Getting started with pSLEUTH

Qingfeng (Gene) Guan

guanqf {at} gmail.com

Department of Geography

University of California, Santa Barbara

Santa Barbara, CA 93106, USA

April, 2008

CONTENTS

1. Introduction to pSLEUTH

1.1 What is SLEUTH?

SLEUTH[1] is a Cellular Automata (CA) model of urban growth and land use change simulation and forecasting, developed in the Department of Geography, University of California, Santa Barbara (Clarke, Hoppen, and Gaydos 1997; Clarke and Gaydos 1998; Silva and Clarke 2002).

A classical Cellular Automata model is a set of identical elements, called cells, each one of which is located in a regular, discrete space, called cellspace. Each cell is associated with a state from a finite set. The model evolves in discrete time steps, changing the states of all its cells according to a transition rule, 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 cell itself, and constitutes its neighborhood.

The urban growth model SLEUTH, uses a modified CA to simulate the spread of urbanization across a landscape. Its name comes from the GIS 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 complex transition rules, the multiple parameters involved in the rules, and the potential vast volume of the datasets, have made SLEUTH a massive computing system.

The basic unit of a SLEUTH simulation is a growth cycle, which usually represents a year of urban growth (Figure 1.1). A simulation (or a Run) consists of a serious of growth cycles that begins at a start year and completes at a stop year (Figure 1.2).

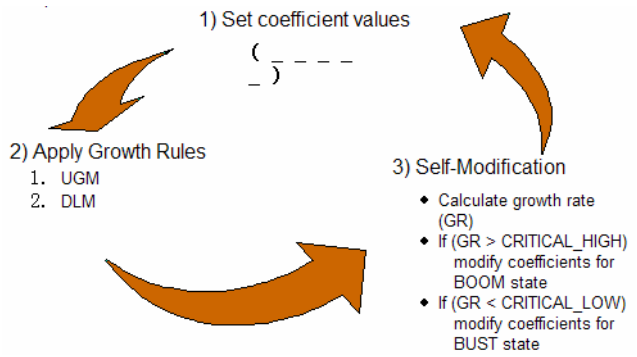


Figure 1.1 A growth cycle of SLEUTH

(http://www.ncgia.ucsb.edu/projects/gig/v2/About/bkStrCycle.htm)

[1] For more details about SLEUTH, see the website of Project Gigalopolis : http://www.ncgia.ucsb.edu/projects/gig/

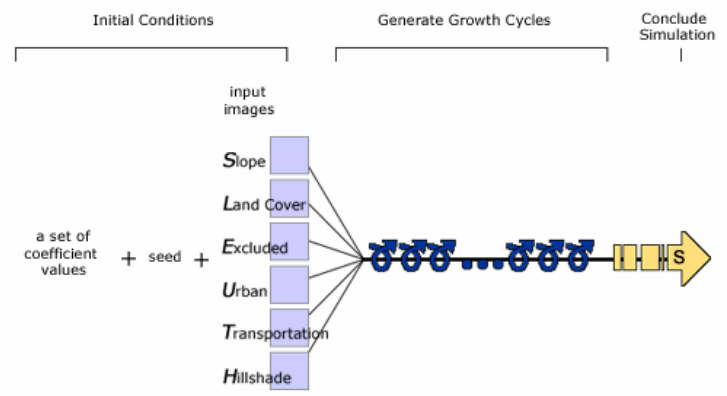
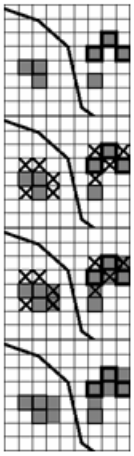


Figure 1.2 A simulation (or a Run) of SLEUTH

(http://www.ncgia.ucsb.edu/projects/gig/v2/About/bkStrSimulation.htm)

There are five parameters (or coefficients) involved in SLEUTH: *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*, *Road-Influenced Growth Rule*. Figure 1.3 shows the process of the *Edge growth*, which is merely a relatively simple one among the four growth rules.



```
Edge Growth:
F(spread_coefficient, slope_coefficient)
{
   for (all non-edge pixels (i,j))
      if ((i,j) is urban) and (random_integer
            < spread_coefficient)
         if (at least two urban neighbors exist)
            if (a randomly chosen, non-urban
                  neighbor is available for urbanization)
               (i,j) neighbor = urban
} end edge growth
```

Figure 1.3 Edge growth example and pseudo code
(http://www.ncgia.ucsb.edu/projects/gig/v2/About/gwEdge.htm)

Calibration processes are 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 testing results produced using a set of parameter combinations with the goal dataset(s) that usually are real historical geospatial data in order to determine the best-fit parameter combination(s).

In addition, to simulate the random processes that happen during urban growth, the Monte Carlo method is used in a simulation run. For a single parameter (or coefficient)

combination, a simulation has to execute multiple times with different Monte Carlo seed values. In practice, *10~100* Monte Carlo iterations for each parameter combination are suggested.

All these above together, make the calibration process a tremendously computing-intensive process (Figure 1.4). In a comprehensive calibration, every possible combination of the five parameter values has to be evaluated with multiple Monte Carlo seeds. If *100* Monte Carlo iterations were applied, a comprehensive calibration over a *20*-year period would consist of $101^5 \times 100 \times 20$ growth cycles. "*The model calibration for a medium sized data set and minimal data layers requires about 1200 CPU hours on a typical workstation*" (Clarke 2003).

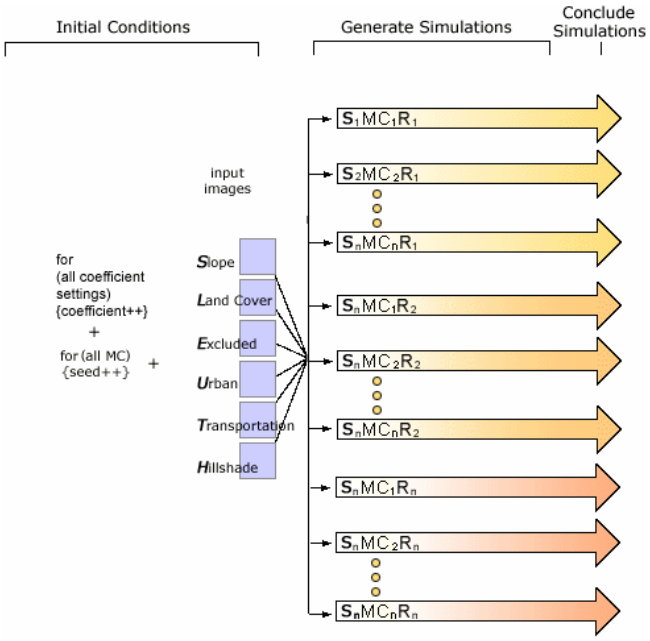


Figure 1.4 A comprehensive calibration process of SLEUTH (http://www.ncgia.ucsb.edu/projects/gig/v2/About/bkStrPrediction.htm)

Apparently, it is infeasible to apply a comprehensive (full) calibration onto a relatively large spatial dataset with a high resolution over a long time period on a single-processor workstation. A few approaches have been developed to solve this infeasibility. One approach is to make some simplifying assumptions to ignore the "unimportant" parameter combinations. The current SLEUTH model uses Brute Force method to seek the best-fit combination, which assumes that the parameters affect the simulation results in a linear manner. However, due to the random processes involved in the CA simulation, the relationship between the parameters and the simulation results is very likely non-linear. Another approach is to deploy "smart" algorithms to seek the best-fit parameter combination(s) without evaluating all the combinations, e.g., Genetic algorithm (Goldstein 2003) and Artificial Neural Networks (Li and Yeh 2001; Guan, Wang, and Clarke 2005). This pSLEUTH project goes to a computing-oriented direction, i.e., deploying parallel computing technologies to improve the performance of CA models, hence making it possible to apply

comprehensive calibrations for large spatial datasets over long-term periods.

1.2 What is pSLEUTH?

pSLEUTH is a parallel version of SLEUTH, developed by Qingfeng Guan in the Department of Geography, University of California, Santa Barbara, to improve the performance of the SLEUTH model, especially the calibration processes, by deploying parallel computing technologies. More importantly, parallel computing is likely to allow the removal of the simplifying assumptions during the calibration processes. Thus, the comprehensive calibration processes might produce different best-fit parameter combination(s) other than the one(s) produced by simplified calibration processes, hence alter the final simulation results.

pSLEUTH is developed using C++ programming language and the open-source general-purpose **p**arallel **R**aster **P**rocessing programming **L**ibrary - pRPL[2]. pRPL encapsulates complex parallel computing utilities and routines specifically for raster processing, and enables the implementation of parallel raster-processing algorithms without requiring a deep understanding of parallel computing and programming, thus it greatly reduces the development complexity. Since the cellspace in a CA model is essentially a grid of pixels, or a raster, a CA model can be easily parallelized using pRPL.

pSLEUTH is also an open-source[3] software and can be freely downloaded and used.

1.3 Parallelization of SLEUTH

The basic idea of parallelizing a CA model is very simple. The transition rules, e.g., the growth rules in the SLEUTH model, are applied to every cell in the cell space independently. In other words, the transition rules applied on a specific cell do not affect the transition rules applied in other cells or are affected by the transition rules applied on other cells. Thus, the transition operations can be easily parallelized. From a parallel algorithm design perspective, the whole cellspace is easy to decompose into sub-cellspaces and assign onto multiple computing units, e.g., processors. Then the transition rules can be applied on these sub cell spaces simultaneously.

pRPL (Guan 2008) provides a simple and powerful interface for users to parallelize almost any raster processing algorithm, with any arbitrary neighborhood configuration. It supports algorithms requiring multiple raster layers which are very common in GI Systems and GeoComputation, and provides multiple options to partition raster datasets, including simple row/column-wise decomposition, simple block-wise decomposition, and workload-based quad-tree decomposition. In addition, pRPL

[2] For more details about pRPL, see “Getting started with pRPL“ (Guan 2008)

[3] pSLEUTH can be freely used for EDUCATIONAL and SCIENTIFIC purposes, and NO COMMERCIAL usages are allowed unless the author is contacted and a permission is granted.

supports static load-balancing and dynamic load-balancing for parallel processing.

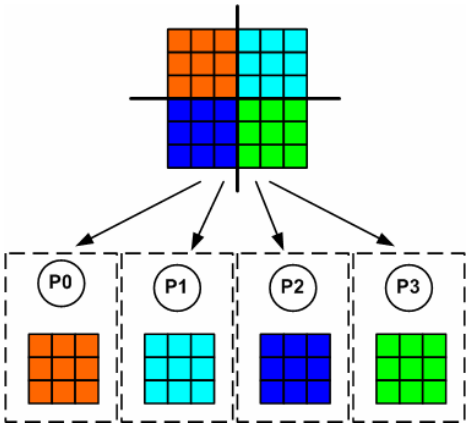


Figure 1.5 Decomposing a cellspace into 4 sub-cellspaces and assigning them onto 3 processors (data parallelization)

pSLEUTH fully utilizes the features of pRPL, and parallelizes the SLEUTH model not only in a data-parallelization manner, but also in a data-task hybrid parallelization manner.

1.3.1 Data parallelization

As mentioned above, data parallelization of a CA model is to divide the cellspace into multiple sub-cellspaces and assign them onto multiple processors so that the processors can apply the transition process on the sub-cellspaces simultaneously (Figure 1.5). The CA model is a theoretically parallel computing model (see the definition of CA). It was born to be parallelized!

In pSLEUTH, the four growth rules in the SLEUTH model, i.e., *Spontaneous Growth Rule*, *New Spreading Centers Rule*, *Edge Growth Rule*, *Road-Influenced Growth Rule*, were implemented using pRPL, so that the datasets used in the model will be decomposed and distributed onto multiple processors.

With pRPL, pSLEUTH provides users multiple options to decompose the cellspace (Figure 1.6), i.e., row-wise, column-wise, block-wise, and workload-based quad-tree decomposition. The quad-tree-based decomposition is likely to produce more evenly distributed workloads for the processors than other simple decomposition methods, but needs some extra computing time to construct the quad-tree which may overweight the speed-up gained by the better distributed workload. Thus caution should be taken when the quad-tree decomposition is to be used. The quad-tree-based decomposition performs the best when the cellspace is extremely heterogeneous, i.e., the urbanized cells are highly clustered in the cellspace (Figure 1.7).

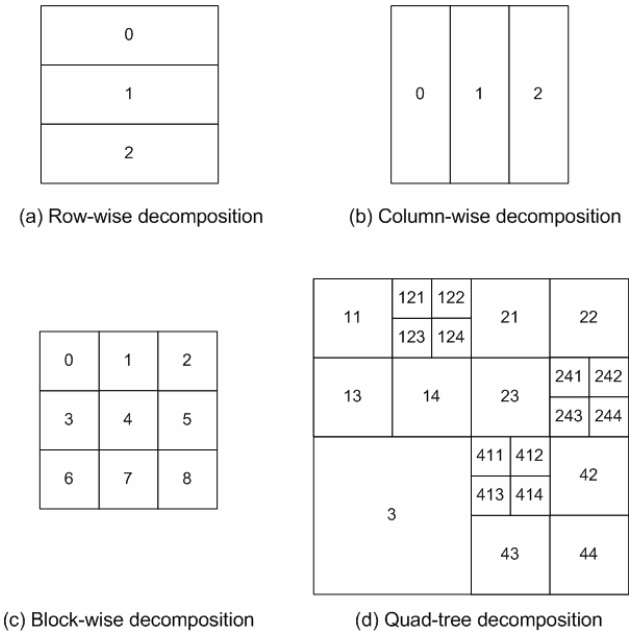


Figure 1.6 Different ways to decompose the cellspace

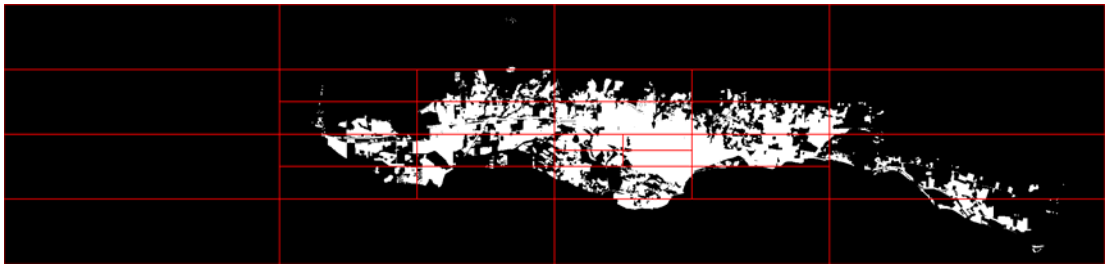


Figure 1.7 An example of quad-tree decomposition

(Santa Barbara urban areas, 1976.

The urbanized cells (in white) will introduce the most workload)

1.3.2 Task parallelization

Task parallelization is to split the problem-solving process (task) into sub-tasks. Task decomposition is often used in two situations. One is when examining multiple scenarios with an identical process, each scenario setting can be processed independently. This method has been used in the current SLEUTH software[4]. Subsets of simulations (or Runs) are assigned to multiple processors, and they execute the simulations on the whole cellspace simultaneously (Figure 1.8). Another is when a sequence of processes work on one dataset, each process can be assigned to a processor and the intermediate data is passed through the processors in a certain order, which is called pipelining. Pipeline parallelization is mainly used on vector

[4] NOTE: this SLEUTH software is NOT the pSLEUTH software, and it can be downloaded at http://www.ncgia.ucsb.edu/projects/gig/

supercomputers.

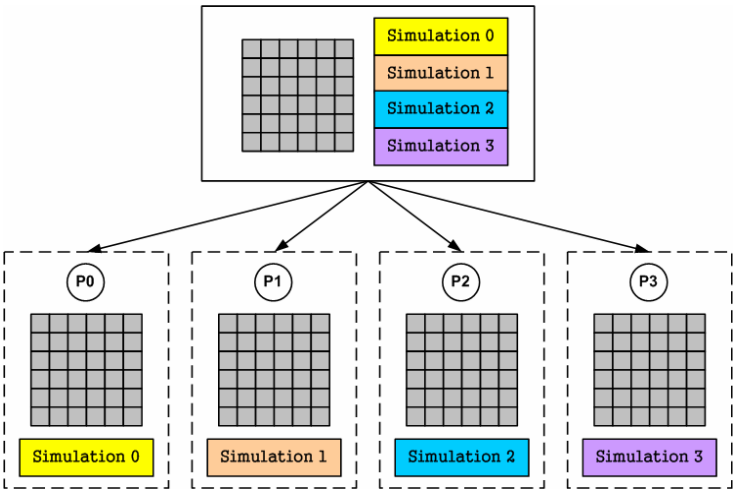


Figure 1.8 An example of task-parallelization

1.3.3 Hybrid parallelization

Hybrid parallelization is to split the dataset into sub-datasets, and split the task into sub-tasks as well. In some cases, neither data parallelization nor task parallelization is sufficient to fully utilize parallel computing resources. Using combinations of them will largely improve the performance. For instance, 10 simulations are to be executed using a 100-processor parallel computer, if only task decomposition is used, 90 processors will remain idle and the theoretical maximal speed-up is merely 10. Adding data parallelization to the algorithm to employ the other 90 processors will increase the speed-up to be much more than 10.

With pRPL, pSLEUTH is able to organize the processors in groups, and assign each group of processors a subset of the global (whole) simulation set to be executed. Within a group, the datasets are divided and distributed among the processors. In this way, the data-task hybrid parallelization is realized (Figure 1.9).

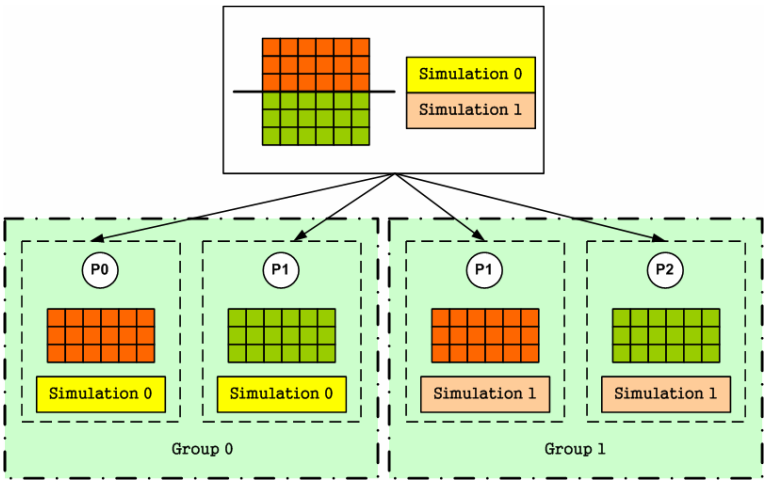


Figure 1.9 An example of hybrid parallelization

1.3.4 Static and dynamic load-balancing

Load-balancing ensures computing units (i.e., processors) get appropriate amounts of workload and work together in an efficient way. Load-balancing techniques can be broadly classified into two categories: static and dynamic (Grama et al. 2003).

Static load-balancing decompose and distribute data subsets or/and sub-tasks among the processors prior to the execution of the process. After the decomposition and mapping processes, processors execute their own computation till the end.

Dynamic load-balancing decompose and distribute data subsets or/and sub-tasks among the processors during the execution of the process. Data subsets or/and sub-tasks are dynamically generated and distributed according to the status of the parallel processors (idle or busy).

pSLEUTH provides options for both load-balancing methods for the task parallelization among the processor groups: static tasking and dynamic tasking. Note that the data parallelization within a processor group is still static, because moving the data of sub-cellspaces among processors may cause massive communication overheads.

When static tasking is used, the subsets of the global simulation set are assigned to the groups before the computation starts (Figure 1.9). When dynamic tasking is used, each group will be initially assigned with a set of a certain number of simulations, and a task farm that contains the remaining simulations will be created on the master processor (or the emitter processor). When a group finishes its initial set of simulations, it requests for a new task, i.e., a simulation, from the task farm (Figure 1.10). The emitter processor keeps sending tasks to the groups until the task farm is drained.

When there are a large number of simulations to execute, and the computing speed and the interconnect transfer rate vary among the processors, dynamic tasking allows the groups with faster computing speed and transfer rate to do more tasks than other groups, hence improves the efficiency and shortens the total computing time.

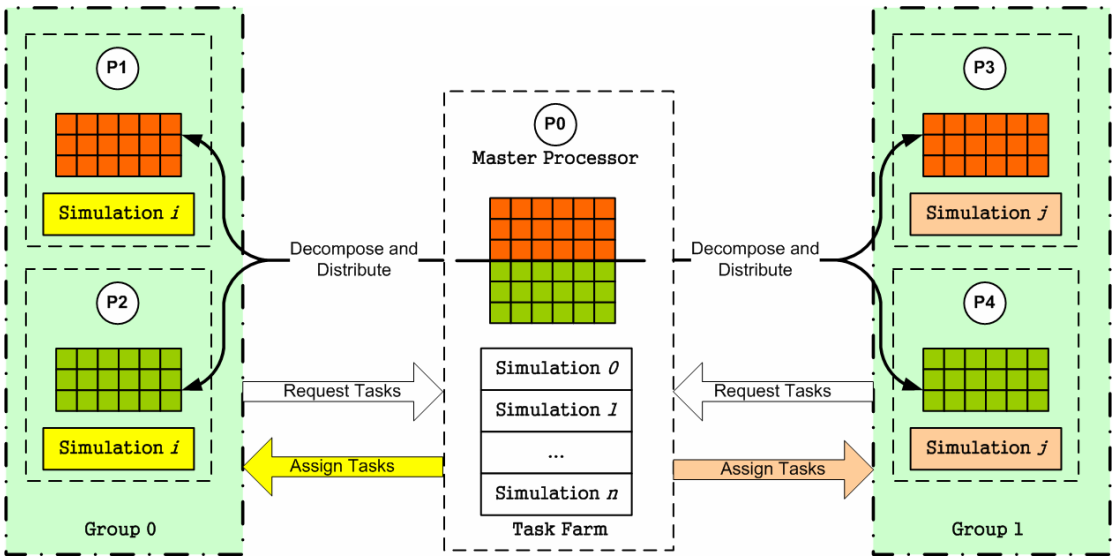


Figure 1.10 An example of dynamic tasking

2. Download and Compile

pSLEUTH can be downloaded at http://www.geog.ucsb.edu/~guan/pRPL/

After you download the compressed file, extract it to a directory on a Unix/Linux-based parallel system[5], e.g., /home/yourname/pSLEUTH. Make sure a C++ compiler (e.g., g++) and a MPI-based parallel computing environment (e.g., LAM, MPICH) are installed on the parallel computer. Since pSLEUTH is based on pRPL[6], you should also make sure that the pRPL library is already compiled. Suppose you already save pRPL in the directory /home/yourname/pRPL, you may need to modify the makefile mk.psleuth in the directory /home/yourname/pSLEUTH/src according to the system setting (i.e., the location of mpi.h) and the location of pRPL.

mk.psleuth

… MPI_LIB = /opt/mpich-gm-gnu/include/ PRPL_LIB = ../../pRPL/ …	… # Specify the location of mpi.h # Specify the location of pRPL …

Then type the following commands to compile the pSLEUTH software, and an executable file psleuth will be created in the directory /home/yourname/pSLEUTH.

```
> cd /home/yourname/pSLEUTH/src
```

[5] Most of current parallel computing systems are Unix/Linux based. pRPL has been tested on several such kind of systems.

6 pRPL can be downloaded at http://www.geog.ucsb.edu/~guan/pRPL/. For how to compile pRPL, see "Getting started with pRPL" (Guan 2008)

```
> make –f mk.psleuth depend          # scan the dependency relations among the codes
> make –f mk.psleuth                 # compile the software
```

Also you can compile a sequential version of SLEUTH if you intend to compare the performance of the parallel version with that of the sequential version. Before compiling the sequential version, you need to modify the source code globalSet.h in the directory /home/yourname/pSLEUTH/src by commenting the line "`#define USE_MPI`", i.e., adding a comment symbol "`//`" at the beginning of the line.

globalSet.h

```
…
//#define USE_MPI
…
```

You may also need to modify the makefile mk.slueth in the directory /home/yourname/pSLEUTH/src according to the location of pRPL.

mk.psleuth

```
…
PRPL_LIB = ../../pRPL/
…
```

```
…
# Specify the location of pRPL
…
```

Then type the following commands to compile the sequential version of SLEUTH, and an executable file sleuth will created in the directory /home/yourname/pSLEUTH

```
> cd /home/yourname/pSLEUTH/src
> make –f mk.sleuth depend           # scan the dependency relations among the codes
> make –f mk.sleuth                  # compile the software
```

Now both the parallel version and the sequential version of SLEUTH are ready to use.

3. Use pSLEUTH and Its Performance

3.1 Input Data files

As mentioned in section 1.1, the SLEUTH model requires six GIS layers: Slope, Land-use, Exclusion, Urban, Transportation, and Hillshade. The land-use data is used for land-use change simulation, and the hillshade data is used as the background of the output image. Since pSLEUTH is only used to simulate the urban growth process, it only requires four layers: Slope, Exclusion, Urban, and Transportation.

All the data must be in grayscale GIF image files, and they must have the consistent dimensions, i.e., the numbers of rows and columns. If it is used for calibrating the parameters, at least two urban time periods, i.e., two GIF images of the urban areas at different historical times, must be used. All layers should be checked for agreement; urban areas should not be present locations defined as undevelopable in the excluded layer.

For more details of preparing the input datasets for SLEUTH/pSLEUTH, see http://www.ncgia.ucsb.edu/projects/gig/v2/About/dtInput.htm

Also, the image files must follow the required naming format: http://www.ncgia.ucsb.edu/projects/gig/v2/Imp/imSetUp.htm#namingConvention

Another input data is an ASCII text file of the neighborhood configuration. The SLEUTH model uses the Moore neighborhood[7], which has nine neighboring cells (including the central cell itself). The neighborhood file (Figure 3.1) contains the number of neighboring cells, and their row-column coordinates relative to the central cell (coordinate [0, 0]), and their weights (all are set to be 1.0 since the neighborhood used in SLEUTH is an equally weighted one). Note that there is a neighborhood file in the directory of the test datasets, i.e., moore.nbr, and you only need to copy this file to the directory of your own input datasets.

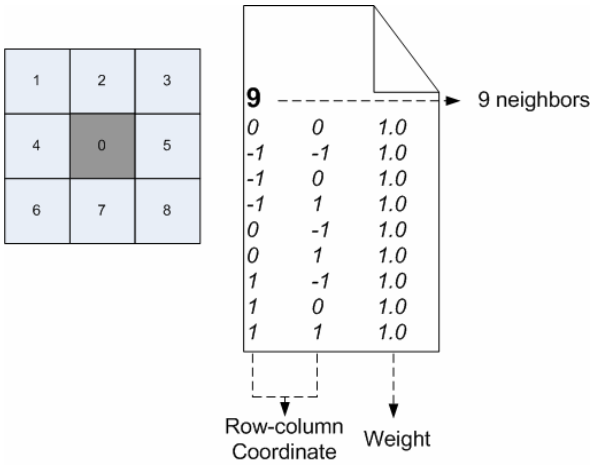


Figure 3.1 The Moore neighborhood configuration

and the neighborhood file used in SLEUTH

3.2 Scenario file

Besides preparing the input image files, users have to specify the execution option flags and variables in the scenario file. A scenario file will be read by the pSLEUTH software, and control the program's behavior.

[7] For more details about neighborhood configurations, see "Getting started with pRPL".

santaBarbara.scen

```
# INPUT_DIR: relative or absolute path where input image files are
#                    located.
# OUTPUT_DIR: relative or absolute path where all output files will
#                     be located.
INPUT_DIR = /data/guan/pSLEUTH/sbInput/
OUTPUT_DIR = /data/guan/pSLEUTH/sbOutput/

# Input Image Files
#      <  >  = user selected fields
#     [<  >] = optional fields
#
# Urban data GIFs
#   format:   <location>.urban.<year>.[<user info>].gif
#
#
URBAN_DATA = sb2074.urban.1976.gif
URBAN_DATA = sb2074.urban.1986.gif
#
# Road data GIFs
#   format:   <location>.roads.<date>.[<user info>].gif
#
ROAD_DATA = sb2074.roads.1976.gif
ROAD_DATA = sb2074.roads.1986.gif
#
# Slope data GIF
#   format:   <location>.slope.[<user info>].gif
#
SLOPE_DATA = sb2074.slope.gif
#
# Excluded data GIF
#   format:   <location>.excluded.[<user info>].gif
#
EXCLUDED_DATA = sb2074.excluded.gif
#
# Neighborhood ASCII file
#
NEIGHBORHOOD_DATA = moore.nbr

#
# The number of  Monte Carlo runs
#
MONTE_CARLO_ITERATIONS = 1
```

```
#
# The run mode is either CALIBRATE or FORECAST
#
RUN_MODE = CALIBRATE
#RUN_MODE = FORECAST

#
# The parameter (coefficient) setting for CALIBRATE mode
# For more details about coefficients of SLEUTH,
# see http://www.ncgia.ucsb.edu/projects/gig/v2/About/gwCoef.htm
#
CALIBRATE_DIFFUSION_START = 0
CALIBRATE_DIFFUSION_STOP = 100
CALIBRATE_DIFFUSION_STEP = 50
CALIBRATE_SPREAD_START = 0
CALIBRATE_SPREAD_STOP = 100
CALIBRATE_SPREAD_STEP = 50
CALIBRATE_BREED_START = 0
CALIBRATE_BREED_STOP = 100
CALIBRATE_BREED_STEP = 50
CALIBRATE_SLOPE_START = 0
CALIBRATE_SLOPE_STOP = 100
CALIBRATE_SLOPE_STEP = 50
CALIBRATE_ROAD_START = 0
CALIBRATE_ROAD_STOP = 100
CALIBRATE_ROAD_STEP = 50

#
# The parameter (coefficient) setting for FORECAST mode
#
FORECAST_DIFFUSION = 40
FORECAST_SPREAD = 100
FORECAST_BREED = 41
FORECAST_SLOPE = 1
FORECAST_ROAD = 23

#
# Data Range for FORECAST mode
# The urban and road input data of the start year FORECAST_YEAR_START must be provided
#
FORECAST_YEAR_START = 1986
FORECAST_YEAR_STOP = 2000

#
```

```
# Self-Modification parameters
# For more details about the self-modification in SLEUTH
# see www.ncgia.ucsb.edu/project/gig/About/gwSelfMod.htm
#
ROAD_GRAVITY_SENSITIVITY = 0.01
SLOPE_SENSITIVITY = 0.01
CRITICAL_LOW = 0.97
CRITICAL_HIGH = 1.03
CRITICAL_SLOPE = 15.0
BOOM = 1.1
BUST = 0.9

#
# Logging options
# OUTPUT_EVERYYEAR: whether output the image of each year during the simulation
# LOG_STATISTICS: whether log the statistics during the simulation
# LOG_COEFFICIENTS: whether log the coefficient modification during the calibration
# LOG_COMPUTING_TIME: whether log the computing time of the execution of the program
# ECHO: whether prompt the status during the execution on the terminal screen
OUTPUT_EVERYYEAR = NO
LOG_STATISTICS = NO
LOG_COEFFICIENTS = NO
LOG_COMPUTING_TIME = YES
ECHO = YES
```

3.3 Run the software

You must use the command mpirun to run pSLEUTH. You must specify the scenario file and the number of sub-cellspaces, and can also specify other options for pSLEUTH.

Syntax[8]

mpirun –np <num_prcs> [mpi_opts] pSLEUTH -sf <scenario_filename> –ns <nSubSpaces> [pSLEUTH_opts]	
-np <num_prcs>	Specify the number of processors to execute in parallel
[mpi_opts]	Other options for **mpirun**, e.g., -machinefile. For more information, read the system manual or consult the system adminstrator
-sf <scenario_filename>	Specify the name of and the path to scenario file
-ns <num_subspaces>	Specify the number of sub-cellspaces that will be produced

[8] <> - user-specified field; [] – optional field

		from decomposition. If the block-wise decomposition is being used, you can specify two –ns flags. For example: "*-bd –ns 2 –ns 4*" will decompose the cellspace into 2X4 sub-cellspaces.
[pSLEUTH_opts]	-ng <num_groups>	Specify the number of processor groups. If not specified, it will be 1 by default
	-dt/-st	Load-balancing option. -dt: dynamic tasking -st: static tasking You can use either of them, but can not use both of them at the same time. Static tasking will be used if this option is not specified
	-rd/-cd/-bd/-qd	Domain decomposition option -rd: row-wise decomposition -cd: column-wise decomposition -bd: block-wise decomposition -qd: quad-tree decomposition You can use any of them, but can not use more than one decomposition method at the same time. Note: If the number of sub-cellspaces is specified and greater than 1, and the domain decomposition option is not specified, the row-wise decomposition will be used.

Example: Suppose we are going to decompose the space into 8 sub-cellspace using row-wise decomposition method, use 16 processors and divide them into 4 groups (4 processors in a group, and 2 sub-cellspaces for each process in a group), and use the static tasking load-balanding method among groups.

```
>cd /home/yourname/pSLEUTH
>mpirun –np 16 ./pSLEUTH -sf ./testInput/test.scen -ng 4 -st -rd -ns 8
```

3.4 Performance

The source code package includes a test dataset. However, it is a very small dataset in terms of volume, and only used to test whether the software runs properly. pSLEUTH shows its performance advantages on massive datasets only.

For the demonstration purpose, a dataset of the US's urban areas is used in this section. Each imput GIF image is 4948×3108 in dimensions (Figure 3.2).

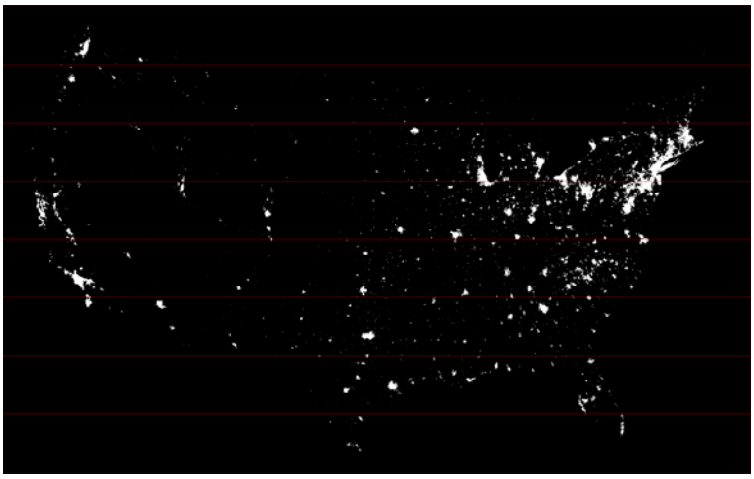


Figure 3.2 The US's urban areas in 1980,

decomposed into 8 sub-cellspaces using row-wise decomposition

To demonstrate the performance of pSLEUTH, a calibration scenario was specified as follows: use only two historical urban area data (1980 and 1990), only three values (0, 50, 100) will be evaluated for all the coefficients, and only 1 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.

The experiments were conducted on a Dell cluster[9] which is composed of a Dell 1750 dual CPU 3.06GHz Xeon servers and a single Dell 1750 monitoring node. The head node has 4GB RAM, 2 mirrored system disks, and a 2TB RAID array that is shared to the cluster. The 128 compute nodes have 2GB RAM each.

By dividing the cellspace into the number of the processors, i.e., each processor only holds one sub-cellspace, the speed-up[10] reached 12.4 when 32 processors were used (Figure 3.3 and 3.4).

By reducing the granularity of the decomposition, i.e., increasing the number of sub-cellspaces, and letting a processor hold more than one sub-cellspaces, the speed-up was largely improved. The speed-up reached 20.7 when 32 processors were used, the row-wise decomposition was used, and each processor held 8 sub-cellspaces (Figure 3.5).

Moreover, by grouping the processors and dynamically assigning tasks to the groups also improved the performance. The speed-up reached 24 when 32 processors were divided into 8 groups with dynamic tasking, the column-wise decomposition was used, and each processor held 8 sub-cellspaces (Figure 3.6). Note that when dynamic tasking is being used, one of the processor will be the emitter processor and will not

[9] For more information about this Dell cluster, see
http://www.cnsi.ucsb.edu/computing/clusters/dell_cluster/dell_cluster.php

[10] Considering an algorithm executes on a parallel system with p processors in a time t_p, and executes on a sequential system (with only one processor) in a time t_1, the speed-up is defined by the ratio: $S_p = \frac{t_1}{t_p}$

participate in the computation. In Figure 3.6, for dynamic tasking experiments, the number of processors does not include the emitter processor.

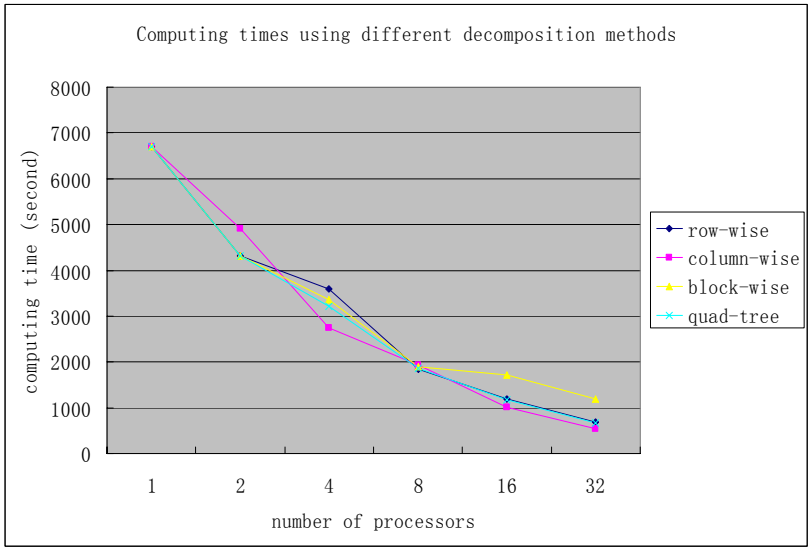


Figure 3.3 Computing times using different decomposition methods

(each processor holds only one sub-cellspace)

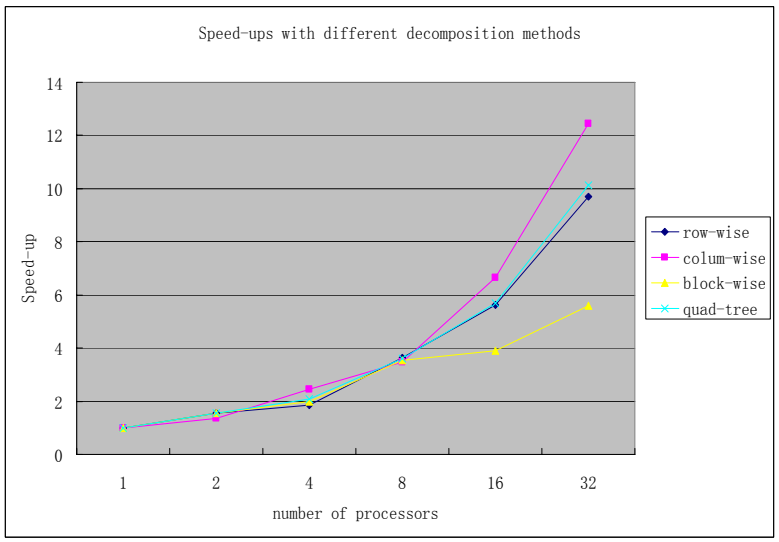


Figure 3.4 Speed-ups with different decomposition methods

(each processor holds only one sub-cellspace)

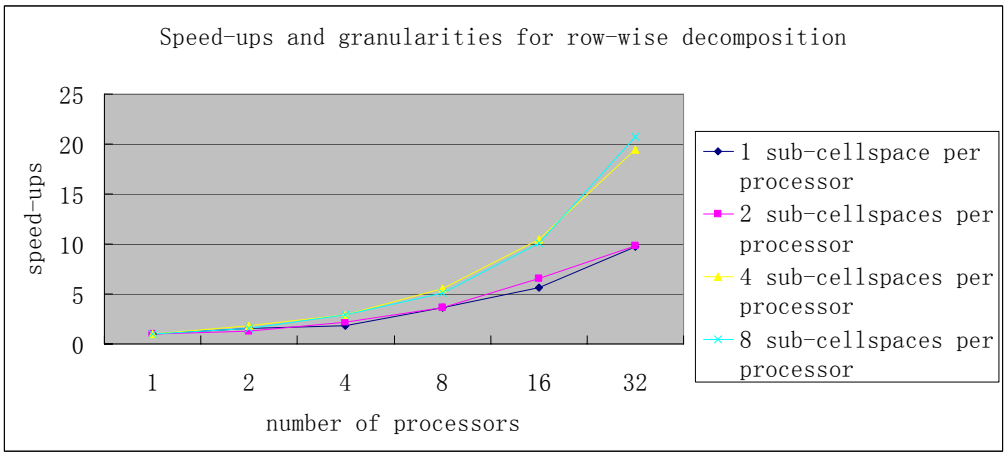


Figure 3.5 Speed-ups with different granularities for row-wise decomposition

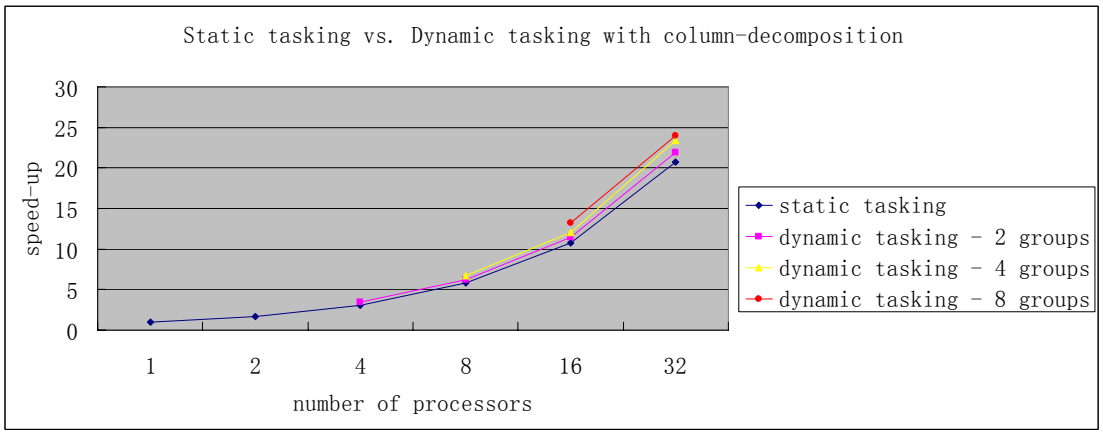


Figure 3.6 Speed-ups with dynamic tasking with column-wise decomposition (each processor holds eight sub-cellspaces)

References

Clarke, Keith C. 2003. Geocomputation's Future at the Extremes: High Performance Computing and Nanoclients. *Parallel Computing* 29, no. 10:1281-1295.

Clarke, Keith C., and Leonard J. Gaydos. 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, no. 7:699-714.

Clarke, Keith C., S. Hoppen, and L. Gaydos. 1997. A Self-modifying Cellular Automaton Model of Historical Urbanization in the San Francisco Bay Area. *Environment and Planning B: Planning and Design* 24:247-261.

Goldstein, Noah Charles. 2003. Brains VS Braun – Comparative strategies for the calibration of a cellular automata-based urban growth model. In *7th International Conference on GeoComputation*, Southampton, England.

Grama, Ananth, Anshul Gupta, George Karypis, and Vipin Kumar. 2003. *Introduction to Parallel Computing*. New York: Addison-Wesley.

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

Guan, Qingfeng, Liming Wang, and Keith Clarke. 2005. An Artificial-Neural-Network-based, Constrained CA Model for Simulating Urban Growth. *Cartography and Geographic Information Science* 32, no. 4:369 - 380.

Li, Xia, and Anthony G. O. Yeh. 2001. Calibration of Cellular Automata by Using Neural Networks for the Simulation of Complex Urban Systems. *Environment and Planning A* 33:1445-1462.

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