spatial dimensions
3.Decompose domain & determine assignment
4.Read assigned sub-domain of data
GPU
Input layers
5.Execute CA transition rule (1) on GPU
GPU
Highest joint probability in the sub-domain
6.Execute CA transition rule (2) on GPU
7.Find the highest joint probability through communications
Highest joint probability
GPU
8.Execute CA transition rule (3) on GPU
Sum of distance-decay probability in the sub-domain
9.Calculate the sum of distance-decay probability through communications
Sum of distance-decay probability
GPU
10.Execute CA transition rule (4) on GPU
Edge cells (no writing) or all cells (writing) of output layer; Number of urbanized Cells in the sub-domain
GPU
Updated Halo cells
11.Exchange data with other processes for halo updating
12.Summarize the number of urbanized cells through communications
Transfer output data to master
No
Is master?
Yes
13.Output?
Yes
No
Receive output data from other processes
14.Reached end year?
No
Yes
Write output data to file
15.Finalize MPI
Iterative evolution

Figure 4. Flowchart of an MPI process for the parallel geospatial CA model.

(11) evoking communications with other processes (i.e., CPU↔CPU transfers) to exchange data of edge cells, and update ‘halo’ cells’ status (including copying the updated halo cells from CPU to GPU, i.e., CPU→GPU transfers);
(12) summarizing the total number of urbanized cells through communications with other processes (i.e., CPU↔CPU transfers) at each time step;

(13) writing the result to the output file if needed (including the master process gathering sub-domains of the output layer from all processes through CPU→CPU transfers, before writing);
(14) checking the iteration status and terminating the computation when the final time step is reached, otherwise going back to step (6);
(15) finalizing the MPI protocol and terminating the program.

A variety of decomposition methods can be used to divide the whole spatial domain of the study area into sub-domains, including regular row-wise, column-wise, and block-wise decomposition methods, as well as irregular ones such as quad-tree-based (QTB) decomposition. As shown in Guan and Clarke's (2010) study, row-wise and column-wise methods usually outperform the block-wise method, because a sub-domain generated by the block-wise decomposition has more neighboring sub-domains to exchange data with for the 'halo' cell updating. Also, even though the QTB method is able to divide the data according to the spatial distribution of workload and generate more workload-balanced sub-domains, it is not suitable for CA computations, because the spatial distribution of workload may change as the values of cells change along the CA evolution. In this study, we used a simple row-wise decomposition method to generate a sub-domain for each MPI process. The decomposition is a collective operation for all processes, meaning all processes first retrieve the spatial dimensions of the data layers and determine their own sub-domains (i.e., ranges of rows) according to their unique MPI rank IDs. Such a decomposition strategy can also be used for column-wise and block-wise decomposition methods.

The Geospatial Data Abstraction Library (GDAL), an open-source raster data I/O library (http://www.gdal.org), is used for reading and writing data. Once the whole spatial domain is decomposed and the sub-domains are assigned to MPI processes, each process evokes a GDAL I/O procedure to read the data according to its assigned sub-domain. Qin *et al.* (2013) have demonstrated that reading a dataset in parallel using GDAL is allowed for some raster formats such as GeoTIFF. However, parallel writing using GDAL may generate incorrect output files. Thus, we implemented a centralized writing mechanism for the parallel CA model. When the computations on all processes are complete, the master process gathers the sub-datasets of simulation results from other processes through data transfer communications, and writes to the final output file using GDAL. Such a writing mechanism assures the correctness of data output, but is not as efficient as true parallel writing, because transferring large amount of data can be time-consuming. Our experiments showed that the writing procedure could take over 50 seconds to output 966 MB of data, while the actual computation took tens of seconds.

As mentioned in Section 1.3, in the end of each time step during the CA evolution, the 'halo' cells of a sub-domain on the output layer (i.e., simulated urban areas) must be updated according to the neighboring sub-domains' edge cells. To implement such 'halo' updating, the edge cells of a sub-domain are first copied from the GPU's memory to CPU's memory (i.e., GPU→CPU transfers), and the MPI process starts communication procedures with other processes (i.e., CPU↔CPU transfers) to exchange the edge cells' data. Once the communications are completed, the updated 'halo' cells are copied from CPU to GPU (i.e., CPU→GPU transfers). The transfers between GPU and CPU will be described in Section 3.2.

The data exchange between processes (i.e., CPU↔CPU transfer) is implemented using the SEND and RECV functions of MPI. To avoid the deadlock communication error, we

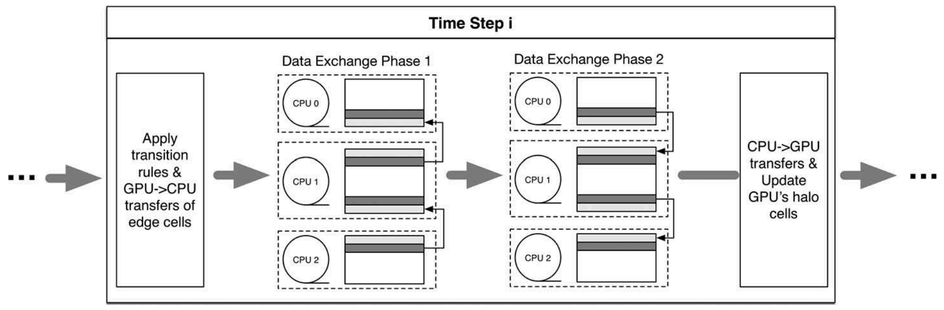

Time Step i
Apply transition rules & GPU->CPU transfers of edge cells
Data Exchange Phase 1
CPU 0
CPU 1
CPU 2
Data Exchange Phase 2
CPU 0
CPU 1
CPU 2
CPU->GPU transfers & Update GPU's halo cells

Figure 5. Two-phase data exchange for 'halo' updating.

used a two-phase data exchange technique for the row-wise decomposition (Figure 5). First, each process P_i sends the data stream of the upper edge to its previous adjacent process (i.e., P_{i-1}), and receives the data stream from its following adjacent process (i.e., P_{i+1}) to update the bottom halo cells. In the second phase, each process P_i sends the data stream of the bottom edge to its following adjacent process P_{i+1}, and receives the data stream from its previous adjacent process P_{i-1} to update the upper halo cells.

3.2. CUDA-based computing

Once an MPI process completes reading the sub-domain of data, it executes CUDA-based computing procedures on the GPU that is associated with the logical node, to apply the transition rules of CA on the sub-domain.

Specifically, a CUDA computing procedure includes the following steps:

(1) the device (i.e., GPU) allocates memory space for the input and output layers with the same spatial dimensions as the assigned sub-domain;
(2) the input layers of data for the sub-domain are copied from the memory of the host (i.e., CPU or CPU core) to the memory of the device (i.e., CPU→GPU transfers);
(3) the device evokes multiple concurrent threads to apply a particular CA transition rule on the cells within the sub-domain, with each thread responsible for evaluating one cell;
(4) once all the cells within the sub-domain are evaluated, the necessary output data (edge cells or all cells, depending on whether writing is required) is then copied from the device to the host (i.e., GPU→CPU transfers);
(5) if no longer needed, a data layer can be deleted on the device and the memory is freed.

In our parallel geospatial CA model, each transition rule is implemented as a CUDA-based computing procedure. Thus, the global conversion probability (i.e., transition rule (1)) is computed using a CUDA procedure at the beginning before the iterative evolution of CA, and the other three transition rules are applied by executing three corresponding CUDA procedures in sequence at each time step during the evolution.

GPUs are equipped with a hierarchy of memory modules, i.e., the global memory of relatively lower accessing speed but the largest size for all threads, the shared memory of medium speed and size for a block of threads, and the local memory of the highest speed but the smallest size for each thread. Also, when the threads of consecutive IDs using the same instruction access consecutive global memory addresses, these consecutive accesses are coalesced into a consolidated access to the memory; thus, the number of global memory accesses can be largely reduced to increase the efficiency (Kirk and Hwu 2010). In our current version of the parallel CA model, the CUDA-based CA transition rules only use the global memory of the GPU. Transition rules (1), (3), and (4) are essentially local-scope raster operations, and able to take advantages of the consecutive accessing mechanism to achieve high performance. However, transition (2), i.e., the computation of joint conversion probability, is a focal-scope operation based on a 3 × 3 neighborhood, leading to inconsecutive accesses of scattered addresses of the global memory. One of our future efforts will focus on utilizing the faster shared memory for the transition rules to further improve the performance.

During the evolution of the CA, the conversion probability layers are intermediate layers, residing in the GPU's memory and being updated by GPU threads. The output layer of the simulated urban areas, reside in both the GPU's and CPU's memory spaces. As shown in Figure 6(a), at the end of each time step, if no writing is required, only the edge cells are copied from GPU to CPU (i.e., GPU→CPU transfers) to be sent to other MPI processes, and the updated 'halo' cells are copied from CPU to GPU (i.e., CPU→GPU transfers) for the next iteration. If writing the output layer is required, all the cells within the sub-domain are copied from GPU to CPU for output (Figure 6(b)).

4. Experiments and performance assessments

4.1. Testing environment and datasets

The above hybrid parallel CA model was implemented using the C programming language, OpenMPI 1.6 (http://www.open-mpi.org), and CUDA 4.0 (https://developer.nvidia.com/cuda-zone). To demonstrate and test the validity and performance of the proposed solution, we conducted a series of experiments on the Keeneland Initial Delivery System (KIDS, http://keeneland.gatech.edu/KIDS) at the Georgia Institute of Technology, a GPU/CPU heterogeneous system with 120 physical nodes connected by an InfiniBand QDR network. Each node is comprised of two Intel Xeon X5660 6-core 2.8GHz CPUs with 12 GB memory per CPU, and two Nvidia M2090 GPUs with 512 thread processors and 6GB memory per GPU. Thus, KIDS allows two logical GPU/CPU nodes (each composed of a CPU core and a GPU) to be initialized on each physical node. The 'GPU/CPU nodes' in the following performance assessments refer to logical nodes.

The datasets of California were used in our experiments, including five normalized site attribute layers (i.e., elevation, slope, distance to major transportation networks, distance to city centers, and land use of 1992), an exclusion layer, and an urban layer of 1992. Each layer contains 40460 × 23851 cells at 30-meter resolution, stored in GeoTIFF format. The model also generates four intermediate layers when executing the transition rules (one layer per rule), as well as the final output layer representing the simulated urban distribution, all with the same spatial dimensions as the input layers. The total memory space required for the simulation was over 20GB, which largely exceeded the memory capacity of a single GPU on the computing system used for the experiments.

The exclusion layer was generated such that the existing urban areas, water bodies, and national/state parks were given the value of 0 (i.e., absolutely excluded from urbanization), and other areas were given the value of 1. The parameters for the global probability calculation (i.e., a and b_k in Equation (1)) were determined using a logistic

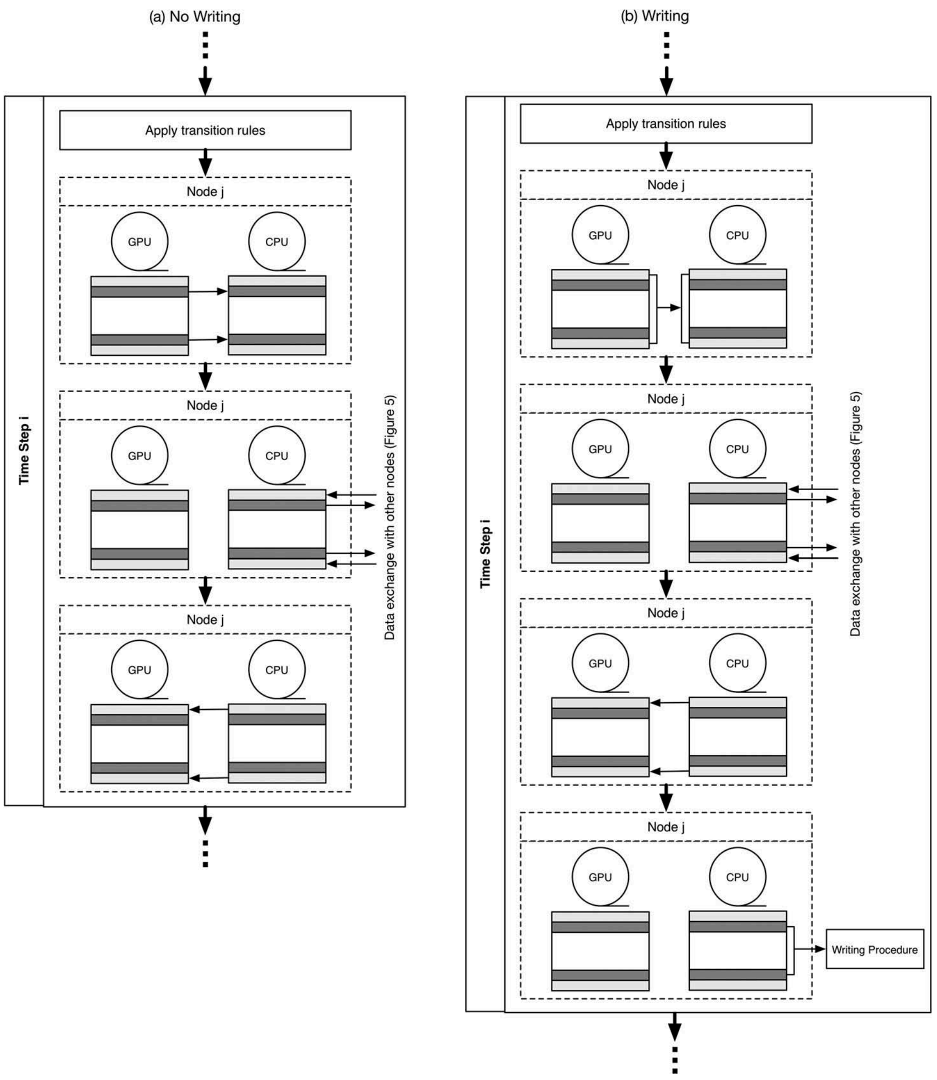


Figure 6. Data transfers between GPU and CPU at each time step.

regression with 10,000 samples out of the urban growth map for the period of 1992–1997 and the five site attribute layers. The dispersion rate in the distance-decay probability calculation (i.e., δ in Equation (3)) was set as 5, following Wu's (2002) suggestion. The maximal number of cells to be urbanized at each time step was set as 16,000, which approximates the average annual growth of urban between 1992 and 1997. Note that the experiments were designed to demonstrate the validity and performance of the hybrid parallel CA model developed in this study. When used in real-world applications for land allocation optimization and planning, the input layers and parameters can be set using alternative methods, such as the optimization methods mentioned in Section 1.1. The hybrid parallelism strategy developed in this study can accommodate various algorithms and parameters in the transition rules, and various input layers, because it decomposes the data, not the algorithm.

4.2. Results and performance assessments

To obtain a baseline for the performance assessment of our parallel geospatial CA, we developed a sequential program that implemented exactly the same model, and conducted a series of simulations on a desktop computer equipped with an Intel Core i7 930 Quad-core 2.8 GHz CPU and 8GB of memory. The same simulations were also conducted using the parallel program on KIDS.

When using a threshold of the conversion probability (e.g., 0.8), instead of the random number generator, in the final conversion computing (i.e., Equation (5)) to eliminate the random selection procedure, the parallel model generated the same simulation results as the sequential model did, which confirmed the correctness of the parallel model. When using the random selection procedure, the results changed slightly for every simulation, no matter whether it was generated by the sequential model or the parallel one. Figure 7 shows the results (with 10-year intervals) generated by a 50-year simulation using the parallel model.

As mentioned in Section 3.1, our parallel CA model uses a centralized output technique. After the computation for a time step is completed, the master process gathers the sub-domains of the output layer from other processes, and writes to the designated file. Our experiments showed that such an output mechanism required an extensive amount of time (often over 50 seconds for each 966 MB map) due to the data transfers between processes, as well as the creation of the output files and the writing operations for large volume of data through a single I/O channel. Parallel I/O, which allows multiple processes to write their own sub-domains of data to the output file in parallel, will be implemented in the next version of our parallel CA model. However, since parallel I/O is not the focus of this study, the following performance assessments will use the computing times of the experiments that disabled the output function, so as to concentrate on the computing performance of the hybrid parallelism of the CA transition rules.

As shown in Table 1, without writing the simulation results, the sequential model took 13,647 seconds (about 3 hours and 47 minutes) to complete a 50-year simulation using the single-CPU desktop computer.

Using 64 logical GPU/CPU nodes (i.e., 64 CPU cores and 64 GPUs on 32 physical nodes, 15 million threads per GPU so that one thread per cell), the parallel model took only 32 seconds to complete the same simulation, achieving a speed-up of 426. The computing time for the actual CA computation (i.e., excluding the reading time[3]) dropped from 13,600 seconds using a single CPU to 31 seconds using 64 logical GPU/CPU nodes, achieving a speed-up of 439. The performance was much higher than other parallel CA models that only use multiple CPUs or CPU cores. For example, Guan *et al.* (2014) implemented the same CA model using the parallel Raster Processing Library (pRPL), and conducted a series of experiments using the same datasets. On a computer cluster composed of 106 nodes, each equipped with four Opteron 16-core 2.1GHz CPUs and 256 GB of RAM, the pRPL-based CA model completed a 5-year simulation (without output) in 77.42 seconds using 1024 CPU cores (i.e., MPI processes). It is quite clear that the GPU/CPU hybrid parallel CA would largely outperform the CPU-only parallel CA, using the same number of physical nodes.

As shown in Figure 8, the computing time decreased as the number of GPU/CPU nodes increased. In most cases, the speed-up exhibited a near-linear increasing pattern, demonstrating the effective scalability of our hybrid parallelism strategy. The obvious exceptions were the 10-year simulations using 32 and 64 nodes, whose speed-ups were lower than those of other simulations for longer periods using the same numbers of nodes.

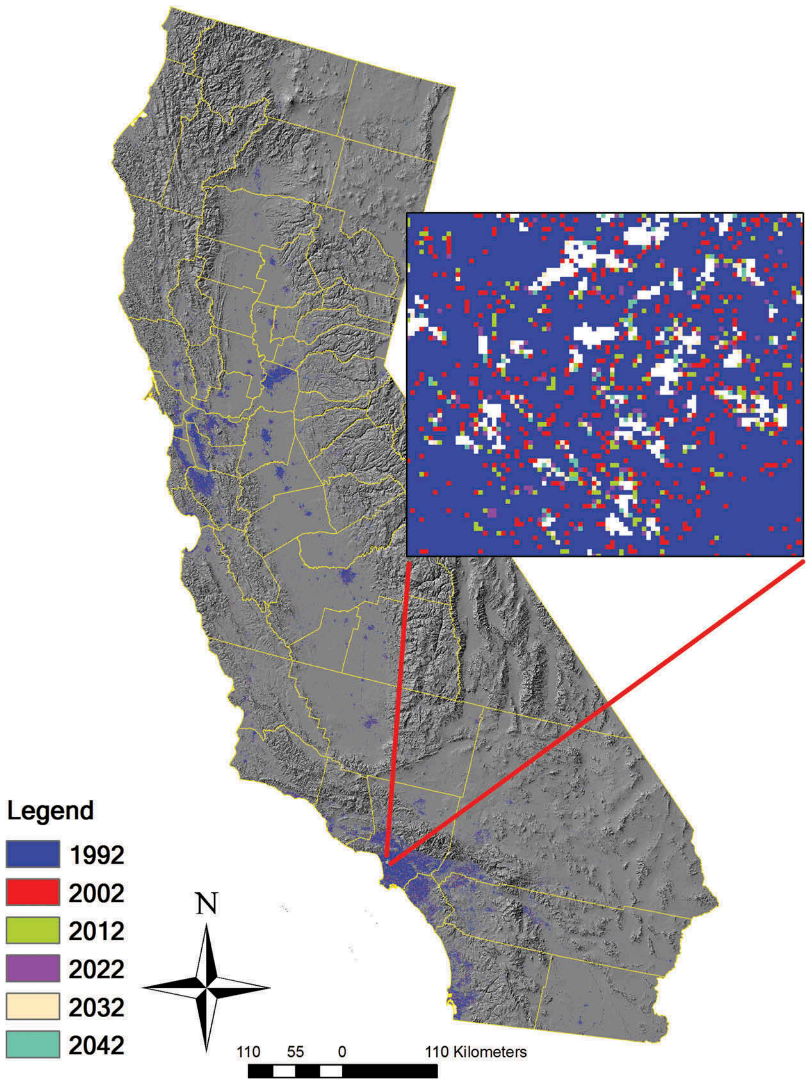
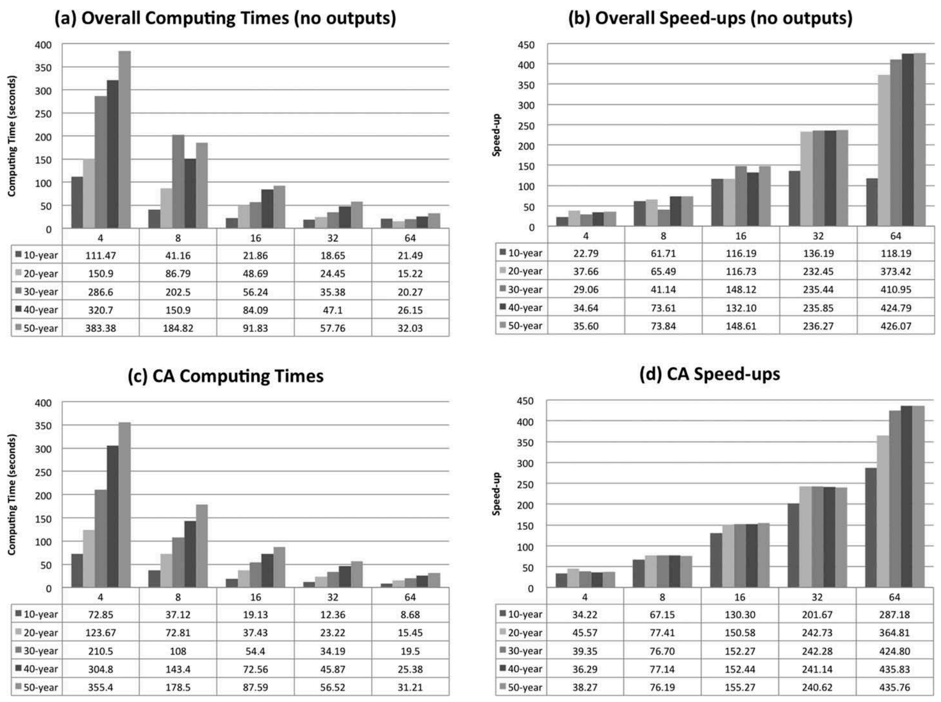


Figure 7. Simulated urban areas generated by the parallel geospatial CA model.

Table 1. Computing times using a single Intel Core i7 CPU.

	Reading (s)	Computing (s)	Total (s)
10-year simulation	47.2	2492.69	2539.89
20-year simulation	47.26	5636.26	5683.52
30-year simulation	46.38	8283.61	8329.99
40-year simulation	47.05	11,061.32	11,108.37
50-year simulation	46.97	13,600.04	13,647.01



(a) Overall Computing Times (no outputs)

	4	8	16	32	64
10-year	111.47	41.16	21.86	18.65	21.49
20-year	150.9	86.79	48.69	24.45	15.22
30-year	286.6	202.5	56.24	35.38	20.27
40-year	320.7	150.9	84.09	47.1	26.15
50-year	383.38	184.82	91.83	57.76	32.03

(b) Overall Speed-ups (no outputs)

	4	8	16	32	64
10-year	22.79	61.71	116.19	136.19	118.19
20-year	37.66	65.49	116.73	232.45	373.42
30-year	29.06	41.14	148.12	235.44	410.95
40-year	34.64	73.61	132.10	235.85	424.79
50-year	35.60	73.84	148.61	236.27	426.07

(c) CA Computing Times

	4	8	16	32	64
10-year	72.85	37.12	19.13	12.36	8.68
20-year	123.67	72.81	37.43	23.22	15.45
30-year	210.5	108	54.4	34.19	19.5
40-year	304.8	143.4	72.56	45.87	25.38
50-year	355.4	178.5	87.59	56.52	31.21

(d) CA Speed-ups

	4	8	16	32	64
10-year	34.22	67.15	130.30	201.67	287.18
20-year	45.57	77.41	150.58	242.73	364.81
30-year	39.35	76.70	152.27	242.28	424.80
40-year	36.29	77.14	152.44	241.14	435.83
50-year	38.27	76.19	155.27	240.62	435.76

Figure 8. Computing performance of parallel geospatial CA.

This may be because that the data transfers for 'halo' updating took larger proportions of the computing times in such short-period simulations.

Also, the speed-ups of the actual CA computation (Figure 8(d)) were slightly higher than the speed-ups of the overall simulation (Figure 8(b)). Even though the MPI processes read their own sub-domains of data in parallel, the efficiency of parallel reading is often lower than the efficiency of parallel CA computing, thereby degrading the performance of the overall simulation. As mentioned above, developing an efficient parallel I/O mechanism for geospatial data will be one of our future research focuses.

5. Conclusion

With their capabilities of simulating spatiotemporal dynamics, geospatial CA models can be used as an effective tool to derive and test spatial optimization strategies. Optimization methods can be used to generate the macro-scale constraints and input layers, search for the optimal parameters, and construct the transition rules for CA models. However, given the complex transition rules and massive amount of data, a geospatial CA model can be both computational and data-intensive, thereby requiring extensive amount of memory space and computing time. Such computational and data intensities make geospatial CA models less feasible and scalable, sometimes even intractable, when simulating long-term spatiotemporal dynamics in large areas with high-resolution spatial data.

In this study, a hybrid parallelism strategy was developed for geospatial CA models on the GPU/CPU heterogeneous computing architecture, which consists of multiple nodes equipped with GPUs and CPUs. The hybrid parallelism strategy uses MPI to form multiple logical GPU/CPU nodes (i.e., MPI processes) to handle the domain decomposition, data I/O, and data exchange between nodes for ‘halo’ cell updating, and uses CUDA to coordinate the data transfers between GPU and CPU, and allocate multiple concurrent threads to apply the CA transition rules on the sub-domain of data on each GPU.

A parallel geospatial CA model for simulating urban growth was implemented using such a hybrid parallelism strategy. The experiments using the datasets of California (requiring over 20 GB of memory) showed that the hybrid parallel CA model was able to handle massive amount of data and greatly reduce the computing time by utilizing multiple GPU/CPU nodes. The experiments also yielded near-linear speed-ups for various simulation lengths (i.e., numbers of time steps) with different numbers of GPU/CPU nodes, demonstrating the effective scalability of the hybrid parallelism strategy.

In conclusion, such hybrid parallelism enables large-scale geospatial CA simulations with complex transition rules and massive amount of data, which were previously infeasible, sometimes impossible, using conventional individual computers, and is able to yield higher performance than CPU-only parallel computers and single-GPU computers. As more GPU/CPU heterogeneous computer systems are integrated in various cyberinfrastructure environments, the hybrid parallelism provides a viable solution to overcome the computational barrier of geospatial CA simulation, thus improving the applicability, feasibility, and scalability of CA-based spatial optimization.

Future work includes (1) developing a parallel I/O mechanism that allows multiple nodes to not only read, but also write the sub-domains of data in parallel; and (2) designing and implementing CUDA-based transition rules to utilize not only the global memory, but also the shared and local memory to further improve the computational performance.

Funding

This research was supported by the Specialized Research Fund for the Doctoral Program of Higher Education, Ministry of Education of China [20130145120013]; the National Science Foundation of USA [OCI-1047916]. This research used resources of the Keeneland Computing Facility, which is supported by the National Science Foundation of USA [OCI-0910735].

Notes

1. The discrete memory architecture allocates separate memory spaces for computing units, and a computing unit cannot access other units’ memory spaces directly. Any data exchange must be implemented through communications between units. On the other hand, the shared memory architecture allows all units to access the same memory space, which eliminates the necessity of communication. However, when multiple units are trying to access (especially updating) the same memory address, a locking mechanism must be evoked to allow one unit to access the address at a time, which will degrade the performance.
2. Ranking from Top500.org as of June, 2014.
3. The reading time refers to the time used for MPI processes to read the sub-domains of input data into the CPU memory, and does not include the time for CPU→GPU transfers.

References

Bandini, S., Mauri, G., and Serra, R., 2001. Cellular automata: from a theoretical parallel computational model to its application to complex systems. *Parallel Computing*, 27, 539–553. doi:10.1016/S0167-8191(00)00076-4

Benjamin, S.C., Johnson, N.F., and Hui, P.M., 1996. Cellular automata models of traffic flow along a highway containing a junction. *Journal of Physics A: Mathematical and General*, 29, 3119–3127. doi:10.1088/0305-4470/29/12/018

Blecic, I., Cecchini, A., and Trunfio, G.A., 2014. Fast and accurate optimization of a GPU-accelerated CA urban model through cooperative coevolutionary particle swarms. *Procedia Computer Science*, 29, 1631–1643. doi:10.1016/j.procs.2014.05.148

Cheng, G., *et al.*, 2012. General-purpose optimization methods for parallelization of digital terrain analysis based on cellular automata. *Computers & Geosciences*, 45, 57–67. doi:10.1016/j.cageo.2012.03.009

Clarke, K.C., 2003. Geocomputation's future at the extremes: high performance computing and nanoclients. *Parallel Computing*, 29, 1281–1295. doi:10.1016/j.parco.2003.03.001

Clarke, K.C. and Gaydos, L.J., 1998. Loose-coupling a cellular automaton model and GIS: Long-term urban growth prediction for San Francisco and Washington/Baltimore. *International Journal of Geographical Information Science*, 12, 699–714. doi:10.1080/136588198241617

Clarke, K.C., *et al.*, 2007. A decade of SLEUTHing: lessons learned from applications of a cellular automaton land use change model. *In*: P. Fisher, ed. *Classics from IJGIS. Twenty years of the international journal of geographical information systems and science*. Boca Raton, FL.: Taylor and Francis, 413–425.

Clarke, K.C., Hoppen, S., and Gaydos, L., 1997. A self-modifying cellular automaton model of historical urbanization in the San Francisco Bay Area. *Environment and Planning B: Planning and Design*, 24, 247–261. doi:10.1068/b240247

Clarke, K.C., Riggan, P., and Brass, J.A., 1995. A cellular automaton model of wildfire propagation and extinction. *Photogrammetric Engineering and Remote Sensing*, 60, 1355–1367.

Couclelis, H., 1997. From cellular automata to urban models: new principles for model development and implementation. *Environment and Planning B: Planning and Design*, 24, 165–174. doi:10.1068/b240165

Dietzel, C. and Clarke, K.C., 2007. Toward optimal calibration of the SLEUTH land use change model. *Transactions in GIS*, 11, 29–45. doi:10.1111/j.1467-9671.2007.01031.x

Feng, Y. and Liu, Y., 2013. A heuristic cellular automata approach for modelling urban land-use change based on simulated annealing. *International Journal of Geographical Information Science*, 27, 449–466. doi:10.1080/13658816.2012.695377

Fotakis, D. and Sidiropoulos, E., 2012. A new multi-objective self-organizing optimization algorithm (MOSOA) for spatial optimization problems. *Applied Mathematics and Computation*, 218, 5168–5180. doi:10.1016/j.amc.2011.11.003

Gropp, W., *et al.*, 1998. *MPI: the complete reference*, Vol. 2. Cambridge, MA: The MIT Press.

Guan, Q. and Clarke, K., 2010. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science*, 24, 695–722. doi:10.1080/13658810902984228

Guan, Q., *et al.*, 2014. pRPL 2.0: improving the parallel raster processing library. *Transactions in GIS*, 18, 25–52. doi:10.1111/tgis.12109

Kirk, D.B. and Hwu, W.W., 2010. *Programming massively parallel processors: a hands-on approach*. San Francisco, CA: Morgan Kaufmann.

Laura, J., *et al*. Parallelization of a regionalization heuristic in distributed computing platforms – a case study of parallel-p-compact-regions problem. *International Journal of Geographical Information Science*. doi:10.1080/13658816.2014.987287

Li, D., *et al.*, 2012. GPU-CA model for large-scale land-use change simulation. *Chinese Science Bulletin*, 57, 2442–2452. doi:10.1007/s11434-012-5085-3

Li, X., *et al.*, 2013. Calibrating cellular automata based on landscape metrics by using genetic algorithms. *International Journal of Geographical Information Science*, 27, 594–613. doi:10.1080/13658816.2012.698391

Li, X., Yang, Q., and Liu, X., 2008. Discovering and evaluating urban signatures for simulating compact development using cellular automata. *Landscape and Urban Planning*, 86, 177–186. doi:10.1016/j.landurbplan.2008.02.005

Li, X. and Yeh, A.G.O., 2000. Modelling sustainable urban development by the integration of constrained cellular automata and GIS. *International Journal of Geographical Information Science*, 14, 131–152. doi:10.1080/136588100240886

Li, X. and Yeh, A.G.O., 2001. Calibration of cellular automata by using neural networks for the simulation of complex urban systems. *Environment and Planning A*, 33, 1445–1462. doi:10.1068/a33210

Li, X. and Yeh, A.G.O., 2002. Neural-network-based cellular automata for simulating multiple land use changes using GIS. *International Journal of Geographical Information Science*, 16, 323–343. doi:10.1080/13658810210137004

Li, X., *et al*., 2010. Parallel cellular automata for large-scale urban simulation using load-balancing techniques. *International Journal of Geographical Information Science*, 24, 803–820. doi:10.1080/13658810903107464

Liu, X., *et al*., 2008. A bottom-up approach to discover transition rules of cellular automata using ant intelligence. *International Journal of Geographical Information Science*, 22, 1247–1269. doi:10.1080/13658810701757510

Liu, X., *et al*., 2010. Simulating land-use dynamics under planning policies by integrating artificial immune systems with cellular automata. *International Journal of Geographical Information Science*, 24, 783–802. doi:10.1080/13658810903270551

Liu, Y. and Phinn, S., 2003. Modelling urban development with cellular automata incorporating fuzzy-set approaches. *Computers, Environment and Urban Systems*, 27, 637–658. doi:10.1016/S0198-9715(02)00069-8

Long, L.N. and Gupta, A., 2008. Scalable massively parallel artificial neural networks. *Journal of Aerospace Computing, Information, and Communication*, 5, 3–15. doi:10.2514/1.31026

Nagel, K. and Schreckenberg, M., 1992. A cellular automaton model for freeway traffic. *Journal of Physics I France*, 2, 2221–2229. doi:10.1051/jp1:1992277

Qin, C.-Z., Zhan, L.-J., and Zhu, A.-X., 2013. How to apply the geospatial data abstraction library (GDAL) properly to parallel geospatial raster I/O? *Transactions in GIS*. doi:10.1111/tgis.12068

Qin, C.-Z., *et al*., 2014. A strategy for raster-based geocomputation under different parallel computing platforms. *International Journal of Geographical Information Science*, 28, 2127–2144. doi:10.1080/13658816.2014.911300

Sanders, J. and Kandrot, E., 2011. *CUDA by example: an introduction to general-purpose GPU programming*. Upper Saddle River, NJ: Addison-Wesley.

Shi, X., *et al*., 2014a. Unsupervised image classification over supercomputers Kraken, Keeneland and Beacon. *GIScience & Remote Sensing*, 51, 321–338. doi:10.1080/15481603.2014.920229

Shi, X., *et al*., 2014b. Geocomputation over the emerging heterogeneous computing infrastructure: geocomputation over the emerging heterogeneous computing infrastructure. *Transactions in GIS*, 18, 3–24. doi:10.1111/tgis.12108

Shi, X. and Ye, F., 2013. Kriging interpolation over heterogeneous computer architectures and systems. *GIScience & Remote Sensing*, 50 (2), 196–211.

Silva, E.A. and Clarke, K.C., 2002. Calibration of the SLEUTH urban growth model for Lisbon and Porto, Portugal. *Computers, Environment and Urban Systems*, 26, 525–552. doi:10.1016/S0198-9715(01)00014-X

Tang, W. and Wang, S., 2009. HPABM: a hierarchical parallel simulation framework for spatially-explicit agent-based models. *Transactions in GIS*, 13, 315–333. doi:10.1111/j.1467-9671.2009.01161.x

Thor, M., 2008. Performance comparison of CPU and GPU based simulation of an avalanche using a cellular automata (Master Thesis). Umeå University, Sweden.

Wang, L., *et al*., 2011. Scaling scientific applications on clusters of hybrid multicore/GPU nodes. *In*: *Proceedings of the 8th ACM international conference on computing frontiers*. New York: ACM.

Wang, S., 2010. A cyberGIS framework for the synthesis of cyberinfrastructure, GIS, and spatial analysis. *Annals of the Association of American Geographers*, 100, 535–557. doi:10.1080/00045601003791243

Ward, D.P., Murray, A.T., and Phinn, S.R., 2003. Integrating spatial optimization and cellular automata for evaluating urban change. *The Annals of Regional Science*, 37, 131–148. doi:10.1007/s001680200113

White, R., Engelen, G., and Uljee, I., 1997. The use of constrained cellular automata for high-resolution modelling of urban land-use dynamics. *Environment and Planning B: Planning and Design*, 24, 323–343. doi:10.1068/b240323

Wu, F., 2002. Calibration of stochastic cellular automata: the application to rural-urban land conversions. *International Journal of Geographical Information Science*, 16, 795–818. doi:10.1080/13658810210157769

Wu, F. and Webster, C.J., 1998. Simulation of land development through the integration of cellular automata and multi-criteria evaluation. *Environment and Planning B: Planning and Design*, 25, 103–126. doi:10.1068/b250103

Yang, J., *et al*., 2013. An intelligent method to discover transition rules for cellular automata using bee colony optimisation. *International Journal of Geographical Information Science*, 27, 1849–1864. doi:10.1080/13658816.2013.823498

Yeh, A.G.O. and Li, X., 2002. Urban simulation using neural networks and cellular automata for land use planning. *In*: D. Richardson and P. van Oosterom, eds. *Advances in spatial data handling*. Ann Arbor: The University of Michigan Press, 451–464.

Zhang, J. and You, S., 2013. High-performance quadtree constructions on large-scale geospatial rasters using GPGPU parallel primitives. *International Journal of Geographical Information Science*, 27, 2207–2226. doi:10.1080/13658816.2013.828840