Chapter 17
Opportunities and Challenges for Urban Land-Use Change Modeling Using High-Performance Computing

Qingfeng Guan and Xuan Shi

Abstract Simulating urban land-use changes involves both high modeling and computational complexities. This paper focuses on a typical spatio-temporal modeling method that has been commonly used in urban land-use change studies—Cellular Automata (CA). After reviewing the recent development of utilizing various parallel computing technologies (e.g., computer clusters and Graphics Processing Unit [GPU]) in CA-based urban models, this paper presents a pilot study, in which a classical CA model, the Game of Life, was implemented as a parallel program over the GPU/CPU heterogeneous cluster architecture, and 300+ speed-up was achieved using 20 GPUs. In conclusion, emerging high-performance computing technologies, such as GPU/CPU heterogeneous cluster architecture, provide promising potentials to overcome the computing obstacle of urban land-use change models, and enable researchers to examine, validate and advance urban land-use change theories and derive sound urban planning strategies. To efficiently utilize the computing power of the GPU/CPU clusters, hybrid parallelism must be implemented to coordinate the computing among GPU/CPU nodes, as well as among the threads on each GPU. However, implementing such hybrid parallelism is challenging for its high development complexity.

Keywords Parallel computing • GPU • Heterogeneous cluster architecture • Urban • Land-use change

Q. Guan (✉)
Faculty of Information Engineering, China University of Geosciences (Wuhan), Wuhan, Hubei 430074, China
e-mail: guanqf@cug.edu.cn

X. Shi
Department of Geosciences, University of Arkansas, Fayetteville, AR 72701, USA
e-mail: xuanshi@uark.edu

X. Shi et al. (eds.), *Modern Accelerator Technologies for Geographic Information Science*,
DOI 10.1007/978-1-4614-8745-6_17,

17.1 Introduction

Simulating urban land-use changes (LUC) has been a challenging task because of the spatiotemporal complexities of interrelationships and interactions between the urban land system and related natural/socioeconomic systems. The difficulty of modeling urban LUC can be aggravated by the massive computational intensity caused by complicated algorithms and large datasets that are often required in the simulation. Some large scale simulations have been infeasible because they are computationally intractable using conventional desktop computers.

In order to reduce both the modeling and computational complexities in spatiotemporal simulations, researchers often had to make subjective and/or simplifying assumptions. However, such simplifying approaches had raised some serious scientific questions in regard to the validity and soundness of the findings resulted from these models, because whether these assumptions could generate reliable calibration and simulation results and lead to unbiased and accurate scientific conclusions has not been sufficiently studied yet. To investigate the potential problems and advance our understanding and theories of urban land dynamics, we must devise approaches to reduce, or even eliminate, these assumptions.

Recent advancements in high-performance computing (HPC) infrastructure provide potential solutions to the above problems. Emerging advanced computing technologies, such as Graphics Processing Units (GPUs) and heterogeneous cluster computing systems that combine multiple GPU accelerators and Central Processing Units (CPUs), have been significantly improving the performance of scientific computation in a variety of domains. Therefore, it is time for geographers and geospatial scientists to examine, validate and advance urban LUC theories as the technological solutions and computing infrastructure are increasingly mature and efficient for such kind of investigations.

17.2 Spatiotemporal Modeling of Urban Land-Use Changes

Many approaches exist to model urban land-use changes and associated natural and socio-economic dynamics. A large proportion of them are based on variants of the Cellular Automata (CA) model, a discrete computational model used to simulate dynamic spatial processes through a set of transition rules. A classical CA model has a set of identical elements, called cells. Each cell is located in a regular, discrete space, called a cellspace. Each cell is associated with a state within a finite set. The model evolves in discrete time steps, changing the states of all its cells according to transition rules, homogeneously and synchronously applied at every step. The new state of a certain cell depends on the previous states of the cells within its neighborhood. CA models have been widely used in geographic research to simulate complex spatiotemporal phenomena, including land-use and land-cover change (Batty

et al. 1999; Couclelis 1997; Wu and Webster 1998; Li and Yeh 2000; Liu and Phinn 2003), wildfire propagation (Clarke et al. 1995), and freeway traffic (Nagel and Schreckenberg 1992; Benjamin et al. 1996).

A typical example is the SLEUTH model, one of the most widely used urban LUC models (Clarke et al. 1997; Clarke and Gaydos 1998; Silva and Clarke 2002). The core of SLEUTH is an urban growth model, which uses a modified CA to simulate the spread of urbanization across a landscape. The behavior of the simulation is determined by five parameters (also termed coefficients), each ranging from 0 to 100. Four growth rules are applied in sequence on the space during each growth cycle, which represents a year of urban growth.[1]

Calibration is needed to determine the appropriate parameter values so that SLEUTH can produce realistic simulation results. The basic calibration procedure of SLEUTH uses the brute-force method, which statistically compares multiple test results produced using combinations of parameter values with the real historical dataset, in order to determine the best-fit parameter combination(s). In addition, to simulate the random processes during urban growth, the Monte Carlo method is applied multiple times, and the outcomes are stored as the cumulative probabilities of change over multiple runs. In practice, 10–100 Monte Carlo iterations for each parameter combination are suggested, although fewer may be better than more (Goldstein et al. 2005).

All of the above together make the calibration highly computationally intensive. A 12-year (1986–1998) simulation over a small-sized dataset (2,074×486) of Santa Barbara County in California took only 1 seconds to complete on a desktop PC. However, a comprehensive calibration over the same dataset and time period to examine all 101^5 parameter combinations with only 1 Monte Carlo iteration was estimated to take over 300 years to complete. This places the SLEUTH model at the edge of computational tractability. The current version of SLEUTH model uses a simplifying assumption to ignore those "unimportant" parameter values during seeking the best-fit combination(s), which is that the parameters affect the simulation results in a linear manner. However, due to the random processes involved in the transition rules, the relationships between the parameters/factors and LUC simulations are very likely non-linear. Thus the calibration results based on such simplifying assumptions are hardly fact-proven (due to the incomprehensive calibration), less reliable, and may lead to inaccurate scientific conclusions and improper land management decisions (Dietzel and Clarke 2007).

Alternatively, researchers have used Computational Intelligence (CI) methods to either seek the best-fit parameter combination(s) without evaluating all the combinations, or construct transition rules for the model (see for example Li and Yeh 2002; Wu and Silva 2010; Liu et al. 2010; Li et al. 2013). However, the computational burden of CA itself is not diminished by CI methods, and the computational intensity may still exceed the capacity of a desktop computer when using complex transition rules and massive datasets.

[1] For details, see http://www.ncgia.ucsb.edu/projects/gig/About/gwRules.htm.

17.3 Exploratory Studies on High-Performance Spatial CA

17.3.1 Parallel Spatial CA

The classical CA model has been recognized to be a natural parallel computing system as the transition rules are applied to the cells homogeneously and synchronously in parallel (Bandini et al. 2001). The cellspace can be easily decomposed into a set of small sub-cellspaces and assigned onto multiple computing units (e.g., CPUs and CPU cores) to be processed simultaneously. Several general parallel CA-based simulation systems have been developed. Examples include the Cellular Automata environment for systEms ModeLing (CAMEL) and CellulAR Programming EnvironmenT (CARPET) language (Spezzano and Talia 1999), and Cell Driver, a CA modeling module of NEMO (Hecker et al. 1999). Both CAMEL and Cell Driver were built based on the Message Passing Interface (MPI), a generic parallel programming library that is available on most parallel computing systems.

Guan and Clarke (2010) developed an open-source general-purpose parallel Raster Processing programming Library (pRPL), for non-specialist scientists to easily parallelize their own raster processing algorithms. pRPL supports multi-layer algorithms that are commonly used in geospatial applications, including spatial CA. pRPL provides multiple data decomposition methods, including a spatially-adaptive quad-tree-based (QTB) decomposition method for situations when the computational intensity is extremely heterogeneous over space. pRPL also automatically takes care of some complicated processes that are required in parallel computing, e.g., communication, synchronization and load-balancing, thus provides transparent parallelism for users. A parallel urban LUC model, pSLEUTH, was developed based on the SLEUTH model using pRPL. Experiments with real-world datasets showed that pSLEUTH greatly reduced the computing time for the calibration process, achieving a speed-up of 24 using 32 CPU cores on a computer cluster composed of 128 dual CPU 3.06 GHz Xeon nodes with 2 GB RAM each.

However, all above parallel CA systems are based on conventional CPU-only parallel computing architectures such as multi-core CPUs and computer clusters. Large-scale parallel computing facilities are extremely expensive and require tremendous amount of financial and labor investments, and very limited to public access. Also, the waiting time in a job queue on a computer cluster may exceed the actual computing time, which makes the performance gain from parallel computing less meaningful. An emerging accelerator technology, GPU with the Compute Unified Device Architecture (CUDA), is able to accelerate the computation processes by deploying hundreds of computing cores on the GPU with very low costs. A PC equipped with a GPU is considerably cheaper than a computer cluster that has the same number of cores. GPUs are very suitable for parallel matrix manipulation and processing, which is similar to CA computing. Some efforts have been made to implement CA models on GPUs, which generated high speed-ups (Thor 2008; Li et al. 2012). Moreover, the heterogeneous computer cluster architecture can

generate massive computing power by coordinating a set of computational nodes that consists of one or more CPU(s) and GPU(s). The heterogeneous cluster architecture has been adopted to build high-end computing platforms to handle super large-scale scientific and engineering computation.

17.3.2 Accelerating CA on GPUs and Heterogeneous Computer Systems

In order to explore the possibility and validity of utilizing the emerging HPC technologies in urban LUC studies, we have successfully prototyped parallel CA models on both GPU-equipped PCs and GPU/CPU heterogeneous clusters. The Game of Life (GOL) is a well-known classical CA model. Based on the transition rule, a cell can live or die depending on the condition of its 3×3 neighborhood. As a result, the living status of the cells can represent various spatial patterns throughout the course of iterations. The pseudo code of the GOL's transition rule is as follows:

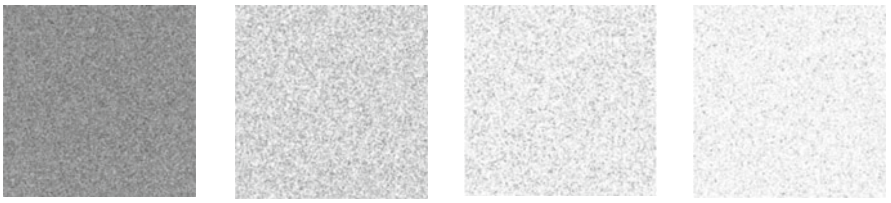
```
FUNCTION Transition (cell, time_t)
  n = number of alive neighbors of cell at time_t
  IF cell is alive at time_t
   IF n ≥ 4
    THEN cell dies of overcrowding at time_t+1
   IF n ≤ 1
    THEN cell dies of loneliness at time_t+1
   IF n = 2 OR n = 3
    THEN cell survives at time_t+1
  ELSE (i.e., cell is dead at time_t)
   IF n = 3
     THEN cell becomes alive (i.e., born) at time_t+1
```

In Table 17.1, the leftmost figure displays the initial status for a 10,000 by 10,000 matrix in which half of the matrix would contain living cells. After 100 iterations, many cells may die and the right-most figure displays the result of the simulation.

Here we introduce the steps taken to create efficient parallel implementations of GOL. In order to ensure that all solutions generate the same result, we create a matrix file that contains the initial living status of randomly generated cells. All versions of the program share a similar initialization phase where this matrix file is read into the appropriate array or arrays in the case of the MPI/CUDA program. Each of the programs was benchmarked against the same set of matrices for 100 iterations. We tested all solutions using the matrix that has a dimension of $10{,}000 \times 10{,}000$, which was initially seeded with half of them as living and half dead.

The GOL was first implemented in a serial C program. A 100-iteration simulation over a $10{,}000 \times 10{,}000$ cellspace was accomplished in about 100 minutes on a desktop PC with a 1.60 GHz dual-core CPU. Within the serial C program, for each iteration, each cell will change its living status by examining the living status of its neighbors. Finally the number of living cells is accumulated.

Table 17.1 $t=0$ [left] through $t=99$ [right]



The GOL was then parallelized into a CUDA program, called GPU-GOL. The GOL's transition rule was implemented as a kernel function. During the simulation, a large number of computational threads are simultaneously invoked on the GPU, each executing an instance of the kernel and applying the transition rule on a small proportion of the whole cellspace.

Since counting the living cells is a sequential process, it was first excluded from the GPU kernel program that implements the transition rules of GOL, while counting the neighbors is a device function. After the result is copied back from device to host, the total number of living cells will be counted in sequential process. The GPU-GOL experiments were conducted on a desktop PC with a NVIDIA GeForce GTX 260 GPU, which has 27 streaming multiprocessors (SM), and is able to run up to 27,648 threads in parallel. The simulation at a size of $10{,}000 \times 10{,}000$ for 100 iterations took about 6 minutes to complete, achieving a speed-up of 16.7.

Further improvement was taken to implement the process of counting the number of living cells through atomicAdd function within the kernel program. GOL simulation at a size of $10{,}000 \times 10{,}000$ for 100 iterations can be completed in about 22 seconds on a single Tesla C2075 Fermi GPU or a single Tesla K20 Kepler GPU, achieving a speedup of 13 in comparison to GTX 260. When shared memory was utilized, better performance could be achieved even over a single GPU, though further examination needs to be conducted to validate the solution over different platforms.

Since a single GPU may not efficiently handle the scalability of computation due to the memory limit on individual GPU, we would like to explore the potential of utilizing multiple GPUs to resolve this problem. Keeneland's hybrid architecture exemplifies its superiority in manipulating the large scale cellular automation computation like GOL. Keeneland is composed of an HP SL-390 (Ariston) cluster with Intel Westmere hex-core CPUs, NVIDIA 6GB Fermi GPUs, and a Qlogic QDR InfiniBand interconnect. The system has 120 nodes, each with two CPUs and three GPUs, while all CPUs and GPUs are bridged together through one I/O hub from which the CPUs can read/write data.

To efficiently utilize and manage the GPU resources in Keeneland, we implemented a combination of MPI and CUDA programs to parallelize the GOL computation on 20 GPUs. Although the CUDA kernel for this implementation is nearly identical to what is implemented in the single-GPU program, data communication become a serious problem due to the strong dependency between the data segments distributed onto different GPU processors.

A row-based data partitioning approach was applied to distribute data segments onto multiple GPUs. We tried to decompose the entire matrix into multiple sections based on the number of GPUs we utilized. In this case, each MPI process reads in a unique portion of the matrix file based on process rank. When the original matrix is split in this way and updated separately on different GPUs, each GPU needs to obtain extra rows of information hosted by the other processors because the state of the cells along the matrix boundaries are dependent upon cells which now are in other sub-matrices handled by different MPI processes.

In order to exchange these boundary rows between the neighboring processors, we applied the SEND and RECV functions in MPI for sending and receiving the boundary rows (i.e. head and tail for each block of the grid) between neighboring processors ranked in MPI, and then copying these rows to the GPU memory. Script 17.1 describes how to handle the data transfer between the host CPU and the GPU, and coordinate the computational threads on the GPU. For each iteration, SEND and RECV functions are first implemented to construct the local data segments to be calculated on each node. The GPU on each node executes the kernel function (i.e., the transition rule) covering one portion of the matrix.

```
for(k = 0; k <ITERATION ; k++) {
if (myrank % 2 == 1) {
     // send tail and receive head
     MPI_Send(...);
     MPI_Recv(...);
     // send head and receive tail
     MPI_Send(...);
     MPI_Recv(...);
}
else {
     // receive head and send tail
     MPI_Recv(...);
     MPI_Send(...);
     // receive tail and send head
     MPI_Recv(...);
     MPI_Send(...);
}
```

Script 17.1 Implementing SEND/RECV for data exchange in MPI program

When 20 GPUs on Keeneland were used, a 100-iteration GOL with a size of $10{,}000 \times 10{,}000$ was completed in 20 seconds. The results were the same as what was generated by the serial C program and GPU-GOL. In short, the computing time was significantly reduced from 100 minutes to 20 seconds, achieving a speed-up of 300. When the atomicAdd approach was applied, GOL over the same size of matrix can be accomplished in about 2 seconds when 20 GPUs were used.

The parallel solution over heterogeneous computer architecture and systems have shown promising prospect to break through the computational bottleneck of CA models that include complex transition rules and use massive datasets. By

simply changing the CA's transition rules to simulate more complex spatiotemporal processes, we may use such an approach to conduct some large-scale urban LUC simulations within a practical length of time.

17.4 Conclusion

Modeling the spatiotemporal dynamics of land use and land cover in the urbanization process often involves complex algorithms and large volume of datasets, which greatly increases the computational intensity, hence sometimes requires unfeasibly long computing time. Simplifying assumptions have been used in previous studies to reduce the computational intensity, but they may generate unreliable results and lead to inaccurate scientific conclusions and improper land management decisions. Emerging high-performance computing technologies, such as GPU and GPU/CPU heterogeneous cluster architecture, provide promising potentials to overcome the computing burden of urban LUC models, thus to enable researchers to examine, validate and advance urban LUC theories and derive sound urban planning strategies. To efficiently utilize the computing power of the GPU/CPU heterogeneous clusters, hybrid parallelism must be implemented to coordinate the parallel computing among GPU/CPU nodes, as well as among the threads on each GPU. However, implementing such hybrid parallelism is challenging for its high development complexity in integrating MPI and CUDA.

In this pilot study, we demonstrated the potential for accelerating CA applications using parallel implementation on hybrid computer clusters. While parallel implementation of CA through MPI + GPU has achieved significant performance improvement, the emerging new architecture of Intel's Many-Integrated Core (MIC) could be another potential accelerator technology for urban LUC simulations. It was found from our other initiatives that the simple MPI-direct-host programming model on Intel MIC cluster can achieve a performance equivalent to the MPI + GPU model on GPU clusters when the same number of processors are allocated for Kriging interpolation calculation and for unsupervised image classification.

Exploring efficient cross-node communication mechanism could be a key component in the future work so as to achieve a strong scalability for CA-based applications running on multiple parallel nodes. For example, the latest Tesla K20 Kepler GPU is able to outperform the Fermi GPU for most applications without special performance tuning. However, K20's direct cross-GPU communication mechanism needs to be explored and deployed to enhance CA-based modeling that has intensive data communication between the nodes. Meanwhile solutions based on Intel MIC architecture is worthy to try since each MIC core has direct support of MPI, making it straightforward to port MPI + CPU code to MIC cluster to achieve significant performance improvement. Exploring a combination of MPI and OpenMP solutions will help handle inter-node and intra-node communications to efficiently utilize the heterogeneous computer architecture and systems.

Acknowledgements This research was supported partially by the National Science Foundation through the award OCI-1047916.

References

Bandini, S., Mauri, G. & Serra, R., 2001. Cellular Automata: From a Theoretical Parallel Computational Model to Its Application to Complex Systems. *Parallel Computing*, 27, pp. 539–553.

Batty, M., Xie, Y. & Sun, Z., 1999. Modeling urban dynamics through GIS-based cellular automata. *Computers, Environment and Urban Systems*, 23(3), pp.205–233.

Benjamin, S.C., Johnson, N.F. & Hui, P.M., 1996. Cellular automata models of traffic flow along a highway containing a junction. *Journal of Physics A: Mathematical and General*, 29(12), pp.3119–3127.

Clarke, Keith C. & 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(7), pp.699–714.

Clarke, Keith C., Hoppen, S. & 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(2), pp.247–261.

Clarke, Keith C., Riggan, P. & Brass, J.A., 1995. A cellular automaton model of wildfire propagation and extinction. *Photogrammetric Engineering and Remote Sensing*, 60(11), pp.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(2), pp.165–174.

Dietzel, Charles & Clarke, Keith C, 2007. Toward Optimal Calibration of the SLEUTH Land Use Change Model. *Transactions in GIS*, 11(1), pp.29–45.

Goldstein, N.C., Dietzel, C. & Clarke, K. C., 2005. Don't stop 'til you get enough–sensitivity testing of Monte Carlo iterations for model calibration. In *Proceedings of the 8th International Conference on GeoComputation*. Ann Arbor Michigan.

Guan, Q. & Clarke, K. C., 2010. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. *International Journal of Geographical Information Science*, 24(5), pp.695–722.

Hecker, C. et al., 1999. System Development for Parallel Cellular Automata and Its Applications. *Future Generation Computing Systems*, 16(2–3), pp.235–247.

Li, D. et al., 2012. GPU-CA model for large-scale land-use change simulation. *Chinese Science Bulletin*, 57(19), pp.2442–2452.

Li, X. et al., 2013. Calibrating cellular automata based on landscape metrics by using genetic algorithms. *International Journal of Geographical Information Science*, 27(3), pp.594–613.

Li, X. & 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(2), pp.131–152.

Li, X. & 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(4), pp.323–343.

Liu, Xiaoping 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, pp.783–802.

Liu, Y. & Phinn, S.R., 2003. Modelling urban development with cellular automata incorporating fuzzy-set approaches. *Computers, Environment and Urban Systems*, 27(6), pp.637–658.

Nagel, K. & Schreckenberg, M., 1992. A cellular automaton model for freeway traffic. *Journal of Physics I France*, 2, pp.2221–2229.

Silva, E.A. & Clarke, Keith C., 2002. Calibration of the SLEUTH Urban Growth Model for Lisbon and Porto. *Computers, Environment and Urban Systems*, 26(6), pp.525–552.

Spezzano, Giandomenico & Talia, Domenico, 1999. Programming Cellular Automata Algorithms on Parallel Computers. *Future Generation Computing Systems*, 16(2), pp.203–216.

Thor, M., 2008. *Performance comparison of CPU and GPU based simulation of an avalanche using a cellular automata*. Master Thesis. Sweden: Ume°a University.

Wu, F. & Webster, C.J., 1998. Simulation of Land Development through the Integration of Cellular Automata and Multi-criteria Evaluation. *Environment and Planning B*, 25(1), pp.103–126.

Wu, N. & Silva, E.A., 2010. Artificial Intelligence Solutions for Urban Land Dynamics: A Review. *Journal of Planning Literature*, 24(3), pp.246–265.