5-2010

A general-purpose parallel raster processing programming library test application using a geographic cellular automata model

Qingfeng Guan
School of Natural Resources, University of Nebraska-Lincoln, qguan2@unl.edu

Keith C. Clarke
Department of Geography, University of California, Santa Barbara, kclarke@geog.ucsb.edu

Published in *International Journal of Geographical Information Science* 24:5 (May 2010), pp. 695–722;
doi: 10.1080/13658810902984228
http://www.informaworld.com

Published online March 2, 2010

A general-purpose parallel raster processing programming library test application using a geographic cellular automata model

Qingfeng Guan
Center for Advanced Land Management Information Technologies,
School of Natural Resources, University of Nebraska-Lincoln, Lincoln, NE, USA

Keith C. Clarke
Department of Geography, University of California, Santa Barbara, Santa Barbara, CA, USA

Abstract
A general-purpose parallel raster processing programming library (pRPL) was developed and applied to speed up a commonly used cellular automaton model with known tractability limitations. The library is suitable for use by geographic information scientists with basic programming skills, but who lack knowledge and experience of parallel computing and programming. pRPL is a general-purpose programming library that provides generic support for raster processing, including local-scope, neighborhood-scope, regional-scope, and global-scope algorithms as long as they are parallelizable. The library also supports multilayer algorithms. Besides the standard data domain decomposition methods, pRPL provides a spatially adaptive quad-tree-based decomposition to produce more evenly distributed workloads among processors. Data parallelism and task parallelism are supported, with both static and dynamic load-balancing. By grouping processors, pRPL also supports data-task hybrid parallelism, i.e., data parallelism within a processor group and task parallelism among processor groups. pSLEUTH, a parallel version of a well-known cellular automata model for simulating urban land-use change (SLEUTH), was developed to demonstrate full utilization of the advanced features of pRPL. Experiments with real-world data sets were conducted and the performance of pSLEUTH measured. We conclude not only that pRPL greatly reduces the development complexity of implementing a parallel raster-processing algorithm, it also greatly reduces the computing time of computationally intensive raster-processing algorithms, as demonstrated with pSLEUTH.

Keywords: parallel computing, raster, cellular automata, SLEUTH

1. Introduction

1.1. Parallel computing for geospatial processing

Armstrong (2000) explicated that computational science has expanded the traditional view of scientific frameworks to include modeling and simulation with a separate and equal role to theory and experimentation. He pointed out that geographers have been exploiting the benefits of advances in computational science, and using modeling and simulation to gain understanding in geography for decades. However, computational complexity and poor algorithmic performance has hampered the use of models to support scientific investigation and decision-making in geographic information science (GIScience). Saalfeld (2000) similarly considered computational complexity and intractability as the limitations of many geographic information system (GIS) and cartography algorithms. Two trends in

geospatial processing have caused an increase in computational complexity: the rapid increase in the volume of geospatial and geography-related information, and the increased sophistication and complexity of geospatial algorithms and models (Armstrong 2000). In many cases, researchers have to choose less-satisfying alternatives because the 'best' solutions appear to be computationally infeasible. Undoubtedly, geospatial practitioners' demand for computing power will increase at an increasing rate.

Parallel computing, in contrast to sequential computing, is the use of multiple processing units (computers, processors, and processes) working together on a common task in a concurrent manner in order to achieve higher performance, usually measured by computing time. A variety of parallel computing systems have been developed and brought into applications, including massive parallel computers, computer clusters, computational grids, and the recently emerged multi-core CPU computers. The adoption of parallel computing has widely spread in many fields, including weather forecasting, nuclear engineering, mineral and geological exploration, astrophysical modeling (Cosnard and Trystram 1995).

For geospatial practitioners, parallel computing provides a means to overcome computational performance limits to solve geospatial problems in a shorter period of time, even to solve problems that are computationally infeasible using sequential computing technology. Besides the high computational throughput, Ding and Densham (1996) pointed out that the principal benefit that geographers can obtain from parallel computing is 'an improved understanding of how to represent space and spatial relations and how to exploit them in analytical procedures'. Openshaw and Turton (2000) also explained that modeling and simulation based on parallel computing are more natural and realistic than sequential modeling because most geographical phenomena are results of multiple concurrent processes and their complex relationships.

Research on parallel GIS began more than 20 years ago, and many studies on geospatial applications of parallel computing have been conducted, for example, in transportation and land-use modeling (Harris 1985), spatial data handling and analysis (Sandu and Marble 1988, Li 1992), least cost path (Smith *et al.* 1989), polygon overlay (Wang 1993), terrain analysis (Peucker and Douglas 1975, Rokos and Armstrong 1992, Puppo *et al.* 1994, Kidner *et al.* 1997), geostatistics (Armstrong and Marciano 1993, 1995-1997, Cramer and Armstrong 1997, Kerry and Hawick 1997, 1998, Wang and Armstrong 2003), and earth observation (Aloisio and Cafaro 2003, Ananthanarayan *et al.* 2003). The research has moved in two main directions: parallelizing existing computationally intensive GIS operations and developing new geospatial analytical methods using additional computational power (Clematis *et al.* 2003).

However, after more than two decades of research, it is still hard to say whether the discipline of geography, especially GIScience, has entered the high-performance computing (HPC) era, or, to be precise, the parallel computing era. An undeniable fact is that most current geographical analytical tools and simulation models are still based on sequential processing and completely outside the parallel computing world. Clematis *et al.* (2003) suggested that two factors 'have had roles to play in obstructing the rise of parallel GIS': the inaccessibility of HPC resources and the lack of parallel GIS algorithms, application libraries, and toolkits. The first obstacle will eventually diminish in light of the newly emerged parallel computing technology that allows GIScientists to either build inexpensive parallel computing systems (e.g., cluster computing and multi-core CPUs) or access open HPC facilities (e.g., grid computing and cloud computing). The solution to the second obstacle may have the following two dimensions.

1. Even though some parallel geospatial algorithms and models have been developed, most of them are simple parallelism of algorithms and models. Very few existing parallel geospatial algorithms and models have included the delicate load-balancing process, i.e. decomposing the data/tasks and mapping the subsets of data/tasks efficiently onto multiple processors. Thus, a study to compare existing load-balancing methods and even develop special ones for parallel geospatial algorithms is necessary.
2. Because computer geospatial processes vary in data requirements, operation scope, and more importantly, workflow, it may be very hard for a generic parallel geospatial programming library or middleware to fully utilize parallel computing hardware for all the different kinds of geospatial computing. On the other hand, designing and implementing a parallel algorithm requires extensive knowledge and experience of parallel computing and programming, which appears to be less possible for GIScientists. General-purpose parallel geospatial programming libraries are in great need for current and future geospatial research and applications, even though special parallelism of a specific algorithm or model may achieve higher performance. Thus, if a generic parallel geospatial library is to be developed, it should be 'general' enough to allow users to develop their specific parallel algorithms for different geospatial processes. In other words, a general-purpose library should greatly reduce the development complexity of parallel algorithms and models without losing much performance improvement. Hutchinson *et al.* (1996) referred to the approach of hiding parallel computing details from nonspecialist users as the 'transparent parallelism' of geospatial processing.

This study aims to tackle these two aspects of the second obstacle by developing a general-purpose parallel geospatial programming library that provides users with multiple load-balancing options, including a workload-adaptive balancing mechanism. The unique features of this library, as well as some computational complexity and dependency analysis and considerations during the designing process, are presented in this article. Also, a parallel geographic cellular automata (CA) model developed using this library is presented to demonstrate the usability and computational performance of this library.

1.2. Parallel raster processing

The raster data model, one of the most widely used data models for geospatial data, represents phenomena in geographical space as a grid of cells with attribute values (Duckham *et al.* 2003). This simple data structure is able to represent a large range of computable geospatial objects, e.g., points, lines, areas, and surfaces or fields as a series of discrete regular cells (Worboys and Duckham 2004). From a programming perspective, a raster can be managed with ease in computers, because it can be mapped directly onto a computer memory structure called the array (Clarke 2003b), and most commonly used programming languages provide utilities to handle arrays (Worboys and Duckham 2004). The raster has been used as a primary data model for analytical geospatial processing and modeling, e.g., land-use and land-cover change modeling, and terrain analysis and modeling. The raster model has formed the basis of entire GIS software suites in the past, and today it is at least supported as a storage and computational data structure in virtually all GIS software. Commonly stated advantages of the raster data model include the fact that the geographic location of each cell is implied by its position in the grid; spatial analyses based on neighboring cells are easy to program and quick to perform; the easy link to the layer concept; large quantities of GIS data are already in this format (e.g., elevation data, imagery); and grid systems are compatible with raster-based output devices (Clarke 1995).

From a parallel computing perspective, the raster was born to be parallelized. A raster data set is a matrix of values at a finite resolution, each of which represents the attribute of a corresponding cell on the land surface. Any matrix can be easily partitioned into sub-matrices and assigned to multiple processors for parallel computing.

However, in real-world applications, the processes can be much more complicated. First, the characteristics of the grids used in geospatial applications raise concerns for parallel computing. We define within raster processing a domain as the grid of cells or *cellspace*. There is a classic division within parallel computing between the partitioning of data across processors (i.e., data parallelism) and the partitioning of operations across processors (i.e., task parallelism), with the result that both require an overhead of communications among the processors. To classify spatial processing and modeling from the data parallelism perspective, Ding and Densham (1996) defined two key characteristics: regularity and homogeneity. The regularity of a spatial domain determines how easily it can be decomposed geometrically into equal areas, whereas the homogeneity of the domain's members affects the balance among partition workloads. A lack of homogeneity or regularity can increase the communication cost, and lead to workload imbalances. For raster processing, the cellspaces are mostly of regular shapes (commonly squares and rectangles), which implies they can be easily divided into sub-cellspaces of equal areas. However, the cells in a cellspace may have different properties. In some applications, different types or values of cells will be processed more or less frequently and so require differing computational intensities across space. For example, when a line-thinning algorithm is applied to a cellspace of lines, only those cells that are identified as lines need to be processed, whereas other cells will be ignored. Heterogeneity in computational intensity over the cellspace is very frequently encountered in geospatial processing and creates workload imbalance among the sub-cellspaces, and so among the processors onto which the sub-cellspaces are assigned by data parallelism.

Second, the scope of raster processing and modeling, which determines the range of data dependencies among sub-cellspaces, needs to be examined in parallel computing. Tomlin's (1990) classification of map algebra is useful in classifying raster processing and modeling, and divides raster operations into those of local scope, neighborhood scope, regional scope, and global scope.

When a local-scope algorithm is used, e.g., selecting cells by attributes, each cell is processed in isolation. In this case a sub-cellspace only needs to hold the block of cells that will be processed by the algorithm. When a neighborhood-scope algorithm is used, for example, calculating slopes and aspects from elevations, the computation for a given cell requires the values of its neighborhood, a set of predetermined surrounding cells (sometimes including the cell itself). In consequence, a sub-cellspace has to hold not only the block of cells to be processed, but also a 'halo' of cells surrounding the block (Mineter 1998). In iterative algorithms such as CA, the 'halos' of a sub-cellspace must be updated after each iteration. Note that local-scope algorithms are special cases of neighborhood-scope algorithms, those where the neighborhood contains only the cell to be processed itself. When a regional-scope algorithm is used, such as calculating the mean elevations of the states in the United States, all the cells within a certain region are required in the computation for the region. Two approaches to decomposition can be used (Mineter 1998). The first divides the cellspace by regions, which means a sub-cellspace holds the cells of a certain region, but this approach is likely to lead to load imbalances as the regions differ in size. The second approach divides the cellspace into regularly shaped sub-cellspaces, but extra computation and communication have to be added to account for the region frag-

mentation. When a global-scope algorithm is used, all the cells in the cellspace are needed to generate the result. This kind of algorithm is the hardest to parallelize because it exhibits a large range of horizontal data dependence (Ding and Densham 1996).

To parallelize a raster-processing algorithm, the characteristics of the data sets (i.e., regularity and homogeneity), and the nature of the algorithm itself (i.e., scope), have to be taken into account. If a general tool is to be developed for parallelizing raster algorithms, it clearly has to be sufficiently general to handle all kinds of data characteristics and processing scopes. Some efforts have been made to provide transparent parallelism of raster processing. One example is the parallel raster-based neighbourhood modelling (NEMO) system developed at Carleton University in Canada (Hutchinson *et al.* 1996), which allows GIS researchers to easily parallelize three types of neighborhood modeling processes: neighborhood analysis, CA, and propagation. However, NEMO was specifically developed for neighborhood-scope raster operations; a more generic parallel programming environment that supports all four types of raster processing in Tomlin's classification is still lacking.

In this study, an open-source general-purpose parallel raster processing programming library called pRPL was developed in the C++ programming language to handle parallel computing processes for massive-volume raster processing. pRPL provides a simple but powerful interface for users to parallelize almost any raster-processing algorithm as long as it is parallelizable, with any arbitrary neighborhood configuration. pRPL enables the implementation of parallel raster-processing algorithms without requiring a deep understanding of parallel computing and programming; thus it reduces the development complexity significantly. To demonstrate the advantages and performance of the library, a parallel geographic CA model, called pSLEUTH, was developed based on the SLEUTH urban growth and land-use change model (Clarke *et al.* 2007). pSLEUTH and therefore pRPL were used to simulate and forecast urban growth over time. The goals were to demonstrate the value of pRPL and get real-world performance data to evaluate our approach.

2. pRPL: an open-source general-purpose parallel raster processing programming library

The goal in developing pRPL was to provide GIScientists who do not have experience with parallel computing with an easy-to-use development toolkit to parallelize their own raster-processing algorithms. From a software architecture perspective, pRPL serves as middleware connecting a general-purpose parallel programming library with application-specific raster-processing algorithms. pRPL hides the technical details of parallel computing from the users, thus relieving them from the time-consuming coding steps of parallel programming, and allowing them to focus on the algorithms themselves. Mineter and Dowers (1999) referred to this kind of architecture as a layered approach for the parallel processing of geographical applications (Figure 1).

pRPL was written in the programming language C++ and built upon message-passing interface (MPI), a standard parallel programming library composed of a set of functions that enable and manage parallel computing by passing messages among processors (Gropp *et al.* 1998). MPI and C++ compilers are available on almost all parallel computing systems (e.g., massive parallel computers, computer clusters, and computational grids), so the portability of pRPL is guaranteed, and applications built upon it should be portable across different parallel computing platforms. Furthermore, pRPL is an open-source programming library and can be freely downloaded from http://sourceforge.net/projects/prpl/

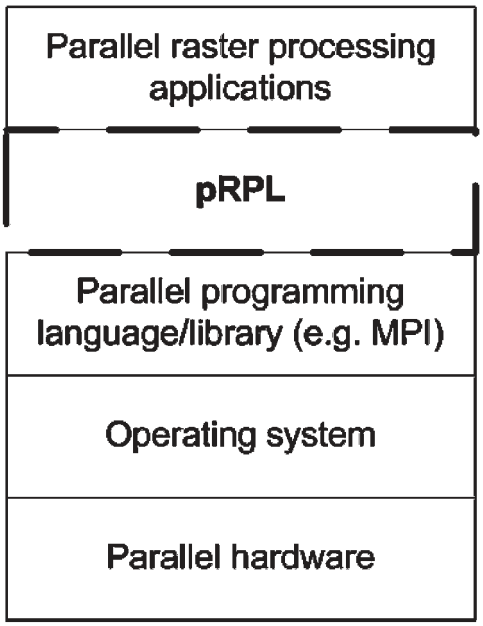

Figure 1. pRPL as a layered architecture.

2.1. Features of pRPL

2.1.1. Object-oriented programming library and class templates

pRPL is an object-oriented Programming (OOP) library, which provides a set of C++ classes for users as the interface to manage and control the underlying parallel processes with relative ease. As it is a template programming library, it supports any type of cell attribute values commonly used in GIS and GeoComputation, e.g., short integer numbers for digital elevation models (DEM), and single and double precision floating point numbers for slopes and aspects. Users are also allowed to define more complex data structures and use them in pRPL. For example, to deal with the mixed pixel problem in land-cover raster data, users may define a complex land-cover data structure that consists of a set of integer numbers representing the possible land-cover types in a pixel (or cell) and a set of double precision floating point numbers representing the corresponding land-cover types' areal proportions within a mixed pixel. OOP and class templates enable pRPL users to efficiently handle various types of geospatial data without losing precision.

2.1.2. General support for raster processing

As a general-purpose library, pRPL was developed with the goal of parallelizing any raster-processing algorithm as long as the computation is parallelizable, i.e., the computation for a certain cell within the output cellspace is independent from that for other cells. [1] Note that computational independence does not imply data independence, which means an algorithm requiring multiple input cells to calculate the value of a certain output cell may still be parallelizable. Thus, neighborhood-scope (or moving-window-based) and lo-

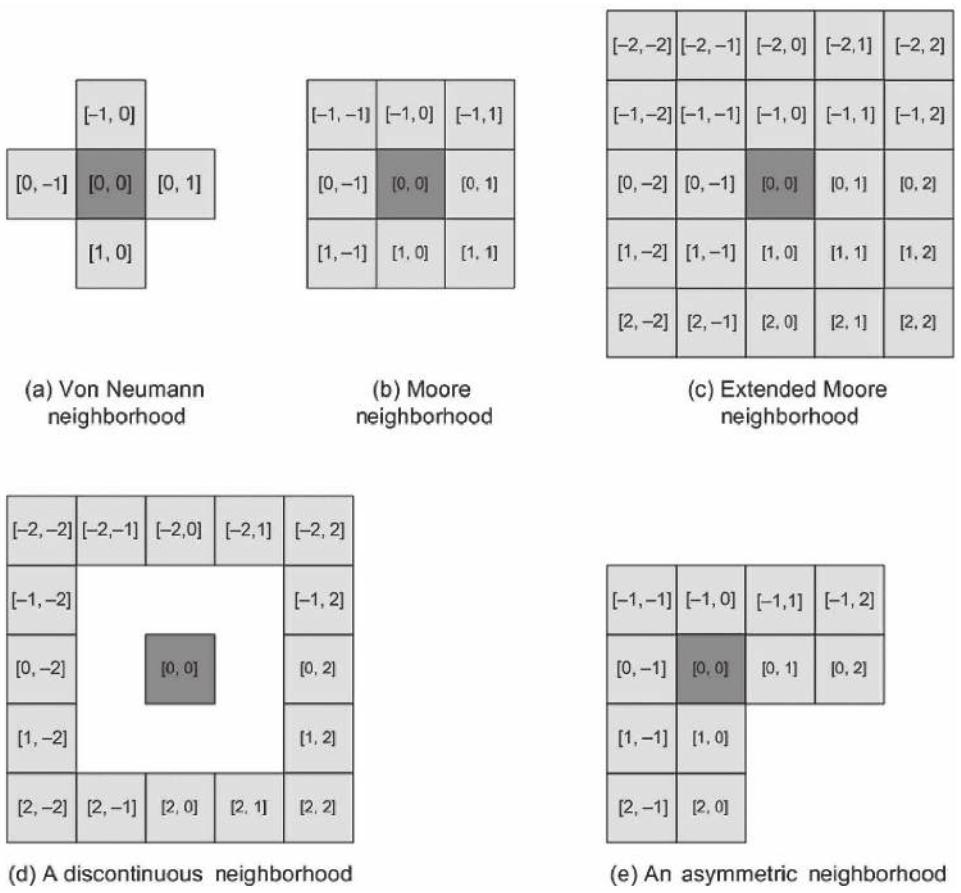

Figure 2. Any neighborhood configuration can be used in pRPL.

cal-scope algorithms are fully supported by pRPL, and some regional-scope and global-scope algorithms that meet the parallelizability criterion are also supported.

At the most basic level, pRPL provides a class called *Neighborhood* for users to specify any arbitrary neighborhood configuration for the algorithm (Figure 2). The neighborhood can be either continuous or discontinuous (Figure 2d), and either symmetrical or asymmetrical (Figure 2e). Most raster-processing applications use regular neighborhood configurations, e.g., von Neumann, Moore, and extended Moore. Some irregular neighborhoods, however, exist in GIS modeling. For example, Batty (2000) discussed the use of irregular neighborhoods in CA models. In pRPL, the spatial configuration of a neighborhood is specified by setting the central cell's row-column coordinate as [0, 0] and the surrounding cells' coordinates according to their locations relative to the central cell (Figure 2). Furthermore, the cells within a neighborhood can be assigned different weights for kernel-based local processing (Lloyd 2007).

To make pRPL general enough to support as many sorts of raster processing as possible, the characteristics of raster-processing algorithms were considered when designing and implementing pRPL: centralization, finalization, and the exchange requirement. In pRPL, the raster-processing algorithm to be parallelized can be either centralized or noncentral-

ized. A centralized algorithm evaluates and updates only the central cell of the neighborhood, whereas a noncentralized algorithm evaluates and updates any cell(s) within the neighborhood or the cellspace. Centralized algorithms are well supported by existing parallel neighborhood-scope raster-processing environments, e.g., NEMO. The support for noncentralized algorithms is, however, a distinctive feature of pRPL. This enables users to parallelize regional-scope and global-scope processing (e.g., linear feature tracing), as well as some special cases of neighborhood-scope processing. Examples of parallelizing noncentralized algorithms will be presented in the following case study. Also, in pRPL, a finalizing process can be included in the algorithm, e.g., assigning a value to the cell based on an intermediate value of the cell calculated during the evaluation process. As mentioned before, an iterative neighborhood-scope algorithm will require data-exchange processes among the sub-cellspaces after each iteration, which is handled by pRPL without any extra programming on the user side.

When defining an application-specific raster-processing operation (termed *Transition* in pRPL), the user simply turns ON/OFF the options provided by pRPL to specify the centralization (centralized/noncentralized), finalization (finalizing/nonfinalizing), and exchange (exchanging/non-exchanging) characteristics of the transition. The pRPL library will automatically optimize the underlying parallel computing routines according to these characteristics.

pRPL also supports multilayer transitions. Multilayer processing is commonly used in raster-based geospatial algorithms, as well as in other kinds of image processing, for example, an AND operation on two binary-coded images. pRPL allows users to use multiple *Layer* objects in a *Transition*, which implements the raster-processing algorithm, where each *Layer* object may contain a cellspace and/or multiple sub-cellspaces. Once a cellspace in a *Layer* is decomposed, the sub-cellspace information (the locations in the global spatial extent and the local extents) can be used for other *Layers* so that they can be decomposed and distributed in exactly the same way so as to ensure the sub-cellspaces in multiple *Layers* will match at geographical locations.

2.1.3. Regular and irregular decomposition

pRPL provides both regular and irregular decomposition methods to divide the domain (Figure 3). Regular decomposition divides the cellspace by rows, or columns, or blocks, without considering the workloads associated with the cells, producing equal-area sub-cellspaces. Regular domain decomposition methods have been widely used in parallel computing for numerical algorithms and image processing, where the workload is usually evenly distributed over the domain. Regular decomposition can also be used on heterogeneous domains using the scattered decomposition technique (Mineter 1998). pRPL allows users to partition the domain into a large number of sub-cellspaces and lets each processor hold multiple sub-cellspaces scattered throughout the domain so as to increase the chance of an even distribution of workload across the processors. However, scattered decomposition will also increase the overhead of communication for data exchange in iterative algorithms as the number of sub-cellspace increases. pRPL leaves the trade off between workload distribution and communication overhead to the user, allowing a choice of how many sub-cellspaces are produced in the decomposition process.

Irregular decomposition (or spatially adaptive decomposition), on the other hand, is suitable for the situation when the workload is clustered within the domain. Irregular decomposition takes into account the workloads of the cells, and is likely to produce unequal-area sub-cellspaces, but also likely to divide the workload more evenly among the

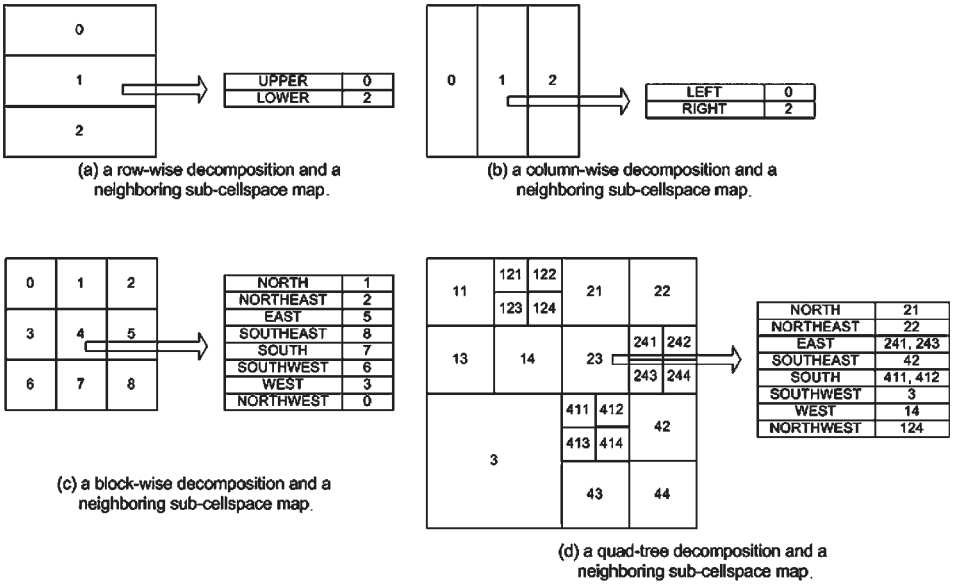

Figure 3. Examples of neighboring sub-cellspace ID map.

sub-cellspaces. Irregular decomposition techniques for use in parallel raster processing already exist, e.g., hierarchies of irregular tessellations (Montanvert *et al.* 1991), and heuristic partitioning (Mineter 1998). pRPL provides a quad-tree-based (QTB) decomposition method, inspired by Wang and Armstrong (2003). Wang and Armstrong used a QTB spatial domain decomposition technique for parallel point interpolation, where the number of control points within a subregion was used to determine the workload. The QTB decomposition is an iterative process. The global cellspace (i.e., the root of the quad-tree) is initially divided into four equal-area sub-cellspaces (termed *leaves*). Then, at each iteration, the workloads associated with the leaves are calculated, and the leaf with the largest workload is divided into four equal-area child leaves. The quad-tree keeps growing until the maximum number of leaves or the minimum workload associated with a sub-cellspace is reached. Note that the calculation of workload is a user-defined process (Section 2.2) and users have the freedom to design workload-calculation algorithms according to their own raster-processing algorithms, which is another distinctive feature of pRPL. Caution must be taken when the QTB decomposition is to be used, because the extra computation required to construct the quad-tree and to calculate the workloads of the sub-cellspaces may outweigh the speed-up gained by a better workload distribution. Irregular decomposition is suitable for heterogeneous domains where the workload is highly clustered. Therefore the QTB decomposition may not yield better performance than other regular decomposition methods when the workload is rather scattered over the space. Also, the workload of a sub-cellspace may change as the cell values change, thus the QTB decomposition is less preferable for iterative algorithms.

Once a cellspace is decomposed, the spatial relationships among the sub-cellspaces are automatically determined by pRPL and stored in a neighboring sub-cellspace ID map within a *SubSpace* object for rapid query (Figure 3). The process of finding neighbors in quad-trees is quite complex and time-consuming (Samet 1982, Samet 1984, Samet 1990). In pRPL, the neighboring sub-cellspace ID map for the QTB decomposition is constructed using an algorithm based on Bhattacharya's (2001) research.

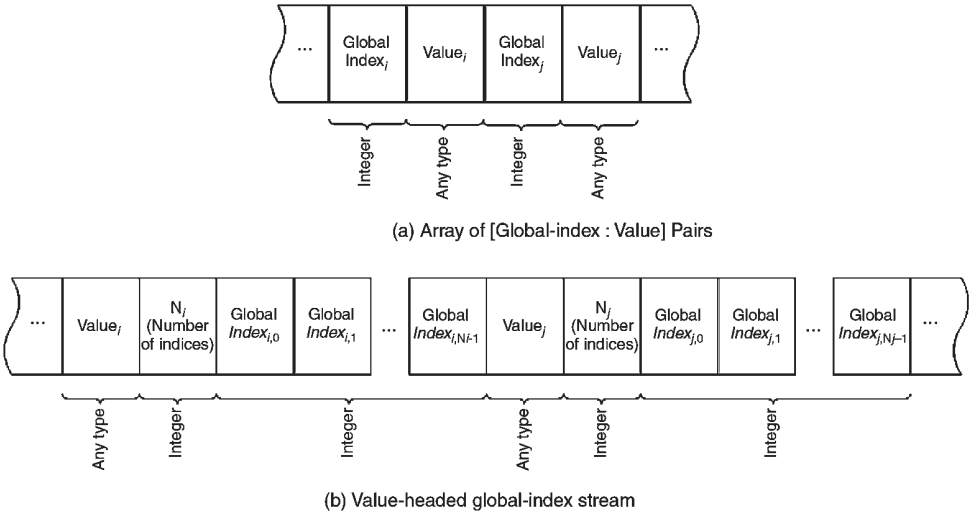

Figure 4. Array of [global-index: value] pairs (a) and value-headed global-index stream (b).

2.1.4. 'Update-on-Change' and the value-headed global-index stream for data exchange

When a neighborhood-scope algorithm is used, the computation of a specific cell value requires the values within its neighborhood. In consequence, a sub-cellspace has to hold not only the block of cells to be processed, but also a 'halo' of cells surrounding the block (Mineter 1998). The halo can be seen as a replica of the cells within the neighboring sub-cellspaces of the sub-cellspace, which may be updated after each iteration if an iterative algorithm is used. When the origins are changed in value, the replicas must be updated accordingly. This inevitably creates communication overhead (i.e., messages between processors in the MPI environment). However, not all the halo cells' values are changed after a given iteration, and it would be inefficient to refresh all the halo cells at every iteration. We therefore developed a data-exchange technique called 'Update-on-Change', which only updates the halo cells whose origins have been changed at an iteration, significantly reducing the volume of messages passed among processors. For the message structure, the array of [global-index : value] pairs (AGVP) may be the most straightforward model for design and implementation (Figure 4a). Note that the spatial indices discussed here are determined in the global spatial extent of the whole cellspace, and can be easily translated into/from local coordinates within sub-cellspaces. However, in many cases there are only a limited number of attribute values that can be assigned to a cell, and many cells will be updated with the same values, thus there is no need to include both the spatial index and the attribute value of every updated cell in the messages as an AGVP does. We therefore developed a message structure called the value-headed global-index stream (VGS) to further minimize the message volume. A VGS (Figure 4b) starts with an attribute value (of any type) and an integer number indicating the number of indices following this attribute value, followed by a string of global indices of the cells that are to be updated with this value; then another attribute value and an integer number, and a string of indices of cells updated with the second value; and so forth. A VGS is usually more efficient than an AGVP in terms of message volume. For example, in a cellspace of integer attributes, if there are M attribute values that the halo cells will be updated with (requiring $2M$ integers for the head information of a VGS), and N halo cells to be updated (requiring N integers for the spatial

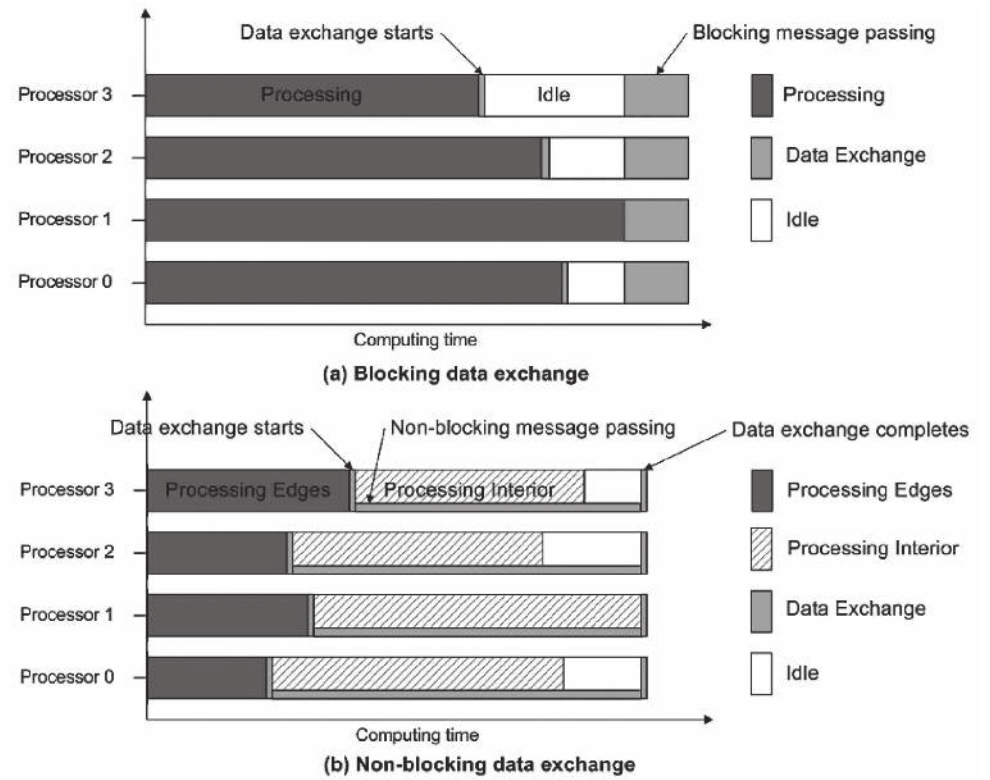

Figure 5. Blocking data exchange (a) vs. non-blocking data exchange (b).

indices of a VGS), then $(2M + N) \times sizeof(integer)$ bytes are needed for a VGS message, but $2N \times sizeof(integer)$ bytes are needed for an AGVP. When M is small and N is big (which is usually the case), the VGS will be much smaller than the AGVP.

2.1.5. Non-blocking communication and 'edgesFirst' for data exchange

In parallel processing, blocking is halting processing on a processor (waiting) until a task is completed on another processor, and a result communicated as a message (Cosnard and Trystram 1995). Instead of blocking communication, pRPL uses the opposite, non-blocking communication routines that allow message passing while the processors are working on other tasks (e.g., computation, I/O process) (Figure 5). pRPL provides an 'edgesFirst' option for users, which forces the processors to first process the sub-cellspace edges, then starts exchanging the update information. During the data-exchange communication, the processors continue working on the interiors of the sub-cellspaces and then finally complete the communications. The non-blocking communication and 'edgesFirst' method provide a means to improve the performance by overlapping communication and computation and to shorten the idle time of processors.

When a *Transition* is used on a sub-cellspace, the sub-cellspace's halos, edges, interior, and send ranges are automatically determined by pRPL according to the *Transition*'s characteristics, the neighborhood configuration, and the sub-cellspace's location within the global spatial extent (Figure 6). The halos, as mentioned before, are the cells surrounding the block of cells to be locally processed. The send ranges include the cells whose val-

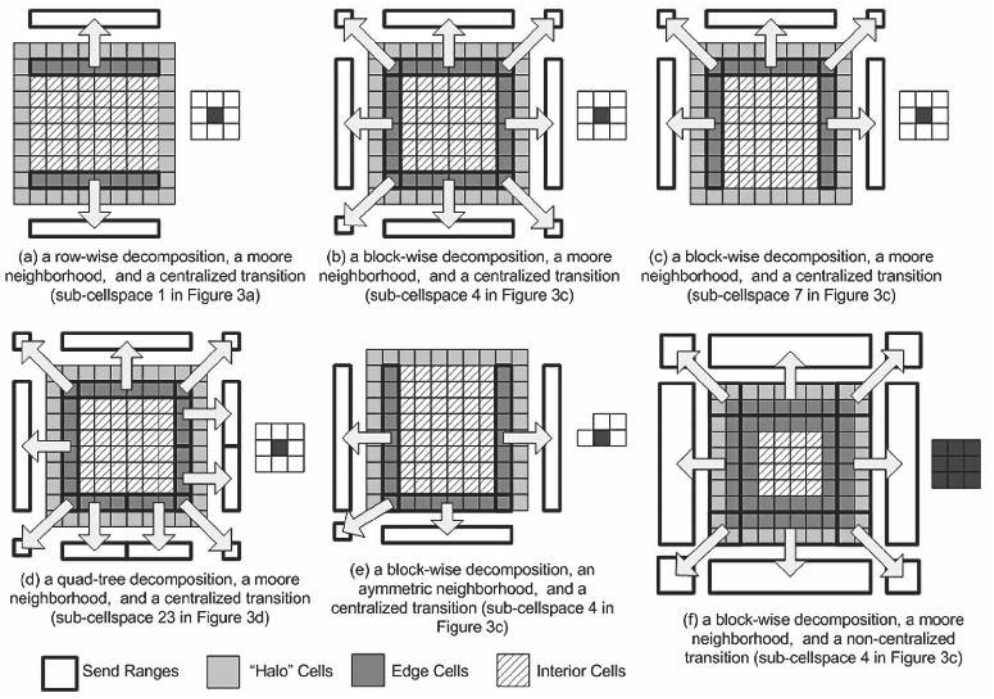

Figure 6. Determining bounding rectangles for sub-cellspaces.

ues are to be sent to the neighboring sub-cellspaces. The edges are the cells that may affect the values of the cells within the send ranges. When centralized algorithms are used, the edges are the same as the send ranges (Figure 6a-e). When noncentralized algorithms are used, the send ranges include the halos and a part of the edges (Figure 6f). The interior is the rest of the sub-cellspace besides halos and edges. When data exchange is needed, for example, after an iteration in an iterative algorithm, the changed cells within the send ranges are compressed into VGSs and sent to the neighboring sub-cellspaces, and also VGSs from the neighboring sub-cellspaces are received and the corresponding cells (usually halos) are updated accordingly.

2.1.6. Processor grouping and data-task hybrid parallelism

pRPL organizes processors into groups. The initial set of processors (i.e., all the processors engaged in the processing) constitute the root group. The root group can be divided into sub-groups, which can be further divided into smaller groups. Each group consists of a handler processor, or master, and a set of worker processors. Groups of processors can be assigned with different data sets and the same transition process, or the same data set and different transition processes, or different data sets and different transition processes. Thus pRPL supports both data parallelism and task parallelism. Data parallelism means dividing the data and assigning the partitions onto multiple processors, whereas task parallelism divides a task set (e.g., different processes or different parameter values) and assigns the subsets to the processors. In the case of pRPL, when a data set is divided and distributed among the processors within a processor group, data parallelism takes place. When

different transition algorithms or transition parameters are assigned to multiple processor groups, task parallelism among groups takes place among groups (within groups, data parallelism takes place because the data set will be divided and distributed among the processors). We call this parallelization approach the data-task hybrid parallelism. The hybrid parallelism has obvious advantages over both data parallelism and task parallelism. If only task parallelism is used, subsets of tasks are assigned to multiple processors, and they execute the processing on the whole cellspaces simultaneously. Given that the data sets are usually large in volume (500+-MBe raster data sets are now commonly encountered in GIS applications), the limited high-speed cache space of a single processor (even the latest multi-core CPUs have less than 20-MB caches) is insufficient to accommodate the whole data set, thus the data will have to be temporally stored in the relatively slow random-access memory (RAM) and even slower virtual memory, i.e., the hard disk, greatly degrading the performance. More importantly, large volume introduces large workload, meaning a considerable number of cells within the cellspace have to be processed, requiring more extensive computing time on a processor. Data parallelism improves the performance by giving each processor only a portion of the whole data set so as to reduce the amount of data to be stored in the less-efficient RAM and virtual memory and reducing the overhead of retrieving data from RAM and virtual memory to the processor cache. This also reduces the workload for each processor, eventually shortening the computing time. However, theoretically the efficiency of data parallelism will decline as the number of processors increases, a common trait of parallel computing and following the Law of Diminishing Returns (Cannan 2001). The cost of communication overhead will eventually outweigh the benefit of computing power as the number of processors increases. Data-task hybrid parallelism divides the data among the processors within a group, and divides the tasks among the groups, thus utilizing the processors in a more efficient way. To the authors' knowledge, no existing similar parallel raster-programming libraries provide such a mechanism to handle data-task hybrid parallelism, and we consider this a major contribution to parallel raster processing.

Processor grouping and hybrid parallelism are especially useful when the application requires a massive volume of data sets and involves a large number of parallelizable tasks (e.g., parameter combinations to be evaluated). However, processor grouping may be less preferable for pure data parallelization of an iterative algorithm, because the data exchange and synchronization among groups will have to be explicitly handled by users and inevitably increase the programming complexity.

2.1.7. Static and dynamic load-balancing

After a data set is divided and distributed among the processors within a group, the sub-cellspaces are statically assigned onto the processors, and will not be reallocated among the processors once the computation starts, because migrating sub-cellspaces between processors could cause extreme communication overheads, given that the sizes of sub-cellspaces are usually large. This load-balancing mechanism is referred to as static load-balancing. [2] However, in some cases, for example, in calibrating the parameters of a model, the same data set will be assigned to a set of processor groups, and a 'task farm' can be used to assign different tasks (for example, parameter scenarios in the demonstration application) to the groups in response to their requests. Thus, a dynamic load-balancing mechanism can be implemented among the processor groups. Both approaches may be desirable in modeling; accordingly pRPL supports both static and dynamic load-balancing.

2.2. Writing pRPL-based programs

Writing a parallel raster-processing program using pRPL is straightforward and requires minimal knowledge of parallel programming. [3] pRPL provides a base class called *Transition* for users to implement their own raster-processing algorithms by writing customized *Transition* classes derived from the base class. The base *Transition* class consists of a pointer to the cellspace to be processed (i.e., the output cellspace) and a pointer to the neighborhood to be used. Also the base class has five methods: *cellspace()*, *nbrhood()*, *evaluate()*, *finalize()*, and *workload()*. To customize the *Transition* class, users can add additional pointers to the extra layers (e.g., layers containing input cellspaces) for multilayer algorithms and overload the five methods according to the algorithms. Overloading is a feature available only to OOP languages. When deriving a customized class from the base class, overloading a method means to specify a distinguishing behavior/process for the customized class but keeping the same method name. Particularly, the *evaluate()* method is where the user implements the algorithm and has to be overloaded; the *finalize()* method needs to be overloaded if a finalizing process is needed by the algorithm; and the *workload()* method has to be overloaded to calculate the workload associated with a (sub-)cellspace if the QTB decomposition is to be used. No parallel programming is required for customizing the *Transition* class.

Writing a main function for a pRPL-based parallel program is as simple as writing a sequential C++ calling program. Simply calling the methods of the classes provided in pRPL will accomplish the decomposition, distribution, updating, and gathering of the cellspace. Particularly, the user uses customized Transition to process the sub-cellspace(s) contained in a Layer object by calling the *update()* method of the *Layer*. pRPL will evoke the customized (overloaded) methods of the *Transition* to update the *Layer*, and automatically handle the necessary data exchange. Thus many existing C++ programs that perform raster applications are candidates for parallelization using pRPL.

2.3. Other similar programming libraries

Programming libraries similar to pRPL for parallel raster-like processing exist. Examples are the global arrays (GA) developed by the Pacific Northwest National Laboratory, and the parallel utilities (PUL) developed by the Edinburgh Parallel Computing Centre at the University of Edinburgh.

The GA toolkit provides a shared memory-style programming environment in the context of distributed array data structures for users to manipulate 2D arrays. GA encapsulates all the details of data distribution, addressing, and data access in the GA objects; thus the GA can be used as if stored in the shared memory (Nieplocha *et al.* 2007).

The PUL is implemented as middleware on top of MPI and provides a suit of utilities to parallelize algorithms. PUL has been used to implement several fundamental vector-based and raster-based GIS operations, i.e., vector-to-raster conversion, raster-to-vector conversion, and vector polygon overlay, and in some GIS applications, such as raster generalization (Healey *et al.* 1998). The major differences between pRPL and these two parallel programming libraries are the following:

1. GA and PUL are written in Fortran and C, and are procedural-based programming libraries. GA is implemented as a library with C and Fortran bindings, and also provides Python and C++ interfaces. PUL provides C and Fortran interfaces. Users write GA-based or PUL-based programs by calling the functions defined in the libraries. On the other hand, pRPL is writ-

ten purely in the C++ language and is an object-based library. A collection of C++ classes is provided as the interface, hiding the parallel-processing details from users.

2. The consensus is that procedural-based programs written in Fortran and C are usually more efficient than object-based programs written in C++, whereas object-oriented programming has the advantage over procedural-based programming for its intuitiveness in the system design process. Also, the encapsulation, inheritance, abstraction, and polymorphism properties of OOP allow pRPL users to develop reusable and complicated algorithm classes with ease. The C++ language was chosen for pRPL in the consideration of general GIScience researchers who do not have much programming experience. The simplicity of usage was given higher priority than performance.
3. pRPL explicitly provides a base *Transition* class for users to implement their own raster-processing algorithms. The methods provided in the base class serve as a programming guideline for users to easily write the code. Furthermore, the implementation of the raster-processing algorithm by deriving from the base class is completely independent of the underlying parallel computing details that are automatically handled by pRPL and requires no parallel programming knowledge. This approach allows users to focus on their algorithm and greatly reduces the programming complexity. Neither GA nor PUL provides such a programming approach. Their users have to mix the implementation of the algorithms with parallel computing handlers, which inevitably increase the programming difficulty.
4. pRPL recognizes the spatial distribution of workload over the cellspace and provides the workload-sensitive QTB decomposition method to divide the data sets in a more balanced way. Both GA and PUL provide regular decomposition methods but do not have any function to directly produce irregular decompositions. To perform a workload-based decomposition, users of GA and PUL have to write their own functions, which can be quite complex.
5. pRPL provides an easy way to organize processors in groups, and supports both data parallelism and task parallelism, and more complicated data-task hybrid parallelism. GA and PUL are primarily developed for data parallelism and provide no such mechanism to support hybrid parallelism. To group processors, the users of GA and PUL must turn to the lower level general parallel-programming environment, e.g., MPI.

GA and PUL are aimed at experienced programmers who wish to fully utilize parallel computing systems and develop portable high-performance programs without spending too much time on the underlying details. pRPL was developed mainly for GIScientists to speed up their raster-processing algorithms and to provide an easy-to-use interface to parallelize raster algorithms. The implementation of raster-processing algorithms is separate from the parallel computing handlers and requires minimal parallel-programming knowledge. The spatial distribution of workload can be easily handled with pRPL, whereas much more effort in design and programming is required with GA and PUL for the same purpose. pRPL provides the processor-grouping mechanism and supports both data and task parallelisms, as well as data-task hybrid parallelism, whereas GA and PUL were primarily developed for data parallelism.

3. Case Study: implementing a geographic CA model using pRPL

3.1. Geographic CA models

A classical CA model has a set of identical elements, called cells, each of which 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 a set of cells, which include the state of the cell itself, and the states within its neighborhood.

One of the most important features of CA is that models can be used to simulate complex dynamic spatial patterns through a set of simple transition rules. CA models have been widely used in geographic research for about two decades to simulate complex spatio-temporal geographical phenomena, e.g., land-use and land-cover change (Clarke *et al.* 1997, Couclelis 1997, White *et al.* 1997, Clarke and Gaydos 1998, Wu and Webster 1998, Li and Yeh 2000, Yeh and Li 2001, Li and Yeh 2002, Silva and Clarke 2002, Yeh and Li 2002), wildfire propagation (Clarke *et al.* 1995), and freeway traffic (Nagel and Schreckenberg 1992, Benjamin *et al.* 1996). In many geographic CA models, multiple geospatial factors are considered while simulating geospatial phenomena. These factors are either presented as the input layers of models (e.g., elevations, slopes, transportation, and vegetation types), making them multilayer systems, or indicated by a set of parameters (e.g., slope sensitivity and road gravity) that reflect their contributions to the model and affect the model behaviors. Previous studies suggested that model parameters have significant impact on the simulation results of CA models (Wu and Webster 1998). Thus, calibration processes are needed to determine the appropriate parameter values so that CA models can produce more realistic simulation results.

Most geographic CA models use variants of the classical CA model and inherit its properties, i.e., a regular and discrete space consisting of a set of cells. Some unique geographic CA models exist, for example, Graph-CA (O'Sullivan 2001), but they are outside the scope of this article. As the classical CA model fits perfectly with raster processing, most current geographic CA models were implemented using raster-processing algorithms and applied to gridded raster data sets.

Many geographic CA models require considerable computing time in real-world applications because of the complicated algorithms and the large volume of their data sets. On the other hand, the classical CA model has been recognized to be a natural parallel computing system as the transition rules can be applied to the cells homogeneously and synchronously in parallel (Bandini *et al.* 2001). Several general parallel CA-based simulation systems have been developed for users to implement parallel CA applications. Examples include the cellular automata environment for systems modeling (CAMEL) and cellular programming environment (CARPET) language developed at the University of Calabria, Italy (Cannataro *et al.* 1995, Spezzano *et al.* 1996, 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 MPI and provide a similar approach to implementing user-defined CA algorithms to pRPL, which attempts to minimize the requirement for parallel programming knowledge. Like GA and PUL, CAMEL and Cell Driver were primarily developed for data parallelism. They neither explicitly support task parallelism nor provide an easy-to-use handler for processor grouping. Also, neither of them provides a workload-sensitive irregular decomposition method for heterogeneous domains.

We therefore chose to develop a geographic CA model using pRPL and to conduct a series of experiments using real-world data sets to demonstrate the usability and computational performance of pRPL.

3.2. The SLEUTH model

SLEUTH[4] is a CA model for urban growth and land-use change simulation and forecasting, developed in the Department of Geography, University of California, Santa Bar-

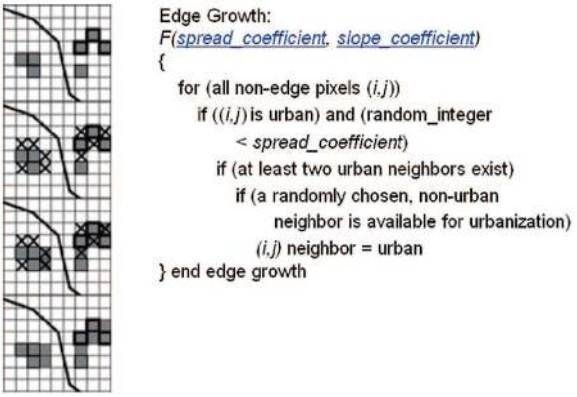

Figure 7. Edge growth example and pseudocode;
http://www.ncgia.ucsb.edu/projects/gig/v2/About/gwEdge.htm

bara (Clarke *et al*. 1997, Clarke and Gaydos 1998, Silva and Clarke 2002, Clarke *et al*. 2007). The urban growth model (UGM) part of SLEUTH uses a modified CA to simulate the spread of urbanization across a landscape. States of cells can be in the set {urban, nonurban} or they can be land-use classes. The model's name comes from the input layers required by the model: slope, land-use, exclusion (where growth cannot occur, e.g., the oceans and national parks), urban, transportation, and hillshade.

The basic unit of a SLEUTH simulation is a growth cycle, which usually represents a year of urban growth. A simulation (or a run) consists of a series of growth cycles that begins in the start year and completes in the stop year.

Five parameters (or coefficients) involved in SLEUTH determine the system behavior: *dispersion*, *breed*, *spread*, *slope*, and *road gravity*. Their values range from 0 to 100 and determine how the growth rules are applied. Four growth rules are applied on the space during each growth cycle: *spontaneous growth rule*, *new spreading centers rule*, *edge growth rule*, and *road-influenced growth rule*. Figure 7 shows the process of the *Edge growth rule*. Note that even though the pseudocode of the *Edge growth* appears simple, it accounts for most of the overall computational intensity because unlike other growth rules that only randomly choose a certain number of cells within the cellspace to evaluate and update, the *Edge growth* rule must examine every non-edge cell in the data (i.e., cells that are not on the edges of the cellspace). The urbanized cells introduce much more computational intensity into SLEUTH than do the nonurbanized cells.

Calibration is needed to determine the appropriate parameter values so that SLEUTH can produce more realistic simulation results. The basic calibration procedure of SLEUTH is to compare multiple test results produced using a set of parameter combinations with the real historical geospatial data set(s), in order to determine the best-fit parameter combination(s). This approach is called the 'brute-force' calibration. In addition, to simulate the random processes during urban growth, the Monte Carlo method is used. The model is applied not once but multiple times, and outcomes are stored as the cumulative probabilities of change over multiple runs. This method is commonly used when stochastic algorithms are in use, to provide a quantitative estimate of output variance. For a single parameter combination, a simulation may be executed multiple times with different Monte Carlo seed values. 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 the above together make the calibration process highly computationally intensive. In a comprehensive (full) calibration, all the possible combinations of the five parameter values (101^5 *in total*) need to be evaluated with multiple Monte Carlo iterations. If 100 Monte Carlo iterations were used, a comprehensive calibration over a 20-year period would consist of $101^5 \times 100 \times 20$ growth cycles. Indeed, 'model calibration for a medium sized data set and minimal data layers requires about 1200 CPU hours on a typical workstation' (Clarke 2003a). This places SLEUTH calibration, especially for large data sets, at the edges of computational tractability.

Apparently, it is infeasible to apply a comprehensive calibration to a relatively large spatial data set with a high resolution over a long time period on a single-processor workstation. A few approaches have been developed to solve this problem. One approach is to make simplifying assumptions to ignore 'unimportant' parameter combinations. The current SLEUTH model uses this method to seek the best-fit combination, which assumes that the parameters affect the simulation results in a linear manner. However, because of the random processes involved in the CA simulation, the relationships between the parameters and the simulation results are very likely nonlinear, which makes the calibration results less reliable (Dietzel and Clarke 2007). Another approach is to deploy 'smart' algorithms to seek the best-fit parameter combination(s) without evaluating all the combinations, e.g., the genetic algorithm (Goldstein 2003) and artificial neural networks (Li and Yeh 2002, Guan *et al.* 2005). This study takes instead a computing-oriented direction, i.e., we deploy parallel computing technology to improve the performance of the CA model, hence making it possible to perform comprehensive calibrations for large spatial data sets over long-term periods. SLEUTH has been identified as being highly suitable for parallelization, although very little such effort has been conducted to date other than adding MPI routines to the calibration process to realize a simple task parallelism. The pSLEUTH project takes a step forward and aims to implement a data-task hybrid parallelism using pRPL.

3.3. Parallelizing SLEUTH using pRPL

pSLEUTH is a parallel version of SLEUTH, developed to improve the performance of the SLEUTH model, especially during the calibration process, by fully utilizing the advanced features of pRPL. More importantly, with the ability to process massive data sets within a shorter period of time, parallel computing is likely to allow the removal of the simplifying assumptions during the calibration processes. Thus, comprehensive or 'exhaustive' calibration processes may produce different best-fit parameter combination(s) other than the one(s) produced by simplified calibration processes, hence altering the final simulation results (Dietzel and Clarke 2007).

The four growth rules of the SLEUTH model were implemented using pRPL, such that the data layers used in the model were decomposed and distributed onto multiple processors. pSLEUTH provides two running modes: FORECAST and CALIBRATE. When running in the FORECAST mode, a user-defined parameter scenario (which can be the best-fit parameter combination resulting from a calibration process) is given to the program and the sub-cellspaces are processed using the given parameters in parallel on multiple processors. The final forecast result (i.e., the whole cellspace) is gathered to the master processor for an image output (Figure 8). In CALIBRATE mode, a calibration strategy is specified by the user to produce a set of parameter scenarios, and the sub-cellspaces are processed using the assigned parameters in parallel on a processor group, and the results are compared with the actual historical data, which are also decomposed and distributed onto the corresponding processors to evaluate the simulation performance, i.e., the similarity between

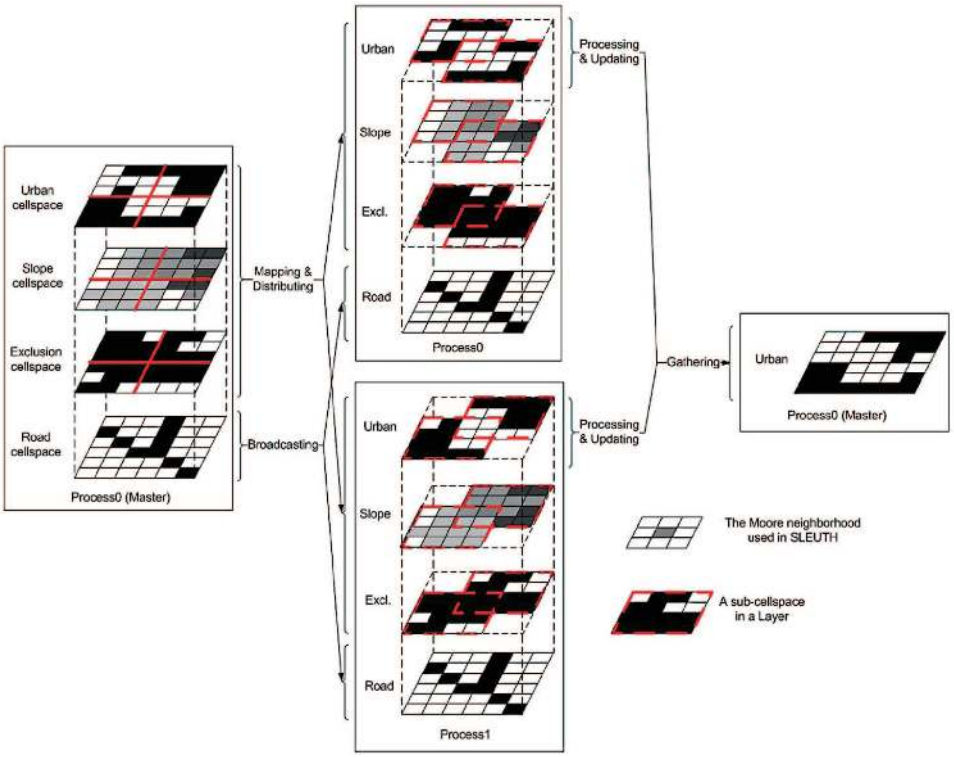

Figure 8. A block-wise data decomposition for pSLEUTH in FORECAST mode.

the simulated data and the historical data. In the end, the calibration results, i.e., simulation evaluations using multiple parameter scenarios, will be gathered to the master processor for an ASCII output.

As mentioned before, the SLEUTH model requires six data layers. Land-use data is used for land-use change simulation, and the hillshade data are used as the background of the output image and animations. In this study, pSLEUTH was only used to simulate the urban growth process, the UGM part of SLEUTH, so it requires only four layers to be decomposed and distributed: slope, exclusion, urban, and transportation.

Table 1 summarizes the centralization and scope characteristics of the growth rules. The customized *Transition* classes for implementing them should be constructed according to these characteristics. For example, the *Road-Influenced growth rule* involves a random walk along the road network, thus the whole road cellspace is required by this rule making it a global-scope transition. In consequence, unlike other data layers, the whole road cellspace is broadcast to all the processors without being decomposed.

Table 1. Characteristics of the growth rules of SLEUTH

Growth rule	Centralization type	Scope type
Spontaneous	Centralized	Neighborhood-scope
New spreading centers	Noncentralized	Neighborhood-scope
Edge	Noncentralized	Neighborhood-scope
Road-influenced	Noncentralized	Global-scope

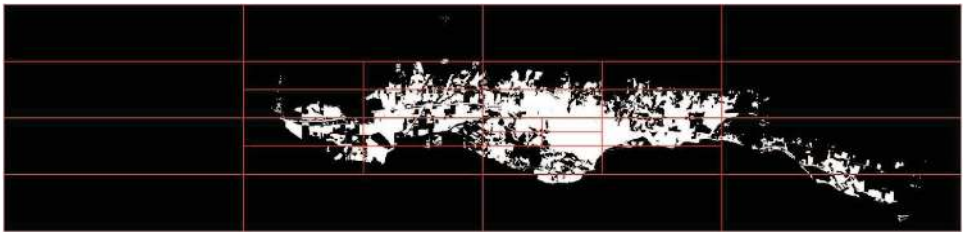

Figure 9. The QTB decomposition of Santa Barbara urban areas in 1976.

With pRPL, pSLEUTH provides users multiple options to decompose the cellspace, i.e., row-wise, column-wise, block-wise, and workload-based QTB decomposition. As mentioned in Section 3.1, the *Edge growth rule* accounts for most of the overall computational intensity, and urbanized cells introduce much more computational intensity than nonurbanized cells do. The current version of pSLEUTH calculates the workload of a sub-cellspace by counting the number of urbanized cells within the sub-cellspace (Figure 9). However, the spatial distribution of the model's workload over the cellspace is far more complex than the clustering of urbanized cells. Wang (2008) has proved that calculating the computational intensity for spatial domain decomposition is an *NP*-hard problem. Simply counting the number of urbanized cells within the sub-cellspaces can hardly reflect the real spatial pattern of the workload; thus the QTB decomposition may not yield better performance than other decomposition methods. This hypothesis was also proved by experiments with real-world data sets in the next section.

With pRPL, pSLEUTH is able to divide the processors into groups and assign each group of processors a portion of the global simulation set (i.e., parameter combinations) to execute, which is task parallelism. Within a group, the data layers are divided and distributed among the processors, which is data parallelism. In this way, the data-task hybrid parallelism is realized.

In addition, pSLEUTH provides options for two load-balancing methods for task parallelism among the processor groups: static tasking and dynamic tasking. When static tasking is used, the subsets of the simulations are assigned to the groups before the actual com-

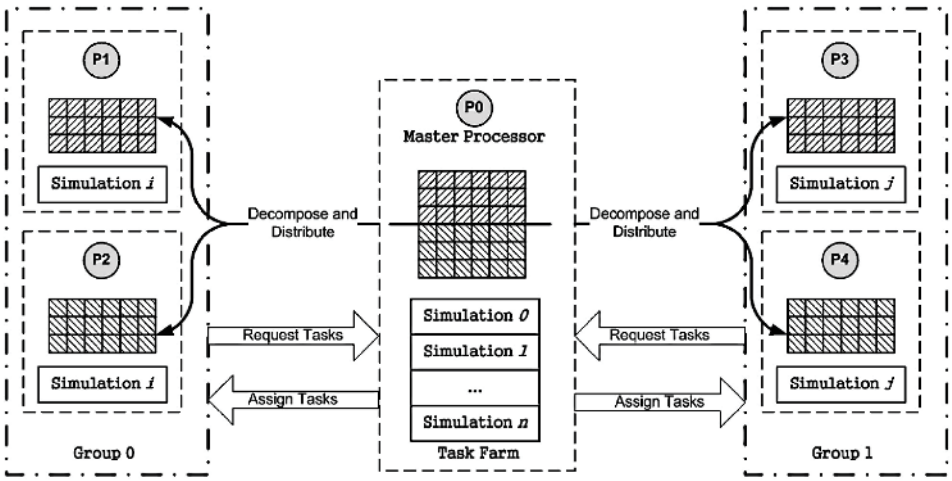

Figure 10. Data-task hybrid parallelization and dynamic tasking for pSLEUTH.

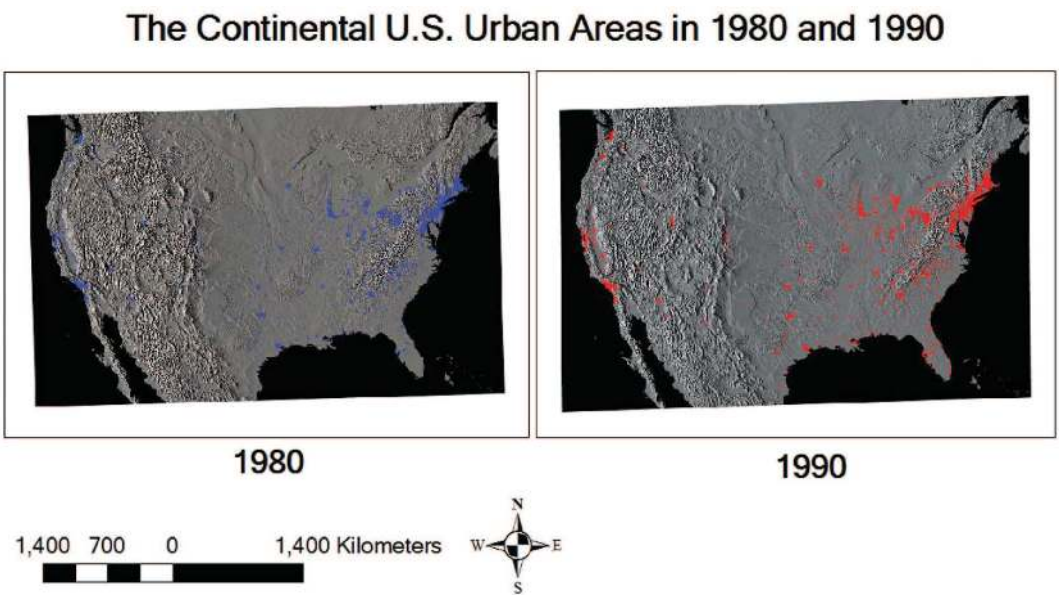

Figure 11. The continental US urban areas (1980 and 1990). Source USGS.

putation starts. When dynamic tasking (Figure 10) is used, a master-worker organization will form, which consists of a master processor (or the emitter processor) and a set of worker-processor groups. Each worker group will be initially assigned a subset of simulations, and a task farm that contains the remaining simulations will be created on the emitter processor. When a group finishes its initial set of simulations, it requests a new task (simulation) from the task farm. The emitter processor keeps sending tasks to the worker groups until the task farm is drained.

Because the computing speed and interconnection transfer rate may vary among the processors, dynamic tasking allows the groups with faster computing speed and transfer rate to do more tasks than other groups, hence it improves the efficiency and shortens the total computing time, especially when there are a large number of simulations to be executed.

4. Experiments and performance analysis

To demonstrate the performance of pSLEUTH, a data set of the continental US urban areas for 1980 and 1990 was used to conduct a series of experiments. Each input GIF image is 4948 × 3108 pixels in dimensions, at approximately a 1-km resolution (Figure 11).

To demonstrate the computational performance of pSLEUTH, a calibration scenario was specified as follows: using only two historical urban area data layers (1980 and 1990), only three values (0, 50, 100) will be evaluated for all the parameters, and only one Monte Carlo iteration will be performed. Thus the total number of simulations is 243 ($=3^5$), and each simulation includes 11 (=1990 - 1980 + 1) years. This calibration scenario is only for performance evaluation purposes and is not exactly how SLEUTH actually calibrates. In real applications, the calibration process of SLEUTH is much more complex and time consuming.

Although other simulation performance evaluation criteria have been used in SLEUTH calibration, only one of the thirteen measures, the Lee-Sallee index (Clarke and Gaydos 1998) was implemented in pSLEUTH. It can be calculated as

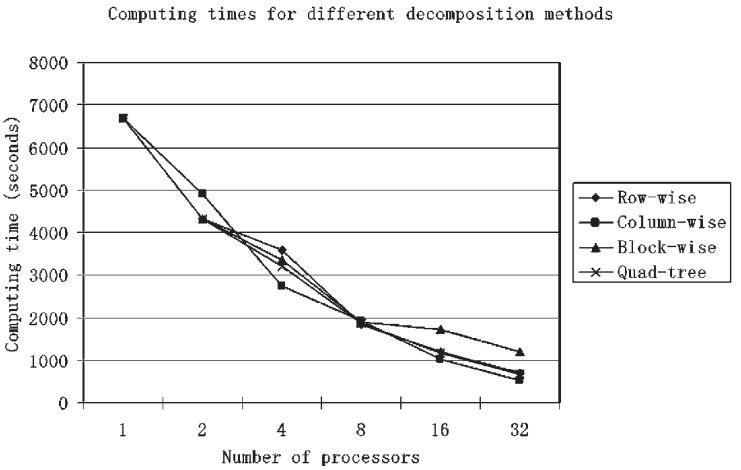

Figure 12. Computing times using different decomposition methods (each processor holds only one sub-cellspace).

$$L = \frac{A_0 \cap A_1}{A_0 \cup A_1} \tag{1}$$

where A_0 denotes the simulation map and A_1 denotes the real map. This index is 1.0 if the simulation map matches the real map perfectly, pixel by pixel. Thus the higher the Lee-Sallee index, the better fit the parameter combination. The simplified goal of calibration was to find the best-fitting parameter combination, i.e., the highest Lee-Sallee index. Note that besides the Lee-Sallee index, the SLEUTH model also provides another twelve indices of model fit, and users can choose which index or index combination to optimize during the calibration process (Dietzel and Clarke 2007). However, this is outside the scope of this article. It has to be stressed that the pSLEUTH presented in this article was developed mainly for the demonstration of the usability and computational performance of pRPL, not for the improvement of simulation performance of SLEUTH. The full parallel implementation of the SLEUTH model will be completed in the future.

The experiments were conducted on a Dell cluster composed of Dell 1750 dual CPU 3.06 GHz Xeon servers and a single Dell 1750 monitoring node. The head node had 4-GB RAM, two mirrored system disks, and a 2TB RAID array that is shared to the cluster. The 128 compute nodes had 2-GB RAM each.

By dividing the cellspace among the number of processors, i.e., each processor only holds one sub-cellspace and only data parallelism is used, the computing time dropped from 6694 s when one processor was used to as low as 537.963 s when 32 processors were used (Figure 12). As Figure 13 shows, column-wise decomposition yielded the highest speed-up (12.44) and efficiency (0.39), and the quad-tree decomposition yielded the second highest (speed-up 10.14, and efficiency 0.32).

Why did the quad-tree method not yield the best performance? The reasons may be fourfold:

1. The QTB requires additional processes to compute the workloads associated with the sub-cellspaces and to construct the quad-tree, which degrades the performance.
2. The computational intensity associated with a sub-cellspace is rather complex to determine for the random processes embedded in the growth rules. Also, the workloads of

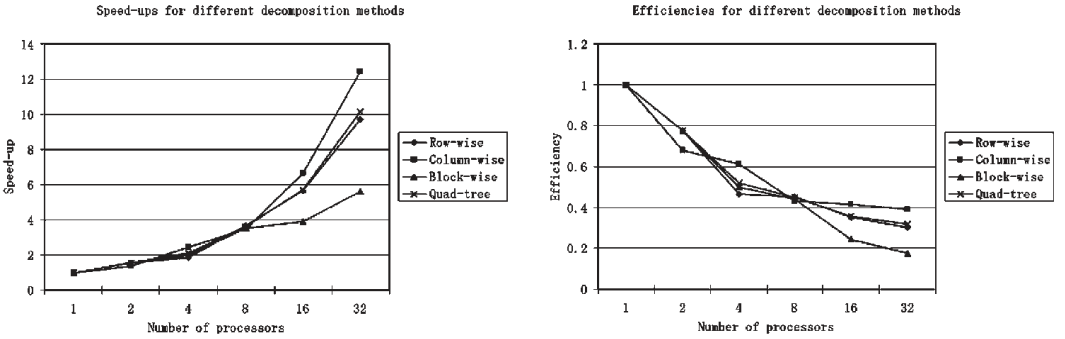

Figure 13. Performances with different decomposition methods (each processor holds only one sub-cellspace).

the sub-cellspaces may change as the cell values change during the iterative process. The workload-calculating algorithm used in pSLEUTH is, however, relatively simple and fully depends on the number of urbanized cells within the initial sub-cellspace, which is likely inadequate to capture all the aspects of the workload as growth unfolds.

3. The quad-tree decomposition works best when the computational intensity is highly clustered over space (Wang and Armstrong 2003), which is not the case in the data set used in this experiment. The urbanized cells are rather more scattered over the space than clustered, because growth takes place at the periphery of the urban areas.
4. The quad-tree decomposition is essentially a special case of the block-wise decomposition. This implies that most sub-cellspaces, i.e., the sub-cellspaces not located at the edge of the global cellspace, have eight or even more neighboring sub-cellspaces to communicate with, which creates much more communication overhead than other decomposition methods (i.e., row-wise and column-wise). By examining the performances of all the decompositions under all circumstances, we found that the block-wise decomposition was likely to yield the worst performance especially when more than eight processors were engaged.

pRPL allows the layer on a processor to hold multiple sub-cellspaces, i.e., scattered decomposition. Theoretically, reducing the granularity of the decomposition will increase the chance of producing better evenly distributed workloads for the processors. By doing so, the performance of pSLEUTH was significantly improved. The speed-up reached 20.7 when the row-wise decomposition (best among all the decomposition methods in this scenario) with 32 processors was used, and each processor held eight sub-cellspaces (Figure 14). Again, only data parallelism was used in this experiment.

Moreover, grouping the processors (i.e., using data-task hybrid parallelization) and dynamically assigning tasks to the groups also improved the performance. The speed-up reached 24 when 32 processors were divided into eight groups with dynamic tasking, the column-wise decomposition was used, and each processor held eight sub-cellspaces (Figure 15). Note that when dynamic tasking is being used, one of the processors acts as the emitter processor and does not participate in the computation. In Figure 15, for dynamic tasking experiments, the number of processors does not include the emitter processor.

With greatly improved computational performance, pSLEUTH actually provides a means to perform a comprehensive calibration to examine all the possible parameter combinations for a SLEUTH application, which is very likely to alter the results found using

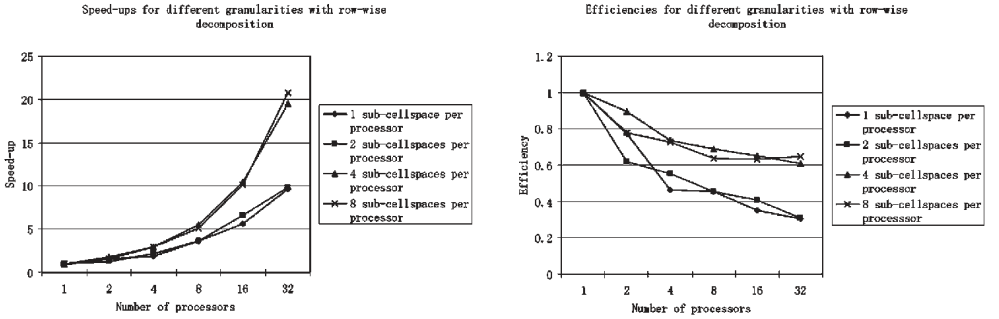

Figure 14. Performances for different granularities with row-wise decomposition.

the linear-effect assumptions, and hence alter the final findings that are very significant for practical applications such as planning and for scientific research. Parallelization of this spatially dynamic model with pRPL has provided a test-bed for proving other spatial models, and perhaps solving new geographical problems (Clarke 2003a).

5. Conclusions

In this study, an open-source general-purpose pRPL was developed such that nonspecialist GIScientists can easily parallelize their own raster-processing algorithms with any arbitrary neighborhood configuration. As a generic parallel raster-processing library, pRPL supports both centralized and noncentralized algorithms, therefore it supports not only local-scope and neighborhood-scope processes, but also some regional-scope and global-scope ones. pRPL supports multilayer algorithms that are commonly used in geospatial applications. pRPL provides multiple data-decomposition methods for users, including a spatially adaptive QTB decomposition method for cases when the computational intensity is extremely heterogeneous over space. The 'Update-on-Change' and VGS techniques developed for pRPL helped to reduce the communication overhead for data exchange among the processors, hence reduce the computing time. Furthermore, the 'edgesFirst' and nonblocking communication techniques overlap the computation and communication, which also helps reduce the computing time. pRPL organizes processors into groups and sup-

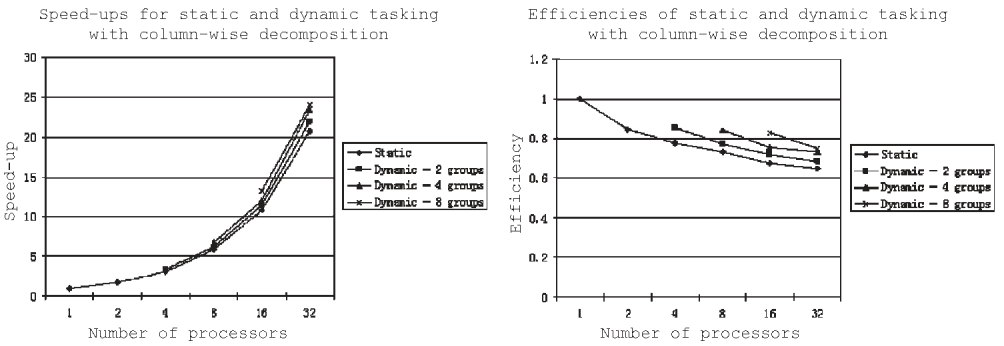

Figure 15. Performances for static and dynamic tasking with column-wise decomposition (each processor holds eight sub-cellspaces).

ports data-task hybrid parallelism, which is innovative for parallel raster processing and especially useful when handling massive-volume data sets and a large number of parallelizable tasks at the same time. With grouped processors, dynamic load-balancing can be implemented with ease. pRPL also provides an intuitive programming guideline for users to implement application-specific algorithms and requires minimal parallel programming knowledge. pRPL provides GIScientists with an easy-to-use toolkit to exploit the great computing power of HPC facilities, i.e., providing transparent parallelism, and more importantly, a test-bed for computationally intensive geospatial models and a problem-solving environment for previously computationally infeasible approaches.

A parallel geographic CA model, pSLEUTH, was developed using pRPL in this study to demonstrate the usability and computational performance of pRPL. pSLEUTH fully utilizes the features of pRPL, parallelizing the SLEUTH model not only using data parallelism but also using data-task hybrid parallelism. pSLEUTH provides options for both static and dynamic tasking as the load-balancing strategy. Experiments with real-world data sets showed that with multiple processors, pSLEUTH greatly reduced the computing time needed for the calibration process, and yields fairly high performances, achieving a speed-up of 24 using 32 processors.

The experiments also revealed that the workload-calculating algorithm used in the QTB decomposition is critical to the performance. The current version of pSLEUTH only accounts for the number of the urbanized cells in the cellspace, which may not be adequate to capture all the aspects of the computational intensity and its dynamics over time.

Furthermore, the experiments showed that reducing the decomposition granularity increased the chance of producing better evenly distributed workloads for the processors, and greatly improved the performance. Data-task hybrid parallelism by grouping processors yielded better performance than did pure data parallelism. Also, dynamically assigning tasks to the processor groups according to their status (busy or idle) yielded better performances than static tasking did.

However, the current pRPL has some limitations. The most critical shortcoming is that it does not support dynamic data load-balancing. The whole data set has to be read, decomposed, and distributed to processors at one time before the computation starts. The next version is under development and will allow sub-cellspaces to be migrated among processors. The master processor is able to read/write and distribute/gather sub-cellspaces dynamically according to the progress of the computation, which will greatly reduce the memory requirement on the master processor and the workload imbalance on the work processors, hence increasing computational performance.

Notes

1. Nonparallelizable algorithms exist in raster processing. One example is deriving flow accumulations from flow directions. The computation of the flow accumulation for a given cell depends on the flow accumulations of its upslope cells that are computed before the current cell.
2. pRPL 1.0 only supports static load-balancing for data parallelism, but pRPL 2.0 supports both static and dynamic load-balancing for data parallelism. This article was written when pRPL 2.0 was under development.
3. For more details about using pRPL, refer to the user manual - Getting started with pRPL (Guan 2008).
4. For more details about SLEUTH, see the website of Project Gigalopolis: http://www.ncgia.ucsb.edu/projects/gig/

References

Aloisio, G. and Cafaro, M. (2003) A dynamic earth observation system. *Parallel Computing* **29**:10, pp. 1357-1362.

Ananthanarayan, A. et al. (2003) Data webs for earth science data. *Parallel Computing* **29**:10, pp. 1363-1379.

Armstrong, M. P. (2000) Geography and computational science. *Annals of the Association of American Geographers* **90**:1, pp. 146-168.

Armstrong, M. P. and Marciano, R. (1993) Parallel spatial interpolation. *Proceedings of the eleventh international symposium on computer-assisted cartography (Auto-Carto 11)* pp. 414-423. American Congress on Surveying and Mapping, Bethesda, MD

Armstrong, M. P. and Marciano, R. (1995) Massively parallel processing of spatial statistics. *International Journal of Geographical Information Science* **9**:2, pp. 169-189.

Armstrong, M. P. and Marciano, R. J. (1996) Local interpolation using a distributed parallel supercomputer. *International Journal of Geographical Information Systems* **10**:6, pp. 713-729.

Armstrong, M. P. and Marciano, R. J. (1997) Massively parallel strategies for local spatial interpolation. *Computers and Geosciences* **23**:8, pp. 859-867.

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**, pp. 539-553.

Batty, M. Openshaw, S. and Abrahart, R. J. (eds) (2000) *GeoComputation using cellular automata,* pp. 95-126. Taylor & Francis, New York, NY

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**:12, pp. 3119-3127.

Bhattacharya, P. (2001) *Efficient neighbor finding algorithms in quadtree and octree* M.T. Thesis. Dept. Comp. Science and Eng., India Inst. Technology, Kanpur

Cannan, E., 2001. *The origin of the law of diminishing returns, 1813-1815*. http://socserv.mcmaster.ca/~econ/ugcm/3ll3/cannan/cannan003.html

Cannataro, M. et al. (1995) A parallel cellular automata environment on multicomputers for computational science. *Parallel Computing* **21**:5, pp. 803-823.

Clarke, K. C. (1995) *Analytical and computer cartography* Prentice Hall, Englewood Cliffs, NJ

Clarke, K. C. (2003a) Geocomputation's future at the extremes: high performance computing and nanoclients. *Parallel Computing* **29**:10, pp. 1281-1295.

Clarke, K. C. (2003b) *Getting started with geographic information systems* Pearson Education, Upper Saddle River, NJ

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**:7, pp. 699-714.

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**:2, pp. 247-261.

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**:11, pp. 1355-1367.

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

Clematis, A., Mineter, M. and Marciano, R. (2003) High performance computing with geographical data. *Parallel Computing* **29**:10, pp. 1275-1279.

Cosnard, M. and Trystram, D. (1995) *Parallel algorithms and architecture* International Thomson Computer Press, Boston, MA

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.

Cramer, B. E. and Armstrong, M. P. (1997) Interpolation of spatially inhomogeneous data sets: an

evaluation of parallel computation approaches. *Proceedings of GIS/LIS'97* American Congress on Surveying and Mapping, Bethesda, MD

Dietzel, C. and Clarke, K. C. (2007) Toward optimal calibration of the SLEUTH land use change model. *Transactions in GIS* **11**:1, pp. 29-45.

Ding, Y. and Densham, P. J. (1996) Spatial strategies for parallel spatial modelling. *International Journal of Geographical Information Systems* **10**:6, pp. 669-698.

Duckham, M., Goodchild, M. F. and Worboys, M. F. (2003) *Foundations of geographic information science* Taylor & Francis, New York

Goldstein, N. C., Dietzel, C. and Clarke, K. C. (2005) Don't stop 'til you get enough-sensitivity testing of Monte Carlo iterations for model calibration. Proceedings of the 8th International Conference on GeoComputation Ann Arbor, MI

Goldstein, N. C. (2003) Brains vs Braun - comparative strategies for the calibration of a cellular automata-based urban growth model. Proceedings of the 7th international conference on GeoComputation Southampton, England

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

Guan, Q., 2008. Getting started with pRPL. http://www.geog.ucsb.edu/~guan/pRPL/Getting_started_with_pRPL.pdf

Guan, Q., Wang, L. and Clarke, K. (2005) An artificial-neural-network-based, constrained CA model for simulating urban growth. *Cartography and Geographic Information Science* **32**:4, pp. 369-380.

Harris, B. (1985) Some notes on parallel computing: with special reference to transportation and land-use modeling. *Environment and Planning A* **17**:9, pp. 1275-1278.

Healey, S. (ed) (1998) *Parallel processing algorithms for GIS* Taylor & Francis, Bristol, PA

Hecker, C. et al. (1999) System development for parallel cellular automata and its applications. *Future Generation Computing Systems* **16**:2-3, pp. 235-247.

Hutchinson, D. et al. (1996) Parallel Neighboourhood Modelling. *Proceedings of the 4th ACM international workshop on advances in geographic information systems* pp. 25-34. Rockville, MD

Kerry, K. E. and Hawick, K. A. (1997) Spatial interpolation on distributed, high-performance computers. *Technical Report DHPC-015* Department of Computer Science, University of Adelaide

Kerry, K. E. and Hawick, K. A. (1998) Kriging interpolation on high-performance computers. *Proceedings of the international conference and exhibition on high-performance computing and networking* pp. 429-438. Amsterdam: Springer-Verlag

Kidner, D. B., Rallings, P. J. and Ware, J. A. (1997) Parallel processing for terrain analysis in GIS: visibility as a case study. *GeoInformatica* **1**:2, pp. 183-207.

Li, B. (1992) Opportunities and challenges of parallel spatial data analysis: initial experiments with data parallel map analysis. GIS LIS-international conference San Jose pp. 445-458.

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**:2, pp. 131-152.

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**:4, pp. 323-343.

Lloyd, C. D. (2007) *Local models for spatial analysis* CRC Press, Boca Raton, FL, USA

Mineter, M. J. Healey, R. D. (ed) (1998) Partitioning raster data. *Parallel processing algorithms for GIS* pp. 215-230. Taylor & Francis, Bristol, PA

Mineter, M. J. and Dowers, S. (1999) Parallel processing for geographical applications: a layered approach. *Journal of Geographical Systems* **1**:1, pp. 61-74.

Montanvert, A., Meer, P. and Rosenfeld, A. (1991) Hierarchical image analysis using irregular tessellations. *IEEE transactions on pattern analysis and machine intelligence* **13**:4, pp. 307-316.

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

Nieplocha, J. *et al.*, 2007. *Global arrays user manual*. http://www.emsl.pnl.gov/docs/global/um/GA.pdf

Openshaw, S. and Turton, I. (2000) *High performance computing and the art of parallel programming: an introduction for geographers, social scientists, and engineers* Routledge, New York, NY

O'Sullivan, D. (2001) Graph-cellular automata: a generalised discrete urban and regional model. *Environment and Planning B: Planning and Design* **28**:5, pp. 687-705.

Peucker, T. K. and Douglas, D. H. (1975) Detection of surface-specific points by local parallel processing of discrete terrain elevation data. *Computer Graphics and Image Processing* **4**, pp. 375-387.

Puppo, E. et al. (1994) Parallel terrain triangulation. *International Journal of Geographical Information Science* **8**:2, pp. 105-128.

Rokos, D. and Armstrong, M. P. (1992) Parallel terrain feature extraction. *Proceedings of GIS/LIS '92* pp. 652-661. American Congress on Surveying and Mapping, Bethesda, MD

Saalfeld, A. (2000) Complexity and intractability: limitations to implementation in analytical cartography. *Cartography and GIS* **27**:3, pp. 239-249.

Samet, H. (1982) Neighbor finding techniques for images represented by quadtrees. *Computer Graphics and Image Processing* **18**:1, pp. 37-57.

Samet, H. (1984) The quadtree and related hierarchical data structures. *ACM Computing Surveys* **16**:2, pp. 187-260.

Samet, H. (1990) *Applications of spatial data structures: Computer graphics, image processing, and GIS* Addison-Wesley Longman Publishing Co., Inc, Reading, MA

Sandu, J. S. and Marble, D. F. (1988) An investigation into the utility of the cray X-MP supercomputer for handling spatial data. *Proceedings of the third international symposium on spatial data handling* pp. 253-266. Sydney, Australia

Silva, E. A. and Clarke, K. C. (2002) Calibration of the SLEUTH urban growth model for Lisbon and Porto. *Computers, Environment and Urban Systems* **26**:6, pp. 525-552.

Smith, T. R., Gao, P. and Gahinet, P. (1989) Asynchronous, iterative, and parallel procedures for solving the weighted-region least cost path problem. *Geographical Analysis* **21**:2, pp. 147-166.

Spezzano, G. and Talia, D. (1999) Programming cellular automata algorithms on parallel computers. *Future Generation Computing Systems* **16**:2, pp. 203-216.

Spezzano, G. et al. (1996) A parallel cellular tool for interactive modeling and simulation. *IEEE Computational Science and Engineering* **3**:3, pp. 33-43.

Tomlin, C. D. (1990) *Geographic information systems and cartographic modelling* Prentice-Hall, Englewood Cliffs, NJ

Wang, F. (1993) A parallel intersection algorithm for vector polygon overlay. *IEEE Computer Graphics and Applications* **13**:2, pp. 74-81.

Wang, S. (September 2008) Formalizing computational intensity of spatial analysis. Proceedings of the 5th international conference on geographic information science Park City, UT pp. 23-26.

Wang, S. and Armstrong, M. P. (2003) A quadtree approach to domain decomposition for spatial interpolation in grid computing environments. *Parallel Computing* **29**:10, pp. 1481-1504.

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**:3, pp. 323-343.

Worboys, M. and Duckham, M. (2004) *GIS: a computing perspective* CRC Press, New York

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* **25**:1, pp. 103-126.

Yeh, A. G. O. and Li, X. (2001) A constrained CA model for the simulation and planning of sustainable urban forms by using GIS. *Environment and Planning B: Planning and Design* **28**:5, pp. 733-753.

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