Geoinformatica (2006) 10: 197–211
DOI: 10.1007/s10707-006-7579-0

Decreasing Computational Time of Urban Cellular Automata Through Model Portability

Charles Dietzel · Keith C. Clarke

Received: 04 October 2004 / Revised: 14 March 2005 /
Accepted: 27 September 2005

Abstract This paper investigates how portability of a model between different computer operating systems can lead to increased efficiency in code execution. The portability problem is not a trivial one, as many geographic models are designed to be run inside a set environment (Unix, Solaris, Linux, and Windows), and are further limited by hardware constraints. The SLEUTH urban growth model [K.C Clarke, S. Hoppen, and L. Gaydos. "A self-modifying cellular automaton model of historical urbanization in the San Francisco Bay area", *Environment and Planning B*, 24:247–261, 1997.] was implemented under three different operating systems (Solaris, Linux, and Windows). Results suggest that the model's performance is optimized by porting it from its original environment (Unix/Solaris) to Linux or Windows where faster processors are available. While the results show optimization of model performance, there are some suggestions of computational bottlenecks that may eventually limit the increased performance of the model. Regardless, the research demonstrates that the portability of a model can lead to not only to a decrease in computation time, but may increase the viability in practical applications and attract a wider user base.

Keywords Cellular automata · Calibration · Portability

1. Introduction

In the early days of computational geography, many models required the use of large computers, usually mainframes, to compute their results. Now many of these

C. Dietzel (✉)
Department of Geography,
University of California, Santa Barbara, CA, USA
e-mail: dietzel@geog.ucsb.edu

K. C. Clarke
National Center for Geographic Information Analysis,
Department of Geography,
University of California, Santa Barbara, CA, USA
e-mail: kclarke@geog.ucsb.edu

models are run as non-computationally intensive process on a standard desktop computer. The remainder of models that are not feasibly run on desktop machines are generally tackled with the widely available resources of Beowulf clusters, grid computing, and supercomputing facilities that are now standard practice at many universities and research centers. But for users that do not have access to these high-end computation resources, the execution of some models on the standard desktop is an arduous waiting game that consumes all computing resources until the process is completed. There must be someway to tackle this problem and provide increased computational power to the basic user while not sacrificing any elements of the model. One possible solution to this problem is portability; the application of a program (or model) in an environment that is not native. There are two main limits to increasing the portability of models: (1) modelers rarely release their entire code to the public for examination and use as open source, and (2) if a program or model works, many researchers are generally satisfied and do not strive to make radical changes to their code. This paper presents results that should encourage modelers wanting to increase the computational power of their models and decrease the computational time of model execution to release their code to others, and not to be satisfied with a model or program simply working—as there are always methods to improve model performance.

During recent years, models of land use change and urban growth have drawn considerable interest. Despite past failures in urban modeling [11], [12] there has been a renaissance of spatial modeling in the last two decades due to increased computing power, increased availability of spatial data, and the need for innovative planning tools for decision support [8]. We need look no further than Lee's 'Requiem' to see the role that computational capabilities played in the downfall of earlier urban models. Many of the 'seven sins,' with the exception of wrongheadedness, were linked to the computational limits of the time.

- Grossness—many models required too much microscopic data. This led to models that were too expensive computationally and irrelevant to broader-scale policy makers.
- Hungriness—too much data was required for models. Lee describes a model requiring too much data when it needs 30,000 datapoints. Datasets this size are now routinely handled inside commercial software packages.
- Complicatedness—many of the interactions within a model were too complicated, linking at times hundreds of factors. Modeling packages such as STELLA currently allow modelers to easily link numerous processes within a systems model, similar to those Lee was working with.
- Mechanicalness—the use of 'digital computers' was not as logical a practice as it is now. Implementing models on a computer was a highly specialized task requiring most if not all of the computational resources of campus or institute to run a model.
- Expensiveness—cost was a major limitation in the development of large-scale urban models. The time required to gather data, develop and run a model was expensive. The Internet now allows us to readily download geographic data, input it into a model, and generate outputs—all within a matter of days if not hours.

While these were many of the common problems of the time, and largely limited by the computational power available, Lee was wrong on one main point—the

increased computation power of computers ***has*** allowed the advancement of urban modeling, specifically the development of cellular automata and agent-based techniques, not to mention the development of the entire field of Geographic Information Science (GIS) that have played a critical element in furthering geographic research. Furthermore, it is our belief that future developments in computer science will allow even more rapid progression in geography than has been seen in recent decades.

With any model, there is an explicit need to calibrate and parameterize the model to fit the dataset; arguably the most computationally intensive part of the modeling process. Recently, modelers have increased their focus on the calibration phase of modeling to gain an increased understanding of how models, in particular cellular automata, work [3], [7], [13], [14], [15], [16]. The calibration phase of modeling is one, if not the most important, stage in modeling because it allows the fitting of model parameters to the input data, to be further used in forecasting. Failure to calibrate a model to the input data results in an application that is not robust or justifiable, so a wide array of possible parameter combinations or sets must be explored. While a preferential method of calibration has not been determined, a method known as 'brute-force' calibration has been used by researchers [5], [14] to calibrate the SLEUTH urban growth model. This method used a massive amount of computational time required to calibrate the model. Clarke and Gaydos [5] report using several *hundred* hours of CPU time to calibrate data for San Francisco, California. The computational time required to calibrate the SLEUTH model has not decreased significantly with advances in computers that host the required environment, and has led to the search for other methods of model calibration, including the use of genetic algorithms [9]. While the use of new techniques, including genetic algorithms, is a major advancement for geographic science and cellular automata modeling, Goldstein [9] still required the use of *twelve* Silicon Graphics O2workstations to complete model calibration. This suggests the question "Are there other methods to decrease the computational time of a model without sacrificing the current structure and features of a model?" Fueled by this question, this paper takes a widely used urban cellular automata model (SLEUTH) known for the vast computational resources required for model calibrations, and seeks to find a new method for decreasing the computational time and resources required for implement the model.

The SLEUTH urban model is a cellular automaton model that has been widely applied [6], [9], [10] and has shown its robust capabilities for simulation and forecasting of landscape changes [4]. The model makes use of several different data layers for parameterisation, e.g., multi-temporal land use and urban extent data, transportation network routes and digital elevation model data. Application of the model necessitates a complex calibration process to train the model for spatial and temporal urban growth [14]. This paper documents research on the role that model portability can play in decreasing the computational resources needed to implement geographic models through using the calibration of SLEUTH as an example. A standard calibration routine was used to evaluate the performance of the model within UNIX, Solaris, Linux, and Windows operating systems on a total of eight different computers, following slight modifications to the code. The computational time of each routine was logged and compared among the machines. These results are then used to examine the role that the broad issue of portability may have in decreasing the computational requirements in other models, as well as some general considerations about the computer hardware that may be best suited for future modeling endeavors using SLEUTH.

2. Model Routines and Hardware Tested

A total of eight different computers were used to determine if the portability of SLEUTH between different platforms would in any way decrease the computational time required to calibrate the model. In the most recent version of the model www.ncgia.ucsb.edu/projects/gig), the SLEUTH model has been recoded into modular ANSI C, the dynamic memory allocation returned to a flat memory structure optimized for the Cray memory model, and the code modified to use the Message Passing Interface (MPI). The model is typically run under UNIX using the standard gnu C compiler (gcc) and may be executed in parallel. It can be formatted for any other standard C compiler. The calibration process iteratively cycles through the five-dimensional control parameter space, evaluating the ability of parameter sets to replicate control data through a series of Monte Carlo simulations. Details on the calibration process and the choice of parameters can be found in [5], [7], [14]. The parameters drive the four transition rules that simulate spontaneous (of suitable slope and distance from existing centers), diffusive (new growth centers), organic (infill and edge growth), and road influenced (a function of road gravity and density) growth. These parameters are:

1. Diffusion—Determines the overall dispersiveness nature of the outward distribution. This parameter controls the number of times that a pixel will be randomly selected for possible urbanization. The maximum value is calculated as:

$$\text{diffusion value} = ((\text{diffusion coefficient} \times 0.005) \times (\text{number of rows})^2 + (\text{number of columns})^2)^{1/2}$$

   This means that the maximum diffusion value will be half of the input image diagonal.
2. Breed Coefficient—The likelihood that a newly generated detached settlement will start on its own growth cycle.
3. Spread Coefficient—Controls how much contagion diffusion radiates from existing settlements.
4. Slope Resistance Factor—Influences the likelihood of development on steep slopes.
5. Road Gravity Factor—An attraction factor that draws new settlements towards and along roads.

For this research a standard coarse calibration routine was used with the 'demo city' data that come as part of the standard SLEUTH package. These data were 50×50, 100×100, and 200×200 pixels in dimension and included data on urban extent, transportation networks, topography, and land use. 3,125 parameter sets were evaluated for each of the three datasets, testing the fit of the parameters to the control data through a set of five Monte Carlo simulations for each parameter set.

2.1. Model Routines

During the calibration of the SLEUTH model, eleven different C language code functions were logged for CPU time and used for comparison between the computer

systems that calibrated the model. Details on these routines can be found within the SLEUTH code, or on the SLEUTH webpage (www.ncgia.ucsb.edu/projects/gig).

1. spr_phase1n3—performs the spontaneous and diffusive growth rules.
2. spr_phase4—loops over the interior pixels looking for urban areas from which to implement organic growth. When the model finds a pixel that passes a random coefficient test, it examines its eight-cell neighborhood. If two of the cells within the neighborhood are urban, then that pixel becomes urban.
3. spr_phase5—searches for new urban growth from spr_phase4. If there is new growth then the model begins to implement road trips. When the model finds a newly urbanized cell it searches for a road within its window, and takes a random walk along that road.
4. spr_spread—main driver that calls spr_phase1n3, spr_phase4, and spr_phase5.
5. gdif_ReadGIF—input layers for the model are in Graphics Interchange Format (GIF). This routine reads the GIFs into the model.
6. delta_phase1—attempts to make land use transitions by picking a pixel to be a spreading center, then randomly choosing a new land use, testing the probability of changing to that land use, building a cluster around that pixel, and transitioning neighboring pixels [2].
7. delta_phase2—searches for pixels that have not changed within a specified number of years, and counts the neighbors that have changed within the previous year. It then ages the deltatrons and kills some of them [2].
8. delta_deltatron—main driver that calls delta_phase1 and delta_phase2.
9. grw_growth—implements cellular automata rules for each year of the calibration time period.
10. drv_driver—creates annual land use probabilities and drives Monte Carlo simulation.
11. main—reads in scenario file, sets coefficients, writes logs, maps memory, reads input files, calculates statistics against which control data are compared, and initializes calibration routine.

2.2. Hardware Specifications

A total of eight different computers were used in testing the portability of the model, and in determining in which environment its performance was maximized. These machines were chosen because they were readily available and found in a ‘typical’ Geography Department; they are though to be machines that are along the line of computers that researchers in other departments would use for research. One machine ran Silicon Graphics Irix 6.5, another used Redhat Linux 8.2, two ran Sun Solaris 8.0, three ran Cygwin within Windows 2000, and the last machine ran Cygwin within Windows XP-Professional (Table 1). Each machine was configured differently, and these differences provided great insight into the hardware requirements necessary to decrease the computational time of the model. When the model was run on each machine, it was the sole process running on that machine at the time, ensuring that other processes would not interfere with it. In an ideal system, the machines used would have been mirrored in their hardware configurations; but even computer science experts agree that achieving such control is hardly possible, so we contend that this is a caveat to the findings of this research, but that the results strongly support the conclusions.

Table 1 Computers and their specifications used to test the portability of the SLEUTH model

Computer name	Manufacturer	Model	Operating system	Processor	RAM
SGI	Silicon Graphics	O2	Irix 6.5 Unix	R10000 MIPS @ 195 Mhz	128 MB
Equator	Sun	Blade 1000	Solaris 8.0	2 × 1 Ghz SPARC	2 GB
Yosemite	Sun	Fire 4810 Server	Solaris 8.0	8 × 700 Mhz SPAARC	32 GB
Redhat	Gateway	GP7-700	Redhat 8.2	700 Mhz Intel Pentium III	384 MB
Tubby	Gateway	GP7-700	Windows 2000/ Cygwin	700 Mhz Intel Pentium III	384 MB
LT	Dell	Inspiron	Windows XP/ Cygwin	2.4 Ghz Intel Pentium 4-M	512 MB
Monkey	Generic	Generic	Windows 2000/ Cygwin	1.8 Ghz Intel Pentium 4	768 MB
Azure	Generic	Generic	Windows 2000/ Cygwin	AMD Athlon XP 2100+	512 MB

Note:

Cygwin is a Linux-like environment for Windows. It consists of two parts: A DLL (cygwin1.dll) which acts as a Linux emulation layer providing substantial Linux API functionality, and a collection of tools, which provide Linux look and feel.

3. Results of Model Calibrations

Results from the calibration of the SLEUTH model on the eight specific computers suggest that the model performs best when implemented within Linux or using Cygwin in a Windows environment for all sizes of input data (Fig. 1), where there were faster processors. The routine 'main' was used to decided which environment provided the best environment for running the model since it (main) was the most computationally intensive, and therefore limiting function (Tables 2, 3, 4).

For all of the computers, running the SLEUTH model inside the Cygwin environment under Windows (2000 or XP) or on a Linux machine, led to the fastest completion of the calibration routine for each dataset tested. This finding is in part biased due to the difference in processor speed between the machines tested; but as it stands now, the processors that one can buy for UNIX machines (SPARC processors from Sun Microsystems) pale in speed to those available to for Windows/ Linux machines (Intel or AMD). As the size of the dataset increased, the difference in performance between the Windows/Linux and Unix/Solaris machines increased with the timing of the Windows machines not increasing in such a steep trajectory. It is important to note that the Linux machine (*Redhat*) performed equally as well as a Windows machine (*Tubby*) with the same configuration and performance settings, so we contend that if as the processor speed on the machine used in testing were increased, the results (timing between Windows and Linux) would still be the same. The individual routines of the calibration run within the Cygwin environment were examined in more detail to determine which individual routines within the model were the most computationally intensive (Fig. 2).

There were differences in the computational time for all machines with respect the routines: spr_spread, spr_phase5, gdif_ReadGIF, delta_deltatron, and delta_

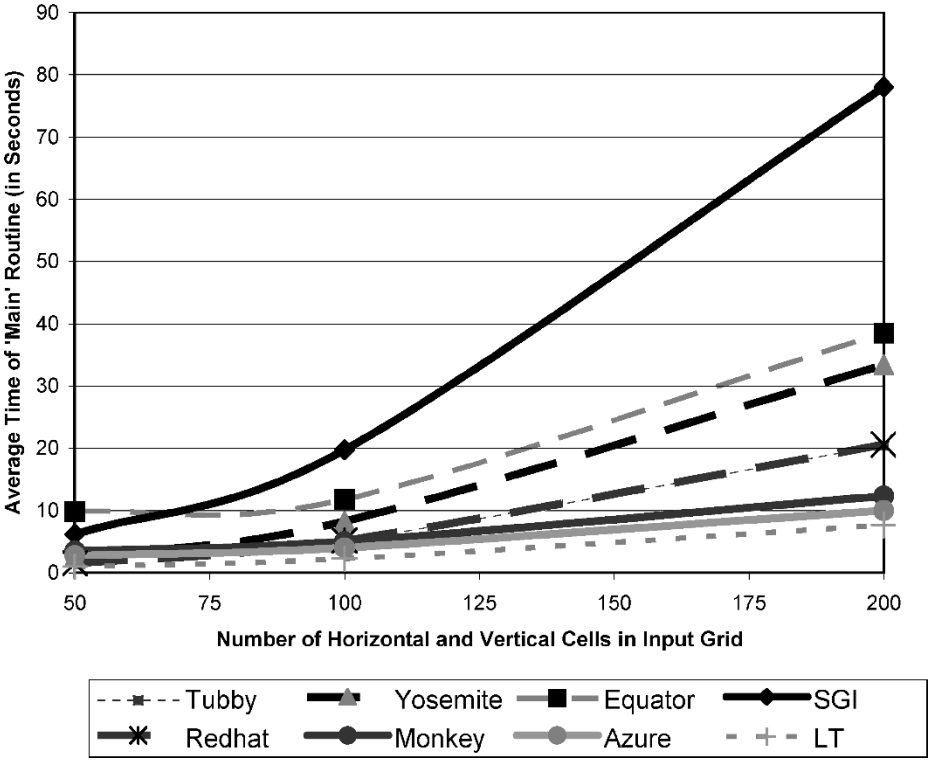

Fig. 1 Computational time in seconds for the eight computers tested to execute the *main* function for a 200 × 200 dataset, using a coarse calibration routine and 5 Monte Carlo iterations

phase2. Despite differences in the configuration of the computers, the timing of the routines spr_phase1n3, spr_phase4, and delta_phase1 were the same for both the machines *LT* and *Azure*. This is an interesting revelation that will be discussed more, since the processors in the machines (Intel 2.4 GHz vs. AMD 2100+) were not

Table 2 Timings in milliseconds for each logged routine for the calibration of a 50 × 50 dataset for the SLEUTH model

	SGI	Equator	Yosemite	Redhat	Tubby	LT	Monkey	Azure
spr_spread	8.18	3.84	3.67	2.56	2.42	0.96	1.27	0.96
spr_phase1n3	0.09	0.05	0.04	0.03	0.03	0.02	0.02	0.02
spr_phase4	2.21	1.11	1.08	0.77	0.66	0.24	0.33	0.24
spr_phase5	5.3	2.43	2.38	1.58	1.5	0.62	0.8	0.59
gdif_ReadGIF	5.38	21.54	1.54	1.54	2.31	2.31	10	7.23
delta_deltatron	3.68	1.68	1.63	0.96	1	0.39	0.53	0.4
delta_phase1	1.05	0.52	0.51	0.64	0.33	0.12	0.18	0.12
delta_phase2	2.61	1.15	1.12	0.32	0.66	0.25	0.34	0.28
grw_growth	1215.85	1921.31	485.38	302.28	353.41	172.08	635.29	513.19
drv_driver	6175.59	9813.03	2464.3	1514.72	1912.3	990.98	3418.4	2771.28
main	6180.46	9856.66	2464.08	1514.7	1916.99	994.51	3455.99	2785.73

Table 3 Timings in milliseconds for each logged routine for the calibration of a 100 × 100 dataset for the SLEUTH model

	SGI	Equator	Yosemite	Redhat	Tubby	LT	Monkey	Azure
spr_spread	22.18	10.57	10.27	7.53	6.19	2.39	3.13	2.42
Spr_phase1n3	0.28	0.1	0.1	0.09	0.08	0.04	0.05	0.04
spr_phase4	7.16	3.57	3.53	2.81	2.14	1.4	1.01	0.75
spr_phase5	12.62	6.21	6.06	3.95	3.31	0.75	1.82	1.33
Gdif_ReadGIF	11.54	13.08	2.31	3.08	3.85	3.08	10	8.38
delta_deltatron	18.2	8.49	8.36	4.34	4.96	1.94	2.58	1.95
delta_phase1	3.87	1.89	1.85	2.69	1.22	0.5	0.67	0.46
delta_phase2	14.29	6.59	6.5	1.65	3.73	1.44	1.9	1.49
grw_growth	3922.63	2330.8	1640.79	1038.25	1004.43	430.72	965.19	765.8
drv_driver	19752.29	11788.66	8266.37	5201.08	5171.02	2283.42	5033.48	4002.54
main	19751.62	11806.05	8264.42	5199.88	5174.63	2286.6	5050.31	4016.36

the same, and comparison between these two speeds is somewhat difficult. Overall, the results suggest that the model performs better when ported from a Unix/Solaris environment to an emulator within a Windows environment or on Linux, where faster processors are available.

4. Ability to Forecast Calibration Time

In calibrating the SLEUTH model, a set of theoretical 'calibration curves' (each unique to the individual machine used for testing) were generated, plotting the time required to calibrate the model (Fig. 1). The curves can be used as a method to determine roughly how long it will take to calibrate a dataset where the number of horizontal and vertical cells was between 50 and 200. After analyzing the calibration results, these curves were used to forecast how long it would take to calibrate a dataset with a larger number of cells in each dimension. Using a second order polynomial trend line, a curve was fit using the datapoints from the *main* function

Table 4 Timings in milliseconds for each logged routine for the calibration of a 200 × 200 dataset for the SLEUTH model

	SGI	Equator	Yosemite	Redhat	Tubby	LT	Monkey	Azure
spr_spread	85.73	43.71	42.13	29.17	24.11	8.61	11.39	9.37
spr_phase1n3	0.63	0.24	0.22	0.22	0.18	0.09	0.12	0.09
spr_phase4	26.34	13.64	13.05	10.3	7.76	2.7	3.55	2.7
spr_phase5	48.72	27	26.52	14.78	12.81	5.15	6.86	5.24
gdif_ReadGIF	43.08	20.77	10	10.77	13.85	10	14.62	12.08
delta_deltatron	83.81	38.94	37.77	18.94	23.98	9.01	12.18	9.51
delta_phase1	16.19	7.37	7.07	10.37	5.09	2.01	2.89	2.01
delta_phase2	67.62	31.63	30.75	8.56	18.87	6.99	9.27	7.49
grw_growth	15551.9	7653.03	6647.47	4125.41	4058.31	1504.29	2413.67	1962.07
drv_driver	78090.77	38497.65	33399.03	20648.44	20453.62	7667.86	12335.48	10020.51
main	78071.65	38506.79	33389.12	20642.31	20452.71	7669.52	12350.48	10032.77

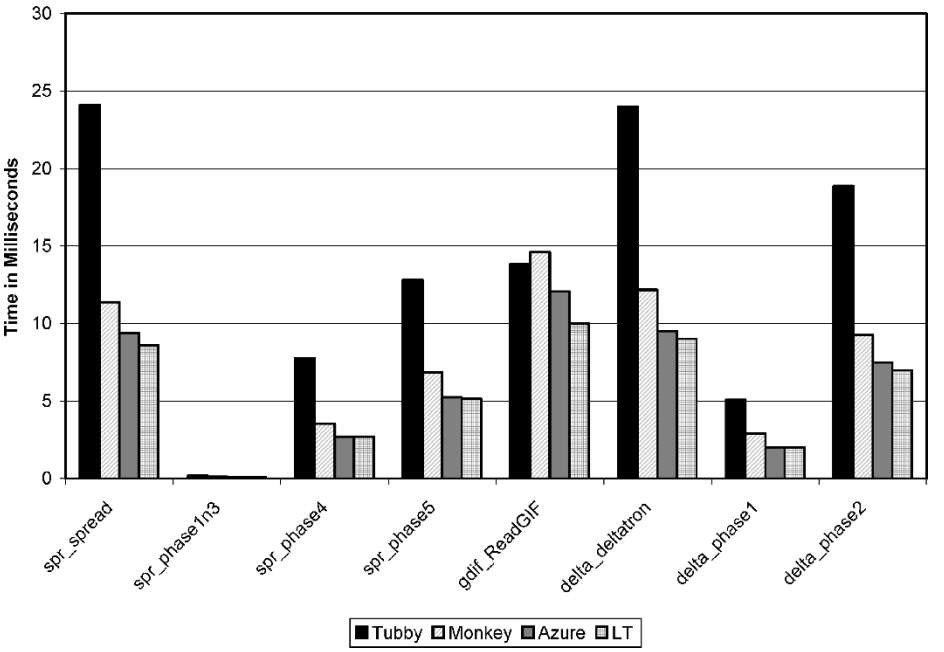

Fig. 2 Timing in milliseconds of calibration routines for the 200 × 200 dataset done using four machines, each running Cygwin within a Windows environment

for Tubby, Monkey, and Azure, and forecast for a dataset that was 500 × 500 cells in size (equation 1).

$$\begin{aligned}&(\mathrm{A})\ Y = 0.5842{*}X^2 - 22.475{*}X + 1580.3\\ &(\mathrm{B})\ Y = 0.2741{*}X^2 - 9.2289{*}X + 3232.2\\ &(\mathrm{C})\ Y = 0.2370{*}X^2 - 10.939{*}X + 2740.1\end{aligned} \tag{1}$$

Equation 1. Equation of second order polynomial trend line to forecast the computational time required to execute the *main* function in the SLEUTH model for a 500 × 500 dataset using the machine Tubby (A), Monkey (B), and Azure (C). Y is computational time, and X is the number of cells in the horizontal or vertical axis of the dataset.

The dataset used in calibration of SLEUTH for the 50 × 50, 100 × 100, and 200 × 200 tests, was resampled to 500 × 500, and calibrated to determine how accurate the forecasting ability of the calibration curves were for much larger datasets. Results from the calibration of the 500 × 500 dataset demonstrate how accurate the calibration curves were able to predict the computational time required to execute the *main* function (Table 5). This method can now be used by researchers to get rough, but accurate, estimates of how long an extensive dataset would take to run before allotting an unknown amount of time to execute the process.

The limit of this test is that the datasets used did not tax the memory capacity of the machines. When the datasets used begin to become more extensive, memory (RAM) becomes an issue. Experience suggests two things. First, if SLEUTH were

Table 5 Estimated computational time (in seconds) for Tubby, Monkey, and Azure to execute the main function for a 500 × 500 dataset, as compared to the actual computational time

Computer Name	Estimated Time	Actual Time	Difference	Percent Error
Tubby	136.3928	136.55012	0.15732	0.115
Monkey	67.14275	65.60108	−1.54167	−2.350
Azure	56.5206	55.63362	−0.88698	−1.594

used to model a system where the data were larger than the memory capacity of the machine, then the model simply would not be able to run and a the modeler must (1) find a machine with more capacity; or (2) resample the data for a coarser resolution to reduce its size so the computer's memory could handle it. The second possibility would be where the memory on the system was taxed to the limits, but not exceeded. While this condition was not examined in this research, it is though that if two machines with equal configurations, with the exception of RAM speed, were used, then the machine with the faster RAM would have the lower computational time. While this is just a hypothesis, it is grounds for future research of testing machine configurations when specific components (in this case, memory) are taxed to the limits of the machine's capacity.

5. Discussion

Findings of this research suggest that by porting a model from its natural environment to an alternative execution time of the model can be greatly reduced if the machine used has a faster processor. This is important when dealing with models that are executed on a timeframe measured on the order of days not minutes. This research may have uncovered one major problem that may ultimately restrict the reduction of execution time. In examining the timings of the calibration routines it was found that three routines (spr_phase1n3, spr_phase4, and delta_phase1) had the same exact timing for the two fastest machines, despite there being a 20% difference between the 'main' routine, which is the overall limiter in model performance. While we cannot directly compare between the processor speeds of *Azure* and *LT* (one is an Intel Pentium IV processor at 2.4 GHz, the other an AMD AthlonXP 2100+), it is known that they are not equal in speed and the AMD is estimated to be equivalent to 1.73 GHz. Because of this difference between the two processor speeds it is necessary to look for other explanations as to why these processes require the exact same amount of computational time.

The first explanation of this maybe related in some form to Amdahl's Law [1]. This law of computer science deals with the performance improvement of using some faster mode of execution in a computer, and the improvement is limited by the fraction of the time the faster mode can be used. As some function within the machine is sped up, some other function will become the bottleneck for the overall process, and for any given functional unit, there is a maximum benefit we can ever get by speeding it up in isolation. This may help to explain why the three call routines (spr_phase1n3, spr_phase4, and delta_phase1) all had the same timing, yet on two processors of different speeds. The three routines in question can be viewed as a computational bottleneck where the computer cannot process the task any faster than a certain point given the size of an input grid. It can be hypothesized that there will be

more computational bottlenecks the larger the input datasets become if this is the explanation for why the two different processors had the exact timing for the spr_phase1n3, spr_phase4, and delta_phase1, despite a 20% difference in the 'main' routine.

The second explanation deals with the difference in architecture between the two chips. There are three facts about the AMD chip that may explain why, despite its slower clock speed (as demonstrated on the 'main' routine), it (the AMD) was able to perform the same as the Intel chip on the spr_phase1n3, spr_phase4, and delta_phase1 routines.

1. AMD's AthlonXP chips have sixteen times the level-1 high-speed cache memory that Intel Pentium IV chips do.
2. The ratio of functional units between the AthlonXP and the Intel Pentium IV is 3:2. This means that the AthlonXP chip can process three floating-point instructions for every two by the Intel.
3. Lastly, the AthlonXP has three times the instruction decoders of a Pentium IV chip, so that for every instruction passed through the processor of a Pentium IV chip, the AMD can process three.

While we have not fully explained what led to the similarity in computational time between *Azure* and *LT* during the spr_phase1n3, spr_phase4, and delta_phase1 routines and the large difference between the 'main,' the true explanation is probably a *combination* of both Amdahl's Law and the difference in chip architecture. That is, while there may be a *slight* computational bottleneck during the spr_phase1n3, spr_phase4, and delta_phase1 routines, the difference in chip architecture between AMD and Intel chips led to the AMD chip having similar computational times during those routines. During other routines, where there was no bottleneck or computational constraint, the Intel chip was able to easily outperform the AMD chip due to faster processor speed. Based on this explanation, if two computers, one with an Intel Pentium IV and the other with an AMD Athlon processor of equal speeds were put head-to-head, the AMD might out perform the Intel due to the ability to more quickly process instructions (calculations) during the most computationally intensive routines of the model. More research is necessary to full substantiate this claim, but based on the findings of this research, and differences in chip architecture, it is a logical hypothesis.

It is also important to reiterate that these computations were the only processes that were running at the time on each machine. Inevitably, this would not necessarily be the case unless a modeler had one machine fully dedicated to running a model. The influence of other tasks on the execution time would undoubtedly have an impact. Some system architectures, specifically Solaris, are much better at handling multiple tasks at a time. But judging from the results of these calibrations, the single most important factor in determining the time required to calibrate is processor speed, and the speed of the Solaris SPARC processors is not nearly as fast as those manufactured by Intel and AMD. Other important factors as the input dataset gets larger will be the amount of RAM and Bus speed. The amount of RAM (random access memory, the place in a computer where the operating system, application programs, and data in current use are kept so that they can be quickly accessed by the computer's processor) necessary to run larger applications (input grids on the order of 10^6 cells) will require much more RAM than the smaller test applications that were applied in this research. Bus speed (the bus speed measures how fast data travels from the

processor to the memory, I/O, and peripherals) will also have an impact on how fast the model can be executed by increasing the read/write time of data in computationally expensive processes. This leads to two conclusions. The first is that if a modeler has the resources to run a model on a single computer, uninterrupted by other processes, then machines with the fastest processors and Bus speed, as well as the most RAM will be optimal. On the other hand, if there are not abundant computational resources available, then the consideration of using a system such as Solaris, which is better at handling multiple processors, may be the best bet for efficient computation. One aspect to recall is that all of the calibrations were done using only one processor, and that none of the processes were implemented in parallel. This was done because there was not access to a parallel machine, and because the findings are intended to help those who have access only to basic and limited resources. With the use of MPI (message passing interface), clusters, and grid computing, the opportunities to use multiple processors are widespread. Even commercial software packages, including the newly released Adobe After Effects, have the ability to incorporate multiple processors into completing computations.

The accuracy of the calibration curves in estimating the time required to execute the *main* function within the SLEUTH model has demonstrated that it is now possible to determine ahead of time, an accurate estimate of how long a calibration will take on any of the machines tested; a process that can be used with some degree of accuracy for any computer. The implication of this is that when developing a dataset, modelers can now estimate how long it will take to calibrate their data at a specific resolution as compared to another. While the results presented do not suggest a universal rule for estimating the computational time, they do demonstrate that it can be readily calculated based on a small sample of calibration times.

While this research has attempted to port the SLEUTH model to as many possible environments as possible, there are still several readily available desktop options that have not been fully explored. The first of these is Apple's OS X. Being built on top of the traditional UNIX APIs (application program interface) and with the complete UNIX environment (including X11) it maybe possible to get the SLEUTH model to run on a Macintosh machine; for this research, there was no Apple machine available for testing. The other environment that was not explored was the Sun Java Desktop system. This operating system is built integrating Linux and the Java 2 Platform together. It offers all of the features that most Linux operating systems (Redhat, SUSE, Mandrake, etc.) have, so it is thought to be readily possible to get the SLEUTH model to run in this environment since it has already been run using Redhat linux.

6. Conclusions

Results from this work suggest that one method for decreasing the computational time of models (or applications for that matter) may be to port them to some other environment where fasters processors are available. The SLEUTH urban growth model was originally intended to be run using either the UNIX or Solaris operating systems, but this research has demonstrated that the model's performance is improved by porting it to another environment (Linux or Windows/Cygwin) where the machines utilize faster processors. Ultimately the performance of a model is

determined by the circumstances that it is implemented under. The results presented were all generated with no other processes (other than background system) running, so it may be necessary to further consider the load of other processes running when choosing which environment to implement a model within.

The main reason that this research was possible was due to the open source nature of the SLEUTH model, which is readily available for free and has a broad user community with a discussion board answering questions. If other modelers have the desire to port their models from one environment to another, it may be wise for them to consider opening up their model code so that a wide array of users can examine it and test using it on a variety of systems. This may especially be possible for those working using heuristics including simulated annealing, genetic algorithms, and other algorithms that typically require large amounts of computational time.

When porting models between different environments, it is important to realize that computational performance is specific to one particular program. Certain programs may be optimized by using one environment versus another, while others may perform more poorly. In either case, the portability of a model may help to attract a wider audience of users, especially in the case when free software (in the case of Cygwin or Linux) takes the place of more expensive licenses. Additionally the portability aspect of a model to a more common environment may help increase its implementation in practical applications outside of the world of scientific computing. Considering the success of this work in reducing the computational time required to execute a model, we encourage other researchers to explore implementing their models in other environments and test the degree that portability can play in reducing the computational time of 'hungry' models.

Acknowledgments The authors would like to thank the wider SLEUTH community for their contribution to this work through their work in porting the model to Linux and implementing it within Cygwin, and the Public Policy Institute of California for the use of their Sun Fire server. The contribution of Claire Jantz (Woods Hole) and Jennifer Small at University of Maryland in determining the modifications required to port the code to Linux was especially appreciated.

References

1. G.M. Amdahl. "Validity of single-processor approach to achieving large-scale computing capability," in *Proceedings of AFIPS Conference*, pp. 483–485, Reston, VA, 1967.
2. J. Candau. "Calibrating a cellular automaton model of urban growth in a timely manner," in *Proceedings of the Fourth International Conference on Integrating GIS and Environmental Modeling (GIS/EM4)*, Banff, Alberta, 2000.
3. G. Caruso, M. Rounsevell, and G. Cojocaru. "Exploring a spatio-dynamic neighbourhood-based model of residential behaviour in the Brussels periurban area," *International Journal of Geographical Information Science*, Vol. 19:103–123, 2005.
4. K.C Clarke, S. Hoppen, and L. Gaydos. "A self-modifying cellular automaton model of historical urbanization in the San Francisco Bay area," *Environment and Planning B,* Vol. 24:247–261, 1997.
5. K.C. Clarke and L. Gaydos. "Loose-coupling a cellular automaton model and GIS: Long-term urban growth prediction for San Francisco and Washington/Baltimore," *International Journal of Geographical Information Systems*, Vol. 12:699–714, 1998.
6. K.C. Clarke. "The limits of simplicity: Toward geocomputational honesty in urban modeling," in P. Atkinson, G. Foody, S. Darby, and F. Wu (Eds.), *GeoDynamics*, Boca Raton, CRC Press, 2005.
7. C. Dietzel. "Spatio-temporal difference in model outputs and parameter space as determined

by calibration extent," in P. Atkinson, G. Foody, and F. Wu (Eds.), *GeoDynamics*, CRC Press, 2005.
8. S. Geertman and J. Stillwell. "Planning support systems: An inventory of current practice," *Computers, Environment and Urban Systems*, Vol. 28:291–310, 2004.
9. N.C. Goldstein. "Brains VS Braun—Comparative strategies for the calibration of a cellular automata-based urban growth model," in P. Atkinson, G. Foody, S. Darby, and F. Wu (Eds.), *GeoDynamics*, CRC Press, 2005.
10. C.A Jantz, S.J. Goetz, and M.K. Shelley. "Using the SLEUTH urban growth model to simulate the impacts of future policy scenarios on urban land use in the Baltimore-Washington metropolitan area," *Environment and Planning B*, Vol. 31:251–271, 2004.
11. D.B. Lee. "Retrospective on large-scale urban models," *Journal of the American Planning Association*, Vol. 60:35–40, 1994.
12. D.B. Lee. "Requiem for large-scale models," *Journal of the American Institute of Planners*, Vol. 39:163–178, 1973.
13. X. Li and A. Yeh. "Calibration of cellular automata by using neural networks for the simulation of complex urban systems," *Environment and Planning A*, Vol. 33:1445–1462, 2001.
14. E.A. Silva and K.C. Clarke. "Calibration of the SLEUTH urban growth model for Lisbon and Porto, Portugal," *Computers, Environment and Urban Systems*, Vol. 26:525–552, 2002.
15. B. Straatman, R. White, and G. Engelen. "Towards an automatic calibration procedure for constrained cellular automata," *Computers, Environment and Urban Systems*, Vol. 28:149–170, 2004.
16. F. Wu. "Calibration of stochastic cellular automata: The application to rural-urban land conversions," *International Journal of Geographical Information Science*, Vol. 16:795–818, 2002.

Charles Dietzel is a doctoral student in the Department of Geography at the University of California, Santa Barbara. He holds a B.A. degree with honors from Washington and Lee University, and a M.E.M. from Duke University. His current research is focused on the calibration of cellular automata models to simulate urban growth, for which he has been awarded a NSF Dissertation Improvement Grant Recipient, as well as a Dissertation Fellowship at the Public Policy Institute of California.

Dr. Keith C. Clarke is a professor and chair of the Department of Geography at the University of California, Santa Barbara. He holds a B.A. degree with honors from Middlesex Polytechnic, London, England, and the M.A. and Ph. D. from the University of Michigan, specializing in Analytical Cartography. Dr. Clarke's most recent research has been on environmental simulation modeling, on modeling urban growth using cellular automata, on terrain mapping and analysis, and on the history of the CORONA remote sensing program. He is the former North American Editor of the International Journal of Geographical Information Systems, and is series editor for the Prentice Hall Series in Geographic Information Science. Since 1997, he has been the Santa Barbara Director of the National Center for Geographic Information and Analysis.