DATA-PARALLEL IMPLEMENTATIONS OF MAP ANALYSIS AND ANIMAL MOVEMENT FOR LANDSCAPE ECOLOGY MODELS

Ethel Jane Comiskey

Computer Science Department

CS-93-207 August 1993

Data-Parallel Implementations of Map Analysis and Animal Movement for Landscape Ecology Models

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Knoxville

Ethel J. Comiskey
August 1993

Acknowledgments

I thank my thesis advisor, Dr. Michael Berry, for his guidance, support and encouragement throughout this project. I also thank Dr. Monica Turner and Dr. David Straight who served on my thesis committee. I am very grateful to Dr. Turner and to Dr. Yegang Wu for the opportunity to work with their Northern Yellowstone Park ungulate model. I thank Dr. Bob Gardner and Karen Minser for the use of their serial mean squared radius program, a modified version of which was used for timing comparisons with my parallel algorithms. I also thank the Joint Institute for Computational Science (JICS), which supports the MasPar MP-2 used for this study, and MasPar representative Sally Chase for her advice and help in using the MasPar MP-2.

Abstract

In this thesis, improved sequential and data-parallel implementations of landscape ecology model components are presented. Parallelization efforts on the (SIMD) MasPar MP-2 focus on three model components: cluster identification, mean squared radius computation (cluster geometry), and animal movement.

The NOrthern YELlowstone Park ungulate model (NOYELP), developed by Drs. Monica Turner and Yegang Wu of the Environmental Sciences Division, Oak Ridge National Laboratory, serves as the example landscape ecology model for the model components studied. Modifications made to the original Fortran-77 NOYELP program as part of this thesis project resulted in a revised serial version which executes 11 times faster than the original (CPU time on a Sun SPARCstation 2).

Parallel implementations were tested and compared to functionally comparable serial algorithms using both random maps and maps extracted from runs of the NOYELP model. Speed improvements of MasPar MP-2 parallel kernels over serial implementations on Sun SPARCstations on the order of 9 and 150 for cluster identification and mean squared radius computation, respectively, were measured on 512×512 random maps with a resource probability of 0.85. Speed improvements generally increased with map size and density. For landscape maps tested, speed improvements were somewhat lower, due largely to the inclusion of map pixels outside the study area (54% of total map pixels) in the data maps analyzed.

Results of this study indicate that parallel adaptation of kernels for cluster identification and geometry is straightforward, but that effective parallelization of animal movements in the NOYELP model and similar individual-based models will involve reconceptualizing the movement rule. Issues involved in the parallelization of landscape ecology models are discussed and suggestions are made for future work in this area.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The impracticality of large scale experimental perturbations of natural systems has made computer modeling an important research tool in landscape ecology ([TWWR+93]). Computer simulations are becoming increasingly important in assessing the degree of habitat fragmentation (clustering phenomena) and its ecological implications in many different contexts and at varying spatial and temporal scales. Unfortunately, most landscape ecology models rely on sequential programming, which imposes practical limitations on the size and density of maps which can be analyzed. Parallel computing can expand the capability of these models to simulate spatial and temporal pattern in large ecological systems ([Haef92]).

Individual-based ecological models represent population dynamics by simulating the behavior and interaction of individuals or small groups of individuals responding as discrete units to pattern changes in the environment. Individual movement is simulated in two- or three-dimensions depending on the particular ecological context ([Lomn92]). These models generally have greater computational requirements than population models and can potentially benefit greatly from parallel programming ([Haef92]).

The NOrthern YELlowstone Park ungulate model (NOYELP), the example landscape ecology model used in this thesis, was developed by Drs. Monica Turner and Yegang Wu of the Environmental Sciences Division, Oak Ridge National Laboratory. NOYELP is an individual-based stochastic model which simulates ungulate population dynamics by representing the movement and foraging behavior of small groups of elk and bison in response to changes in the environment that impact the availability of forage and the distance a group can travel in a single day. Computing time for the original serial NOYELP model increases with the number of animals included in the model, as well as with the level of available resources. Parallel computing can be used to increase the number of individuals and the range of environmental conditions considered in individual-based models like NOYELP.

1.2 Objectives

An important objective of this research effort is to produce scalable map analysis algorithms for the identification and characterization of clusters for large, complex maps on massively-parallel SIMD computers. In landscape ecology models, cluster analysis kernels are often called at the end of each model time step to record and evaluate pattern changes, making them key components of many models. For example, the NOYELP model performs cluster analysis twice per model day over 180 days and over 5 replications, for a total of 1800 calls per execution cycle. Cluster radius, a measure of the compactness or density of clusters, has many potential uses in landscape ecology. However, determining radius measures can be computationally intensive, resulting in their exclusion from many applications because of constraints associated with computing time. An efficient parallel kernel for radius computation could make the use of cluster radius (and other measures derived from radius) feasible in landscape ecology modeling.

A second thesis objective is to investigate issues associated with parallelization of the animal movement component of landscape ecology models. Animal movement is typically implemented in serial programs as nested loops of activity repeated over each time-step of a model cycle, a situation for which parallel processing appears well-suited. However, many subjective decisions are made in formulating the rules governing the search and movement of animals in serial models. Some of these rules are based on the serial paradigm and may be unsuited to the constraints of parallel computing. Reconceptualization of movement rules may be necessary for parallel implementations of individual-based models.

1.3 Thesis Overview

Improved sequential and data-parallel implementations of landscape ecology model components are presented in this thesis. Parallel kernels are implemented on the MasPar MP-2, a single instruction, multiple data (SIMD) massively parallel machine. Three model components are examined: cluster identification, mean squared radius calculation (cluster geometry), and animal movement. The first two components are not model-specific and could serve as kernels or modules in other landscape ecology models. The animal movement component is specific to the NOYELP model. Parallel implementations of these landscape model components are tested on random maps and on landscape maps extracted from runs of the NOYELP model. Performance of parallel kernels is compared to that of optimized serial programs.

Chapter 2 of this thesis briefly describes the MasPar MP-2 system, the MasPar

Programming Language (MPL) and the two data-mapping strategies (i.e., hierarchical and cut-and-stack) supported by MasPar. Chapter 3 provides background information about the NOYELP model. Procedures and concepts common to the development and testing of all kernels are presented in Chapter 4. Chapters 5 through 7 address cluster identification, cluster geometry, and animal movement, respectively. Both serial and data-parallel algorithms, along with performance comparisons, are discussed in these chapters. Chapter 8 states conclusions drawn from parallelization efforts and suggests future work in this area. Supplementary information is provided in Appendices A–D.

Chapter 2

Data-Parallel Programming on the MasPar MP-2

The MasPar MP-2 is a massively data-parallel distributed memory processing system consisting of a front end machine and a Data Parallel Unit (DPU). The front end of this single instruction, multiple data (SIMD) system is a DECstation 5000 model 200 workstation with an ULTRIX operating system, windowing capabilities, and standard I/O devices. The DPU, which handles all parallel processing, consists of the Array Control Unit (ACU), the processor (PE) array, an 8-way X-net communication mesh and a global router. Figure 2.1 is a schematic diagram of the MasPar MP-2 system.

The ACU has 24 32-bit registers for user-declared register variables, 128 KBytes of data memory, and 1 MByte of physical instruction memory (RAM), expandable to 4 GBytes of virtual instruction memory. The ACU performs operations on *singular* (shared) variables which are visible to all processors, and controls the PE array, sending data and instructions to each PE simultaneously via the dedicated ACU-PE bus.

The MasPar MP-2 used for this study[1] has 4096 processors arranged in a 64×64 grid. Each PE is a RISC-based processor with 32 32-bit registers available for user-declared variables and 64 KBytes of private (unshared) memory. Each PE has a 16-bit datapath connecting local memory to PE registers. During program execution, all PEs receive the same program instruction from the ACU. All PEs which are active (enabled) at that point in the program execute the same operation simultaneously on their private data. PEs are connected by an 8-way X-net communications mesh and by a global router. In order for one PE to access the private data of another PE, special communication constructs must be used.

[1]supported by the Joint Institute for Computational Science (JICS) at the University of Tennessee, Knoxville.

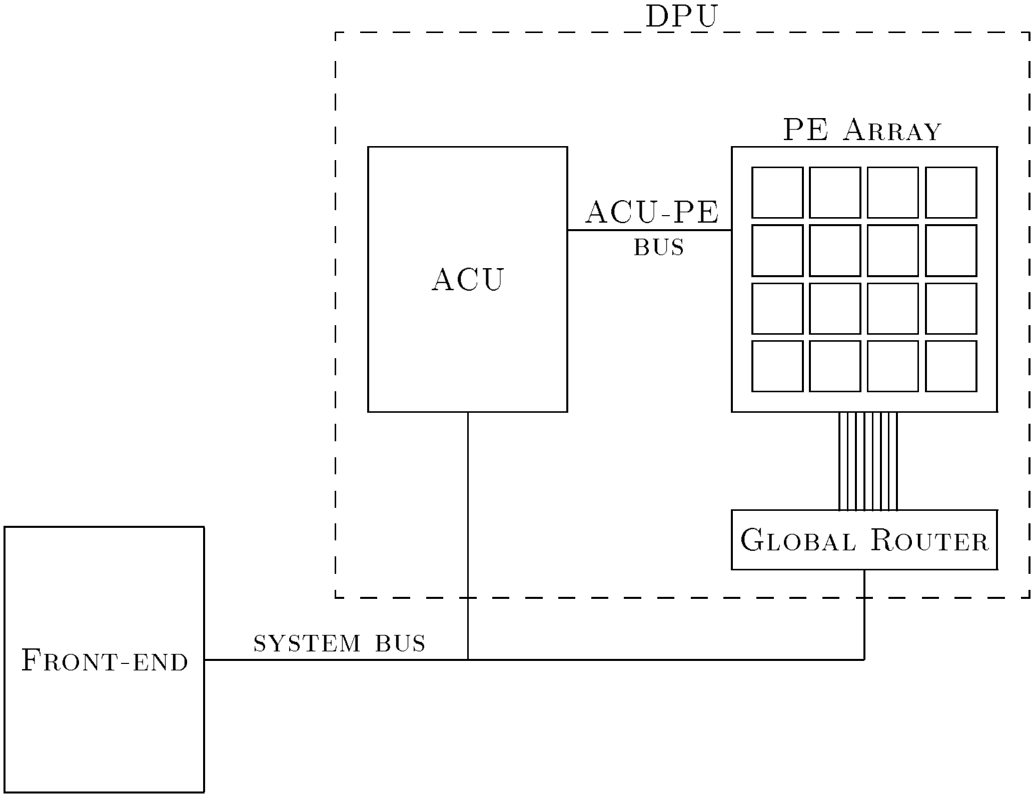


Figure 2.1: Diagram of the MasPar MP-2 system.

2.1 MasPar Programming Language

All kernels developed for this thesis were written in the MasPar Programming Language (MPL), MasPar's ANSI-C compatible language for programming the DPU. MPL is the most efficient and flexible language supported by MasPar. It is also MasPar's lowest level language, allowing more user control over communication and data mapping to processors than can be obtained with MasPar Fortran (MPF). MPL extensions to ANSI C include the capability to allocate *plural* variables across PEs and the ability to perform operations on these variables. MPL adds the keyword `plural` to specify that the associated variable is parallel. An example is provided below.

```
int i;          /* allocates 4 bytes in the ACU's memory */
plural int j;   /* allocates 4 bytes in each PE's memory */
```

Three communication constructs are provided for sending and receiving values between sets of PEs: `iproc`, `xnet`, and `router`. The `iproc` construct allows access to a plural variable on a single PE. The `xnet` construct is used to access processors which are a uniform distance away from active processors in one of eight directions: north, south, east, west, northeast, northwest, southeast, or southwest, requiring both a distance and a direction specifier. Automatic toroidal wraparound is employed with `xnet` to allow circular shifting of data values. East-west borders and north-south borders are connected for shift purposes. For example, the bottom row of PEs is 1 xnet shift north of the top row of PEs. The `global router` is used for communication between a particular PE and any other member of the PE grid.

2.2 Data Virtualization on the MasPar MP-2

Typical cluster analysis applications involve data sets larger than the size (64×64) of the MasPar MP-2 processor array. In situations such as this, where individual processors must handle more than one data point (i.e., map pixel), data must be mapped onto the processor array in some fashion. MasPar systems provide two general data mapping strategies for allocating multiple data points to individual processors: cut-and-stack and hierarchical. Figures 2.2 and 2.3 show in schematic form MasPar MP-2 cut-and-stack and hierarchical virtualization, respectively.

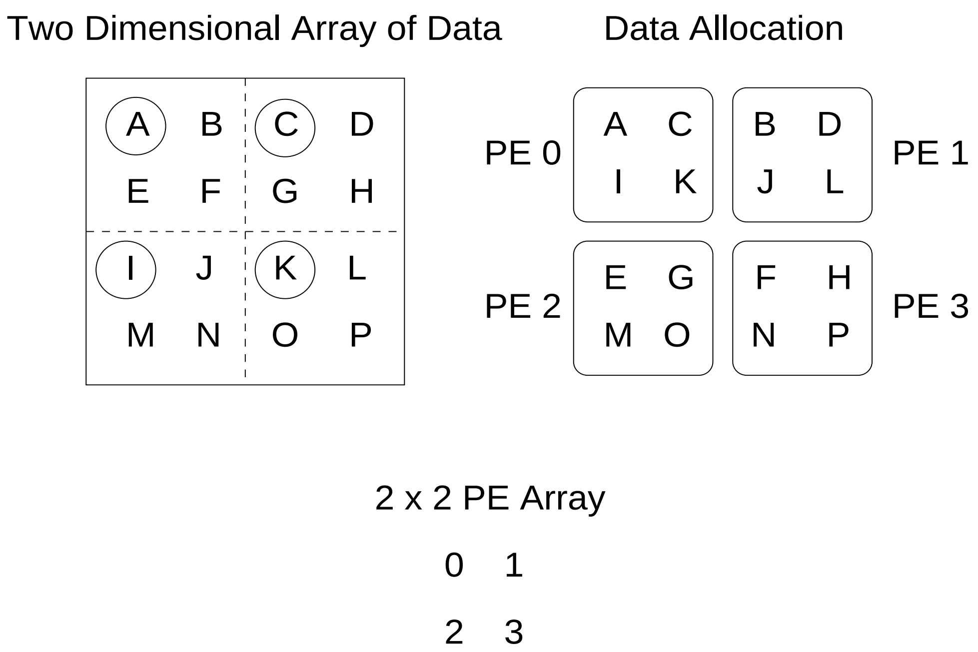


Figure 2.2: Cut-and-stack virtualization on the MasPar MP-2.

With cut-and-stack mapping (Figure 2.2), the data set is divided into a number of segments, called *pages*, equal to the total number of pixels divided by the total number

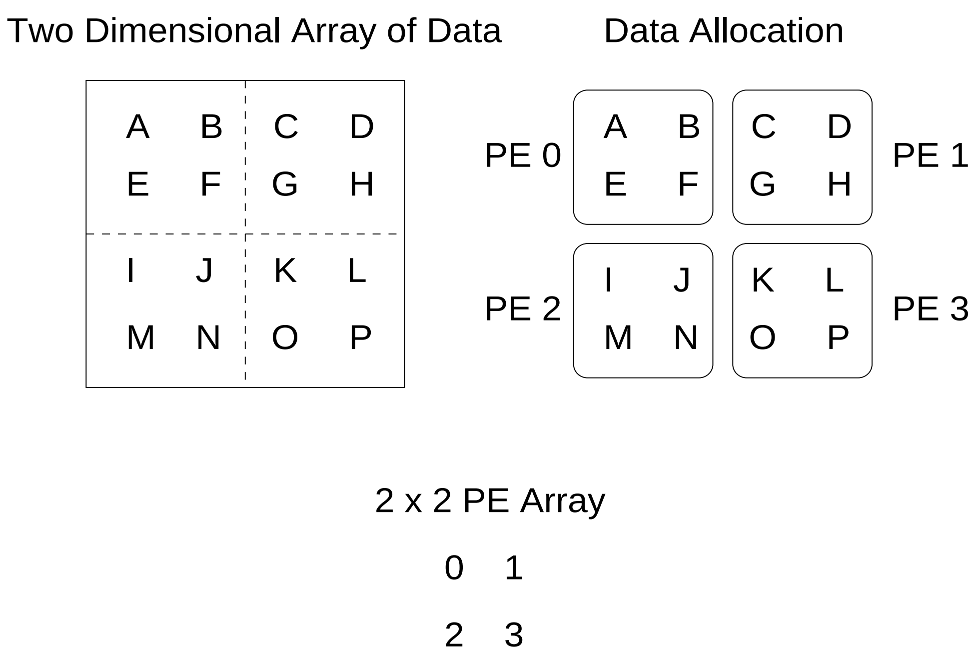


Figure 2.3: Hierarchical virtualization on the MasPar MP-2.

of PEs (i.e., 4096). Each page is the size of the PE array and each PE receives one piece of data from the same relative position in each page. Therefore, if the data pages were stacked, the data assigned to each PE would be an array whose elements form a column running vertically through the stacked pages. In cut-and-stack mapping, logically consecutive data points (i.e., adjacent map pixels) are assigned to physically adjacent processors, accessed by using the MPL communication construct `xnet`. In Figure 2.2, circled data items are assigned to PE 0.

To implement hierarchical data mapping, the data set is divided into as many equally-sized rectangular blocks of adjacent elements as there are PEs, with each PE being allocated one logically contiguous block representing a sub-grid of the original data set. No communication constructs are required for operations within each PE's sub-grid of values. The `xnet` construct is used to communicate between border row and column elements of adjacent sub-grids. As with cut-and-stack mapping, for maps greater than 64×64 processors are assigned more than one pixel, and these pixels are stored as data arrays at each processor.

In choosing between these two strategies, one tries to maximize processor utilization by balancing the workload across processors, while at the same time minimizing communications between processors. Hierarchical mapping is generally more efficient when communication needs are localized in subareas of the data map. Algorithms implementing cut-and-stack mapping are generally simpler to encode, and are often more efficient when there is no advantage to having adjacent data in each PE memory (i.e., when communication requirements are not localized) because work is distributed more evenly across processors.

There exists a family of MasPar `mpi` conversion functions which allow data configurations to be changed within a program. If different data mapping strategies are more efficient for different parts of a program, the programmer can switch between cut-and-stack and hierarchical mapping as needed. See Appendix A for a list of these functions.

Data-parallel cluster analysis kernels employing both hierarchical and cut-and-stack data mapping were developed on the MasPar MP-2 for this study and performance comparisons were made for the two strategies. These results are discussed in Chapters 5, 6, and 7.

Chapter 3

Northern Yellowstone Park Ungulate Model

The programming efforts discussed in this thesis use as an example the spatially explicit, individual-based NOYELP model. The NOYELP model simulates the search, movement, and foraging activities of individuals or small groups of free-ranging elk (*Cervus elaphus*) and bison (*Bison bison*) on the part of their winter range which lies in northern Yellowstone National Park (NYNP). The model was developed to explore the effects of fire scale and pattern on winter foraging and survival of ungulate populations on the heterogeneous, multi-habitat NYNP landscape. The information presented in this chapter is a summary of [TWWR+93], which provides a detailed description of the model and its application to the assessment of impacts of fire on ungulate survival in the context of related studies. A summary list of NOYELP subroutines and formulas used for computation of forage biomass and animal energetics are included in Appendix B.

NOYELP simulations are conducted for each of 180 days during the (approximate) period of November 1 through April 30. Within a day, an animal group makes one to several moves in its search and foraging activities. Available forage biomass varies as a function of foraging activity and snow cover. Ungulate body weight is decreased whenever daily forage intake does not meet energy requirements. Starvation during winter, the main factor influencing ungulate mortality in the study area, occurs when calculated body weight falls below survival thresholds. The model does not project ungulate reproduction or plant growth. For each year simulated, new data (e.g., weather conditions during the 180-day period, number of ungulates present at the beginning of winter, and amount of forage in kg/hectare present in each habitat category at the beginning of winter) are input to the model. Because NOYELP is a stochastic model, five replications of the 180-day simulation period are run with each set of input conditions, and results are summarized statistically over the set of replicates.

3.1 Description of the Study Area

Yellowstone National Park (YNP) was established in 1872 as the nation's first National Park. It covers 9000 km^2 (900,000 hectares) of the landscape in the northwest corner of Wyoming and immediately adjacent parts of Montana and Idaho. The NYNP study area encompasses 77,020 ha in the north central part of YNP. Approximately 83% of the elk winter range is included within the NYNP study area. Ecological dynamics on the winter range largely control ungulate survival and population sizes in YNP.

YNP is characterized by long, cold winters and short, cool summers. The climate is somewhat warmer and drier in the study area compared to the rest of the Park. The northwestern-most part of the study area lies in a precipitation shadow. Snowfall in this area is typically lower than in the winter range as a whole. While elevations in YNP range from 1500 m to more than 3000 m, those characteristic of the NYNP study area are in the lower end of the range. The vegetation of the study area consists primarily of lower-elevation grassland or sagebrush steppe interspersed with aspen and conifer woodlands.

3.2 Model Description

In the NOYELP model, the NYNP landscape is represented as a gridded irregular polygon with a spatial resolution of 1 hectare. The irregular shape of the study area requires a 285 by 584 grid (166,440 grid cells) to span the 77,020 1-hectare grid cells for serial implementations.

Spatial heterogeneity across the NYNP landscape is represented by a series of data maps. Some of the environmental data (e.g., elevation, slope, aspect) are constants, while others (e.g., baseline snow depth and forage biomass) may vary from simulation to simulation. Many of the abiotic data were obtained from the YNP geographic information system (GIS). Elevation is used primarily to initialize bison locations at the beginning of the simulation. Slope, aspect and baseline snow depth are used to estimate effective snow depth.

Initial quantities of pre-winter forage assigned to each habitat grid were derived from data collected during late summer and early fall of 1990. Unlike other habitat variables, forage biomass values are influenced by animal foraging activities and effective snow depth. Ungulate groups search the study area for forage according to a detailed serial movement rule. When snow depth exceeds the brisket height of the particulate ungulate category, foraging cannot occur. Daily forage intake is balanced against daily energy expenditure in estimating weight loss by ungulate individuals. When weight drops below a survival threshold, mortality occurs.

The NOYELP model was originally calibrated by adjusting the values of two parameters, maintenance energy (`enmb`) and the upper threshold of snow equivalent at which foraging is precluded (`swhi`), for which field data were not available ([TWWR+93]).

3.2.1 Habitat types

To simulate ungulate foraging and survival, the NYNP landscape is discretized into six habitat types (Figure 3.1). Four of these six habitat types are grasslands, differentiated primarily on the basis of moisture availability, species composition and biomass production. At elevations characteristic of NYNP, the sagebrush-grassland habitat types form a patchwork mosaic with the two woody habitat types, aspen stands and coniferous (canopy) forests (dominated by pine and fir). In general, ungulates appear to respond to forage quantity rather than quality or subtle community differences, which simplifies the modeling process. At the start of the model year, forage is distributed within each habitat type by assigning to each grid cell in the habitat type a forage biomass value drawn randomly from the 95% confidence interval (i.e., $\pm$ 2 standard errors) of a normal distribution around the mean for the particular vegetation class, as determined by field sampling.

3.2.2 Ungulate Categories and Distribution

Six ungulate categories were defined for simulation purposes. Bison groups consist of 9 cows, 9 calves, or 2 bulls, while elk cow, calf and bull groups each include four individuals. Since calf groups follow the foraging pattern of the cow groups to which they are assigned, the model effectively simulates groups of 8 and 18 combined cow/calf groups for elk and bison, respectively. The model places no constraints on the number of groups that may occupy a grid cell at one time. All animals within the same ungulate group are assigned the same initial body weight.

Elk are initially distributed randomly within grid cells containing forage (i.e., resource sites) across the winter range. Bison groups are initially distributed randomly in grassland habitats at elevations $\leq 2100m$, resulting in approximately 90% being assigned to the eastern portion of the study area.

3.2.3 Snow Simulation

Snow conditions are extremely important in determining the winter dynamics and survival of ungulate populations in the NYNP. Both foraging and movement can be affected by snow. Snow depth and snow density (%) are used to determine energetic costs of travel and maximum daily moving distances. Snow water equivalent (`swe`), the product of snow depth and density, influences daily forage intake. When snow depth exceeds brisket height, ungulates generally cannot forage. Even shallow, densely packed snow may limit foraging. There is an upper limit of snow/water equivalent (`swhi`) above which no foraging can take place. This limit is category-specific, depending largely on the size of the animal (i.e., brisket height). For example, bulls can travel and graze in deeper, denser snow than calves.

To simulate snow conditions, the northern range is first subdivided into two regions (the snow-shadow area and the rest of the study area), based on amount of winter precipitation, and baseline snow depths and densities are then projected within each area, assuming no slope or aspect effects. These baseline projections are subsequently modified according to the slope and aspect of each grid cell. Snow conditions are updated at 3-day intervals.

3.2.4 Foraging

Daily ungulate forage intake on a grid cell is a function of maximum daily intake. This intake is a product of two constants, initial body weight (`bw`) and maximum daily foraging rate, (`feed`), and one of two negative feedback terms representing the amount of available forage at the site (`fbbio`) and the depth and density of snow (`fbswe`). Each feedback term is a number between 0 and 1. Whichever feedback term is smaller has the greater impact on foraging and is allowed to operate.

The hyperbolic forage availability feedback term (`fbbio`) reflects a direct relationship between the amount of available forage and the instantaneous rate of feeding. As ungulates graze, the feedback term decreases, reflecting a decrease in the amount of available forage on the grid cell. Because no regrowth of vegetation occurs during the winter (dormant) season, and no other sources of forage attrition are considered in the model, biomass can only remain the same or decrease. The feedback function utilizes the concept of a refugium value of biomass not available to ungulates. When forage biomass falls to the refugium value (13% of Fall biomass), the value of the feedback term is set to zero. When the refugium value is greater than 0, the feedback term will be less than 1.

The snow water equivalent (`swe`) feedback term reflects the effect of both snow depth and snow density on the ability of ungulates to obtain forage. Two thresholds (the value at which foraging is set to zero and the value at which limitation of foraging begins) are used in defining the snow feedback term. At `swe` values greater than those which limit foraging, an animal can forage at its maximum rate. Between the two thresholds of `swe`, a linear change in the value of the feedback term is assumed.

3.2.5 Search and Movement Rules

The NOYELP model utilizes a simple algorithm to simulate an ungulate's search and movement strategy. If an ungulate group is located on a grid cell containing available forage at the start of the day, the animal grazes. If forage intake on that cell is less than the daily maximum, the animal searches for another grazing site. Because the forage feedback term has an upper limit at 0.87 (with the refugium level set to 13%), the ungulate group cannot attain its maximum daily intake of forage from one grid cell, and must move at least once per day. An ungulate group is prohibited from remaining at the same grid cell, revisiting a grid cell during the course of one day's movement, or moving to a grid cell outside the boundary of the study area. Searching procedures are described in detail in Chapter 7.

During fall and early winter, when forage is generally available, ungulate groups will typically move only once per day. Later in the winter, when forage becomes less available due to foraging and/or the presence of snow cover, an ungulate group may move several times per day in its search for food. Maximum movement distance (and, therefore, the number of moves an ungulate group can make in one day) decreases as snow conditions become more extreme because of increases in energy costs associated with travel in snow.

3.2.6 Energetics

Daily energy balance is the difference between daily energy gain, `engain`, and daily energy expenditure, `encost`. `Engain` is the product of total intake of forage in kg (`fd`) and habitat type specific forage energy content in kcal/kg (`enpk`). `Encost` is the sum of maintenance energy cost (`enme`) and travel energy cost (`enmov`). Maintenance energy cost, represented as a power function of current body weight, includes the energy costs associated with all the animal's daily activities which occur within the grid cells. For initial model parameterization, estimates of maintenance energy cost obtained from the literature were used, but these estimates were subsequently adjusted during model calibration. Travel energy cost is computed by first calculating the (per unit distance) energy cost of travel in the absence of snow (a function of body weight), and then modifying this value to account for the relatively higher travel costs associated with travel in snow (a function of snow depth and snow density). These costs increase exponentially as a function of relative sinking depth, and may limit maximum daily distance traveled and (consequently) the number of cells a group can search during the day.

Whenever forage intake is insufficient to meet the animal's energy expenditures, ungulate body weight is adjusted downward. No weight gain is permitted. Death by starvation is assumed to occur when ungulates lose both 70% of their fat and 30% of their non-fat body weight. Since no predators are included in the model, death by starvation is the only significant source of population attrition on the winter range.

3.3 Example Data Set

For the example data set used in simulations for this study, 19,972 animals in 5,015 groups were input to the model. Greater than 96% of the total number of ungulates were elk. Table 3.1 presents a listing of ungulate cow, calf and bull groups, with

Table 3.1: Sample composition of animal groups in a NOYELP model simulation.

Category	Fraction of Population	Size of Groups	Number of Individuals	Number of Groups
Elk:				
Cows	0.65	4	12524	3132
Calves	0.16	4	3084	771
Bulls	0.19	4	3660	915
Total	1.00		19268	4819
Bison:				
Cows	0.38	9	270	30
Calves	0.18	9	126	14
Bulls	0.44	2	308	154
Total	1.00		704	196
All ungulates			19972	5015

a count of individuals of each species belonging to each of the categories. Since movement of calf groups follows that of the cow groups to which they are assigned, the model effectively simulates the movement of 4,230 groups.

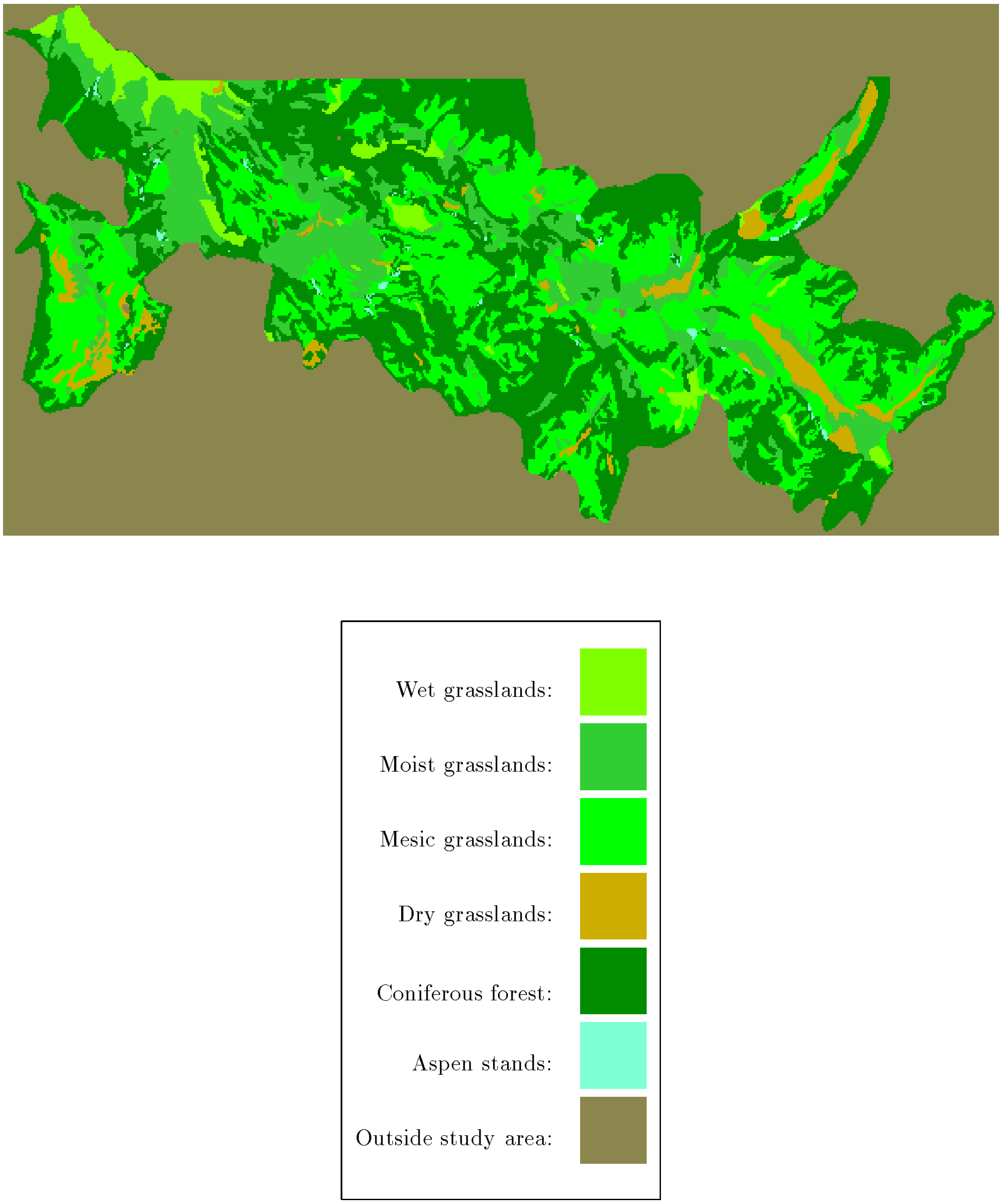


Figure 3.1: Map of NOYELP study area showing habitat types.

Chapter 4

Preliminaries

In developing kernels for diverse applications requiring cluster identification and geometry, the programmer must make the kernels adaptable to the needs of the application. Clusters might represent resources, animals, landscape patterns, or dispersal patterns of pollutants ([TWWR+93]). Map size and density may be static or may change during program execution.

To simulate cluster analysis on landscape maps, random maps are generated having a proportion, p, of 1's. The proportion of 1's (or non-zero elements) in a map is called the p *value* of that map ([StAh91]). These random maps are then used to facilitate algorithm development and to test parallel implementations for accuracy. Using random maps, performance of an algorithm can be predicted on prospective real world maps of various sizes and densities from actual landscape ecology models. For testing purposes, $m \times m$ random maps of density p are generated, where $m =$ 64, 128, 256, 512, 768, 1024, and $p = 0.10$, 0.30, 0.62, 0.85, and 1.00 for each value of m.

In landscape ecology models, the non-zero pixel elements generally represent an assigned level or range of some specific habitat parameter (e.g., moisture less than 10%, biomass greater than 100 kg/hectare, or a flammability index of 7), animal group density, or some aggregate of parameters representing habitat suitability on the unit area of landscape.

For this study, 1 hectare landscape units with available resource (i.e., forage biomass) above a pre-assigned threshold level is represented by setting the grid elements of the random maps to 1 with a probability of p. For these random maps, a setting of 0 (with probability $1 - p$) represents habitat with resource levels below the threshold. Thus, pm^2 is the number of pixels of suitable resource habitat (or 1's) within the map. For irregularly-shaped real world study areas, such as NYNP, the rectangular or square data grid will include pixels outside the study area, thereby decreasing the p value of the map as a whole.

4.1 Map Density

Table 4.1 summarizes cluster size distribution, size of the largest cluster, and total number of clusters for the five p values and six map sizes used in this study. All clusters for maps with $p = 0.10$ and $p = 0.30$ have fewer than 100 members (or pixels), while maps with $p \geq 0.62$ have a large dominating cluster, along with smaller clusters. This change in maximum cluster size is explained by percolation theory ([StAh91]). According to this theory, maps with p values greater than a threshold of 0.5928 are characterized by a large dominating cluster that *percolates* across the map from boundary to boundary. A random map with $p = 0.10$ is a sparse map with small, isolated clusters; $p = 0.30$ yields a map with many fragmented clusters; a map with $p > 0.59$ is dominated by one large cluster.

Figure 4.1 illustrates the differences in cluster numbers and sizes associated with p values of 0.30 and 0.62 for 64×64 maps.

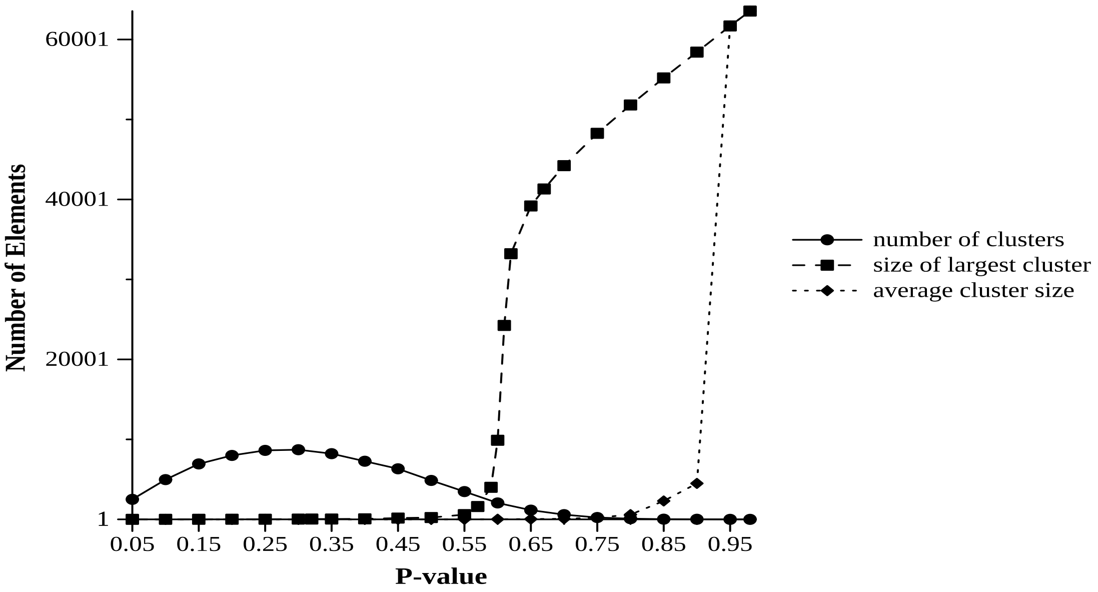
(a) (b)

Figure 4.1: Sample 64×64 random maps with (a) $p = 0.30$ and (b) $p = 0.62$.

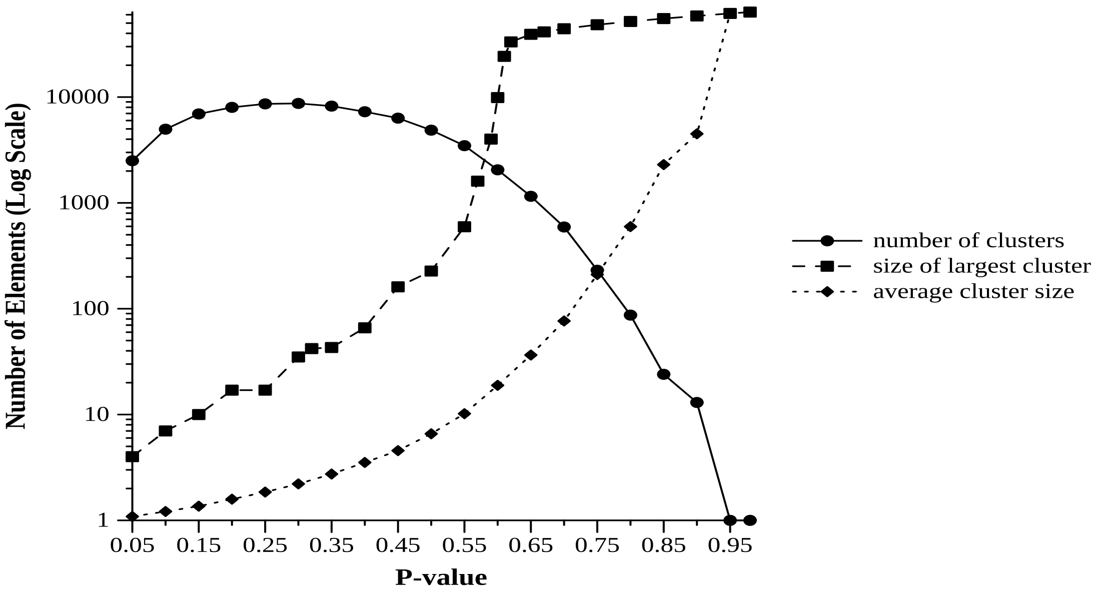
As the p value of a map increases, the amount of the work involved in cluster identification and geometry computations changes. The relationships of number of clusters, maximum cluster size, and average cluster size to p value for 256×256 maps are illustrated in greater detail in Figure 4.2. The number of clusters gradually increases to a peak around $p = 0.33$, after which it gradually decreases to 1. The average cluster size remains small until p values exceed 0.80 and the number of clusters drops to about 100. Maximum cluster size stays relatively small until the percolation threshold of $p = 0.5928$ is reached, after which the size of the largest cluster increases dramatically and remains large. For larger p values, the average cluster size

Table 4.1: Distribution of cluster sizes for randomly generated maps of six sizes and five p values.

		Size of	Total	Cluster Size						
p	Map Size	Largest Cluster	No. of Clusters	<100	101-500	501-1000	1001-10000	10001-100000	100001-500000	>500000
0.10	64	8	325	325						
	128	6	1305	1305						
	256	7	5227	5227						
	512	12	20917	20917						
	768	8	47281	47281						
	1024	15	84140	84140						
0.30	64	25	534	534						
	128	29	2157	2157						
	256	33	8484	8484						
	512	42	33891	33891						
	768	44	75941	75941						
	1024	55	135122	135122						
0.62	64	1981	110	109			1			
	128	6609	382	376	4	1	1			
	256	34363	1400	1393	5	1		1		
	512	141190	5503	5490	10	1	1		1	
	768	323676	11989	11965	22	1			1	
	1024	577501	21141	21099	37	4				1
0.85	64	3455	4	3			1			
	128	13862	9	8				1		
	256	55523	30	29				1		
	512	222417	129	128					1	
	768	500409	259	258						1
	1024	890080	510	509						1
1.00	64	4096	1				1			
	128	16384	1					1		
	256	65536	1					1		
	512	262144	1						1	
	768	589824	1							1
	1024	1048576	1							1



(a)



(b)

Figure 4.2: Comparison of cluster characteristics across p values for 256×256 maps (a) standard scale and (b) log scale.

approaches the size of the largest cluster as the number of clusters approaches one.

Execution times for cluster identification and geometry programs can be related to changes in cluster characteristics as a function of map density. Different cluster analysis algorithms developed during the course of this study generally reach performance peaks at characteristic p values. Some algorithms performed well on sparse maps (p values below about 0.15) or handled large numbers of small clusters well (p values near 0.30), but did not process dense maps containing a small number of large clusters efficiently. Algorithms which perform well on the large dominating clusters found in maps with p values above 0.59 often were not well adapted to handle the numerous smaller clusters found in sparser maps. Testing with maps of widely varying densities is necessary to identify the combination of algorithms which provide optimal performance over the entire range of densities found in real landscape maps.

The relationship between cluster characteristics and p values is somewhat different for non-random landscape maps. While the size of the largest cluster (and hence computational complexity) is predictably tied to the p value in random maps, this is not the case with landscape maps. For example, calculation of the mean squared radius of a cluster of size n requires computing $\mathcal{O}(n^2)$ squared row and column differences. The random map of size 64×64 (4096 pixels) with $p = 0.30$ (1229 resource pixels) generated for this study contains 534 clusters, all with fewer than 25 pixels and an average cluster size of 2.3. A total of 6103 row and column comparisons are required for all cluster geometry computations. By comparison, a 64×64 (non-random) landscape map with a p value of 0.30 could have 1229 clusters of 1 pixel each, requiring no row/column comparisons for mean squared radius calculation or, at the other extreme, could have 1 cluster with 1229 pixels, requiring $1229 \times 1229 = 151044$ comparisons. Because the relationship of p value and cluster size is not predictable for landscape maps, the size of the largest cluster is a more accurate indicator of performance than p value for these maps.

While results from random maps might not be directly comparable to those from landscape maps of the same size and density, the trends in algorithm performance over a set of random maps of varying sizes and densities can serve as a general guide to performance on landscape maps with known cluster characteristics. The similarity between performance results for random and landscape maps increases as the p value increases, since the range of possible cluster configurations decreases as maps become more dense.

Some modeling applications require analysis of clusters in a static map of known density while in other applications, such as NOYELP, changes in a given landscape feature (i.e., available biomass) over time must be evaluated by identifying and characterizing clusters at each time step. A typical dynamic application might start with a dense map (high p value) and analyze progressively sparser maps as resources are depleted; other applications will involve a map that is initially sparse, with the p value increasing as complexity is added. In these dynamic applications, a small improvement in cluster analysis time could save hours in total computing time.

4.2 Map Size

The relationship between random maps and landscape maps is complicated by the irregular shape of real landscape unit, and the resulting need to include pixels in real maps which are outside the study area to attain a regular (i.e., rectangular) grid for spatial analysis. For example, the 77,020 hectare NYNP study area is an irregularly-shaped polygon. Because of its irregular shape, the study area must be represented in the serial NOYELP simulation model as a 285×584 grid. If the study-area pixels were contained in a square, it could be represented as a 278×278 grid, with a spatial resolution of 1 hectare. Pixels outside the NOYELP study area constitute 54% of the model grid and are assigned the value -1 to distinguish them from habitat pixels.

When NOYELP input files are adapted to the 4,096-processor MasPar MP-2 implementations, the model grid must be further extended to 320×640, so that all rows and columns are multiples of 64, the row and column dimensions of the MasPar PE grid. This increases the proportion of non-study area pixels to 62% of the total map area. For MasPar MP-2 implementations, each processor is assigned 50 pixels, with an average of only 19 of these representing study area pixels. As a result, speed improvements for parallel NOYELP implementations will be smaller than for random maps of the same size. Processors which have been assigned map segments outside the study area will be idle, causing performance degradation. For this reason, the shape of the study area is a much more serious concern for parallel simulations than for serial models.

Algorithm performance for NOYELP maps will also differ from that of random maps of the same size and density because of the non-random distribution of forage biomass, a product of environmental heterogeneity and non-random foraging activities. This clumping of biomass distribution will affect different cluster analysis algorithms in different ways (see discussion in Section 4.1 above).

4.3 Verification

In developing parallel cluster analysis and ungulate movement algorithms, testing with maps of varying size and density is important not only for predicting performance, but also for verification of results. Error detection in parallel programming is not as straightforward as it is with serial programming, especially when data virtualization (via MPL) is involved. Care must be taken to exercise all permutations of inter-layer communication to ensure that no data are lost. Implementations which function correctly for dense maps (i.e., which "turn on" most or all virtualized layers to the active state) may fail when maps are sparse. On the other hand, these dense maps may generate large intermediate sums or products which test the limits of variable sizes and the accuracy of variable casting. During the algorithm development phase of this thesis effort, several errors appeared in only 1 of the 30 size/density test combinations, pointing to the need for testing over a range of combinations.

4.4 Serial Computing Environment

The computing environment used for serial program development and testing for this thesis consists of two architectures: the Sun SPARCstation 2 and the SPARCstation IPX. The SPARCstation 2 has a SunOS 4.1.2 operating system, a clock speed of 40 MHz, and 64 Mbytes of RAM. The Sun SPARCstation IPX is very similar, with a SunOS 4.1.3 operating system, a clock speed of 40 MHz, and 16 Mbytes of RAM. Both are capable of a peak computation rate of 4.2 Mflops (Millions of FLOating-Point operations per second). Performance of these machines on SPEC benchmarks is shown in Table 4.2.

Table 4.2: Performance specifications for architectures used in the sequential computing environment.

Machine	Sun SPARCstation IPX	Sun SPARCstation 2
MIPS	28.5	28.5
SPECmark89	24.4	25.0
SPECint92	21.8	21.8
SPECfp92	21.5	22.8
Mflops	4.2	4.2

The serial timings presented in the results sections of Chapters 5 through 7 were obtained from one of these two architectures, as specified in discussions and in table and figure captions. The Sun SPARCstation 2 was generally used, except for programs which compute mean squared radius. These programs required many hours of computing time, and were run on the Sun SPARCstation IPX because of the availability of dedicated time. Dedicated wall-clock times are used when times for serial implementations are compared to those for parallel implementations. When serial implementations are compared with other serial implementations, CPU times for programs run on the same machine (Sun SPARCstation 2 or Sun SPARCstation IPX, as specified) are used. When computing times are specified as excluding *I/O* time, *I/O* refers to reading data from input files and writing results and times to a standard output device (screen).

Chapter 5

Cluster Identification

Cluster identification is not unique to landscape ecology. It is important in such diverse fields as image processing and lattice field theory in physics ([ApCM92]). In physics, cluster identification is performed on n-dimensional maps and is referred to as *connected component labeling*, with map elements considered as boolean variables set to *on* or *off*. The goal is to have the same unique label on all connected sites and a different label for each disconnected cluster.

In landscape ecology applications, cluster identification typically involves locating and labeling clusters in a 2-D grid, and determining cluster characteristics such as total number of clusters, size of each cluster, size of the largest cluster, and average cluster size. Adjacent pixels are considered to belong to the same cluster if they have the same value (e.g., habitat or resource level), as defined by a particular nearest-neighbor rule. The neighbor rule implemented in the serial and parallel algorithms investigated in this thesis considers pixels containing the same value to belong to the same cluster if they are north, east, west or south (NEWS) neighbors of each other or of some other element in the cluster. Diagonal adjacency is not considered in this rule. In the following discussion, grid cells or pixels having a positive value indicating membership in the map class being analyzed are called resource pixels, consistent with the NOYELP model example utilized in this effort (where forage biomass is the resource variable of interest). Figure 5.1 shows a simple grid with cells belonging to each cluster (according to the neighbor rule) in a common enclosure.

5.1 Serial Algorithms

Serial cluster identification algorithms fall into one of two classes: (1) those which build entire clusters in sequence (i.e., one at a time) in the order in which they are encountered during grid traversal, and (2) those which build clusters incrementally, as members are encountered in grid traversal.

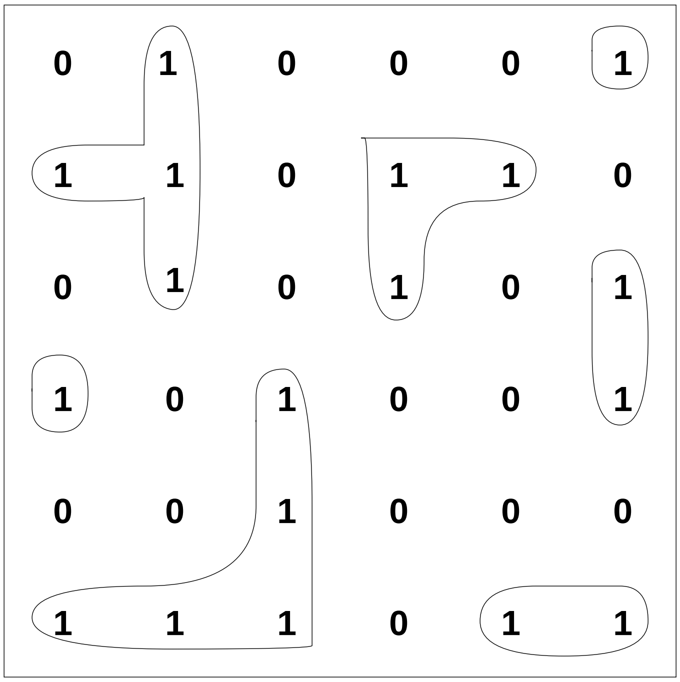


Figure 5.1: Two-dimensional spatial grid showing showing 7 individual clusters.

Recursive and pseudo-recursive cluster identification programs fall into the first class (i.e., they build cluster in sequence). Recursive programs have a cluster-building function which labels one pixel, then calls itself recursively with the location of any nearest neighbors of that pixel. Pseudo-recursive programs simulate recursion by storing pixel locations in arrays, which serve as *stacks* of cluster elements from which clusters are sequentially built. Cluster labeling is accomplished by traversing the grid one element at a time. When a pixel containing a resource value is encountered, it is labeled with a unique cluster number. Then all nearest neighbors of this pixel are examined and added to the growing cluster if they are also resource pixels. This process continues with examination of nearest neighbors of nearest neighbors, until all pixels in a particular cluster are identified and labeled. Traversal of the grid (and building of the next cluster) then continues with the next unlabeled pixel and the process continues until all clusters members are labeled.

Algorithms which fall into the second class embody an alternate approach to cluster identification, incremental cluster building. The grid is traversed from top down and left to right. Pixels are given temporary labels as they are encountered, and labels are updated as clusters take shape concurrently. Examples of this approach are the Hoshen-Kopleman algorithm, discussed in more detail below, and the *local diffusion* or *label propagation* method ([ApCM92]).

In the following subsections, (5.1.1 to 5.1.3), five serial cluster algorithms are discussed. Three of these (the original NOYELP function, a revision of this function, and the Hoshen-Kopleman algorithm) implement incremental cluster identification, while the other two, based on recursion and pseudo-recursion, respectively, process clusters sequentially.

5.1.1 Original NOYELP incremental algorithm

The original cluster identification algorithm used in the serial NOYELP model is a type of *local diffusion* algorithm, wherein repeated local nearest-neighbor comparisons (using the NEWS rule) result in correct labels diffusing throughout the grid. As the resource grid is traversed one element at a time, each element compares its resource value with those of its nearest neighbors. Elements with matching values are included in the same cluster by assigning to all the lowest (numerical) cluster label in the cluster. Cluster labels are updated as cluster membership changes. In the original NOYELP version, grid traversal is repeated four times, with four comparisons made for each pixel on each pass. It was assumed that all clusters had been properly labeled at this point.

As implemented in the original NOYELP model, this cluster identification algorithm had several deficiencies. One involved a minor array index error which would have led to aberrant results, but was easily corrected. The other, which involved the inefficient and incomplete implementation of the basic algorithm, was more significant. Testing with several 10×10 data files representing several known levels of cluster complexity showed that this algorithm correctly identified simple cluster patterns, but complex clusters were incompletely labeled. Testing with larger random maps revealed that errors in cluster identification began to appear as resource pixel density approached the percolation threshold ($p = 0.59$). Large clusters which spread from border to border were incorrectly labeled as several smaller clusters. These results indicated that four traversals of the grid were not sufficient for complete cluster identification when densities were near the percolation threshold.

The original algorithm was modified to identify clusters correctly by adding an activity flag and making repeated passes through the data until no cluster-building activity was detected. As many as 32 4-comparison (NEWS) passes through a 1024×1024 map with $p = 0.62$ were required to accurately label all clusters, resulting in a substantial increase in execution time. Clearly, a more efficient algorithm was needed to deal with the range of map densities which might be encountered in modeling natural environments.

Yegang Wu, author of the original NOYELP program, subsequently developed another incremental cluster-building Fortran-77 algorithm (ORFOR) which correctly identified all clusters and showed significant speed improvements over the original algorithm (modified for accuracy by adding the activity flag, described above) but required large arrays and performed poorly on maps with large clusters.

5.1.2 Recursive and pseudo-recursive algorithms

Cluster identification is most easily conceptualized in recursive terms, but recursion is not available in Fortran-77. Nonrecursive versions of recursive algorithms are often more efficient, even in recursive languages, because they lack the overhead of repeated parameter passing, and because nonrecursive code is more easily optimized by compilers ([HoSa83]).

A two-step approach was taken to develop this more efficient Fortran cluster identification algorithm:

1. A recursive C program (RECRC) was written and tested for accuracy. This program provided the conceptual basis for development of a functionally comparable pseudo-recursive Fortran program.

2. Following a procedure outlined in Horowitz and Sahni ([HoSa83]), the recursive C program was translated into a pseudo-recursive Fortran-77 program (PRFOR). The pseudo-recursive version builds clusters in the same way as the recursive program, but uses only iteration to control program flow. Arrays of row and column numbers (x- and y-coordinates) are used to simulate a stack. Pixel coordinates are pushed onto the stack by adding elements to the arrays; coordinates are popped from the stack by decrementing the count of items, which serves as the maximum array index.

After testing for accuracy and performance on random maps of various sizes and densities, the revised algorithm proved to be an acceptable alternative for cluster identification. PRFOR was included in the revised NOYELP model and is used for performance comparisons with the parallel kernels discussed below.

Figure 5.2 traces the performance of PRFOR for 1024×1024 random maps as p value increases from 0 to 1. A direct linear relationship exists between execution time and map density. Memory requirements for arrays which record the number of elements per cluster peak near $p = 0.32$, while working stack array sizes increase with increasing maximum cluster size and p value.

5.1.3 Implementation of the Hoshen-Kopleman algorithm

Another serial algorithm for incremental cluster identification, the Hoshen-Kopleman algorithm ([HoKo76]), was implemented later in the thesis effort to serve as a basis for the hierarchically-mapped parallel version of cluster identification. This algorithm (HKFOR) traverses the map to be analyzed pixel by pixel, assigning pixels to temporary clusters as they are encountered. Two working arrays, `level` and `label`, are maintained to keep track of clusters-in-progress. `Level` is the length of a row in the

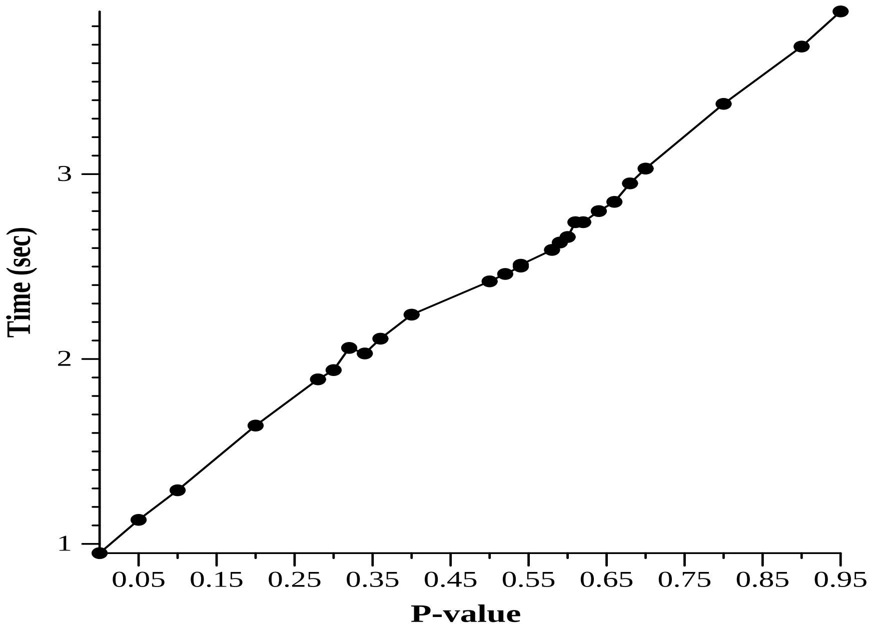


Figure 5.2: CPU time versus p-values ranging from 0 to 1 for 1024×1024 random maps using the serial PRFOR algorithm on a Sun SPARCstation 2 (excluding I/O).

main map, and records the cluster identification label of elements in the previous row analyzed. `Label` is an array which has an entry for each cluster and records an accumulating total of cluster membership. Figure 5.3 illustrates the status of these arrays after one row of the grid has been analyzed. Shaded circles in the `level` array indicate pixels which are not in the map class being analyzed.

Each cluster identification label stored in the `level` array serves as an index into the `label` array, pointing to either a positive or negative number. If the number is positive, it is the number of members to date in that cluster; if it is negative, the absolute value of the number represents the true cluster label/index. As each row of the map is traversed, the ID number of each map element is compared to that of the previous element in that row (west neighbor) and that of the element in the same column of the previous row (north neighbor). If they are all in the same cluster, the smallest label is assigned to all three elements, and any changes in the status of the west and north neighbors are recorded in the working arrays. When the map has been completely traversed, the `label` array holds all final assignments of cluster labels and the number of pixels in each cluster. This innovative 1-pass algorithm is efficient and does not require the large amount of stack space which makes recursive approaches for maps with very large clusters prohibitive on many machines.

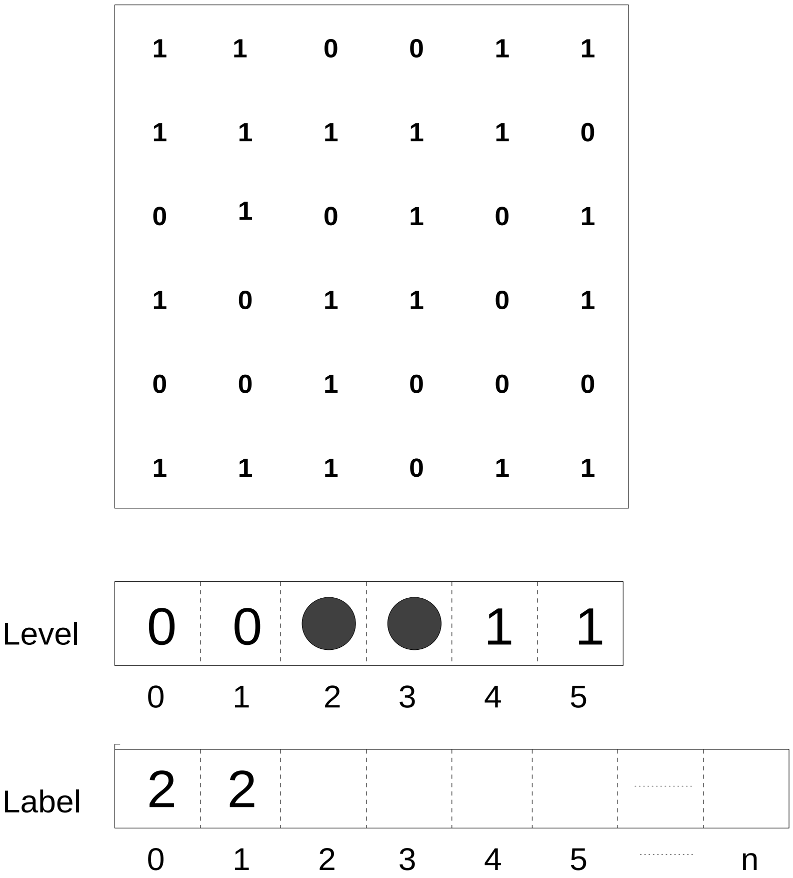


Figure 5.3: Status of *level* and *label* arrays after one row of grid has been traversed using the Hoshen-Kopleman algorithm.

5.1.4 Comparison of serial algorithm performance

Table 5.1 compares the elapsed CPU times (in seconds, excluding I/O) for these four serial cluster identification algorithms on a Sun SPARCstation 2 for seven map sizes and five p values. Execution times for all algorithms increase with map size and density.

The relative efficiency of the four cluster identification algorithms was consistent across map size and density. The incremental cluster identification HKFOR was consistently the most efficient of the four algorithms. The pseudo-recursive PRFOR sequential algorithm demonstrated efficiency comparable to that of HKFOR for maps with the lowest densities, and was marginally but consistently slower at high p values. The recursive C sequential cluster identification algorithm, RECRC, was comparable in efficiency to PRFOR and HKFOR for maps with low densities, but was clearly slower at higher p values. The incremental cluster identification algorithm ORFOR was consistently the slowest over all p values and map sizes.

Cluster identification methods which build clusters incrementally are preferred for SIMD parallel implementations because they embody more inherent parallelism. Even the *local diffusion* method, which is very slow in serial applications, is more efficient as the serial component of SIMD parallel implementations on the MasPar MP-2 than either the recursive or pseudo-recursive algorithms developed in this thesis effort. A modification of the Hoshen-Kopelman algorithm ([FlTa92]), the more efficient of the two serial incremental approaches evaluated, was chosen as the serial component in the parallel cluster identification kernels developed for this thesis.

5.2 MasPar MP-2 Algorithms

Cluster identification algorithms implemented on the MasPar MP-2 employed the two data mapping (virtualization) strategies supported by MasPar in order to handle maps larger than the 64×64 PE grid: cut-and-stack and hierarchical (see Chapter 2 for a discussion of these strategies). For both implementations, work is divided into two distinct tasks: (1) labeling cluster elements and (2) collecting information from the PEs. In Sections 5.2.1 and 5.2.2, the cluster labeling task is discussed for cut-and-stack and hierarchical data mapping, respectively. Collection of cluster data, which is similar for both mapping strategies, is discussed in Section 5.2.3.

5.2.1 Cluster labeling in implementations with cut-and-stack data mapping

In implementing cluster labeling with cut-and-stack mapping, each map pixel with a resource value greater than 0 is initially assigned a unique ID number based on its position in the map. Repetitive comparisons of north-south and east-west pairs of pixels are then made. This is very similar to the serial *local diffusion* method (also called *label propagation* ([ApCM92]), except for the fact that adjacent map pixels are on adjacent PEs, and label comparisons are made using the `xnet` communication construct. Contiguous map pixels in the same cluster are given the cluster ID corresponding to the smallest pixel label in the group of contiguous pixels.

Figure 5.4 shows a simple 4×4 map at four stages of the cut-and-stack labeling process: (a) starting state, (b) after labeling with a unique ID, and after each PE looks (c) north and (d) west. This process of comparison and relabeling continues until all adjacent cluster elements have the same label and no label updating activity is detected (using an activity flag).

Table 5.1: Comparison of CPU times for serial cluster identification algorithms on a Sun SPARC-station 2 (all times are in seconds).

Algorithm	p-value	Map Size						
		64	128	256	512	768	1024	2048
HKFOR	0.10	0.00	0.02	0.07	0.27	0.61	1.25	5.27
	0.30	0.00	0.02	0.10	0.39	0.91	1.67	7.00
	0.62	0.01	0.03	0.13	0.61	1.37	2.60	10.79
	0.85	0.02	0.04	0.19	0.76	1.69	3.23	13.08
	1.00	0.02	0.05	0.21	0.88	1.96	3.73	14.74
ORFOR	0.10	0.01	0.02	0.17	0.70	1.60	3.00	13.45
	0.30	0.02	0.05	0.27	1.13	2.54	4.59	20.00
	0.62	0.02	0.10	0.44	1.79	4.13	7.23	30.34
	0.85	0.03	0.12	0.59	2.32	5.41	9.41	38.88
	1.00	0.04	0.14	0.69	2.74	6.35	11.11	45.72
PRFOR	0.10	0.00	0.01	0.06	0.31	0.74	1.30	5.22
	0.30	0.01	0.02	0.11	0.46	1.13	1.91	7.80
	0.62	0.01	0.03	0.17	0.68	1.56	2.75	11.09
	0.85	0.01	0.04	0.22	0.88	2.04	3.56	14.62
	1.00	0.02	0.06	0.25	1.08	2.54	4.49	18.03
RECRC	0.10	0.01	0.02	0.10	0.41	0.93	1.69	10.68
	0.30	0.01	0.03	0.13	0.52	1.18	2.08	12.26
	0.62	0.01	0.04	0.17	0.70	1.62	2.87	22.32
	0.85	0.02	0.16	0.59	1.62	5.32	7.21	*
	1.00	0.03	0.23	0.92	3.76	8.78	9.38	*

*exceeds stackspace limits

HKFOR: Hoshen-Kopleman (Fortran-77)
PRFOR: Pseudo-Recursive (Fortran-77)
ORFOR: NOYELP (revised) (Fortran-77)
RECRC: Recursive (C)

5.2.2 Cluster labeling in implementations with hierarchical data mapping

The hierarchical cluster labeling strategy adapted for parallel algorithm development was influenced by the work of Tamayo ([FlTa92]) in quantum physics. The process, which involves three-steps, is shown in the context of the entire cluster identification process in Figure 5.5 and is discussed below. In Figure 5.5, the two large boxes represent adjacent PE subgrids, each with 9 data elements (map pixels). The number in the center of each small box (one map pixel) represents the resource level in the *Starting State*, and the cluster label thereafter. The number in the lower right corner of each pixel represents the unique pixel label. The circled number in the upper left corner represents the total number of pixels in a cluster (stored at the head pixel of each cluster). The box in the upper right-hand corner denotes a local cluster head and contains a local cluster sum.

1. **Step 1.** As was the case for implementations employing cut-and-stack mapping, each map pixel is initially assigned a unique ID number (label) based on its position in the map. Clusters are first resolved locally (within the subgrid of each PE) using an adaptation of the Hoshen-Kopelman algorithm ([StAh91]) for incremental cluster identification. Modifications to the serial Hoshen-Kopleman algorithm required for parallel implementation included labeling each pixel with its local cluster number and storing local cluster sums at the local head pixels for local clusters. Each pixel is assigned a pointer to the local head of its cluster (i.e., the cluster member on its PE with the smallest label).

   Alternative recursive or pseudo-recursive algorithms for this serial step in cluster labeling are not well-suited to an SIMD approach and are much less efficient. With the sequential cluster labeling pseudo-recursive algorithm, subgrid traversal stops when a local cluster head is encountered and local building of that cluster takes place. As a consequence, some processors are idle while local cluster building takes place on one or more other processors, resulting in poor performance. Since the Hoshen-Kopleman adaptation adds new members to each cluster as they are encountered in local subgrid traversal, work is distributed more evenly over local subgrids for more efficient SIMD performance.

2. **Step 2.** For each PE (in parallel), comparisons are made of local cluster labels in border rows and columns with those in immediately adjacent rows and columns of adjacent PEs The smallest cluster label is assigned across PE boundaries to border elements of the adjacent PE which are in the same cluster. Label changes are then transferred (by pointer) to the local head of the cluster to which each border pixel belongs. An activity flag is set whenever relabeling occurs.

3. **Step 3.** The process of border comparison and relabeling is continued until an equilibrium state is reached in which corresponding border cluster elements on all adjacent PE's share the same cluster label. This state is detected when no activity flags are set recording relabeling activity. Total number of pixels in each cluster are stored at the cluster's head pixel.

   The number of iterations required for complete label propagation depends on sizes and densities of the clusters. For random maps, p values near the critical region (0.5928) result in cluster characteristics requiring the maximum number of iterations for resolution. When maps are sparse, clusters are smaller and labels do not have far to propagate. When maps are denser, most pixels belong to the same cluster, requiring little relabeling.

5.2.3 Collection of cluster data

In serial implementations of cluster identification, **collection** of cluster data is accomplished within the cluster labeling function. Cluster size is accumulated in arrays with one entry per cluster. For parallel implementation, where memory is somewhat limited on both the ACU and the DPU, potentially large arrays such as these cannot be maintained. Cluster data must be maintained on the PEs, and are not collected until after cluster labeling has been completed.

For both mapping strategies, collection of cluster data is accomplished by having all members of a cluster report to the local head pixel on their PE, which then reports to the global head pixel for that cluster (i.e., the cluster member with the smallest original label over all processors, whose ID number has been used to label all other members). Local collection for hierarchical implementations is shown in Step 1 in Figure 5.5. The boxes in the upper right-hand corner of each grid cell denote local cluster sums. Implementations of both mapping strategies would be expected to benefit from the local collection of cluster information for each cluster represented on a PE before sending the sums to the head element. Local collection would be expected to improve efficiency of the hierarchical algorithm more for sparse random maps ($p = 0.10$ and 0.30) than for dense random maps because of the high likelihood that clusters would be confined to individual PEs. Local collection would improve the cut-and-stack version only for denser maps ($p > 0.59$), since only the large clusters typically found in dense maps are likely to have a significant number of members on the same processor.

For either mapping strategy, each member of a cluster can calculate the address (PE number and subgrid position) of the head pixel of that cluster from its own final cluster label, as follows:

$$\text{PE number} = \text{label} \ \texttt{mod} \ \text{nproc},$$
$$\text{layer} = \text{label} \ / \ \text{nproc},$$

where PE *number* is a unique processor identifier, *layer* is a virtualized data layer on that processor (see Chapter 2 for a discussion of virtualization), and *nproc* is the total number of PEs, which is 4096 for the MasPar MP-2. The MPL communication construct `p_sendwithAdd()` is used to report membership of the pixel to the head element of the cluster to which it belongs (by sending a 1). Sums representing total cluster membership are maintained by each head element. Global collection is denoted by circled integers in the upper left-hand corner of cells, shown in Step 3, Figure 5.5. The MPL function `reduceMax()` is then used to find the size of the largest cluster, as follows:

```
largestcluster= reduceMax(clustersize).
```

where `largestcluster` is a singular variable and `clustersize` is a plural value allocated on all PEs. Number of clusters is determined by counting head elements, as follows:

Table 5.2: Speed improvement of MasPar MP-2 versions over pseudo-recursive Fortran version on a SPARCstation 2 for cluster identification (excluding I/O).

Mapping strategy	p value	Map Size 64	128	256	512	768	1024	2048
Cut-and-stack	0.10	0.00	0.62	4.00	2.48	1.74	1.27	0.41
	0.30	0.83	1.25	2.35	2.19	1.87	1.34	0.55
	0.62	0.23	0.35	0.47	0.54	0.60	0.50	0.34
	0.85	0.18	0.51	1.13	1.33	1.42	1.36	1.24
	1.00	0.39	0.73	1.47	1.95	2.14	2.14	2.23
Hierarchical	0.10	0.00	2.50	5.00	8.21	8.44	7.06	4.81
	0.30	0.83	2.50	7.50	7.12	6.89	5.31	3.24
	0.62	0.21	0.73	1.67	3.16	4.08	4.04	2.14
	0.85	0.20	0.65	3.14	8.98	12.68	15.08	12.40
	1.00	0.36	1.02	4.58	15.41	28.83	38.84	52.29

```
if(myclusterlabel==myoriginallabel) reduceAdd32(one).
```

The average cluster size is simply calculated by dividing the number of non-zero elements by the number of clusters.

In the early stages of parallel algorithm development, collection time dominated cluster identification time, largely because of the improper functioning of the initial MasPar MPL version of the `p_sendwithAdd()` function (used to collect data on head elements). The error was reported to MasPar, and a less efficient collection strategy which required all receiving PEs to be in an active state was devised as a temporary alternative. The `p_sendwithAdd()` error was corrected in subsequent software releases, and modifications to the collection algorithm utilizing the corrected function resulted in a substantial decrease in cluster data collection time.

5.3 Results

For both parallel cluster identification programs implemented on the MasPar MP-2 (i.e., hierarchical and cut-and-stack), total elapsed wall-clock time was compared with that of the pseudo-recursive serial Fortran version (PRFOR) on a Sun SPARCstation 2 for random maps of seven sizes and five densities (Table 5.2). For these comparisons, *work* time is defined as the sum of *label* and *collect* times for cluster identification, excluding time for reading data from a binary input file and writing results to the screen (I/O). Speed improvements are calculated by dividing work time for the serial program by work time for each parallel implementation. Tables C.1 through C.3 in Appendix C list actual read, work, and total wall-clock times for both MasPar MP-2 versions and for PRFOR.

Trends in performance of the two parallel kernels on these random maps across

p values within a given map size and across map sizes within a given p value were generally consistent. For the smallest maps (i.e., 64×64), both parallel algorithms were generally less efficient than the serial algorithm (i.e., speed improvement values less than 1.00). The cut-and-stack algorithm was also slower than the serial algorithm for maps with p values near the percolation threshold ($p = 0.62$) regardless of map size, and for all 2048×2048 maps of densities $<= 0.62$. In contrast, the hierarchical algorithm showed speed improvements over the serial algorithm for all p values for map sizes larger than 128×128.

Figure 5.6 graphically presents the speed improvements of the hierarchically-mapped MasPar MP-2 implementation for these same mapsizes and p values. Of the five densities considered, the worst parallel algorithm performance was typically seen for maps with $p = 0.62$, near the percolation threshold (Table 5.2 and Figure 5.6). For both parallel algorithms, an overall trend of maximum speed improvement at smaller than maximum map size was evident for all but the densest maps (i.e., those with $p = 1.00$). The greatest speed improvement for the hierarchical algorithm (52.29) was observed for the densest and largest map, while the greatest speed improvement for the cut-and-stack algorithm (4.00) was observed for the sparsest map (i.e., $p = 0.10$) of size 256×256.

For all random maps which were virtualized (i.e., those larger than 64×64), the hierarchically-mapped kernel was more efficient than its cut-and-stack counterpart, with the relative performance of the hierarchical algorithm generally increasing with increasing map size and density (Table 5.2). For sparse maps ($p = 0.10$ and 0.30), hierarchical mapping benefited from the local collection of cluster data on PEs, which reduces the number of `p_sendwithAdd`s required to transmit data to the head element of each cluster when cluster elements are sparsely distributed. Local collection is useful for cut-and-stack mapping only when clusters are large enough to have multiple members on each PE. For dense maps dominated by a single large cluster ($p \geq 0.62$), the cluster labeling component dominates cut-and-stack execution time. The hierarchical mapping strategy outperforms cut-and-stack mapping for these denser maps because it labels large clusters more efficiently.

An increase in the speed of `xnet` communications projected for future MasPar releases would improve the relative performance of the cut-and-stack algorithm, which requires more inter-processor communication than the hierarchical algorithm.

Figure 5.7 shows how the labeling and collection components of cluster identification using the two mapping strategies perform as p values increase from 0 to 1 for maps of sizes 256×256 and 2048×2048. Elements (pixels) for these graphs were produced by MPL programs using the MPIPL routine `mpigenrand` to generate and distribute appropriate random values on MP-2 PEs for each p value. Note the y-axis (time) scale differences for graphs showing results from the two mapping strategies. This reflects the consistently better performance of the hierarchical algorithm across all p values and both map sizes considered. Hierarchical total time is superimposed on cut-and-stack graphs for comparison. Performance differences for the hierarchical and cut-and-stack algorithms are most evident for cluster labeling of maps with densities near the $p = 0.59$ threshold, especially for the larger maps.

A comparison of these results with the changes seen in cluster characteristics across p values for random maps (e.g., compare Figure 5.7 with Figure 4.2(a) in Chapter 4) illustrates their relative importance in determining execution times of cluster identification algorithms. For the random maps analyzed, peak execution (i.e., work) times for both parallel cluster identification algorithms occur at p values near the 0.59 threshold, when the maximum cluster size begins to climb dramatically and the number of clusters is still relatively large. This is the point at which the maximum amount of border updating and label reassignment within PE subgrids is required. Execution times decrease as map density increases and one cluster becomes dominant. Under these conditions, most PE subgrid elements and border elements on adjacent PEs belong to the same dominant cluster, requiring much less border updating and label reassignment.

Memory requirements for the two SIMD algorithms are constant for a given map size over all p values. This is in sharp contrast to results for the serial algorithm PRFOR, which is characterized by increasing CPU time and memory requirements as the maximum cluster size increases (see Figure 5.2). These differences in the relationship of execution time to cluster characteristics for serial and parallel implementations largely explains why speed improvements of parallel kernels were generally lowest for random maps with p values near the percolation threshold. In fact the cut-and-stack cluster identification algorithm was slower than the serial Fortran algorithm at $p = 0.62$ for all map sizes (Table 5.2). By studying these relationships during code development and by anticipating cluster characteristics of non-random maps for a particular application, parallel processing bottlenecks can be pinpointed, and an optimized cluster identification strategy can be developed for each parallel application.

A more efficient cut-and-stack approach to cluster identification has been proposed ([ApCM92]) which includes power-of-2 neighbors (on processors which are 2^n units away, $0 < n < m - 1$, where map size $= 2^m \times 2^m$) in the list of neighbors which are checked on each iteration of cluster label updating. This approach should reduce the number of iterations required for complete label propagation for the dendritic clusters characteristic of maps with p values near the critical threshold.

5.4 Performance of MasPar MP-2 algorithms on NOYELP maps

5.4.1 Test map characteristics

Figure 5.8 illustrates the temporal pattern of density variation (expressed as p value) in maps of available biomass (resources) generated during a 180-day cycle of the NOYELP model. Resource maps of the study area extracted from the same 180-day cycle of the NOYELP model at day 1, day 90, day 120, and day 180, respectively, are presented in Figures 5.9 through 5.12.

All portions of the maps which are light green represent available resource pixels. Pixels which have high resources levels, defined as sufficient available biomass to satisfy at least 50% of the daily forage intake requirements of a bison bull, are represented by red.

At the beginning of the model year (November), resource levels, p values, and maximum cluster size are all high (Figures 5.8 and 5.9). As winter progresses, resources are depleted by the grazing of ungulates or are made inaccessible to the ungulates by heavy snowfall. This is reflected in lower p values for the resource maps (Figure 5.8), smaller, more fragmented clusters, and lower levels of available biomass (Figure 5.10 and 5.11). The model year ends in April before spring regeneration of vegetation begins; however, melting snow exposes ungrazed resources, causing map density to increase near the end of the 6-month cycle (Figures 5.8 and 5.12).

5.4.2 Results

Cluster identification is performed twice at each time step in the NOYELP model, once for clusters of any available resource level and again for clusters with available resources above a fixed limit (high resources).[1] Table 5.3 compares performance of the two MPL cluster identification algorithms with that of the serial NOYELP algorithm (PRFOR) on 285×584 resource maps extracted from the NOYELP model runs for seven days in the 180-day cycle. Maps were expanded to 320×640 for input to the MasPar MP-2 kernels, so that row and column dimensions are multiples of 64, the size of the PE grid. The p values calculated in Table 5.3 are based on the total number of map pixels in the expanded resource map, and include pixels outside the study area, while the graph of p values in Figure 5.8 includes only study area pixels.

Wall-clock time for the hierarchical parallel implementation is typically 4 to 5 times faster than that for the serial algorithm, while the cut-and-stack parallel implementation is slower than the serial algorithm for most maps. These results are not inconsistent with the relative performance of the parallel kernels on random maps of

[1] This is a simplification of what the actual resource matrix represents. Cluster analysis is performed on the matrix of feedback modifiers which limit the maximum daily intake of forage by ungulates, based on limitations associated with levels of forage biomass and snow depth/density.

Table 5.3: Comparison of wall-clock times for parallel implementations of cluster identification with the serial NOYELP function PRFOR on Sun SPARCstation 2 on 285 × 584 resource maps extracted from NOYELP model runs (all times are in seconds).

Time step	Resource level	p value	Largest Cluster	Serial PRFOR	Parallel hierarchical	Parallel cut-and-stack
1	any	0.30	55554	0.41	0.08	0.41
1	high	0.27	51035	0.38	0.08	0.43
30	any	0.30	55554	0.40	0.08	0.41
30	high	0.27	51045	0.37	0.08	0.43
60	any	0.29	53800	0.39	0.07	0.40
60	high	0.22	29945	0.33	0.07	0.45
90	any	0.18	16515	0.31	0.06	0.50
90	high	0.04	4701	0.21	0.05	0.18
120	any	0.06	6158	0.23	0.05	0.23
120	high	< 0.01	394	0.19	0.03	0.04
150	any	0.03	6153	0.23	0.05	0.23
150	high	< 0.01	0	0.19	0.02	0.02
180	any	0.29	55190	0.39	0.07	0.41
180	high	0.14	4672	0.28	0.06	0.47

similar map size and maximum cluster size, except for the relatively poor performance of the cut-and-stack implementation compared to PRFOR. Future work in this area should involve a closer examination of the the range of cluster configurations and characteristics encountered in landscape maps, the relative performance of parallel kernels on these configurations, and methods of optimizing parallel performance over the range of configurations.

5.5 Conclusions

Speed improvements of SIMD parallel algorithms on the MasPar MP-2 over serial algorithms for cluster identification are modest, largely because of the high communication requirements associated with labeling of pixels. These results indicate that the serial algorithms developed and evaluated are efficient enough for many purposes. However, the hierarchical parallel algorithm consistently outperformed both the serial and cut-and-stack algorithms, and could be useful in applications such as the NOYELP model which call cluster identification functions many times within a single program execution. NOYELP identifies clusters twice at each time step (once for high resource patches and once for patches with any resource) over 180 time steps and over 5 replications of the 180-day cycle. This requires a total of 1800 calls to the cluster identification function. Modest time savings per cluster analysis can result in significant savings in total execution time for models such as NOYELP. For a typical serial NOYELP simulation involving 20,000 ungulates, approximately 20% of total execution time for the revised model is spent identifying clusters (see Chapter 7 for a discussion of serial modifications). A simulation involving fewer animals would require that less model time be spent in the animal movement component and proportionately more time be spent in cluster identification. The presence of fewer animals would also mean less resource depletion due to grazing, and hence more pixels with available biomass (i.e., higher p values) over the course of model execution. This would further increase the proportion of total execution time allocated to cluster identification. Therefore, the fewer animals input to NOYELP, the more important the efficiency of the cluster identification component becomes.

To utilize the parallel cluster identification kernels for NOYELP model simulations, the serial Fortran-77 NOYELP program could be run on the front end machine of the MasPar MP-2 (DECstation 5000-200 workstation) and make calls to the MPL cluster identification function running on the DPU, which would then *blockIn* the data matrix to be analyzed. Since cluster identification results do not feed back into the main NOYELP program (results are stored in an array for later output to a file), an asynchronous call via `callAsync()` would allow the main program to continue execution on the front end while cluster identification is being accomplished on the DPU. Output from the parallel cluster identification kernels could be input to other parallel modules to compute various cluster geometry indices, such as mean squared radius, on distributed data on the DPU. Mean squared radius algorithms are discussed in detail in the following chapter.

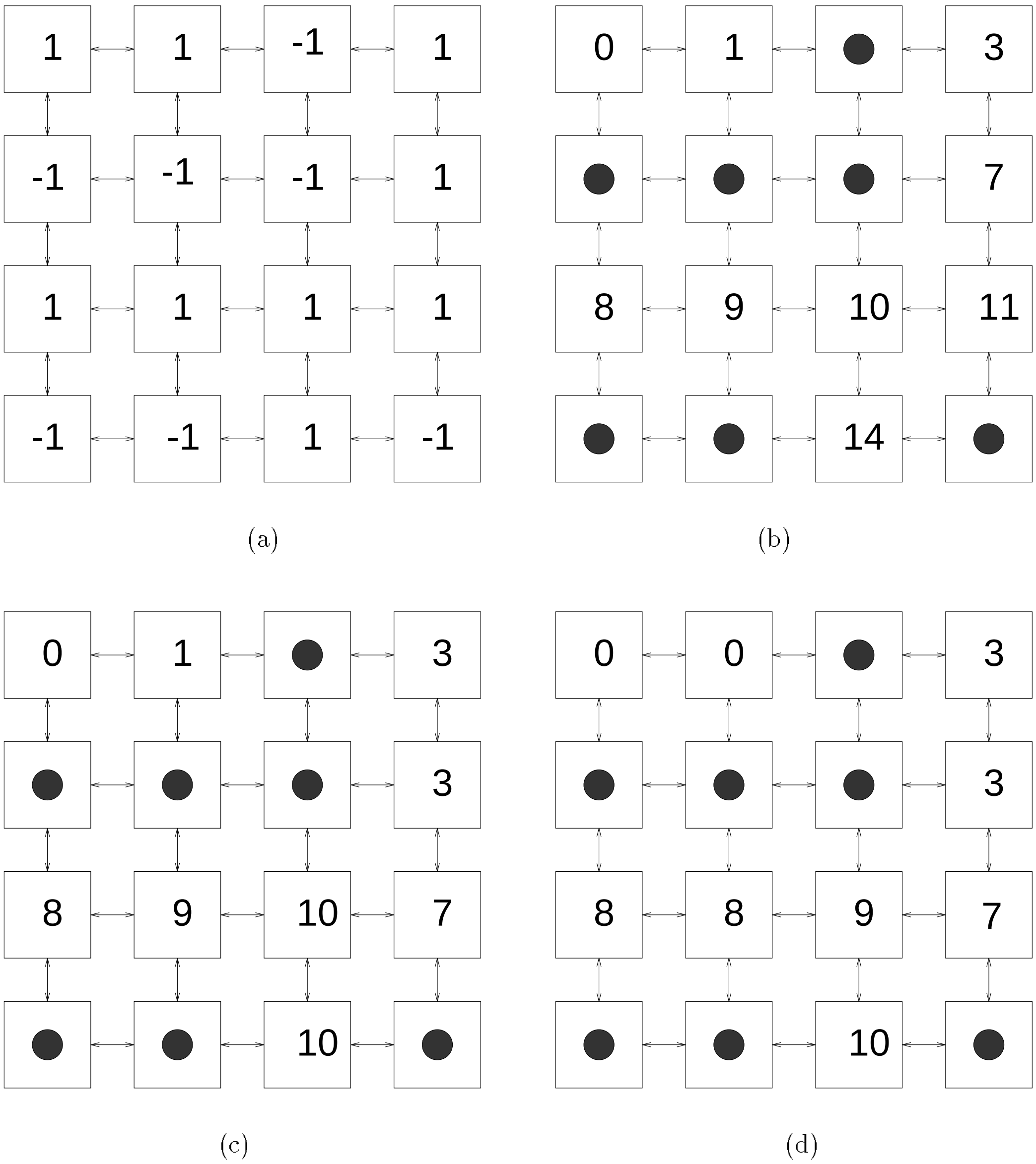


Figure 5.4: Four stages of the cut-and-stack labeling process: (a) starting state, (b) after labeling with unique ID, and after each PE looks (c) north and (d) west.

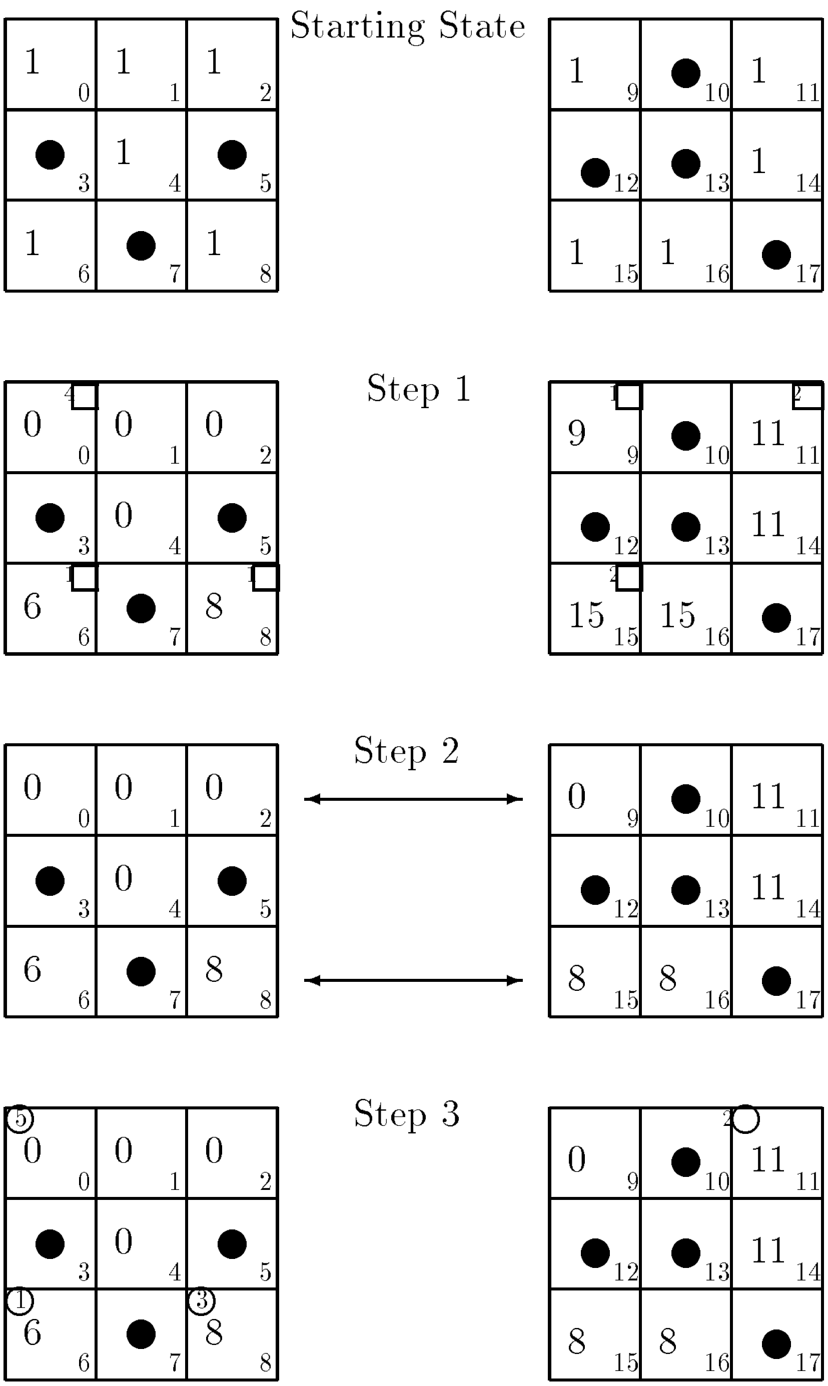


Figure 5.5: Step-wise procedure for cluster identification in hierarchically-mapped MPL implementation of cluster identification.

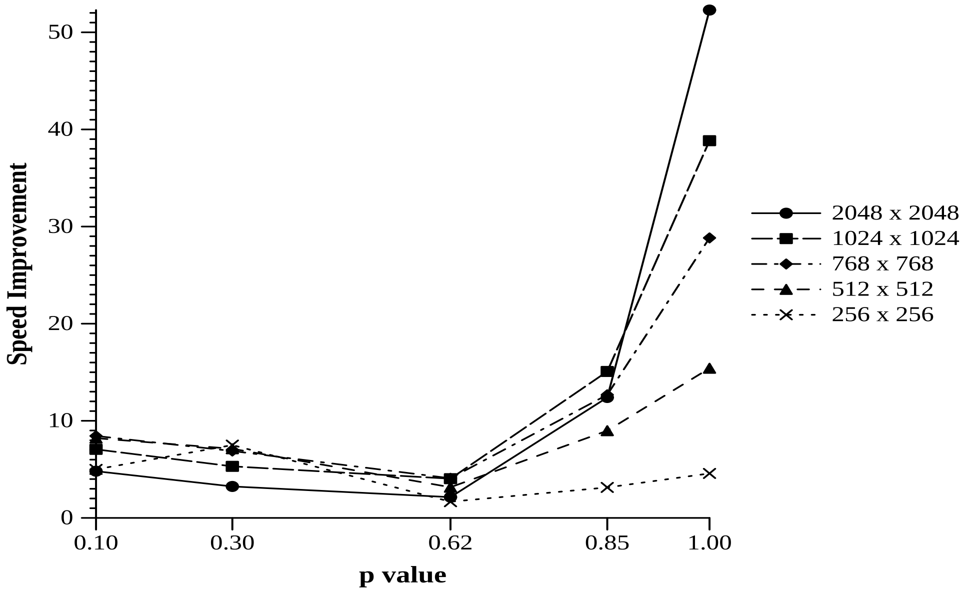


Figure 5.6: Speed improvement of MP-2 hierarchically-mapped implementation over the sequential PRFOR version on a SPARCstation 2 for cluster identification (work time, excluding I/O).

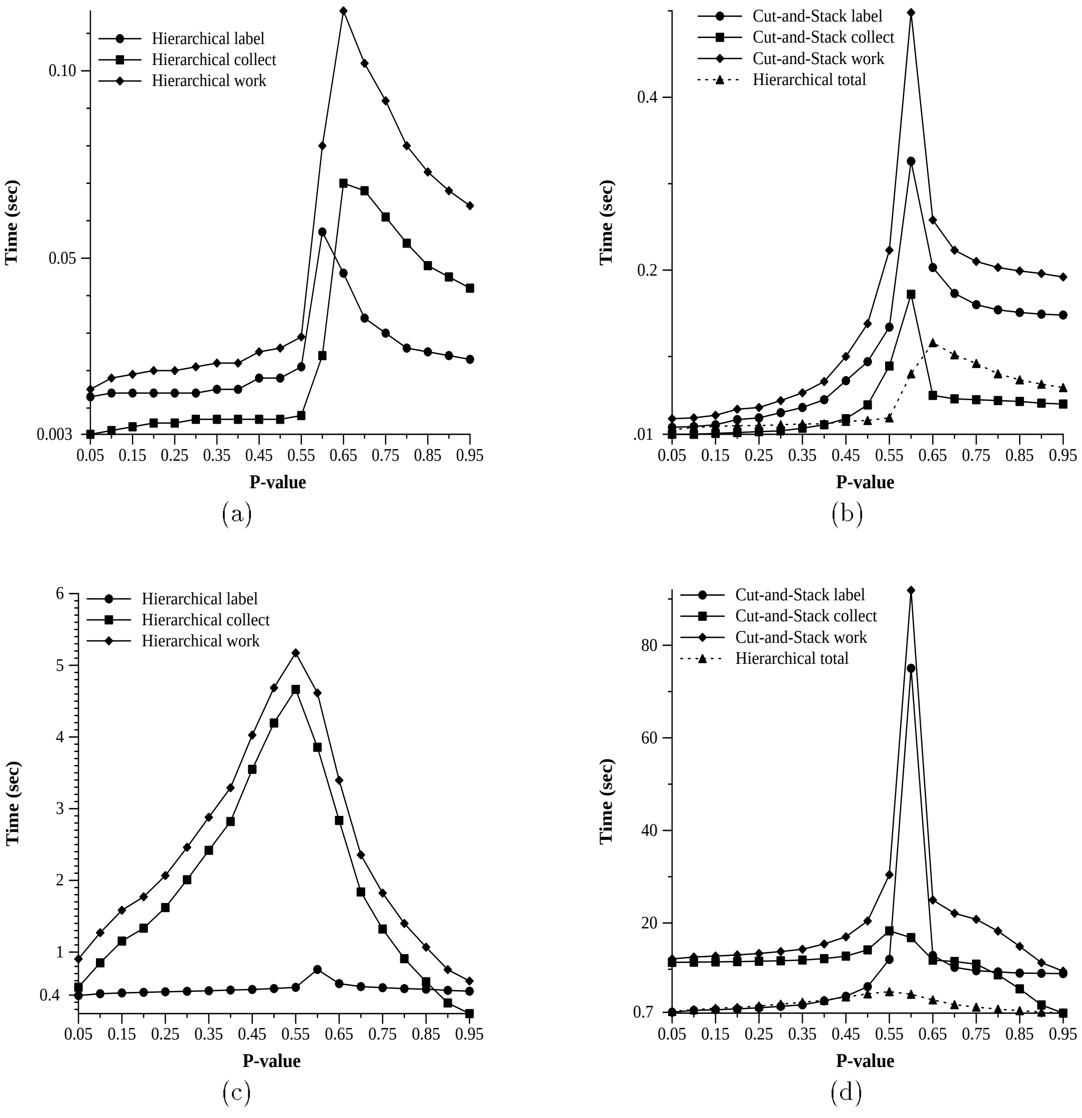


Figure 5.7: Comparison of wall-clock labeling, collection, and total work times versus p values for 256×256 maps ((a) and (b)) and 2048×2048 maps ((c) and (d)) using hierarchical ((a) and (c)) and cut-and-stack ((b) and (d)) virtualization.

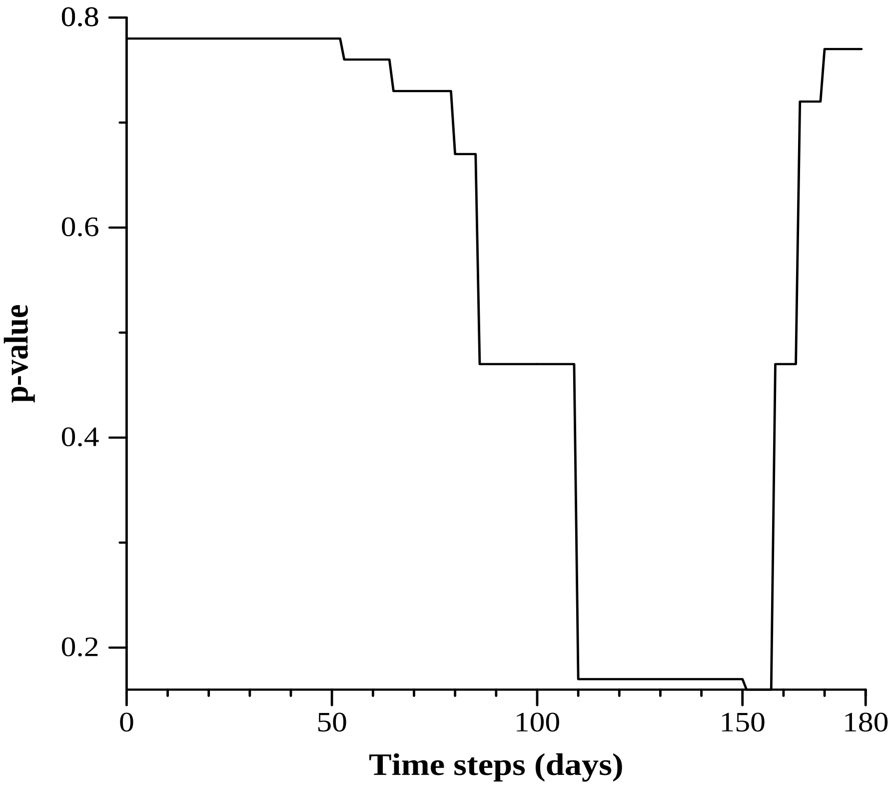


Figure 5.8: P values of the available resource matrix generated daily for a 180-d ay cycle of the NOYELP model (excluding pixels outside the study area).

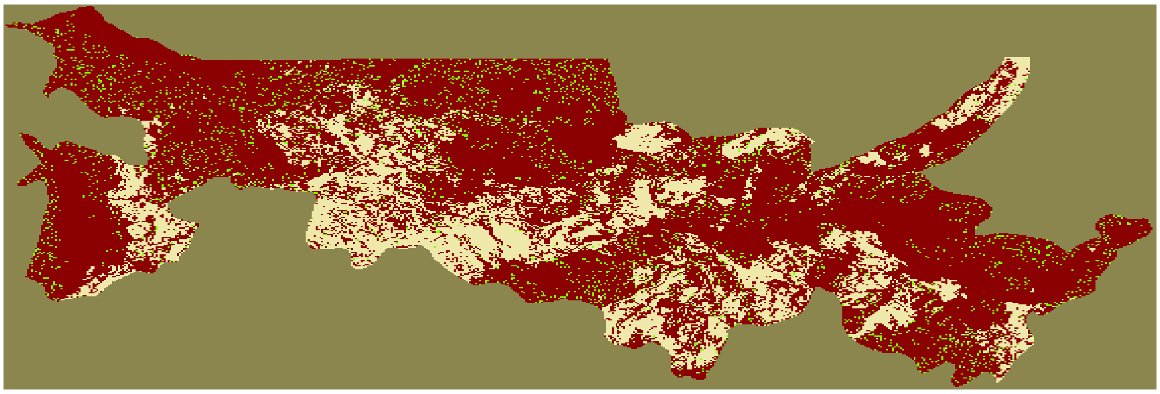


Figure 5.9: Resource map of the NOYELP study area extracted at day 1 from a 180-day cycle of the NOYELP model.

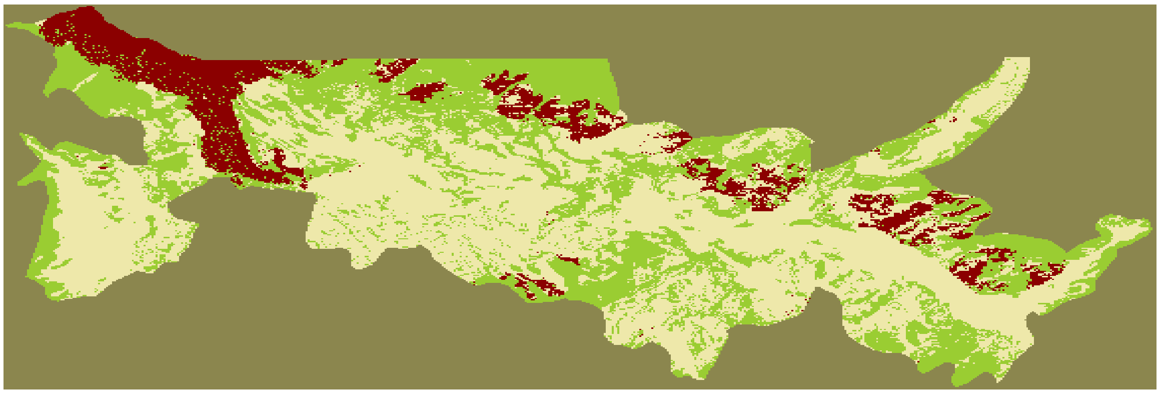


Figure 5.10: Resource map of the NOYELP study area extracted at day 90 from a 180-day cycle of the NOYELP model.