The Design of the Land-Use Change Analysis System (LUCAS): Part II – Parallel and Distributed Implemetation

B.C. Hazen & M.W. Berry

Computer Science Department

CS-95-298 August 1995

The Design of the Land-Use Change Analysis System (LUCAS): Part II - Parallel and Distributed Implementation ⋆

B.C. Hazen

Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN, 37996-1301, hazen@cs.utk.edu

M.W. Berry

Department of Computer Science, University of Tennessee, 107 Ayres Hall, Knoxville, TN, 37996-1301, berry@cs.utk.edu

Abstract

Computer models are used in landscape ecology to simulate the effects of human land-use decisions on the environment. Many socioeconomic as well as ecological factors must be considered, requiring the integration of spatially explicit multidisciplinary data. The **Land-Use Change Analysis System** or **LUCAS** has been developed to study the effects of land-use on landscape structure in such areas as the Little Tennessee River Basin in western North Carolina and the Olympic Peninsula of Washington state. These effects include land-cover change and species habitat suitability. The map layers used by LUCAS are derived from remotely sensed images, census and ownership maps, topological maps, and output from econometric models. A public-domain geographic information system (GIS) is used to store, display and analyze these map layers. A parallel version of LUCAS (pLUCAS) was developed using the Parallel Virtual Machine (PVM) on a network of workstations giving a speedup factor of 10.77 with 20 nodes. A parallel model is necessary for simulations on larger domains or for maps with a much higher resolution.

Keywords: ecological simulation, Geographic Information System (GIS), habitat suitability, land-cover map, landscape-change model, land ownership, parallel implementation, socioeconomic model

⋆ This research has been supported by the Southeastern Appalachian Man and the Biosphere (SAMAB) Program under U. S. State Department Grant No. 1753-000574 and University of Washington Subcontract No. 392654.

1 Introduction

People affect the environment in which they live in many subtle and complicated ways. In order to better understand the effects of human land-use decisions on the environment, both ecological and socioeconomic factors must be considered. This multidisciplinary approach is taken by the Man and the Biosphere (MAB) project [3], which analyzes the environmental consequences of alternative land-use management scenarios in two geographically distinct regions: the Little Tennessee River Basin (LTRB) in North Carolina and the Olympic Peninsula in Washington State.

The MAB project integrates the diverse disciplines of ecology, economics, sociology, and computer science to evaluate the impacts of land-use. This integration of ideas also requires that data from the various disciplines share a compatible representation. Such forms include tabular and spatial databases, results of mathematical models, spatial models and expert opinions [3]. A *geographic information system* or GIS, such as the Geographic Resources Analysis Support System (GRASS), can be used to easily store and manipulate the spatially explicit representation of this data. The Land-Use Change Analysis System (LUCAS) is a prototype computer application specifically designed to integrate the multidisciplinary data stored in GRASS and to simulate the land-use policies prescribed by the integration model.

1.1 Terminology

Familiarity with common terms from landscape ecology and geographic information systems helps to better understand the LUCAS model. The spatially explicit multidisciplinary data is stored in *raster maps*. Raster maps, also known as *data layers*, are matrices of integers. Each entry in the matrix is called a *grid cell* or *pixel* and corresponds to the value of an attribute, such as elevation, for a particular parcel of land. Contiguous pixels with the same value are called *patches* or *clusters*. Many other geographic terms are discussed in Section 2 in which GRASS is introduced.

The concept of *transition* is central to the LUCAS model. A transition is a change, usually in land cover, from a given state to a new state as dictated by the land-use scenario. *Transition probabilities* are the probabilities of a transition occurring for a particular grid cell. The generation of transition probabilities is discussed in Section 3. A *scenario* is a predefined land management policy. LUCAS defines many scenarios for each watershed it simulates. Two *watersheds*, distinct geographic regions, are currently supported: the LTRB and the Hoh River on the Olympic Peninsula.

As discussed in Section 3, a stochastic simulation requires multiple *replicates*, repeated trials, to statistically verify the simulation. Usually many *time steps*, five year intervals, are simulated for each replicate to model change over an extended period of time.

1.2 Sample Scenario and Validation

In LUCAS, scenarios describe particular land-use policies to be simulated. As an example, suppose that a natural resource manager in the LTRB would like to determine the impact of not logging any trees for 25 years on the habitat of the Wood Thrush (*Hylocichla mustelina*). The scenario is formally defined to use the historical transition probabilities based on existing map layers from 1975, 1980 and 1986 along with the restriction that once a grid cell of land is forested, it will remain forested. For example, the land manager may run LUCAS with 10 replicates for 5 time steps each to simulate the change over 25 years. She can quickly examine the graphical statistics plotted on the screen or more carefully analyze the statistics saved to a SAS [30] file. Other scenarios with different constraints can be investigated and their results compared. In this way the investigator can better understand the effects of potential land-use decisions. The simulation and resulting statistics produced by LUCAS are discussed in Section 3.

To validate the LUCAS model, historical data are compared against the simulated data. Starting with the oldest existing map, the period of time up to the year for which the newest map exists must be simulated. The degree to which the statistics for the simulated and historical land cover layers agree determines the accuracy of the model for this period. Section 5 addresses this validation procedure and explains the results.

1.3 Development and Parallelization

As a solution to the problem of modeling landscape change, LUCAS was implemented as an "object-oriented" C++ application to promote modularity. It was envisioned that different or additional software modules could easily be added to existing code as the needs of investigators changed. Section 5 discusses this future expansion and Section 4 examines the details of the modular implementation along with the development of a parallel version of LUCAS. The creation of a distributed version of LUCAS, Parallel LUCAS (pLUCAS), was motivated by the ever more demanding and time consuming calculations necessary to extend the LUCAS model to other larger regions. pLUCAS utilizes the Parallel Virtual Machine (PVM) [12], an applications which allows a network of arbitrary workstations to behave as a single computational unit.

This "virtual machine" can then simulate many land-use scenarios in a fraction of the time required on a single processor.

2 Geographic Information System

The Geographic Resources Analysis Support System (GRASS) [37], developed by the United States Army Construction Engineering Research Laboratory (USACERL), was selected to be the Geographic Information System (GIS) for LUCAS. Like many GISs, GRASS provides the tools necessary to manipulate and display spatially explicit data and presents a standard format for data representation.

2.1 GRASS and LUCAS

GRASS was chosen to be the GIS used in the development of LUCAS for several reasons:

First, GRASS is able to import a variety of data types. The remote satellite imagery used to generate the GRASS vegetation map layers used in LUCAS was purchased from EOSAT, a company which distributes Thematic Mapping (TM) and Multi-spectral Scanner (MSS) satellite data. These images were interpreted for land cover using a software package called ERDAS [7]. This format could readily be converted to a native GRASS format with the GRASS utility `r.in.erdas`. Other geographical map layers, such as slope, aspect, and elevation, were originally in Digital Elevation Model (DEM) format and were imported via `m.dem.extract`. The land ownership data was in ARC/INFO [1] format and was imported using `v.in.arc`. Finally, the population density maps were originally in TIGER/line™ format and were subsequently converted to ARC/INFO and imported into GRASS in the same manner. The ecologist preparing these maps also used many of GRASS's map manipulation tools to create data layers, such as the distance from each point to the nearest road, which required a simple distance calculation for each grid cell.

Second, the source code for GRASS is provided in the software distribution. In developing LUCAS, there were many instances in which features or techniques were not fully documented which made the availability of the GRASS source code invaluable. A greater understanding of the functionality of GRASS was thus also gained. The source code also enabled a few GRASS routines to be

[1] ARC/INFO is a registered trademark of ESRI, Environmental Systems Research Institute, Inc. 380 New York Street, Redlands, CA 92373.

adapted and integrated into LUCAS, avoiding the unnecessary rewriting of programs. Another benefit of the availability of the source code is the relative ease of software portability.

The final reason GRASS was selected was one of sheer economics. Because GRASS was developed with governmental funds, it and its source code are freely available from USACERL. If a commercial package such as ARC/INFO were selected, the source code would not necessarily be included in the distribution and licensing fees would effectively make each copy of LUCAS cost-prohibitive.

GRASS is not a perfect tool, however. As a non-commercial package, many bugs persist in the code. For example, the GRASS X-windows monitor often functions properly under SunOS 4.1.3, but not under Solaris 2.3. Some of the features of GRASS are not well documented, which again made the availability of the source code invaluable. The GRASS environment works well for someone with knowledge of UNIX and programming but would be rather challenging for an ecologist or land manager without such skills. In spite of its many foibles, GRASS is a useful environment in which to work and program.

LUCAS was built on top of the GRASS libraries, so switching to another GIS would be difficult, but not impossible. Replacing the map access routines would be straight forward, but the graphical display code would need to be completely rewritten. Naturally the algorithms and methods comprising the essence of the LUCAS simulations would remain the same, regardless of the GIS being used.

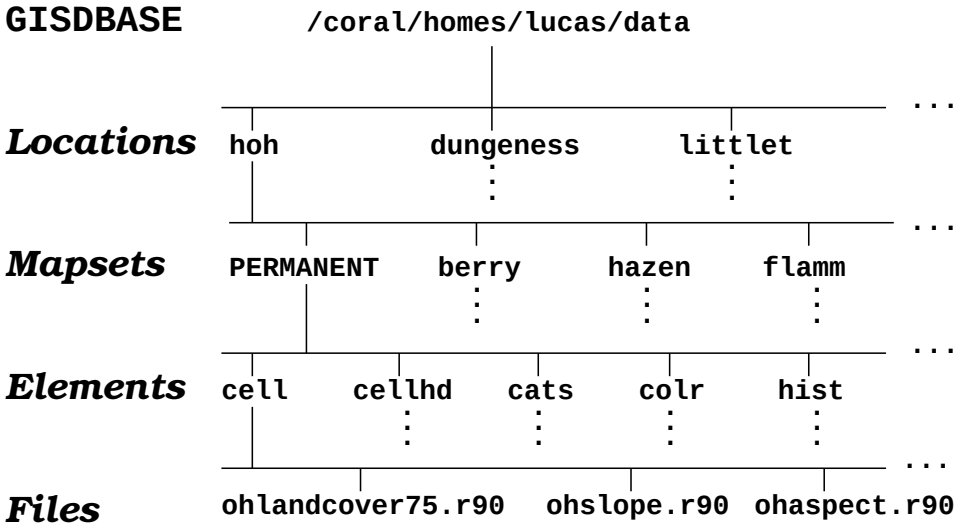


Fig. 1. Example of the GRASS storage hierarchy

The GRASS data hierarchy is illustrated in Figure 1. The root of the GRASS file system is defined by the UNIX environment variable `GISDBASE` which is the path name of the GRASS database. Specifying which database to use is therefore trivial, making the parallel implementation much easier (see Section 4.2). Within the database there are many GRASS *locations*, which correspond to watersheds in LUCAS. A location is an independent set of data, usually associated with a distinct geographic region. Within a given location there are many *mapsets*. Mapsets are private repositories for maps and their supporting files. Users may write only to mapsets which they own, but they may access information in any mapsets which they have permission to read. Within each location there is a special mapset called `PERMANENT` which holds specific information about that location. All users must be able to access `PERMANENT`, therefore this is the mapset in which all of the original LUCAS maps are stored.

Each mapset contains a series of files and directories (called *elements*) shown in Table 1 which correspond to map components. For a particular map, each element will contain a file with the same name as that map.

Table 1
Table of selected GRASS mapset elements

Element	Function
`cell`	Binary raster file
`cellhd`	Header files for raster maps
`cats`	Category value information for raster maps
`colr`	Color table for raster maps
`colr2`	Secondary color table for raster maps
`cell_misc`	Miscellaneous raster map support files
`hist`	History information for rater maps
`dig`	Binary vector data
`dig_ascii`	ASCII vectory data
`dig_att`	Vector attribute support
`dig_cats`	Vector category label support
`dig_plus`	Vector topology support
`reg`	Digitizer point registration
`windows`	Predefined regions
`WIND`	Current region

All of the input maps used in the LUCAS simulations are *raster files*; discrete grids (or matrices) of numeric values each corresponding to a square parcel of land called a grid cell. In GRASS these raster maps are accessed by row.

Any row may be read independently, but each row must be written sequentially because each row is usually compressed using a row-oriented compression scheme, *run-length encoding* (RLE) [27]. For raster maps, the *cell* directory is where the actual binary map is stored. Maps can be stored in an uncompressed (32-bit) format for each grid cell (pixel) or in the RLE compressed format.

The GRASS *header* contains such information as the map resolution, its geographic coordinates, and the number of rows and columns of grid cells contained in the map. It is assumed that all of the maps used in LUCAS will have identical headers so that the maps can be assured of being properly overlayed. If the maps do not have the same region, the map may be resampled within the correct region using the GRASS utility `r.resample`.

The *category values* are the names of the categories assigned to each numeric value found in a map. For example, in the Dungeness Watershed land cover map, the value (1) corresponds to coniferous forest; (2) corresponds to deciduous or mixed forest; (3) to grassy, brushy or new growth regions; (4) to unvegatated areas; (5) to water and (6) to snow, ice or clouds. The value of (0) is reserved to indicate that no data is present for that grid cell.

The *color table* of each map assigns a specific color to each category value. If these are not specified, GRASS will assign arbitrary colors automatically.

The *region* contains the geographic boundaries of the area being examined by GRASS as well as the sampling resolution for this area. There is one current region which must be specified (via `g.region`) before running LUCAS. For information on installing GRASS for use with LUCAS see Appendix B and [13].

2.3 Programming Interface

USACERL provides a series of full–featured C libraries with GRASS which are carefully documented in the *GRASS Programmer's Manual* [31]. There are numerous major libraries with several smaller support libraries as shown in Table 2. The existence of these GRASS libraries meant that many low-level I/O, graphics and map processing routines already existed and could be incorporated into LUCAS. The GIS, Raster Graphics, Display Graphics, and D libraries were the only ones used in the development of LUCAS.

GRASS programs can specify information about the current state of GRASS by using environment variables. These are separate from UNIX environment variables and are actually stored in a file, usually `$HOME/.grassrc`, where `$HOME` designates the user's default home directory. This technique is used by the LUCAS graphical user interface (GUI) to communicate with the simulation

Table 2
C libraries provided by GRASS

Library Name	Description
Binary Tree	Simple contiguous-memory binary tree routines
Bitmap	Basic support for the creation and manipulation of two dimensional bitmap arrays
Convert Coordinates	Converts from one cartographic coordinate system to another
D	Wrapper functions for some of the Display library routines
DLG	Reads and writes U. S. Geological Survey DLG-3 format maps
Digitizer	Allows control of external digitizer
Display Graphics	Graphics display routines; higher level than Raster Graphics library
GIS	General-purpose GRASS routines
Icon	Simple manipulation of bitmapped icons
Imagery	Integrated image processing routines
Linked List	Generic linked list memory manager; more efficient than `malloc()` and `free()` implementations
Lock	Provision for advisory locking mechanism
Projection	Cartographic projection filters
Raster Graphics	Video graphics display utilities
Rowio	Routines to access multiples rows of large input maps
Segment	Utilities to allow for random access of large maps
Vask	*Virtual ask*; allows for Curses-like text I/O
Vector	Routines to process binary vector files

program.

GRASS is written in traditional Kernighan and Ritchie C (K&R C) [15] for portability, but LUCAS is written in C++, which required a few adjustments to be made to the programming interface. C++ is a strongly typed language and therefore all functions must be prototyped before they are used. K&R C, on the other hand, does not impose this requirement and consequently GRASS header files are devoid of type information. Therefore, before any GRASS routines could be used in LUCAS, they first had to be prototyped. This is another reason why the availability of GRASS's source code was so crucial.

GRASS also allows for hardware-independent graphical displays, called *monitors*, using MIT's X-windows, Sun's OpenWindows, Silicon Graphics' IRIS, Tektronix 4105, raster file or other formats. GRASS monitors are used to collect user input regarding which impacts should be examined and to graphically display resulting maps. The GRASS library routines allow for the complex displaying of spatially explicit data without additional laborious programming.

3 Functional Design of LUCAS

LUCAS was designed to model landscape change stochastically using historical data as a basis for prediction. Its modular design is intended to increase the program's flexibility and to allow for future modifications or additions.

3.1 Stochastic Modeling

The econometric model used in LUCAS is a dynamic, *stochastic* model which, by definition, has at least one random variable, namely land cover, and deals explicitly with time-variable interaction. A stochastic simulation [2] uses a statistical sampling of multiple replicates, repeated simulations, of the same model. Simulations are used when the systems being modeled are too complex to be described by a set of mathematical equations for which an analytic solution is attainable. Simulation is, however, an imprecise technique and provides only statistical estimates and not exact results. The LUCAS computer simulation serves as a tool to help evaluate land-use management policies before actually implementing them on the real system.

An effective model need only have a high *correlation* between its prediction and the actual outcome in the real system, not necessarily approximate the real system, so statistical results are sufficient to understand the results of the simulated model [29]. Therefore, the primary output produced by LUCAS are statistics collected during the simulation which can be viewed as a graph or analyzed with a statistical package, such as SAS [30]. Figure 2 outlines the modular model implemented in LUCAS; the validation of which is discussed in Section 5.1. Aside from predicting future behavior, models also allow investigators to explore the nature of potential scenarios. Each module of the LUCAS model is described in detail in the following sections.

3.2 Socioeconomic Model Module and TPM

LUCAS takes a multidisciplinary approach to simulate change in a landscape. Ecologists and economists used knowledge from both disciplines to develop a land management simulation. Many discrete and continuous ecological and sociological variables were used empirically in calculating the probability of change in land cover (a discrete variable): land-cover type (vegetation), slope, aspect, elevation, land ownership, population density, distance to nearest road,

[2] Such simulations are also sometimes referred to as *Monte Carlo* simulations because of their use of random variables [23,29].

distance to nearest economic market center (town), and the age of trees. For an analysis of the influence of these economic and environmental factors on landscape change see [36]. Each variable corresponds to a spatially explicit map layer stored in the GIS (see Section 2). A vector of all of these values for a given grid cell is called the *landscape condition label* [8,9]. An example landscape condition label (LCL) is shown in Table 3.

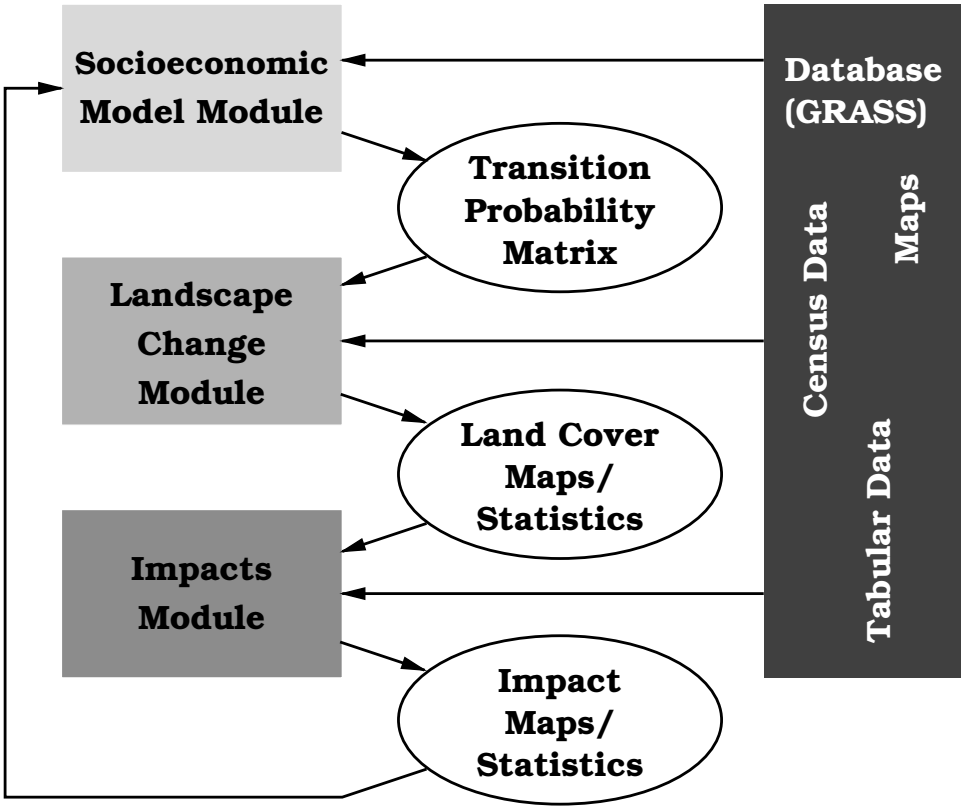


Fig. 2. Relationship among LUCAS modules

Table 3
Example Landscape Condition Label in the Hoh Watershed on the Olympic Peninsula

ℓ	x_ℓ	Meaning	Attribute
1	2	Public Lands	Land ownership
2	73	73 years old	Tree age (Olympic Peninsula only)
3	4	Unvegetated	Land cover (vegetation)
4	19	19° incline	Slope
5	1	True	Steep slope ($>$ 17° incline; Olympics only)
6	1531	1531 meters	Elevation
7	1089	1089 meters	Distance along roads to nearest market center
8	21	1890 meters	Distance to nearest road

Each element of the LCL $\vec{x} = (x_1, x_2, \ldots, x_8)^T$ is used to determine the probability of change using the multinomial logit equation found in Equation (1)

[39,38,3].

$$\Pr(i \rightarrow j) = \frac{\exp(\alpha_{i,j} + \vec{z}^T \vec{\beta}_j)}{1 + \sum_{k \neq i}^{n} \exp(\alpha_{i,k} + \vec{z}^T \vec{\beta}_k)}, \quad (1)$$

where n is the number of cover types, $\vec{z}$ is a 5×1 column vector composed of elements $x_4, \ldots, x_8$ of the LCL $\vec{x}$ in Table 3, $\vec{\beta}_j = (\beta_{1,j}, \beta_{2,j}, \ldots, \beta_{5,j})^T$ is a vector of logit coefficients, $\alpha_{i,j}$ is a scalar intercept, and $\Pr(i \rightarrow j)$ is the probability of unvegetated land cover remaining the same ($i = x_3 = 4 = j$) at time $t + 1$ or changing to another cover class (i.e., $j = 1, 2, 3$). The land ownership (x_1) determines which table of logit coefficients should be used and the tree age (x_2), used only for coniferous forest cover types, determines if the trees have aged sufficiently to be harvested, i.e., change to another cover type. The *null-transition* or probability of no land cover change is defined by Equation 2.

$$\Pr(i \rightarrow i) = \frac{1}{1 + \sum_{k \neq i}^{n} \exp(\alpha_{i,k} + \vec{z}^T \vec{\beta}_k)}, \quad (2)$$

where the symbols have the same meaning as in Equation (1). Example vectors of coefficients for the Hoh Watershed are shown in Table 4.

Table 4
Example Multinomial Logit Equation Coefficients for unvegetated land cover in the Hoh Watershed on the Olympic Peninsula

Pixel Attribute	**Transition to**		
	Conifer	**Deciduous/Mixed**	**Grassy/Brushy**
intercept	$\alpha_{1,1} = -3.14560$	$\alpha_{1,2} = -3.22460$	$\alpha_{1,3} = -1.81020$
slope	$\beta_{1,1} = 0.03760$	$\beta_{1,2} = 0.00496$	$\beta_{1,3} = 0.03180$
steep slope	$\beta_{2,1} = -0.51363$	$\beta_{2,2} = 0.47170$	$\beta_{2,3} = -1.18990$
elevation	$\beta_{3,1} = -0.00018$	$\beta_{3,2} = -0.00237$	$\beta_{3,3} = 0.00488$
distance to town	$\beta_{4,1} = 0.00245$	$\beta_{4,2} = 0.00490$	$\beta_{4,3} = 0.03040$
distance to road	$\beta_{5,1} = -3.14560$	$\beta_{5,2} = -3.22460$	$\beta_{5,3} = -1.80200$

The multinomial logit coefficients and intercept values were calculated empirically by Wear *et al.* [39] from existing historical data stored in the GRASS database (see Figure 2). The table of all probabilities generated by applying Equation (1) to all cover types is called the *transition probability matrix* (TPM), an example of which can be found in Table 5. If the TPM in Table 5 were used, for example, a random number from the closed interval $[0, 1]$ less

than 0.5839 would change the land cover to grass/brushy, otherwise the land cover would remain unvegetated. For a discussion of logistic regression and a basis for Equation (1) see [34].

Table 5
Example Transition Probability Matrix based on the example multinomial logit coefficients.

From Unvegetated	Changing to	Probability
$4 \rightarrow 1$	Coniferous	$< 1 \times 10^{-30} \approx 0$
$4 \rightarrow 2$	Deciduous/Mixed	$< 1 \times 10^{-30} \approx 0$
$4 \rightarrow 3$	Grassy/Brushy	0.5839
$4 \rightarrow 4$	Unvegetated	0.4161

3.3 Landscape Change Module

The Landscape Change Module in Figure 2 is the heart of the LUCAS software.

It takes the multinomial logit coefficients generated in Socioeconomic Model Module, implements the actual landscape change, and produces new land cover maps and statistics as output. The first step in designing LUCAS was to develop the method to simulate one time step, a five year period, of landscape change over multiple replicates. Figure 3 shows the general algorithm used by LUCAS to simulate a given scenario.

3.3.1 Pixel-Based Transitions

Two types of transitions are simulated by LUCAS: grid cell (or pixel-based) and patch-based. The determination of the pixel-based landscape transitions is relatively trivial because each grid cell changes independently. The transition probabilities from the initial cover type to all other cover types are calculated using Equation (1) and the value of the landscape condition label of a grid cell. A pseudorandom number is then drawn from a uniform distribution between zero and one (see Section 3.3.3). This number, in turn, determines the new land cover type for this grid cell via the calculated probabilities.

3.3.2 Patch-Based Transitions

Patch-based transitions are considerably more difficult because of the task of patch identification. A patch (or cluster) is a group of contiguous grid cells with identical landscape condition labels. The North-East-West-South (NEWS)

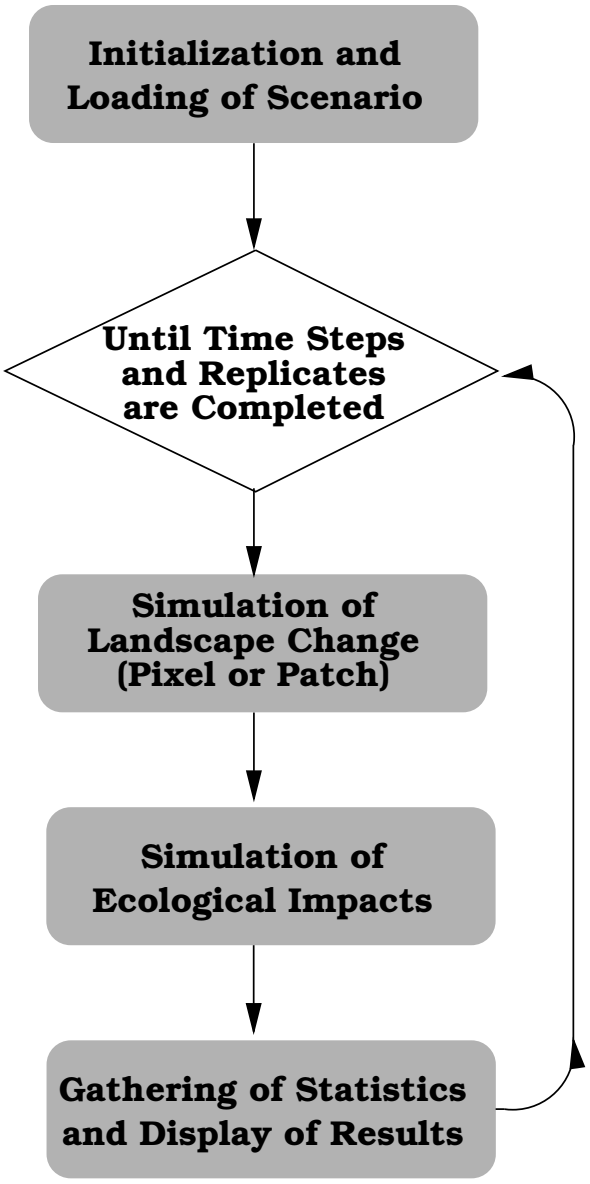


Fig. 3. General LUCAS algorithm

Neighbor Rule [2] was used to define adjacency. The patch-identification algorithm in Figure 4 is used by the GRASS program `r.clump` and was incorporated into LUCAS because of the availability of its source code and its relatively meager memory usage. It is a two-pass algorithm which requires only the current and previous rows to be in memory at one time. The transition is calculated in the same manner as in the pixel-based model when a new patch is created and its value is passed on to the other members of the current patch.

```
begin
   for  row := 1 to NumberOfRows do begin
      for  column := 1 to NumberOfColumns do begin
         if this cell <> cell to left and above then begin
            Start a new patch and determine new value after transition
         end;
         else if this cell = cell to left, not above then begin
            Continue the patch to the left
         end;
         else if this cell = cell above, not left then begin
            Continue the patch in the row above
         end;
         else if this cell = cell above and left then begin
            if number of patch above = number of patch to the left
            then begin
               Continue the patch
            end;
            else begin
               Unify the two patches
            end;
         end;
   end;
   if second pass then begin
      Write out row of new land cover
   end;
end.
```

Fig. 4. Patch-based landscape transition algorithm

3.3.3 Random Number Generation

To implement the stochastic model mentioned in Section 3.1, uniformly distributed pseudorandom numbers are needed to accurately predict a new land-cover value for a cell based on the the transition probabilities. Unfortunately, most vendor-provided random number generators (RNGs) are not sufficient for such modeling [26,4]. "Good" random numbers should be uniformly distributed, statistically independent and reproducible [29,24]. Truly random numbers are not reproducible without storing them, so pseudorandom numbers are more suitable for simulations. Repeating a sequence of random numbers is desirable to debug simulations and two policies can be more accurately compared statistically if both use the same "random" sequence [4].

According to Knuth and others [4,17,26,29], a very standard technique for generating pseudorandom numbers is the use of linear congruential generators (LCG) which were first introduced by Lehmer in 1949 [18]. LCGs compute the ith integer in a pseudorandom sequence by the recursion found in Equation (3).

$$x_i = (ax_{i-1} + c) \bmod m, \tag{3}$$

where multiplier a, increment c, and modulus m determine the statistical quality of the generator. This generator meets many of the criteria established for a "good" RNG including the fact that it requires very little memory to implement. The problem with LCGs is that their *period* is limited to m [29]. All pseudorandom number generators suffer because the sequences they generate are periodic, i.e., there exists an integer t such that $y_{n+t} = y_n$ for all $n \geq 0$, where y_n is the nth number in a random sequence [24]. Without using costly multiple-precision arithmetic, the cycle generated by LCGs is bounded by the length of the machine word. Thus on a machine with 32-bit integer arithmetic (31-bit with the sign bit), this means that a maximum of $2^{31} - 1 =$ 2,147,483,647 numbers can be generated before the sequence repeats itself. Two billion may seem sufficient for a single simulation, but if the transitions for large maps with millions of cells are to be calculated for many time steps and replicates, this period could be exhausted.

Park and Miller propose that Lehmer's LCG with $a = 7^5 = 16807$, $c = 0$ and $m = 2^{31} - 1$, the Mersenne prime, be adopted as a minimal standard random number generator because it has been exhaustively tested and its characteristics are well understood [26].

Another revolutionary generator using a feedback shift register (FSR) was introduced in 1965 by Tausworthe [32]. It could generate arbitrarily long sequences of random numbers without the multidimensional non-uniformities found in LCGs. The kth bit, a_k, of a random bit sequence is defined to be

$$a_k = c_1 a_{k-1} + c_2 a_{k-2} + \ldots + c_{p-1} a_{k-p+1} + a_{k-p} \pmod 2 \tag{4}$$

and has its maximum possible cycle length if and only if the polynomial $1 + c_1 x + c_2 x^2 + \ldots + c_{p-1} x^{p-1} + x^p$ is primitive over GF[2].[3] However, Toothill *et al.* discovered negative results running a statistical test of the Tausworthe FSR method in 1971 [33]. Two years later Lewis and Payne refined Tausworthe's generator to create a generalized FSR (GFSR) [19]. Because this algorithm uses base two, the addition operation without carry is the same as the binary *exclusive OR* (XOR, denoted $\oplus$). Thus Equation (4) for integers (groups of bits) is equivalent to

$$x_k = c_1 x_{k-1} \oplus c_2 x_{k-2} \oplus \ldots \oplus c_{p-1} x_{k-p+1} \oplus x_{k-p}. \tag{5}$$

Using primitive trinomials, $1 + x^q + x^p$, $p > q$, Equation (5) reduces to

$$x_k = x_{k-q} \oplus x_{k-p}. \tag{6}$$

[3] For more information about Galois Fields, see [10].

This requires only one XOR operation and some address calculations for each integer generated by the GFSR and therefore is just as fast, if not faster than LCGs. It also requires careful initialization of the vector before generating new random numbers. Kirkpatrick and Stoll presented a specific implementation of the generator described by Equation (6) (denoted **R250**) using $q = 103$ and $p = 250$ [16]. In 1994 Carter created a C++ class library [5] of **R250**[4] which uses Park and Miller's minimal standard LCG to initialize the R250's bit vector. LUCAS uses Carter's C++ implementation of **R250** as its RNG. It has an extremely long period (2^{249}) and is faster than an LCG. It does require more memory, i.e., the storage of 250 integers, but this cost is negligible.

3.3.4 Statistics

Once the map of new land cover has been generated, the ecologist or land manager can use the results to determine the impact of the policy defined in the Socioeconomic Model Module. As stated in Section 3.1, statistics are the only true metric for analyzing a stochastic simulation. They also provide a convenient method for understanding the impact of the particular land management policy or scenario. The statistics in Table 6 are collected by LUCAS for each time step.

The stack-based, non-recursive algorithm in Figure 5 is used to identify patches for statistics purposes. This method differs from the one used for patch-based transitions, because the statistics algorithm requires the entire map layer to be in memory at one time, which is not feasible for the composite map used in Section 3.3.2. This routine was originally developed by Gardner [11] and later modified by Minser [2].

[4] The source code is available on the World Wide Web at URL `http://taygeta.oc.nps.navy.mil/random.html`.

[5] This is the corrected patch perimeter/area metric: $(0.282 \times \text{perimeter}/\sqrt{\text{area}})$ [1].

Table 6
Statistics collected by LUCAS

	Statistic
Pixel Statistics	Proportion of landscape in each cover type Area (ha) of landscape in each cover type Edge:area ratio for each cover type Amount of edge (km) for each pair of cover types Total edge (km) in the whole landscape
Patch Statistics	Number of patches Mean patch size Standard deviation of patch size Size of largest patch Cumulative frequency distribution of number of patches by size Mean patch shape (normalized shape index) [5]

```
begin
   for row := 1 to NumberOfRows do begin
      for column := 1 to NumberOfColumns do begin
         if cell is type being analyzed then begin
            Push cell onto stack of cells in patch
            repeat
               Pop cell off top of patch stack
               Mark cell as visited
               Designate cell as center of a new patch
               if neighbors of cell are same type then begin
                  Push neighbors onto patch stack
                  Increment patch size, perimeter values
               end;
            until patch stack is empty
         end;
      end;
   end;
end.
```

Fig. 5. Patch-based statistics collection algorithm for a given land cover type

3.4 Impacts Module

The land cover map produced by the Landscape Change Module (see Section 3.3) is analyzed by the Impacts Module. This module may determine the effect the changed landscape has on species habitats, water quality, or other environmental impacts. Currently LUCAS is designed to perform only species habitat suitability analyses. Dr. Scott M. Pearson at Mars Hill College (Mars Hill, NC) created a list of species and habitat identification algorithms for each of the watersheds currently simulated by LUCAS in Table 7 [28]. The output from this module is a binary map; either a grid cell is suitable for a species or it is not. The statistics in Table 6 are again collected for each impact map.

Table 7
Species habitats impacts modeled by LUCAS

Watershed	**Species**	
Little Tennessee River Basin	Catawba Rhododendron	*Rhododendron catawbiense*
	Cranefly Orchid	*Tipularia discolor*
	European Starling	*Sturnus vulgaris*
	Mountain Dusky Salamander	*Desmognathus ochrophaeus*
	Northern Flying Squirrel	*Glaucomys sabrinus*
	Princess Tree	*Paulownia tomentosa*
	Southeastern Shrew	*Sorex longirostris*
	Wood Thrush	*Hylocichla mustelina*
Olympic Peninsula: Hoh and Dungeness	Cascade Oregon Grape	*Berberis nervosa*
	Heather Vole	*Phenacomys intermedius*
	Honeysuckle	*Lonicera ciliosa*
	Horsetail	*Equisetum telmateia*
	Licorice Fern	*Polypodium glycyrhiza*
	Mountain Alder	*Alnus sinuata*
	Mountain Huckleberry	*Vaccinium membranaceum*
	Red Squirrel	*Tamiasciurus hudsonicus*
	Twinflower	*Linnaea borealis*

3.5 Parallel/Distributed Design

The parallel version of LUCAS (pLUCAS) uses the same model as the serial version, but is designed to run on a network of high-end workstations via the Parallel Virtual Machine (PVM) software package [22]. This implementation takes a very *coarse-grained* approach to parallelism. For accuracy purposes, the stochastic simulation of LUCAS requires multiple independent replicates which can easily be run independently on separate machines. The difficulty arises in managing the interprocessor communication, the scheduling of tasks

and the handling of data.

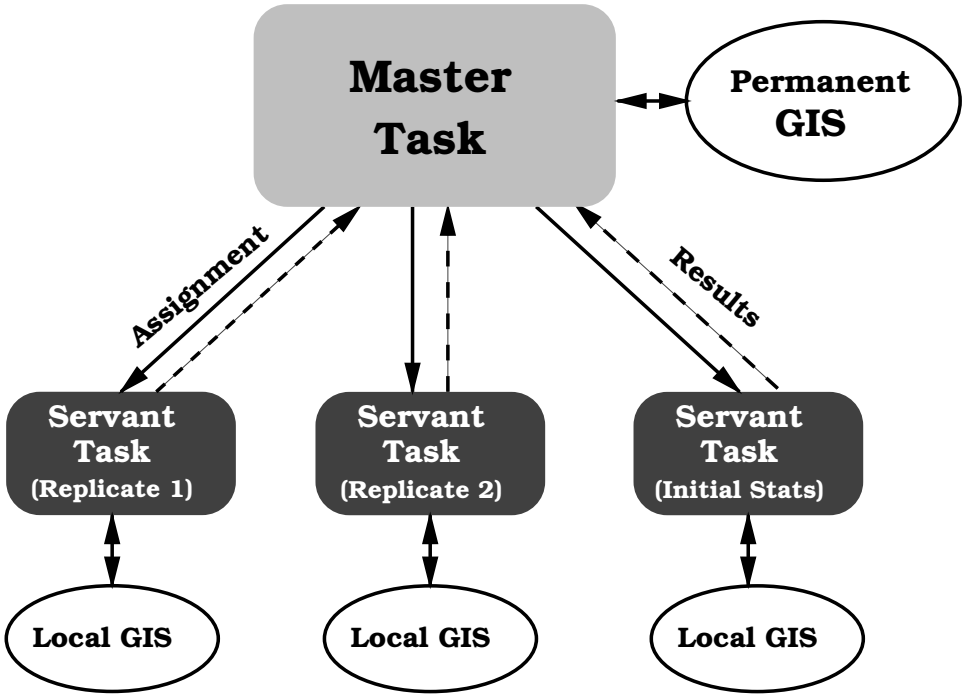


Fig. 6. Relationship between *master* and *servant* tasks in pLUCAS

To simplify the problem, centralized task management is used by pLUCAS (see Figure 6). One task[6] on the virtual machine is known as the *master* while all other tasks are *servants*. The servant tasks are almost identical to the serial version of LUCAS with only a few minor modifications. Each servant accesses its own copy of the GIS on its local disk because of the intense I/O required. The master, on the other hand, handles all task management and works directly with the permanent GIS. The master manages a FIFO job queue which consists of all of the independent replicates of one or multiple simulations. In the serial version of LUCAS, the statistics for the original maps are collected at the beginning of each run, but the parallel version has a single job dedicated to that purpose as shown in Figure 6. Two other queues are maintained to manage available processors: the run queue and the idle queue. When a servant is available, i.e., at the head of the idle queue, it is assigned the next task on the job queue. This continues until all tasks have been dispatched. When a task has completed, the host is moved from the run queue to the idle queue. After all tasks have completed, the data from the servants are shipped back to the master to be reassembled and installed in the permanent GIS.

The pLUCAS master task uses the Park and Miller generator mentioned in

[6] A *task* in PVM is comparable to a UNIX process, i.e., an independent thread of execution on a given host.

Section 3.3.3 to create random number seeds which insure that the sequences generated by the servant tasks' **R250** random number generators are independent [6]. These seeds are passed to the servant tasks as they are spawned.

4 Serial and Distributed Implementations

As mentioned in Section 3, stochastic simulations require multiple replicates to create sufficient data for statistical analysis. These repeated simulations can be very time consuming on a single workstation if the maps are large and many scenarios are to be examined. To remedy this, a distributed version of LUCAS was created using PVM on a network of workstations. This section first discusses the details of the serial C++ implementation of LUCAS and later its evolution into a parallel version.

4.1 C++ Classes

GRASS was written in K&R C [15] which necessitated LUCAS to be written in either C or C++ to use the GRASS libraries. C++ was chosen as the programming language for LUCAS to allow for an object-oriented solution. While C++ is not a fully-object oriented language, it provides the C-compatibility and useful object-oriented extensions necessary for LUCAS.

LUCAS consists primarily of objects which contain both data and methods (procedures) which operate on that data. Figure 7 shows all of the major C++ classes found in LUCAS and their interaction.

In the following sections, each class is carefully outlined and its relationship to the other classes is described in more detail.

4.1.1 DString Class

The DString class is a general-purpose class developed to dynamically manage strings of text. C is notoriously awkward for performing string manipulations, so this class helps to make their usage more convenient. Many of the file names, GRASS variables, labels and similar information used in LUCAS are text-based, therefore almost every instance of a string is a DString object. Instantiating a DString object will automatically claim a portion of the system heap to store the string; actually an array of characters.

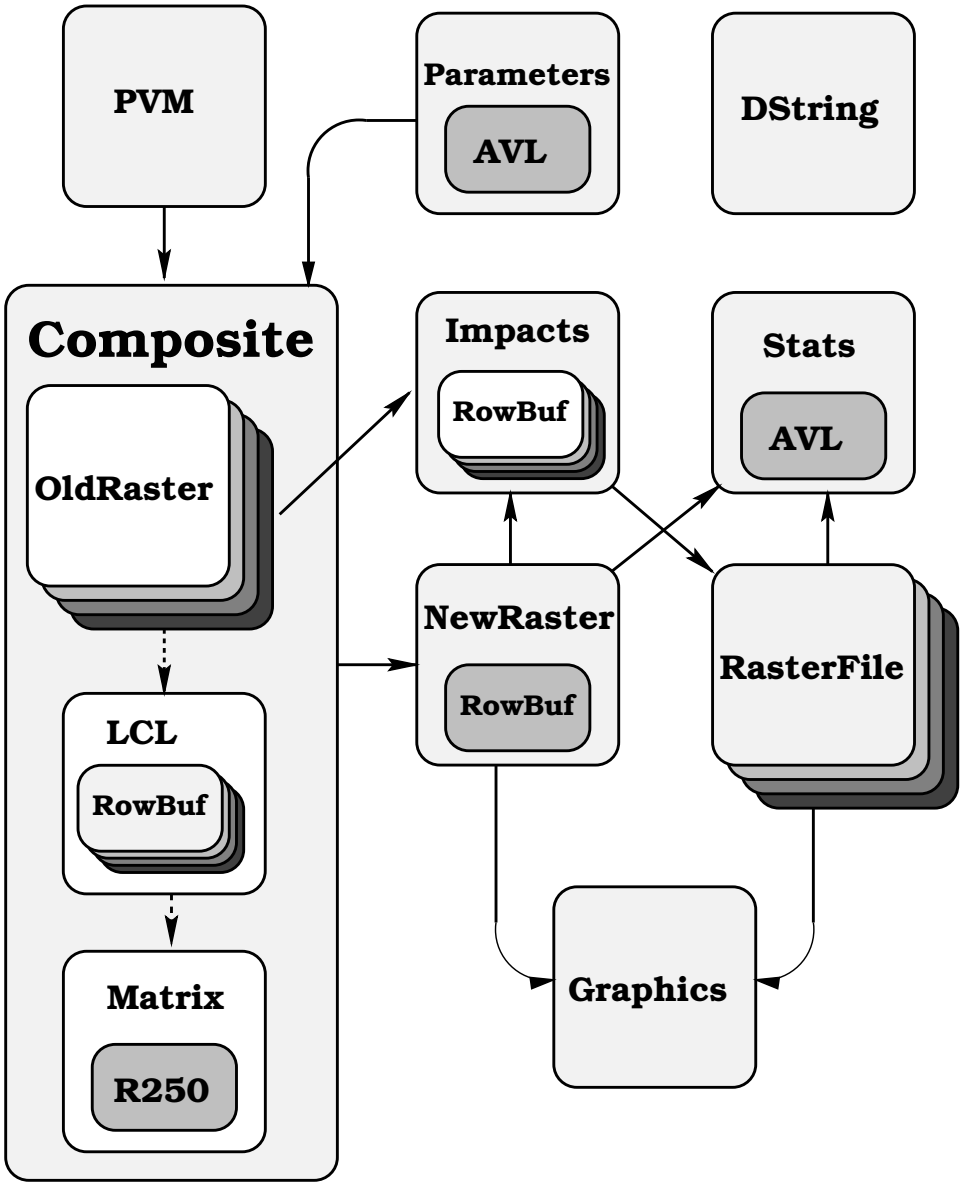


Fig. 7. Relationship among LUCAS C++ classes

Aside from automatic storage allocation and release of memory, the DString class also provides much superior operators for string manipulation. In order to concatenate strings or other objects the insertion operator `<<` is used. To compare two strings, the intuitive `==` operator is used. For example:

```
DString some_string;
DString other_string = "Something else";
int     intvalue     = 5;

some_string << "X has a value of " << intvalue << " now.";
if ( some_string == other_string ) {
  cout << "The strings are equal.\n";
}
```

The only difficulty in using such objects is that the GRASS routines require `char*` arguments, instead of DString objects, to be passed to them. This problem is overcome by creating an `operator` to return the internal character array.

4.1.2 Parameters Class

One of the few global objects in LUCAS is an instance of the Parameters class called `globals`. This object provides an interface between the GUI and the rest of LUCAS. All of the information gathered from the user in the GUI is stored in GRASS variables which are read by the Parameters object. This object stores those values and makes them available for other objects, thus creating a central repository for user-selected parameters. These values are stored in a height-balanced AVL tree, implemented by the AVL class, providing $O(\log_2 n)$ access of global parameters.

The Parameters class also reads the impacts file (Section A.2), which describes each impact module, and the scenario file (Section A.1) prescribed by the user via the GUI. All of the multinomial logit equation coefficients of the Transition Probability Matrix (TPM) are read in from the scenario file and stored in an internal matrix. This matrix is then used to construct the instance of the Composite class (Section 4.1.3) which in turn constructs its internal Matrix class object (Section 4.1.5). As each of the input map layers specified in the scenario file are read, a new OldRaster object (Section 4.1.7) is created, and an array of these objects is then used to initialize the Composite class.

4.1.3 Composite Class

The Composite class is really the heart of the LUCAS program, containing both input maps and the TPM. It creates a virtual "composite map" of land-

scape condition labels (LCLs) by forming an array of RasterFile objects. The composite is then used to calculate transitions and produce a new land cover map object.

Instances of the LandscapeConditionLabel class (Section 4.1.4) are used by the Composite object to access individual LCLs. Within each LCL, the values are extracted by the Composite object's Matrix object (Section 4.1.5) to generate the necessary transition probabilities.

4.1.4 LandscapeConditionLabel Class

The LandscapeConditionLabel class allows a grid cell on multiple maps to be viewed as a single vector (see Figure 8 and Table 3). The LandscapeConditionLabel class is actually a *virtual class*, i.e., it is never instantiated.

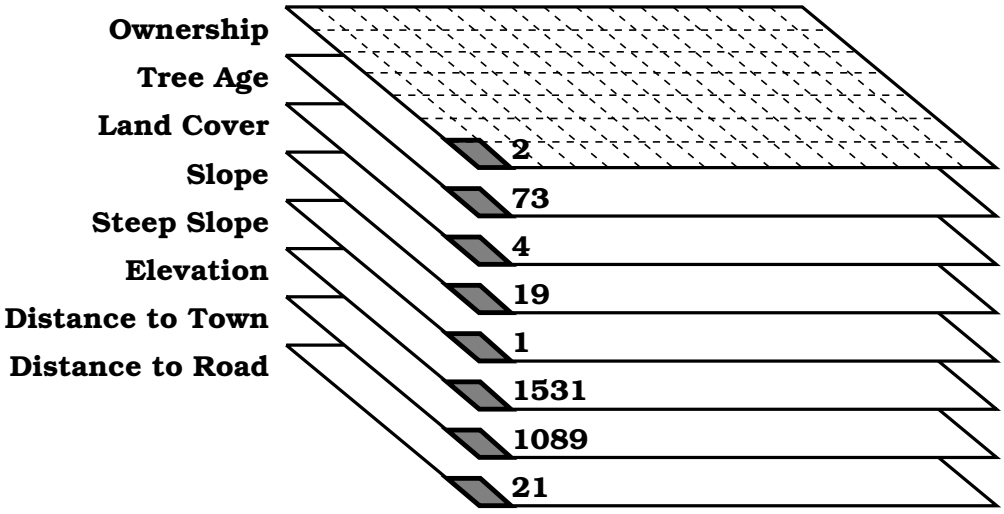


Fig. 8. Landscape Condition Label (LCL) as a vector across multiple map layers

Two other classes are derived from this base class: the LCL class and the LCLrow class. These two classes have nearly identical methods, but one is designed to operate on a single landscape condition label, and the other on a row of landscape condition labels. The two classes are `friend`s to each other to allow access to their `private` data members. This also allows for operators to improve code readability.[7] The LCL class manages its own storage, while the LCLrow class relies on its own RowBuf class to manage its memory usage. The RowBuf class provides for simple management of a single or multiple row

[7] These operators allowed the source code for the GRASS program `r.clump` to be used for the patch-based transition module (Section 3.3.2) with relatively few changes. Instead of single grid cells, the code operates on landscape condition labels with identical operators.

buffers; vectors each containing a row of grid cells. Because GRASS uses row-oriented access to raster maps, all of the map I/O for LUCAS is performed through the use of row buffers.

4.1.5 Matrix Class

The matrix of multinomial logit coefficients of the TPM are kept in the Matrix class. The primary function of this class is to calculate transition probabilities for a grid cell using the logit coefficients and the LCL corresponding to that grid cell. Once the transition probabilities have been calculated, a random integer is generated by the R250 class object (Section 4.1.6). This determines which value the new grid cell should be assigned. The method for calculating transition probabilities to determine transitions is discussed at length in Section 3.2.

4.1.6 R250 Class

This class for random number generation was created by Carter [5]. Details on the implementation of this class can be found in Section 3.3.3. The seed for the random number generator is the system clock for the serial version and a number from a random sequence in the parallel version (see Section 3.5 and Section 4.2.5).

4.1.7 RasterFile Class

The RasterFile class is another virtual base class. Two classes are derived from this class: the OldRaster class and the NewRaster class, each used to read or write GRASS raster maps. GRASS differentiates between existing raster files and new raster files. This fact along with the different methods required for the two types of maps motivated two separate classes.

The OldRaster class deals with the file management of existing raster files. It does not have any internal row buffers, rather it relies on external RowBuf objects (Section 4.1.4). This is because the existing raster objects are accessed from a number of other objects. The OldRaster class also stores other information about existing map layers, such as the GRASS header, category values and the color map (Section 2.2). These GRASS parameters are used by the Stats (Section 4.1.8), Graphics (Section 4.1.10) and NewRaster classes when analyzing the raster map.

The NewRaster class deals with newly created maps (either land cover or tree age) and manages its own RowBuf object to write out new map rows. The color map and category values are copied from an OldRaster object to insure

the new map is properly displayed.

4.1.8 Stats Class

The Stats Class is responsible for analyzing an existing OldRaster object (see Section 4.1.8) and reporting various statistics about its associated map (Section 3.3.4). These statistics are stored in two forms: A SAS-formatted [30] text file and a special format for XGraph. [8] The graphics file is actually many smaller graph files in one, with each graph file having each line preceded by a unique keyword. This way the GUI is able to scan a single file and create a menu of graphs to be displayed on the GRASS monitor.

4.1.9 Impacts Class

The function of this class is to determine the suitability of regions as habitats for various species. Other impacts, such as water quality, could be incorporated into this class. Table 7 shows which species' habitats are modeled for each watershed in LUCAS. A grid cell is deemed suitable habitat for a particular species if it meets certain criteria. For example, the habitat for the Catawba Rhododendron in the Little Tennessee River Basin is defined as balds (grassy) with an elevation of more than 900m or on the edge of forests with the same elevation. Each species has its own statically (as opposed to dynamically) defined routine because of performance considerations.

4.1.10 Graphics Class

The Graphics Class is responsible for managing the GRASS monitor window. When LUCAS is run, the monitor window in Figure 9 allows the user to choose the format for the output during the simulation. Figure 10 shows the two formats for the monitor. Each time step the monitor displays either the original land-cover map next to the permuted land cover map or it displays the permuted land cover map next to a species impact map.

4.2 Distributed Implementation

The distributed version of LUCAS, Parallel LUCAS or pLUCAS, is derived from the existing serial version. Modifications were required to integrate the

[8] XGraph version 11.3.2 was written in 1989 by David Harrison at the University of California, Berkeley.

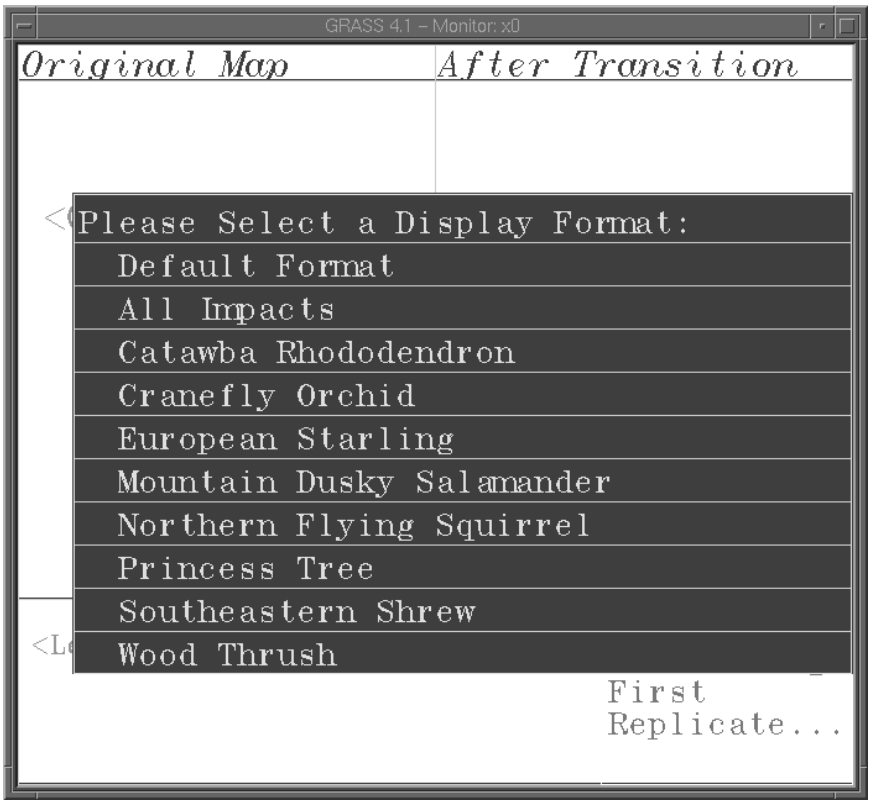


Fig. 9. GRASS monitor used to select output format

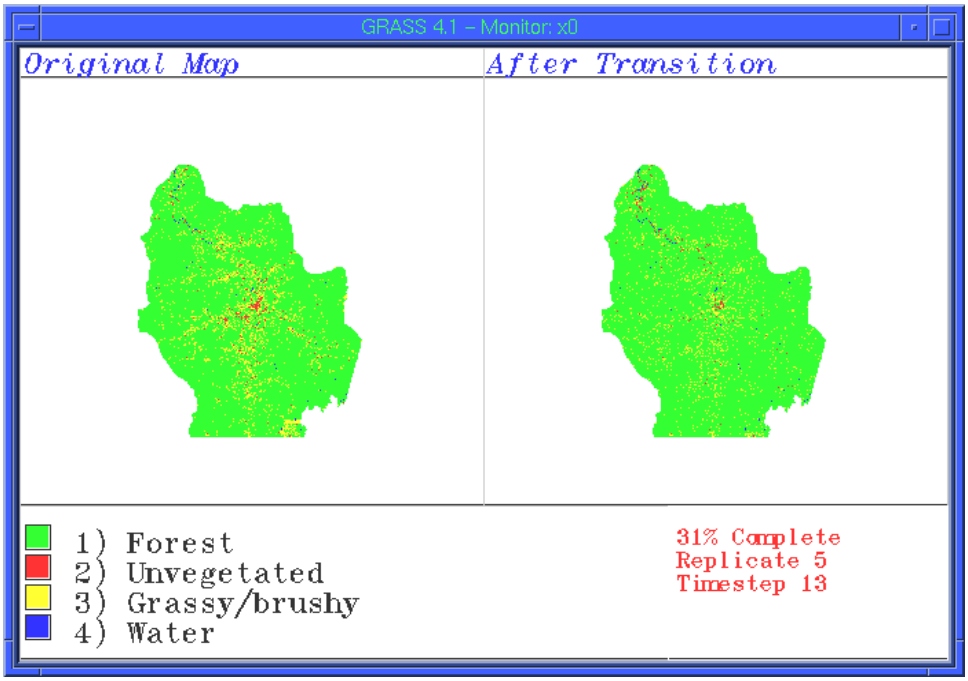
Parallel Virtual Machine[9] (PVM) software environment [22] for distributing information across a network of workstations. Measures were also taken to manage the tasks on independent machines and collect results from the hosts comprising the virtual machine.

4.2.1 PVM

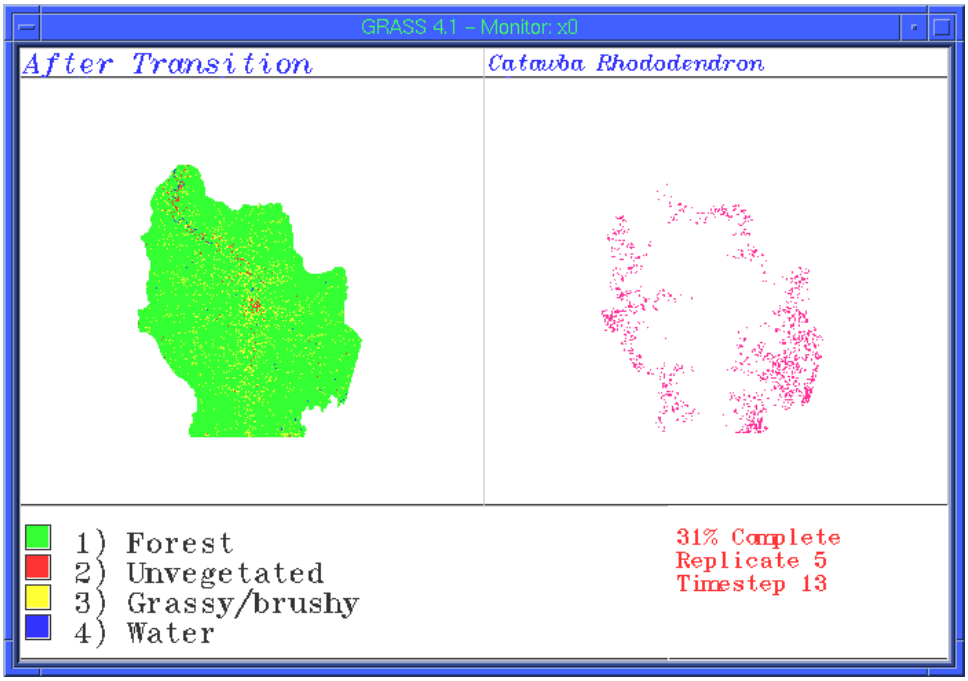
PVM allows tasks to be spawned on independent, heterogeneous workstations and permits them to communicate over a network via message passing. It handles such problems as guaranteed message delivery, the correct sequence of message delivery, and differences in internal data representations on dissimilar architectures, thus freeing the programmer from such concerns. PVM guarantees delivery by using the TCP/IP protocol suite. This also aids in portability as any machine connected to the Internet must communicate via TCP/IP.

The developer is presented with a simple programming interface to create messages of a specified type. First a message buffer is initialized, next the message components are packed according to type and finally the message is

[9] PVM version 3.3.7 was used in pLUCAS.



(a) Land cover maps before and after simulation



(b) Maps of land cover and impacts after simulation

Fig. 10. GRASS monitor showing the two possible types of display formats

sent with a designated message tag. On the receiving end, either a blocking or non-blocking receive can be used to retrieve any message or messages with a specific tag. The elements of the message are then unpacked by the receiver

according to the same types in which they were packed.

To initialize PVM, a `pvmd` daemon process must be started on each of the machines participating in the virtual machine. Once the virtual machine is running, all messages are communicated from the user process to the local `pvmd` daemon via TCP sockets, which then sends the messages across the network to the remote `pvmd` daemon via UDP sockets. The remote `pvmd` then relays the information to the remote version of the user process again via TCP. The PVM libraries [12] are required for a user program to be able to communicate with the local `pvmd` daemon.

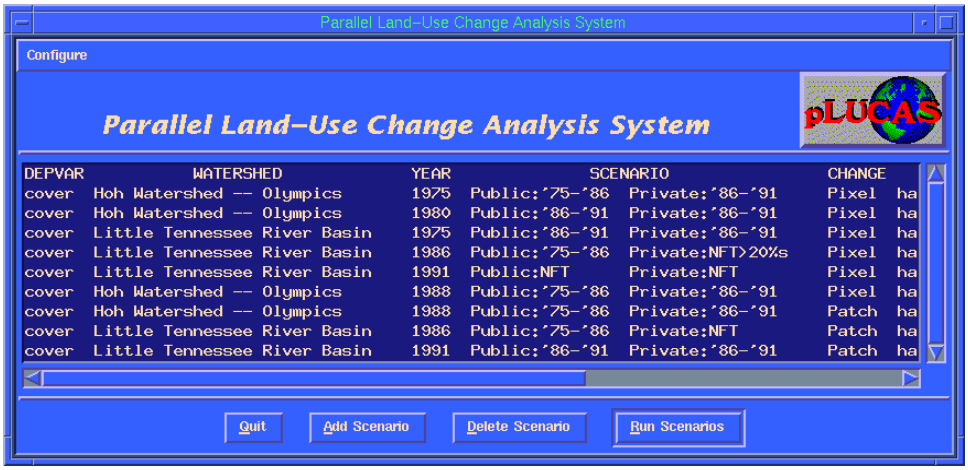
4.2.2 pLUCAS Graphical User Interface

The serial GUI was originally written by MacIntyre [21,20] in K&R C and Motif 1.2 and was later converted to ANSI C and the Motif User Interface Language (UIL) by Levy [3]. The existing user interface requires the proprietary Motif Developer's Libraries which are not available on all of the machines on which the parallel version was intended to be run. Instead, the public-domain packages TCL 7.4b3 and Tk 4.0b3 were used.

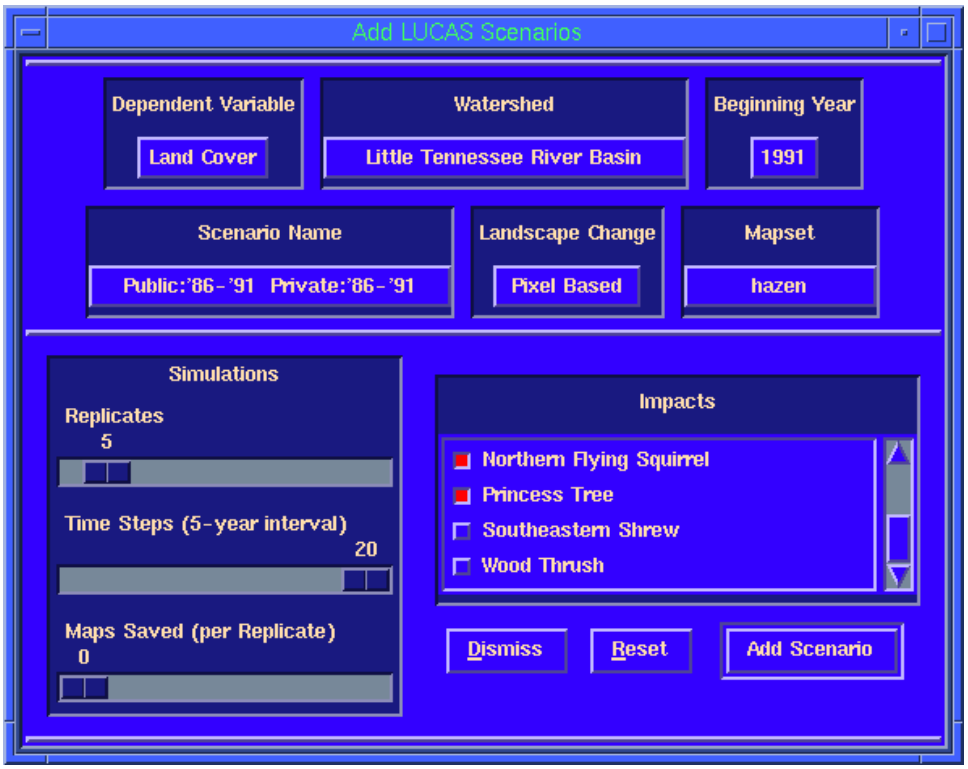
The Tool Command Language (TCL) [25] and its graphical Tool Kit (Tk) allow for the rapid development of MIT X11 Windows-based interfaces as well as provide a rich variety of scripting commands. This made the ideal choice for pLUCAS because its GUI is relatively simple and it primarily manipulates files and strings; tasks which are more difficult in C. Since TCL is a scripting language, changes made to the interface were immediate, without requiring time-consuming recompiling. Developing a new interface in TCL/Tk required a number of days, instead of months, to implement. The primary disadvantage of using TCL/Tk over Motif or the Xt libraries is that it is interpreted and therefore slower and somewhat less robust, but the maintenance is simpler and the time required to learn the package is much less.

4.2.3 Main Window

The main window of pLUCAS is shown in Figure 11(a). The user can select the PVM host file used to initialize the virtual machine as well as adding and removing hosts from this list via the *Configure* menu. This menu also allows the user to copy the data files to the remote hosts, remove these files and start and stop PVM. The large listbox in the center of the window displays scenarios which have been selected in the *Add Scenario* Window. Once all of the scenarios have been selected, the *Run Scenarios* pushbutton starts the virtual machine and calls the pLUCAS C++ program to manage the remote PVM tasks.



(a) Main interface window of pLUCAS GUI



(b) Add scenario window of pLUCAS GUI

Fig. 11. pLUCAS graphical user interface

4.2.4 Add Scenario Window

This window (See Figure 11(b)) is displayed when the *Add Scenario* pushbutton is selected in the main window. The design of this window was strongly influenced by the existing serial LUCAS GUI. Each field has a menubutton which pops up a list of radio buttons used to select the dependent variable, watershed, mapset, beginning year, and scenario name. The scales in the bot-

tom left corner select the number of replicates and time steps to be run and the number of maps to be saved per replicate. A scrollable listbox on the right allows the user to select any impacts to be run each replicate.

Once the scenario has been properly selected, the *Add Scenario* pushbutton adds this current scenario to the list on the main window. The *Reset* pushbutton resets the window to the default values and the *Dismiss* pushbutton unmaps the entire window.

4.2.5 Modifications to LUCAS

In the serial version of LUCAS, the main program directed the classes in Figure 7 to perform the desired number of replicates and time steps of simulation. In the parallel version, this responsibility is passed to the PVM class (Section 4.2.6) and the main program determines which incarnation of pLUCAS is currently running. Under PVM it is typical to have a single executable perform many different functions. pLUCAS has three major modes: *master*, *servant* and *send data*, each of which are examined in Section 4.2.6 in more detail.

The Stats class also had to be altered in the distributed version of LUCAS because all of the statistics collected each time step were originally stored in a single file. The implicit parallelism of pLUCAS required that each servant create its own, independent statistical files which are later reassembled by the master.

The final major alteration necessary to make LUCAS a parallel application was the creation and distribution of unique seeds for the random number generator. As discussed in Section 3.3.3, the R250 generator uses the standard Lehmer linear congruential generator to create seeds. This seed must be unique to each servant process to insure that a replicate is not repeated. Two hundred fifty integers are used to initialize the R250 generator, so a new seed for the LCG is generated simply by taking the value of the sequence after every 250 iterations.

4.2.6 PVM Class

As discussed in the Section 3.5, there are three major states of the pLUCAS executable (see Figure 6). Each serves a different function which is outlined in the following subsections.

4.2.7 Master mode

There is only one *master* task in the virtual machine. It is responsible primarily for task scheduling and result collecting. The master task is run directly from the GUI, but all other PVM tasks are spawned by the master task itself.

Once the number of hosts running in the virtual machine has been determined, the master task spawns *servant* tasks to perform the jobs defined by the user in the GUI. Because the master task spends most of its time in a blocking wait state for the servant tasks to complete their jobs, a servant task also runs on the master node to increase the overall throughput.

Once the servant tasks have performed all of the desired simulations, a *send data* task is spawned by the master on each host. After the results are all back on the host node, a single SAS and graphics output file is generated for each scenario from the many independently generated statistics files. These files along with any saved land cover maps are then installed in the permanent GIS. This final step is cumbersome and costly in terms of communication bandwidth, but the results need to be centrally stored in order for them to be of any use.

4.2.8 Servant mode

A PVM task running in *servant* mode is essentially a scaled-down version of the serial version of LUCAS. It is responsible for simulating multiple time steps of a single replicate. The only difference between this mode and the serial version is that the master task assigns a unique replicate number and random number seed to a particular servant task. It notifies the master task upon completion of its calculations.

4.2.9 Send data mode

This mode copies data from the servant nodes to the master node and was created to reduce the amount of time spent relaying information back to the master. One send data task is assigned to each servant node. If many replicates are run on the same host, it is much more efficient to send all of the information back to the master at once, rather than piecemeal. The send data mode also performs rudimentary housekeeping on the remote node, cleaning up after the servant tasks and preparing for the next time pLUCAS is run.

5 Results and Conclusions

Although it is not necessary in land management for a model to closely approximate the system being modeled, validation is useful to test the model's realism. In Section 5.1 the validity of the implementation of the LUCAS model is discussed. Such a simulation must be reasonably fast, as well as valid, to efficiently investigate a large number of land use scenarios. In order to reduce the computational time to a minimum, a parallel version of LUCAS was created which is significantly faster than its serial counterpart. To determine the benefits of this distributed implementation, both LUCAS and pLUCAS are compared in Section 5.2. Although parallelization is a research advancement, much more is yet to be accomplished with this ecological simulation model. Potential future directions for LUCAS are outlined in Section 5.3.

5.1 Validation of the LUCAS model

To test the validity of the LUCAS model, ten pixel-based replicates of the historical transition simulations for the Little Tennessee River Basin were analyzed. Each replicate began with an initial land cover map for the year 1975. Two 5-year time steps using the historical transition probabilities from 1975–1986 were simulated followed by one time step using the 1986–1991 historical transition probabilities. The simulated 1980, 1985 and 1990 maps were then compared against the historical 1980, 1986 and 1991 maps.[10]

Turner [35] reported that the proportion of land cover of each cover type simulated by LUCAS corresponded closely to the actual historical land cover types. The simulation for forest, which was the dominant land cover type, had little room to redistribute because it comprises 90% of the landscape and hence was spatially distributed accurately. For the other cover types, however, LUCAS simulated many more patches than were present in the historical maps, i.e., the landscape became more fragmented. This is not surprising considering that the pixel-based model used in the simulation examines only independent grid cells. For example, after one time step 3417 patches of grassy land cover with an average patch size of 1.09 cells were simulated, whereas the actual map contained 1583 grassy patches with an average patch size of 2.4 cells. If a patch-based simulation were used, it would likely cause somewhat less fragmentation. However, due to the strict definition of a patch in the current implementation, i.e., contiguous grid cells with identical landscape condition labels, many patches tend to degenerate into one-cell patches or pixels. This is because the chance of at least one of the components of the LCL differing from

[10] Due to the availability of only a select number of land cover maps, historical maps are not available for each simulated year.

its neighbor is great, thus creating small patches. A complete set of statistics from the model validation is available in [13].

5.2 Distributed Results

To test the relative speed of LUCAS and pLUCAS on a varying number of hosts, 10 replicates of 20 time steps for each of the four historical, pixel-based scenarios of the Hoh Watershed on the Olympic Peninsula shown in Table 8 were used. Figure 12 shows the Hoh land cover map before and after a 100 year simulation.

Table 8
Scenarios of land-cover change for Hoh Watershed according to historical transition probabilities

	Ownership Type	
Scenario	**Public**	**Private**
1	1986–1991	1986–1991
2	1986–1991	1975–1986
3	1975–1986	1986–1991
4	1975–1986	1975–1986

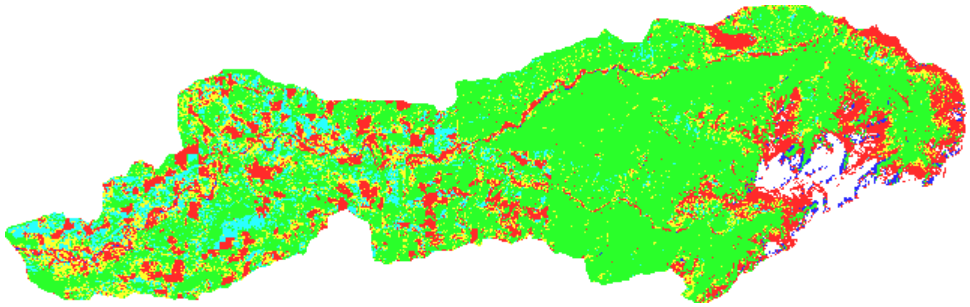
5.2.1 Computing environment

Both LUCAS and pLUCAS were tested on a shared 10 Mb/sec Ethernet network of 20 dedicated 70 MHz Sun SPARCstation 5 machines each with 32 Mb of memory. pLUCAS and a bare essential subset of the GIS were locally installed in the `/var/tmp` directory, a local disk, on each machine which required 6 Mb of disk space for the initial installation and approximately 25 Mb during execution.

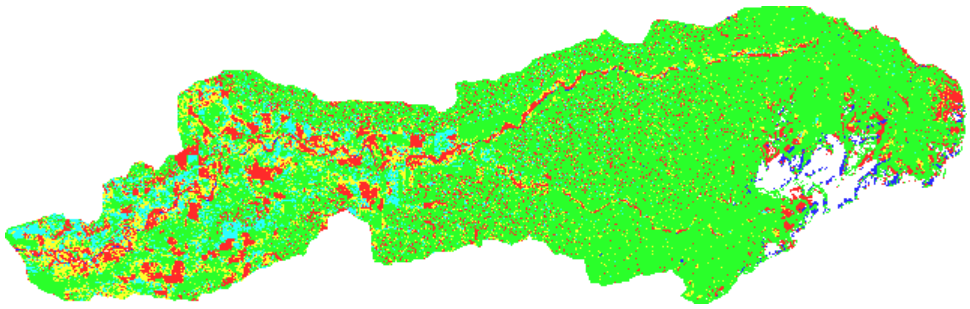
5.2.2 Scalability, Speedup, and Efficiency

Three trials of the same set of scenarios were performed using one host for LUCAS and 4, 8, 12, 16, and 20 hosts for pLUCAS. The elapsed wall-clock time for program execution found in Figure 13 and Appendix D were compared and relative speedups are shown in Figure 14. These times do not include the minimal 1-2 minute one-time setup overhead for pLUCAS found in Table D.1.

Speedup factor is defined as $S(n) = T(1)/T(n)$, [14] where $T(n)$ is the elapsed wall-clock time, and n is the number of nodes (or hosts in PVM). Figure 14 shows an asymptotic behavior for speedup: for a small number of hosts the speedup is very dramatic, but as more and more hosts are added, the relative



(a) Before simulation of Scenario 4



(b) After simulation of Scenario 4

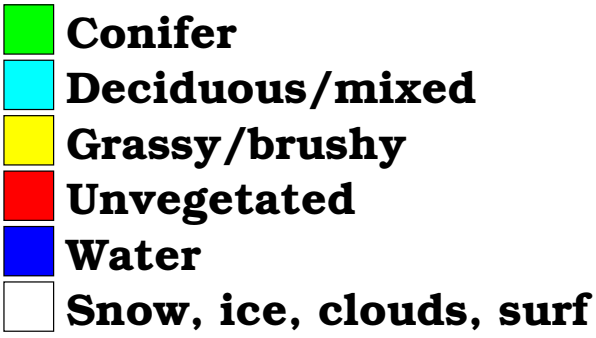


Fig. 12. Hoh Watershed maps before and after a 100-year simulation

speedup becomes increasingly less. The speedup appears to approach a factor of approximately 11 over the serial version. Naturally, a linear speedup would be ideal, but is not realized due to increasing overhead and the communication bottleneck inherent in a master-servant model of parallelism.

From the speedup factors, the efficiencies for the various number of processors is easily determined. Efficiency is calculated by $E = S(n)/n$, where S is the speedup factor and n is the number of hosts. Figure 15 shows that the relative average efficiency steadily decreases as more hosts are added to the virtual machine. As the speedup factor approaches its asymptote, the efficiency plummets. This means that small gains in speed come at the cost of inefficient machine use.

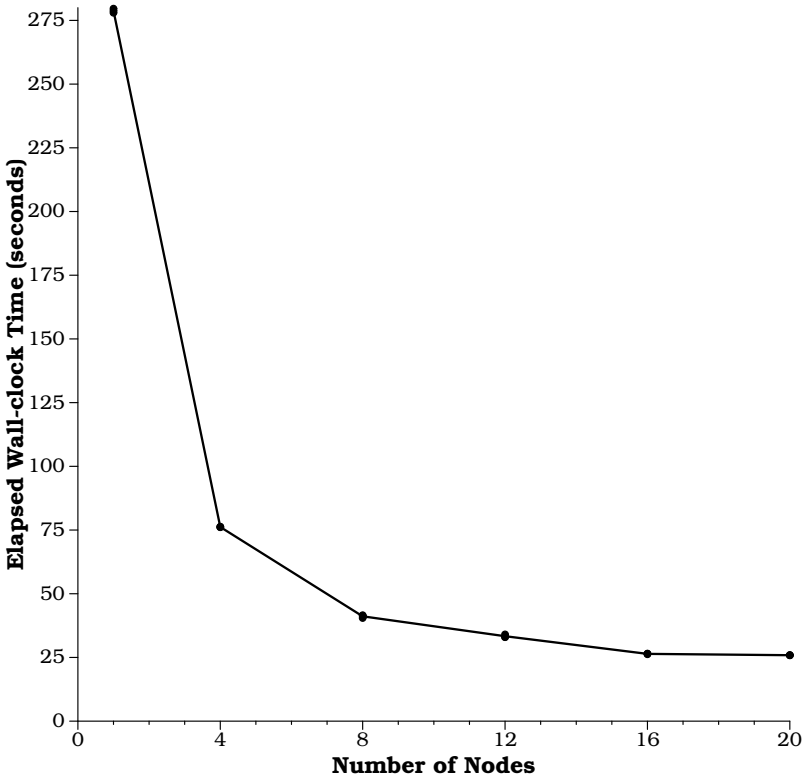


Fig. 13. Average wall-clock execution time for pLUCAS on multiple hosts

Clearly pLUCAS is scalable for a moderate number of hosts (≈ 20). Running pLUCAS on a much larger number of machines, however, would not necessarily be beneficial as speed increases would likely be small. Porting the program in its current form to a supercomputer without concurrent I/O would also be rather impractical because pLUCAS is an I/O-intensive application. It would be reasonable, however, if each node had access to either a local disk or a shared, striped disk array with parallel I/O. This is why PVM made such an excellent choice as a parallel platform for pLUCAS.

5.3 Future Development

Although the LUCAS Project has set a precedent for landscape change simulation, it is not the final solution to ecological landscape modeling. Much more research needs to be done both ecologically and computationally in this field. An immediate addition to LUCAS could be the integration of the multinomial logit coefficients for the Dungeness Watershed, also on the the Olympic Peninsula. Data already exists and is in place, so these additional scenarios could be immediately incorporated into the package.

The first extension to the existing code itself could be the addition of other impact modules, such as water quality or additional species impacts. The species

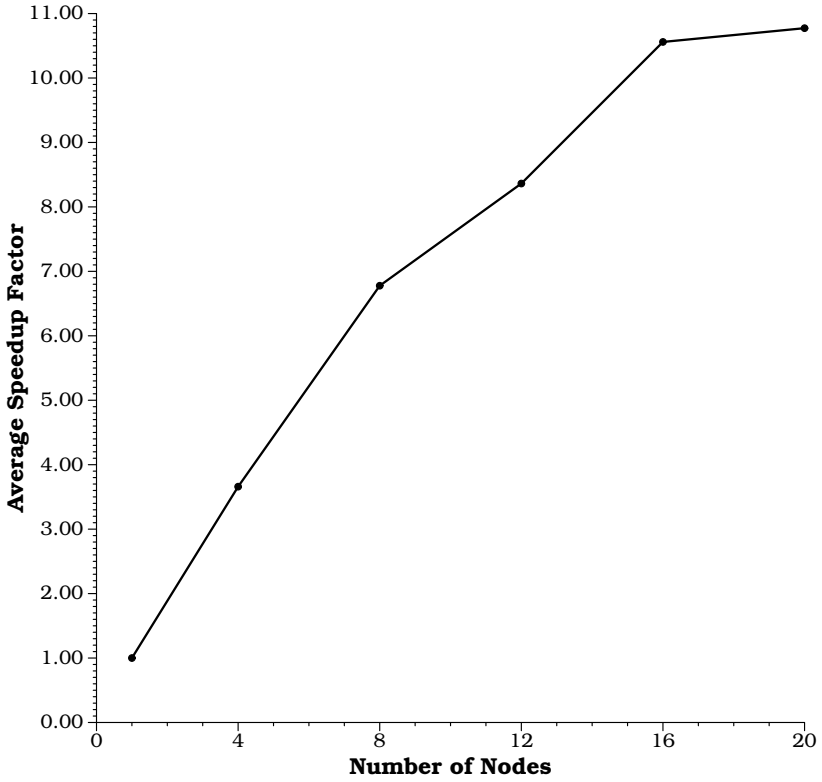


Fig. 14. Average speedup factor for pLUCAS vs. serial LUCAS

habitat module could also be reconstructed in such a way that the user could easily and dynamically define a habitat, thereby alleviating the need for predefined routines. Naturally performance will continue to be an issue, making the migration to such a dynamic system more challenging. Similarly, other transition probability modules could be created which would not require an economist to generate a table of coefficients *a priori*, rather a land manager could simply define a scenario and run it in the same sitting. The economist's knowledge would then have to be encoded in some other fashion than is currently employed.

The next most obvious future need will be simulations on larger maps or at higher resolution which will increase the already sizeable demand on the computer's resources. A natural solution would be to spatially distribute portions of the maps across many processors of a supercomputer or a network of workstations, having each node calculate a portion of the map. For pixel-based simulations this would be fairly straightforward, but patch-based modeling would be much more challenging. Issues such as patch unification, I/O, and data reassembly would be other major hurdles.

Currently, the Integrated Modeling System (IMS) for the Southern Global Change Program (sponsored by the USDA Forest Service) is adapting LUCAS for the study of forest response to environmental stress, disturbances, and land use changes in the southeastern United States. The study region is orders of

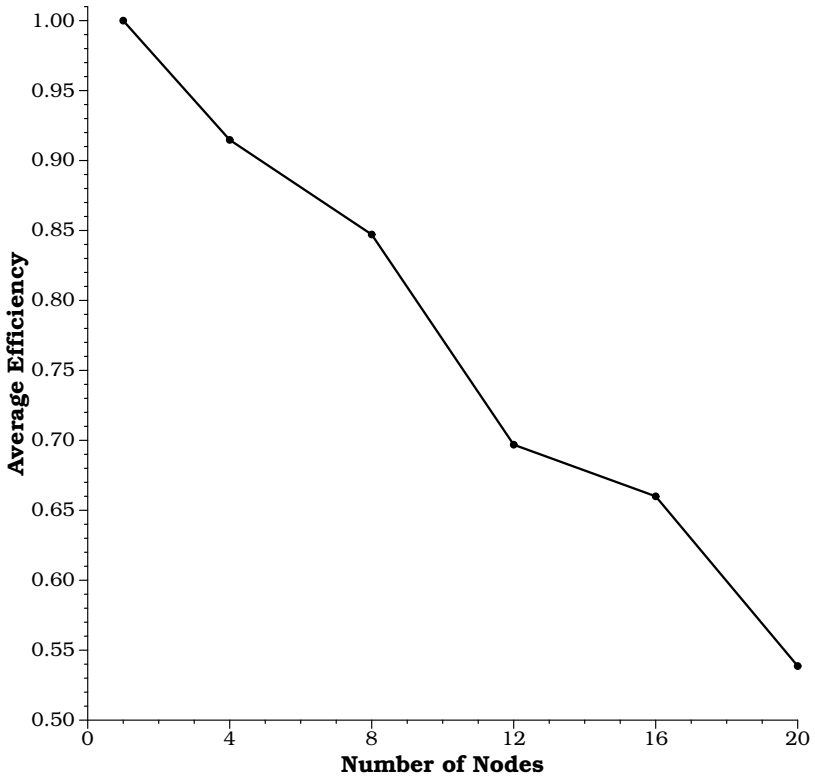


Fig. 15. Average efficiency of pLUCAS vs. serial LUCAS

magnitude larger than the initial test watersheds already implemented, so parallelism may be the only solution to challenges on this scale.

5.4 Conclusions

The Land-Use Change Analysis System is a valuable prototype tool for modeling changes in a landscape to better understand human influence on the environment. LUCAS has already been used by several investigators to better understand land management and change in the LTRB and the Hoh watersheds [28,36,38,39]. Based on the validation results, the environmental scientists working on the project have confirmed their model. Future development of the LUCAS concept will be facilitated by other projects such as the Strategic Environmental Research and Development Program (SERDP) at Oak Ridge National Laboratory (ORNL) and the IMS for the Southern Global Change Program.

As a computer application, LUCAS takes an object-oriented approach to simulation in hopes of promoting code-reusability and versatility for future adaptations. It addresses the problem of efficiently managing a large quantity of spatially explicit data, applying a stochastic model (both pixel- and patch-based), and collecting meaningful statistics. pLUCAS offers a distributed so-

lution to the same problem, thus opening the door for larger, more complex simulations on a network of workstations or a supercomputer.

References

[1] W. L. Baker and Y. Cai. The r.le programs for multiscale analysis of landscape structure using the GRASS geographical information system. *Landscape Ecology*, 7(4):291–302, 1992.

[2] M. Berry, J. Comiskey, and K. Minser. Parallel Analysis of Clusters in Landscape Ecology. *IEEE Computational Science and Engineering*, 1(2):24–38, 1994.

[3] M. W. Berry, R. O. Flamm, B. C. Hazen, and R. L. MacIntyre. The land-use change analysis system (LUCAS) for evaluating landscape management decisions. *IEEE Computational Science and Engineering*, June 1995. To appear.

[4] P. Bratley, B. L. Fox, and L. E. Schrage. *A Guide to Simulation.* Springer-Verlag, New York, 2nd edition, 1987.

[5] E. F. Carter. The generation and application of random numbers. *Forth Dimensions*, XVI(1 & 2), 1994.

[6] E. F. Carter. Personal Correspondence, February 1995.

[7] ERDAS, Inc., Atlanta, GA. *ERDAS Field Guide*, 1994.

[8] R. O. Flamm and M. G. Turner. Alternative model formulations for a stochastic simulation of landscape change. *Landscape Ecology*, 9(1):37–46, 1994.

[9] R. O. Flamm and M. G. Turner. GIS applications perspective: Multidisciplinary modeling and GIS for landscape management. In V. Alaric Sample, editor, *Forest Ecosystem Management at the Landscape Level: The Role of Remote Sensing and Integrated GIS in Resource Management Planning, Analysis and Decision Making*, pages 201–212. Island Press, Washington, D.C., 1994.

[10] J. A. Gallian. *Contemporary Abstract Algebra*, chapter 24, page 310. D. C. Heath and Co., Lexington, MA, 1990. Galois Field of order 2.

[11] R. Gardner, R. V. O'Neill, and M. G. Turner. Ecological implications of landscape fragmentation. In S. T. A. Pickett and M. J. McDonnell, editors, *Humans as Components of Ecosystems: Subtle Human Effects and the Ecology of Populated Areas.* Springer-Verlag, New York, 1992.

[12] A. Geist et al. *PVM: Parallel Virtual Machine A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, MA, 1994.

[13] B. C. Hazen. A Distributed Implementation of the Land-Use Change Analysis System (LUCAS) Using PVM. Master's thesis, University of Tennessee, Knoxville, August 1995.

[14] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, Inc., New York, NY, 1993.

[15] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1988.

[16] S. Kirkpatrick and E. P. Stoll. A very fast shift-register sequence random number generator. *Journal of Computational Physics*, 40:517–526, 1981.

[17] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*, chapter 3. Addison-Wesley, Reading, MA, 2nd edition, 1981.

[18] D. H. Lehmer. Mathematical methods in large-scale computing units. In *2nd Symposium on Large-Scale Digital Calculating Machinery*, pages 141–146, Cambridge, MA, 1951. Harvard University Press.

[19] T. G. Lewis and W. H. Payne. Generalized feedback shift register pseudorandom number algorithm. *Journal of the Association for Computing Machinery*, 20(3):456–468, 1973.

[20] R. L. MacIntyre. A graphical user interface for the Land-Use Change Analysis System. Master's thesis, University of Tennessee, Knoxville, December 1994.

[21] R. L. MacIntyre, B. C. Hazen, and M. W. Berry. The Design of the Land-Use Change Analysis System (LUCAS): Part I - Graphical User Interface. Technical Report CS–94–266, University of Tennessee, Knoxville, December 1994.

[22] R. J. Manchek. Design and implementation of PVM version 3. Master's thesis, University of Tennessee, Knoxville, May 1994.

[23] D. D. McCracken. The Monte Carlo method. *Scientific American*, 192(5):90–96, May 1955.

[24] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.

[25] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, April 1994.

[26] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31(10):1192–1200, October 1988.

[27] T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, Rockville, MD, 1982.

[28] S. M. Pearson. Simulating the impacts of land cover change on native species in two forested watersheds. *Ecological Applications*, July 1995. Submitted.

[29] R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. Wiley series in probability and mathematical statistics. John Wiley & Sones, New York, 1981.

[30] SAS Institute Inc., Cary, NC. *SAS User's Guide: Statistics*, 1992.

[31] M. Shapiro, J. Westervelt, D. Gerdes, M. Larson, and K. R. Brownfield. *GRASS 4.1 Programmer's Manual.* United States Army Construction Engineering Research Laboratory, Champaign, IL, March 1993.

[32] R. C. Tausworthe. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19:201–209, 1965.

[33] J. P. R. Toothill, W. D. Robinson, and A. G. Adams. The runs up-and-down performance of tausworthe pseudo-random number generators. *ACM Journal*, 18(3):381–399, 1971.

[34] J. C. Trexler and J. Travis. Nontraditional regression analyses. *Ecology*, 74(6):1629–1637, 1993.

[35] M. G. Turner. Personal Correspondence, June 1995.

[36] M. G. Turner, D. N. Wear, and R. O. Flamm. Land ownership and land cover change in the Southern Appalachian Highlands and Olympic Penninsula. Submitted to *Ecological Application*, August 1994.

[37] U.S. Army Construction Engineering Research Laboratory, Champaign, IL. *GRASS 4.1 User's Reference Manual*, Spring 1993.

[38] D. N. Wear and R. O. Flamm. Public and private forest disturbance regimes in the southern Appalachians. *Natural Resource Modeling*, 7(4):379–397, Fall 1993.

[39] D. N. Wear, M. G. Turner, and R. O. Flamm. Ecosystem management in a multi-ownership setting: Exploring landscape dynamics in a southern Appalachian watershed. *Ecological Applications*, October 1994. Submitted.

Appendices

A Support Files

Two files are required to run a LUCAS simulation: the *scenario file*, which defines all of the parameters of a particular land management policy and the *impacts file*, which lists the available impacts. This allows the GUI to determine which impacts are supported.

A.1 Scenario File

The graphical user interface (GUI) allows the user to select which scenario she would like to run. Each scenario is defined by a specific *scenario file* in the `SCENARIOS` directory. The scenario file specifies:

- the name of the scenario
- the dependent variable (currently only land cover)
- the watershed in which to run the scenario
- the year in which the starting map was created
- the maps required for the scenario
- the types of the above maps (e.g. impacts map, land cover, etc.)
- the number of values the dependent variable can assume
- the number of independent variables, i.e. the columns in the table of multinomial logit coefficients
- the logit coefficients of the Transition Probability Matrix (TPM)

An example scenario for the Little Tennessee River Basin uses historical transitions for both public and private lands over the period 1975–1986. The dependent variable is land cover and the year of the original map is 1980. The following maps are required:

```
LAYER   slandcover80.r901rcm    cover
LAYER   spubprivate.r90rcm      matrix
LAYER   sslope.r90m             slope
LAYER   selevation.r90m         elevation
LAYER   spopden80.r90m          indep
LAYER   sdistfranklin.r90m      indep
LAYER   sdistroads.r90m         indep
IMPACT  saspect.r90m            aspect
IMPACT  shyd.r90m               hydro
```

The first column is the type of map layer (either standard map `LAYER` or impacts map `IMPACT`), the second column is the name of the map and the

third column is the type of map (`cover` is a land cover map, `matrix` determines which table of coefficients is to be used, `indep` are general independent variables and `slope`, `elevation`, `aspect` and `hydro` are specific independent variables). Land cover can change to one of three cover types and there are six rows of coefficients:

```
# Private Lands ( 1975-1980 )
#       slope    elevation  popdens    disttown   distroads intercept
1 1     0.0       0.0        0.0         0.0        0.0        0.0
1 2    -0.125    -0.00163    0.0017     -0.0019    -0.0242    -0.2845
1 3    -0.10794  -0.0025    -0.0001     -0.00533   -0.0114     1.623
2 1    -0.0028    0.0012    -0.0034      0.0066    -0.064     -1.446
...
```

The first column is the starting land cover type and the second column is the ending land cover type. The transition probability matrix and the multinomial logit equations are discussed in Section 3.2.

A.2 Impacts File

The impacts file contains a colon-delimited list of parameters for each impact module in LUCAS. For each module the following information is specified: the watershed to which it belongs, the module number, the name of the module and the name of the output file associated with the module. Currently only the impact of land cover change on the habitats of certain species is modeled.

```
littlet:0:Catawba Rhododendron:rhodo
littlet:1:Cranefly Orchid:orchid
...
hoh:8:Twinflower:flower
hoh:9:Horsetail:tail
...
```

Module numbers are necessary to indicate which hard-coded routine should be used to simulate the desired impact. While it would be much more desirable to have the impacts file itself define the impact routine, the predefined routines are necessary because of performance considerations.

B Installation of LUCAS

First GRASS must be installed on the local UNIX system. GRASS 4.1 is freely available from USACERL via anonymous FTP at

ftp://moon.cecer.army.mil/lucas/grass4.1. Information about GRASS, the *GRASS User's Reference Manual* and an installation guide can be found at URL **http://www.cecer.army.mil/lucas/** on the World Wide Web. If LUCAS is to be compiled, the Motif Developer's Library version 1.2 or later, GNU `g++` version 2.5.8 or later (or equivalent C++ compiler) must be installed on the system.

Once GRASS and LUCAS have been installed and the LUCAS mapsets have been placed in the GRASS database, the user will need to follow the following procedure:

(i) Set the UNIX environment variable `GISBASE` to the path leading up to the GRASS directory, e.g., in the user's `$HOME/.cshrc` file:
`setenv GISBASE /coral/homes/lucas/gis`.

(ii) Set the UNIX environment variable `GISDBASE` to the path leading up to the GRASS database, e.g., in the user's `$HOME/.cshrc` file
`setenv GISDBASE /coral/homes/lucas/data`.

(iii) Set the UNIX environment variable `GISRC` to point to the user's GRASS resource file also in `$HOME/.cshrc`. This file is usually `$HOME/.grassrc`.

(iv) Run the `grass4.1` shell. Set the name of the location, mapset, and database path name.

(v) From within the GRASS shell, run `g.region` to set the current region for the mapset. Select the "Set from a raster map" option and choose the map for the appropriate location:

Location	**Region Raster**
dungeness	odunge.r90
hoh	ohoh.r90
littlet	littlet.mask

(vi) Exit the GRASS shell.

(vii) Run `lucas` to start the GUI.

Steps iv–vi should be repeated for each location.

C Installation of pLUCAS

The serial version of LUCAS, outlined in Appendix B, should be installed, as much of the same software is needed in both installations. Additionally, TCL 7.4 and Tk 4.0 or later must be installed to use the parallel GUI. They can be obtained via anonymous FTP at **ftp://ftp.cs.berkeley.edu/ucb/tcl**. Information about TCL/Tk, its use and installation can be found at URL **http://www.sco.com/Technology/tcl/Tcl.html**. The Parallel Virtual Machine (PVM) version 3.3.7 or later also needs to be installed on each host

participating in the virtual machine. PVM can be downloaded from **ftp://ftp.netlib.org/pvm3** and **http://www.epm.ornl.gov/pvm/pvm_home.html** has on-line information.

Once the software is installed:

(i) Edit the first line of `gui.tcl` to point to the local installation of TCL. (e.g., `#!/home/lucas/local/bin/wish`)
(ii) Make sure the `PVM_ROOT` and `PVM_ARCH` environment variables are set in the user's `$HOME/.cshrc` file.
(iii) Make sure each host is in the user's `.rhosts` file.
(iv) Install PVM `lib` directory under pLUCAS.
(v) Place pLUCAS `analysis` executables for each architecture in the appropriate PVM `bin` directory
(vi) Install the optional `$HOME/.plucasrc` file.
(vii) Run the `pack` script to create the export tar files.
(viii) Run the `plucas_install` script on the master node. This installs the master node in `/var/tmp`.
(ix) Run the `plucas` script to launch pLUCAS.

NOTE: It is assumed that the host running the GUI will be the master node. The `$HOME/.plucasrc` file is filled with X resources:

```
*IconName:                      LUCAS
*background:                    LightSlateBlue
*foreground:                    NavajoWhite
*highlightBackground:           LightSlateBlue
*activeBackground:              SlateBlue
*activeForeground:              Gold
*troughColor:                   LightSlateGrey
*sunkenBackground:              MidnightBlue
*selectColor:                   Red
*Scale*highlightBackground:     MidnightBlue
```

D Execution times of LUCAS and pLUCAS

The one-time setup overhead times, which include installing the files on the remote nodes and starting and stopping PVM, are found in Table D.1. Table D.2 shows the average elapsed wall-clock times for the calculation of the the four pixel-based historical scenarios for the Hoh Watershed in Table 8 with LUCAS running on a single host and pLUCAS running on 4, 8, 12, 16, and 20 hosts. The averages were taken over three independent timings. The calculation of speedup and efficiency are discussed in Section 5.2. Timings were taken using the UNIX utility `/usr/5bin/time`.

Table D.1
Elapsed wall-clock time (minutes) for setting up pLUCAS

Nodes	Setup times			Average time
4	1:15.9	1:18.8	1:12.1	1:15.6
8	1:30.2	1:23.5	1:27.2	1:27.0
12	1:34.9	1:37.7	1:37.3	1:36.8
16	1:44.2	1:48.6	1:48.8	1:47.2
20	2:10.2	2:17.2	2:08.0	2:11.8

Table D.2
Elapsed wall-clock time (minutes), speedup, and efficiency for Hoh historical scenarios

Nodes	Actual times			Average time	Speedup	Efficiency
1	277:59.6	278:46.5	279:36.7	278:47	1.00	1.000
4	76:05.9	76:19.4	76:09.7	76:11	3.65	0.915
8	40:28.5	41:22.2	41:34.2	41:08	6.78	0.847
12	34:02.4	33:01.4	32:57.1	33:20	8.36	0.697
16	26:15.0	26:27.8	26:29.5	26:24	10.56	0.660
20	25:53.9	25:55.0	25:49.2	25:53	10.77	0.529