Computational Models of White-Tailed Deer in the Florida Everglades

C.A. Abbott, M.W. Berry, E.J. Comiskey,
J.C. Dempsey, L.J. Gross, & H.-K. Luh

Computer Science Department

CS-95-296 August 1995

Submitted to IEEE Computational Science & Engineering, (1995)

Computational Models of White-Tailed Deer in the Florida Everglades*

C.A. ABBOTT, M.W. BERRY, J.C. DEMPSEY
Department of Computer Science, University of Tennessee, Knoxville TN 37996-1301, {**cabbott,berry,dempsey**}**@cs.utk.edu**

E.J. COMISKEY, L.J. GROSS, AND H.-K. LUH
Institute for Environmental Modeling and Department of Mathematics, Knoxville TN 37996-1300, {**ecomiske,gross,luh**}**@math.utk.edu**

Abstract. Computer models can be used to simulate the interactions between animals and their environments. The SIMPDEL (**S**patially-Explicit **I**ndividual-Based Simulation **M**odel of Florida **P**anther and White-Tailed **D**eer in the **E**verglades and Big Cypress **L**andscapes) model has been developed to analyze and predict the effects of alternative water management scenarios in South Florida on the long-term populations of white-tailed deer and Florida panther. This model simulates the aging, reproduction, foraging, growth, and mortality of individual animals over a period of 23 years. The primary focus of this work is the use of parallel processing in the three main computational components of SIMPDEL: hydrology, vegetation, and deer foraging. Very similar results for both sequential and parallel SIMPDEL models have been obtained and speed improvements ranging from 8.9 to 27.0 were achieved for the parallel model over the sequential model executing on a Sun SPARCstation 5.

Keywords: computer modeling, Florida Everglades, individual-based, parallel computing, spatially-explicit, white-tailed deer

1. Introduction

The Florida Everglades is considered to be one of the most threatened ecosystems in the nation, due in part to decades of intense, adverse water management impacts [4]. Restoration alternatives are currently being developed to restore natural water flows to the area. Hence, there is a need to predict and compare the effects of alternative hydrologic restoration scenarios on the Everglades wildlife. Future projections of these effects can be realistically accomplished only by computer modeling [8].

The SIMPDEL (**S**patially-Explicit **I**ndividual-Based Simulation **M**odel of Florida **P**anther and White-Tailed **D**eer in the **E**verglades and Big Cypress **L**andscapes) model was jointly developed by a group of modelers and biologists at the Institute for Environmental Modeling, University of Tennessee [3]. SIMPDEL is a member of the ATLSS family of models (**A**cross-**T**rophic-**L**evel **S**ystem **S**imulation for the Everglades and Big Cypress Swamp) under development at a variety of institutions with support from the National Biological Service. Upon completion, the SIMPDEL model will be used to analyze the effects of alternative water management scenarios on the long-term populations of white-tailed deer and Florida panther in the Everglades. The model tracks the

* This research has been supported by the National Science Foundation under Grant No. NSF-BIR-93-18160. The serial implementation of the model was developed with support from the National Park Service through Cooperative Agreement No. CA-5460-0-9001.

growth, movement, foraging and reproduction of individual deer and panther across South Florida as affected by the underlying landscape, the spatial dynamics of hydrology, and the interactions between individuals. Typical model simulations include $20,000$ deer and 50 panthers, with the population sizes varying throughout the simulation due to stochastic mortality and fecundity.

Execution time of the SIMPDEL model increases as the number of deer increases and as the abundance of forage decreases. For example, execution times ranging from 16.5 to 39 hours were recorded for increasing population sizes. Hence, a primary goal of this work is to reduce substantially execution time of the sequential SIMPDEL model via parallel programming while producing population trends similar to those produced by the current sequential model. An additional design goal is to construct the parallel model in such a way as to ensure that the same results will be achieved regardless of the number of processors used, in order to simplify code conversion for a future implementation of the parallel model designed for a distributed network of workstations using Parallel Virtual Machine (PVM) [9]. As the panther component of the sequential SIMPDEL requires minimal CPU time, the focus of this study is the parallelization of the deer portion of the model.

The parallel computing environment used in this research is described in Section 1.1. Section 2 introduces the sequential SIMPDEL model and describes the Everglades study area. Section 3 discusses the parallel SIMPDEL model, and Section 4 presents results and speed improvements of the parallel model over the sequential model. Finally, a summary and future work considerations are provided in Section 5.

1.1. Computing Environment

The parallel machine used in this study was a Thinking Machines Corporation CM-5. The CM-5 is a general-purpose multicomputer providing both SIMD and MIMD capabilities, allowing users to write both data parallel and message-passing programs. The CM-5 at the University of Tennessee, Knoxville contains 32 *processing nodes* (although tens to thousands are possible) connected via a fat-tree architecture. Processing nodes (PNs) are the processors that perform actual computations on parallel data and communicate with each other as necessary, to share data [14]. Each PN has a 32 MHz SPARC 2 processor, 32 Mbytes of memory, and a 128 Mflops vector processing unit capable of performing 64-bit floating-point and integer operations [11]. Control processors (CPs) manage the processing nodes and I/O devices.

The CM-5 supports two programming models: hostless and host/node. The host/node model involves two programs executing simultaneously, one on the host and a second executing on each processor. The host performs necessary initializations and then invokes the node program. In the hostless programming model, the host is utilized only as an I/O server and to initiate and terminate program execution. A single program runs independently on each node, and communication is achieved through message-passing [13]. For parallelizing SIMPDEL, the hostless programming model was used. Message-passing was accomplished through the use of communication routines supplied by the CMMD library, version 3.0.

1.2. Notations

In the following sections, PN_i refers to the current processor only, and PN_j and PN_k refer to any other processor. A $500m \times 500m$ grid cell is referred to as a $500m$ grid cell, and a $100m \times 100m$ grid cell is similarly referred to as a $100m$ grid cell.

2. Sequential Everglades Model

SIMPDEL is a spatially-explicit individual-based simulation model developed to analyze the effects of alternative water management strategies on the long-term populations of white-tailed deer and Florida panther in the Florida Everglades. An individual-based model is one in which each member of a population (and its interactions with other members) is simulated individually [5]. There are four tightly coupled components of the SIMPDEL model: hydrology, vegetation, deer, and panthers. Hydrology inputs influence vegetation growth and restrict animal movement. Deer depend on high quality vegetation for maximum energy intake and growth, and panther depend on deer for prey. The focus of this work is on the first three components: hydrology, vegetation, and deer. For comparison with the parallel implementation, to be described in Section 3, the sequential SIMPDEL program was executed without simulating a panther population.

2.1. Study Area

The portion of the Everglades used in this model encompasses about 7,500 mi^2, from Lake Okeechobee to Florida Bay. The area includes eighteen vegetation types (Figure 1), with the most prevalent being freshwater marsh and wetlands. The climate in South Florida is subtropical, with a dry season from mid fall through late spring and a wet season during the summer. Precipitation is a major route by which water enters the Everglades, with approximately 75% falling from May through October [6]. Average annual rainfall in the area ranges from 55-65 inches. The study area terrain is almost flat, with elevations ranging from $10m$ in the north to $1m$ in the south [3].

2.2. Landscape Representation

Since the study area is large, the landscape is modeled at two spatial scales: $100m$ grid cells and $500m$ grid cells, where each $500m$ grid cell represents 25 $100m$ grid cells. At the $500m$ scale, the landscape is composed of 68,085 grid cells represented by a 420 row (**`RPIXEL`**) by 265 column (**`CPIXEL`**) grid cell map. Data maps containing vegetation types, forage quantities, and water depths are maintained at both spatial scales. Maps at the $500m$ scale are represented as two-dimensional arrays (**`RPIXEL`** $\times$ **`CPIXEL`**), and maps at the finer $100m$ scale are represented as three-dimensional arrays (**`RPIXEL`** $\times$ **`CPIXEL`** $\times$ 25) in the sequential model.

2.3. Hydrology Component

The water level inputs to the SIMPDEL model were derived from the South Florida Water Management Model (SFWMM), developed by the South Florida Water Management District and Everglades National Park. SFWMM is a large-scale mathematical model of the present network of canals, structures, levees, and pumps that make up the highly managed water system controlling water levels and flows throughout the area [7]. The current water level inputs are averaged weekly data from the period 1966 through 1989, although daily data are also available. The spatial level of resolution is at the $2mi$ scale, which is too coarse to model individual animal movements. Therefore, spatial data interpolation is used to redistribute water levels at the $2mi$ scale to the $100m$ scale, based on expected depths for the vegetation types represented in each $2mi$ cell. This results in a $100m$

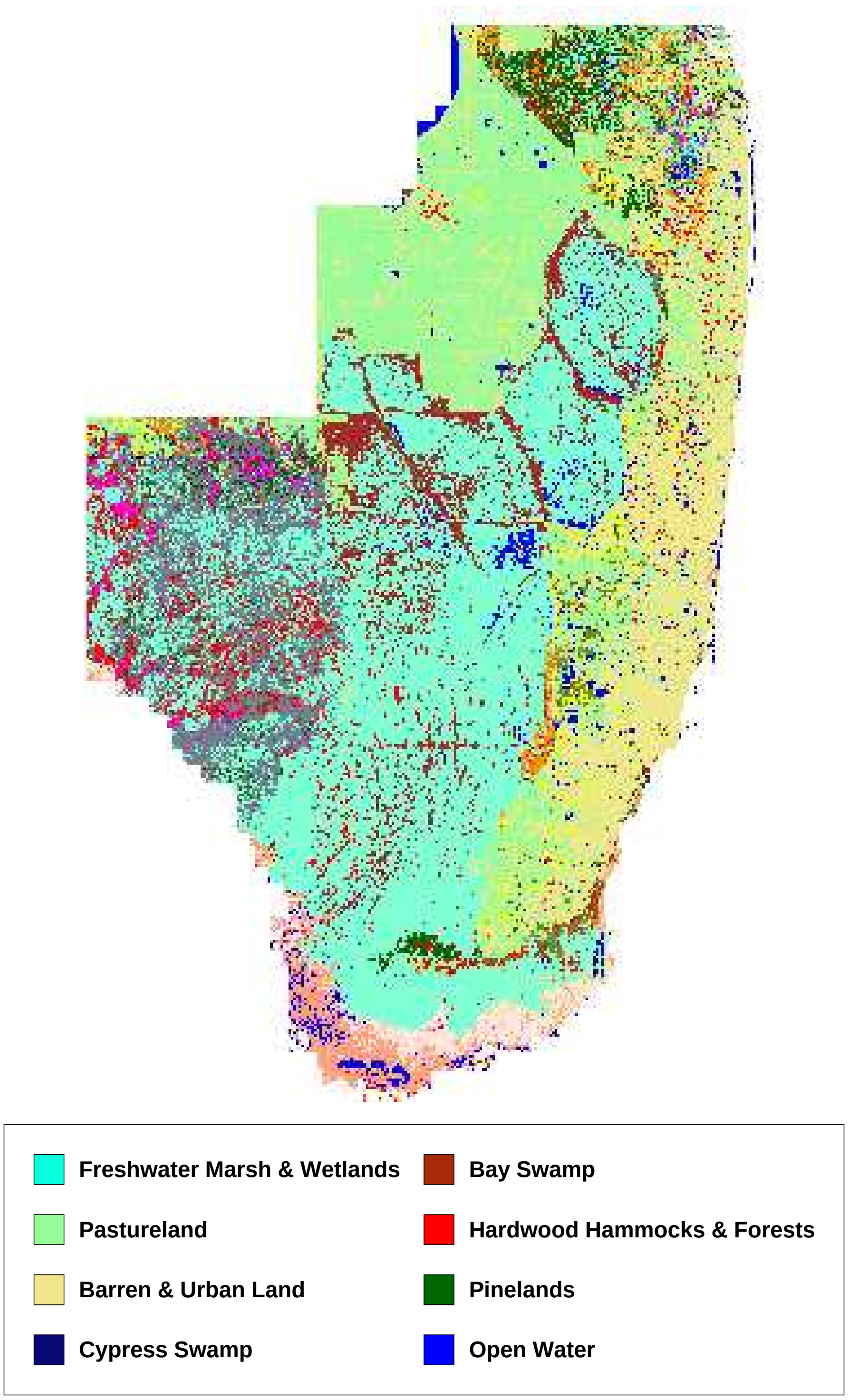


Figure 1. SIMPDEL landscape with 18 vegetation types. The legend provides a description of some of the most significant areas.

resolution map that approximates the water level in each vegetation cell. Water levels at the $500m$ scale are then computed by averaging the levels in the 25 constituent $100m$ cells.

2.4. Vegetation Component

The vegetation distribution data used in the model were derived using the South Florida portion of the Florida Department of Transportation vegetation map, which is a satellite image at the $30m$ scale. Vegetation data is represented at both the $100m$ and $500m$ scales, which are aggregations of the $30m$ satellite image [8].

Each $100m$ and $500m$ grid cell contains two quality classes of forage: high quality and medium quality. High quality forage has a higher caloric value, but is lower in quantity. Medium quality forage, on the other hand, has a lower caloric value and is higher in quantity. In addition to high and medium quality forage, there is unlimited low quality forage of the least caloric value. This class of forage can satisfy only about 80% of a deer's daily energy requirements [3].

On the first day of the simulation, initial quantities of high and medium quality forage are assigned to each $100m$ grid cell based on its vegetation type and average water level for that year. These quantities are updated at the $100m$ scale daily, weekly, or monthly for growth effects, depending on a user-selected option. Depending on the time of year, forage amounts either increase or decrease due to seasonal vegetative growth and deterioration. After computing available quantities at the $100m$ scale, $500m$ forage quantities are computed by averaging values in the 25 constituent $100m$ cells. In addition to seasonal growth effects, forage amounts in specific $100m$ and $500m$ grid cells are updated daily after each deer grazes.

2.5. Deer Component

The white-tailed deer (*Odocoileus virginianus seminolus*) is the only large herbivore in South Florida [8]. Everglades deer are considerably smaller than deer found farther north, at about two-thirds the weight of those found in Pennsylvania or Wisconsin. Deer are most numerous in the prairies and rocky pinelands. The animals often wade in water to feed on marsh plants; however, extremely high water may force them onto elevated tree islands, possibly resulting in death by starvation if flooding is prolonged. Deer are a major prey item for the endangered Florida panther. A panther will kill about one deer per week when hunting is good [15].

The SIMPDEL deer component simulates the aging, reproduction, foraging, growth, and mortality of each individual deer on a daily time step over a period of 23 years (the time period over which hydrology data are available). The deer component is driven by hydrology and vegetation inputs. High water levels restrict deer movement and water levels also restrict vegetation growth.

At the start of the simulation, an initial population of deer is generated using a user-selected population size read from an input file. Age, location, sex, body weight, mating day, and other characteristics are determined randomly and stored in a data structure for each individual.

2.5.1. Fawn Independence and Dispersal

At age eighteen months, male deer become independent and disperse from the natal range. Dispersal location is determined randomly using the deer's current location and a maximum dispersal distance. Potential grid locations for dispersal are continually generated up to a maximum number or until a *suitable dispersal location* is found. A suitable dispersal location is a 500m grid cell that contains high quality forage, has a water depth below the maximum level a fawn can withstand, and has no deer already assigned to that grid cell location. If no suitable grid cell is found, the fawn will remain at its current position. A female deer at eighteen months of age will remain at the same location as the mother, but will forage independently and move to a new area if forage is scarce [8]. Fawn independence under the age of eighteen months is also possible, and occurs either when the mother gives birth to new fawns, or when a female fawn still under the care of its mother gives birth to fawns.

2.5.2. Mating and Reproduction

Successful reproduction depends on several factors. Pregnancy can occur only on a female's mating day and depends on her age and health, as determined by comparing her body weight with her maximum weight ever attained. In addition, an available male deer who has not mated within the rutting period (currently set at one year) must be located within mating distance from the female's location. If no male is found, the female does not become pregnant, her mating day is incremented by 28 days, and the process repeats itself each month until pregnancy results.

2.5.3. Forage Search and Movement Rules

White-tailed deer are extremely efficient foragers, eating highest quality vegetation first before eating that of a lower quality. An algorithm to simulate this behavior, which is based on the assumption that deer find the best quality forage [8], is illustrated below.

Each deer's daily search for food begins at its current location. If the current location contains *available forage* above the threshold, **`ALPHA_1`**, deer grazing is simulated on the constituent $100m$ cells within the current $500m$ location, effectively reducing the forage levels at both resolutions. Available forage within a $500m$ grid cell is a function of the actual forage level, water level, and the maximum water depth that the deer can withstand. Since maximum water depths differ for bucks, does, and does with fawns, available forage levels in the same grid cell will vary for different deer. If the available forage level in the current grid cell is less than the threshold, or there is not enough forage in the cell to satisfy the deer's maximum daily intake, which is a function of body weight, a search of the concentric squares centered at the deer's current location is performed, (see Figure 2(a)) for the nearest $500m$ grid cell with the maximum level of available high quality forage. If multiple grid cells along the same concentric square perimeter have the same maximum available forage level, one of them is chosen at random. The deer is then assigned to this new location and its daily travel distance incremented by the search radius (the number of grid cells between the old and new locations). If the chosen cell does not contain enough high quality forage above the threshold to satisfy the deer's maximum daily intake, grazing upon *medium* quality forage within the same grid cell is simulated. If the maximum forage intake is still not satisfied, the search for high quality forage will continue up to the deer's maximum moving distance.

If the maximum moving distance is reached while searching for high quality forage, but no forage above the threshold has been found within the last search perimeter, the forage search is restarted from the deer's current location. This time, however, the search will proceed among the concentric squares for a $500m$ cell with available *medium* quality forage above a new threshold,

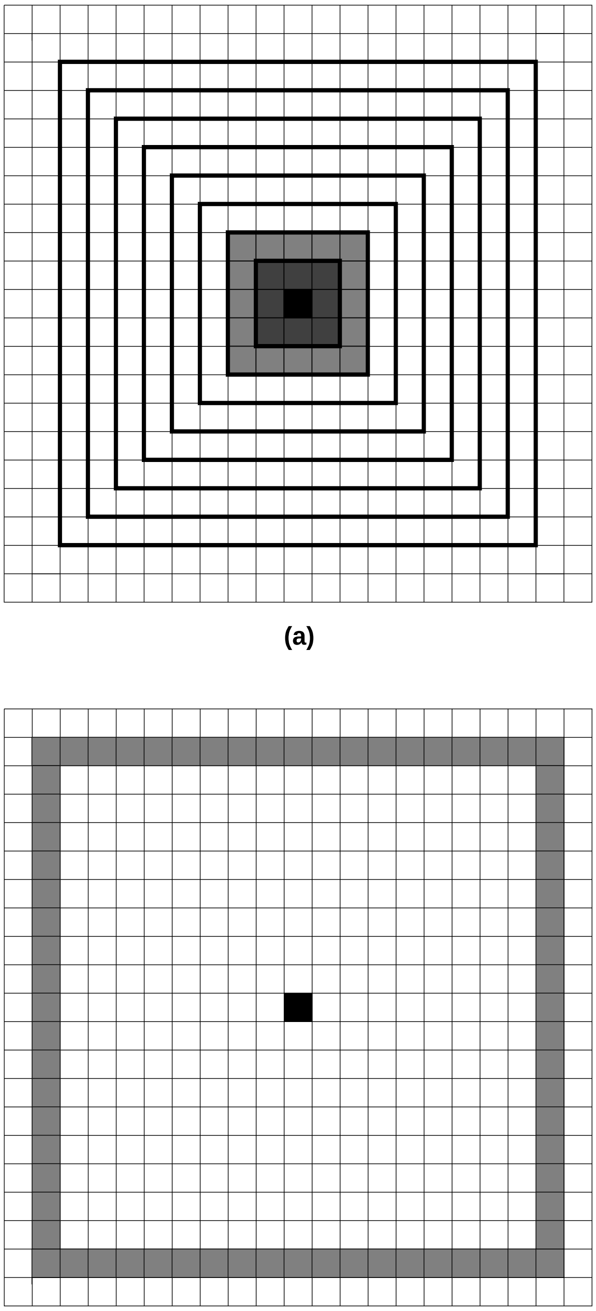


Figure 2. SIMPDEL forage search diagram. (a) High/medium quality search area with a maximum travel distance of 8 grid cells. Dark gray cells represent the first search perimeter and light gray cells represent the second search perimeter. Each additional search perimeter is surrounded by a thick black line. (b) Low quality forage search area on the outer search perimeter.

ALPHA_2. Due to the lower caloric content of medium quality forage, **ALPHA_2** is larger than **ALPHA_1**.

If no grid cells with available high or medium quality forage above the thresholds are located within the maximum travel distance, the deer is forced to graze on low quality forage, and an attempt is made to place the deer randomly at a *suitable habitat* location on the outer perimeter of the search area (see Figure 2(b)). A suitable habitat location is a 500m grid cell with a water level below the maximum level the deer can withstand. If no suitable habitat location is found, a perimeter grid cell will be randomly selected from those not located within the habitat type "open water". The deer will remain at its current location only if all perimeter cells are located within this habitat type.

If the deer is not forced to consume low quality forage, the high/medium quality forage search will continue until one of three conditions is satisfied: the maximum daily intake is reached, the maximum travel distance is reached, or the maximum forage time is reached.

Each adult deer or independent fawn will forage sequentially each day of the simulation according to the above rules. Order is chosen randomly and is computed daily, so that no deer has preferential access to food sources.

2.5.4. Growth and Mortality

Deer growth is dependent upon weight, caloric intake, and energy expended during maintenance, travel, grazing, and reproduction. If energy intake from foraging is greater than energy expenditure, the deer gains weight, and if energy intake is less than energy expenditure, the deer loses weight [3].

Deer mortality is the result of several factors. Death by weight loss occurs if the deer's weight drops below 70% of its maximum weight ever attained [8]. Death by natural age-related factors occurs if the natural mortality rate for the deer's age is greater than a randomly generated number. Since panther are not simulated, death by panther predation is determined randomly, in order to maintain the deer population at somewhat realistic levels. Fawn mortality, in addition to the three factors described above, occurs if the mother dies while the fawn is still nursing.

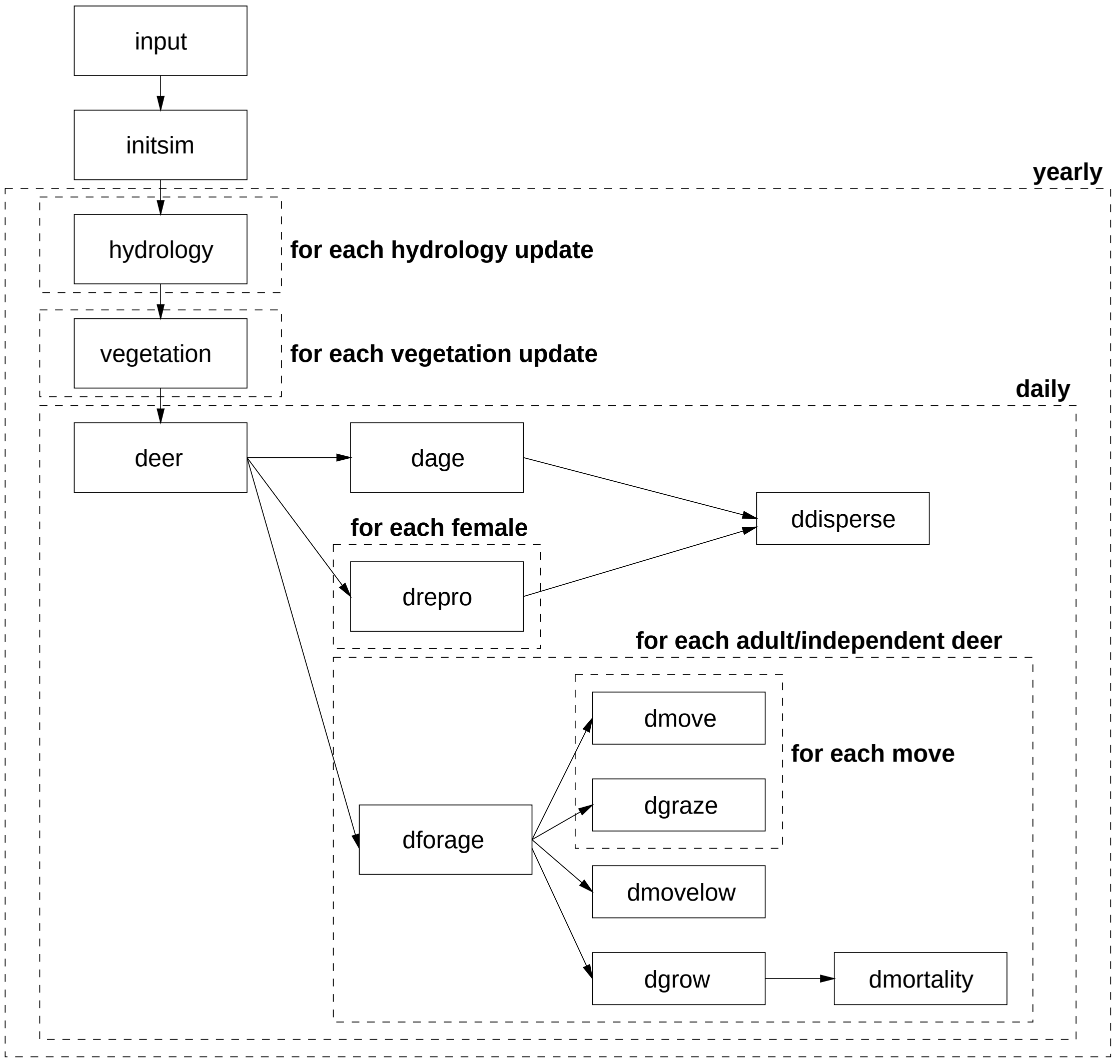


Figure 3. SIMPDEL program flow by major subroutine

2.6. Program Flow

SIMPDEL program flow is illustrated in Figure 3. Input files are read by subroutine **`input`**, and simulation variables initialized in **`initsim`**. Hydrology and vegetation updates are performed once every seven days by subroutines **`hydrology`** and **`vegetation`**. The deer component is simulated daily by subroutine **`deer`**, which invokes **`dage`**, **`drepro`**, and **`dforage`**, in sequence. Subroutine **`dage`** increments deer ages daily for deer under two months of age, and monthly for all others, and calls **`ddisperse`** for all male deer turning eighteen months of age. Subroutine **`ddisperse`** simulates the dispersal process by randomly selecting dispersal locations and analyzing the locations for suitability. The reproduction process is handled by **`drepro`**, which simulates deer mating and fawn birth, and calls **`ddisperse`** for all male deer whose mothers are giving birth to new fawns. The foraging process is comprised of subroutines **`dforage`**, **`dmove`**, **`dgraze`**, and **`dmovelow`**. Subroutine **`dmove`** locates a new $500m$ grid cell position with available forage and **`dgraze`** simulates deer grazing on the $100m$ grid cells within the selected $500m$ cell. Subroutine **`dmovelow`** selects a $500m$ grid cell along a deer's maximum travel distance perimeter, and is called only after no high or medium quality forage is found. Finally, after each deer's foraging process is completed, subroutine **`dgrow`** updates the deer's body weight based on daily energy expenditures and invokes **`dmortality`** if death results from weight loss or other factors.

3. Parallel Model

In order to parallelize the SIMPDEL model for execution on the 32 processor Thinking Machines CM-5, several modifications to the sequential model are required. The following sections describe the landscape partitioning method, revised data structures, and parallelization of the hydrology, vegetation, and deer components.

3.1. Landscape Partitioning

A rowwise block-striped partitioning strategy (Figure 4) is used to divide the landscape across the 32 processors. With this data partitioning method, the landscape is divided into groups of complete contiguous rows [12]. Due to the large computational demand of both the hydrology and vegetation

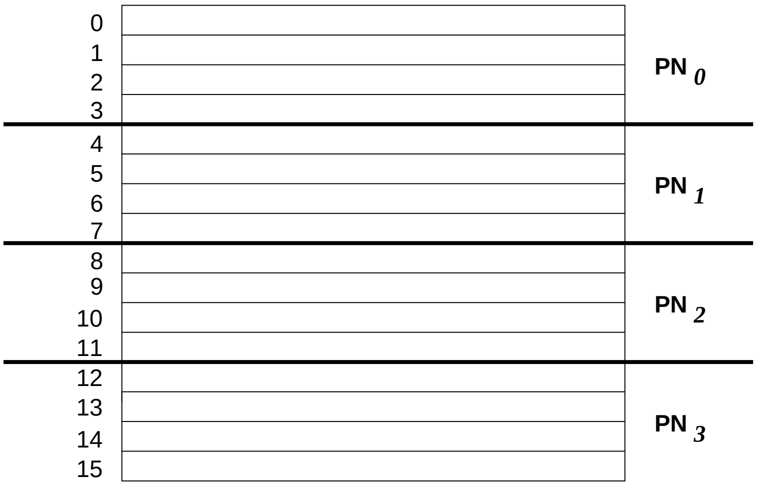


Figure 4. Example rowwise block-striped data partitioning with 16 rows and 4 processors.

components and the irregular shape of the study area, each processor is assigned a group based on the total number of 500m grid cells (Figure 5), as opposed to the total number of rows, in order to achieve a similar initial workload balance. This partitioning method simplifies the deer movement process (to be discussed in Section 3.3.5), since each processor has only two nearest neighbors. Processor PN_i's nearest neighbors are PN_{i-1} and PN_{i+1}, with the exception of PN_0, which has no nearest neighbor to the north, and PN_{31}, which has no nearest neighbor to the south.

3.1.1. Map Data Structure

In the sequential SIMPDEL model, map data are stored as a series of grids, represented as two or three-dimensional arrays, depending on the grid cell resolution, as explained in Section 2.2. To facilitate dynamic grid repartitioning, to be explained in Section 3.1.2, a more efficient data structure is required. The map data structure used in the parallel model is shown in Figure 6.

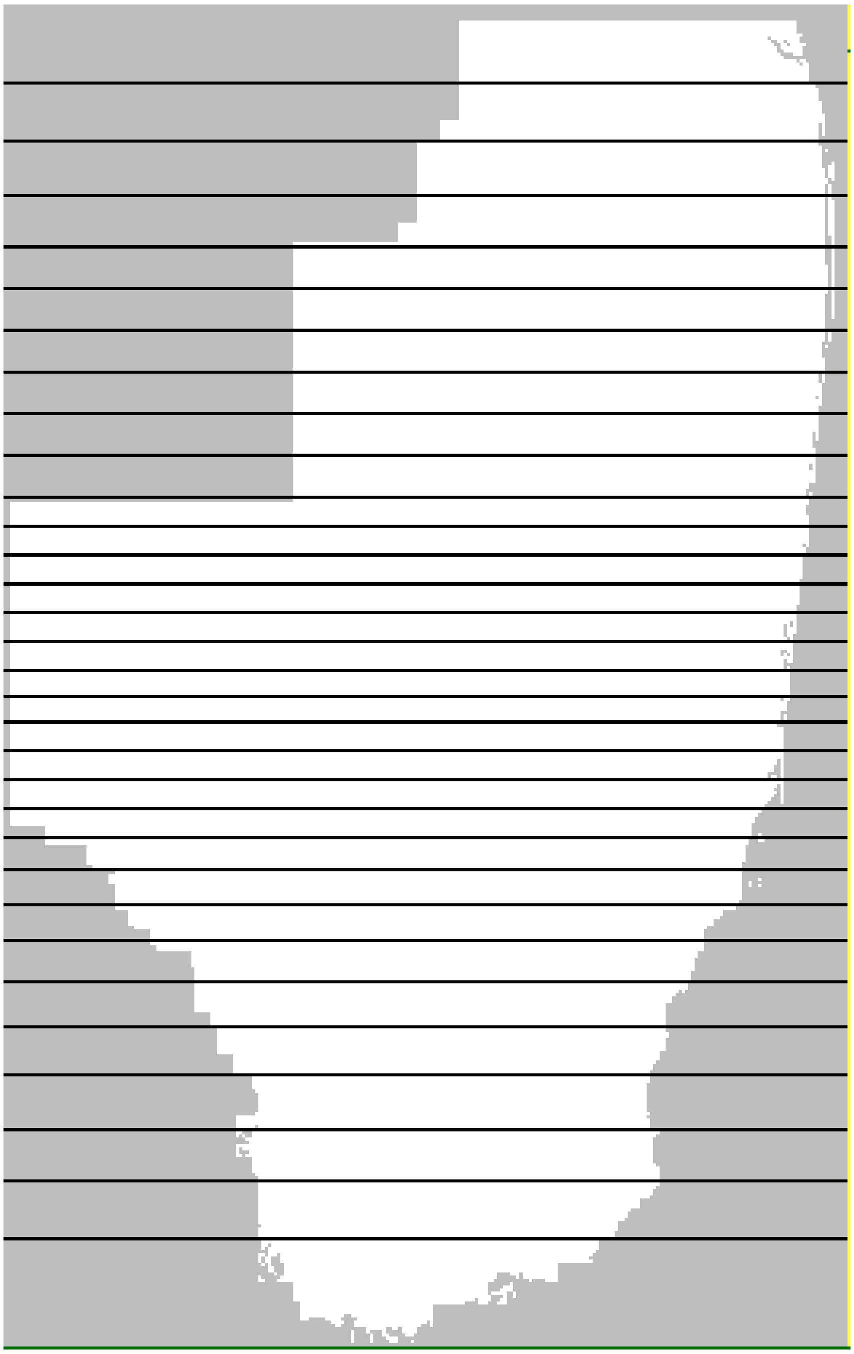


Figure 5. Landscape partitioning according to area.

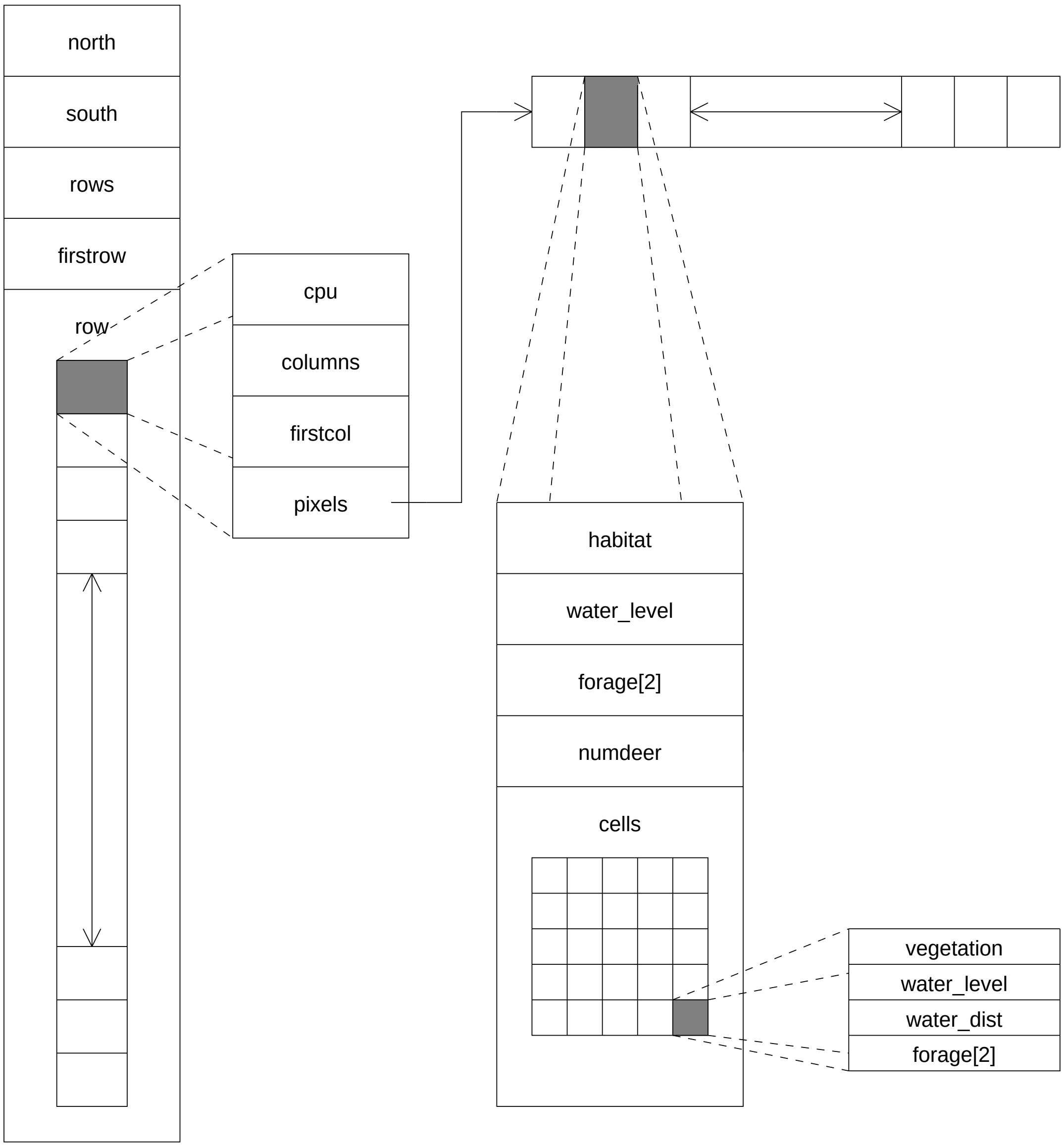


Figure 6. Parallel Map Data Structure

Each processor has its own map data structure, **`map`**, containing an array of rows, **`row[RPIXEL]`**, which corresponds to the total number of rows at the 500m scale. Each element of this array is itself a data structure containing the fields: **`pixels`**, **`cpu`**, **`firstcol`**, and **`columns`**.

The **`pixels`** array is of length **`columns`** containing map data at the 500m resolution, such as habitat type, water level, forage quantity, and a 5×5 array of 100m cells, each containing vegetation type, water level and forage quantity at the finer scale, as well as water distribution data required for hydrology updates. Each processor contains a **`pixels`** array for every row it owns, while all others are set to NULL. In order to decrease memory requirements for map data storage, only grid cells within the study area boundaries are stored, as opposed to the sequential model which stores every grid cell, regardless of location. The data structure field **`firstcol`** is used to map column indexes of the **`pixels`** array onto their actual positions in the study area. For example, if **`map.row[`**i**`].firstcol`** is 5 then **`map.row[`**i**`].pixels[0]`** contains the data corresponding to the 5th column of row i (where columns begin at 0), and if **`map.row[`**i**`].columns`** is 50 then **`map.row[`**i**`].pixels[49]`** contains data corresponding to the 54th column of row i.

Although each processor stores only a portion of the map, and each portion is mutually exclusive, all processors contain identical data in the fields: **`cpu`**, **`firstcol`**, and **`columns`**, for each of the 420 (**`RPIXEL`**) rows. The value stored in **`map.row[`**i**`].cpu`** is the processor identification number on which row i is located so that the processor location of any grid cell can easily be determined, given the row position. The values stored in **`map.row[`**i**`].firstcol`** and **`map.row[`**i**`].columns`** are used to determine whether a grid location owned by another processor is within the study area boundaries.

3.1.2. Map Initialization

In order to allow the number of processors to be scaled up or down in future parallel model versions designed for PVM, the landscape map is partitioned dynamically, after the 500m habitat data has been read from an input file. After an initial data partitioning according to the number of rows, each processor creates and initializes a **`pixels`** array for each row assigned to that processor, as the habitat data is read. The map data is then dynamically repartitioned according to area.

The landscape partitioning strategy allows for simple repartitioning by sending an entire row's contents as a message to a north or south processor. The repartitioning phase steps though processor pairs, beginning with PN_0 and PN_1, followed by PN_1 and PN_2, sending and receiving map rows between pair members until each processor contains an *approximately equal* area. Since map rows must be complete, an equal area on each processor is not possible with the shape of the current landscape map.

3.2. Hydrology and Vegetation Components

The parallel versions of the hydrology and vegetation components are very similar to those of the sequential model. Since each processor owns a portion of the map, and map data in any grid cell are independent of any other, no interprocessor communication is involved. For hydrology updates, the same interpolation method to redistribute water levels from the $2mi$ scale to the $100m$ scale is used (as explained in Section 2.3). However, each processor computes water levels for its map portion only. Similarly, each processor is responsible for updating forage growth in both the high and medium quality classes of forage in only the locally owned grid cells. Thus, each of the 32 processors updates hydrology and vegetation values in approximately 1/32 of the study area. Excellent speed improvements for both components were achieved through parallelization and are presented in Section 4.

3.3. Deer Component

Unlike the hydrology and vegetation components, execution of the parallel SIMPDEL deer component is dominated by interprocessor communication, since only a portion of the landscape is available to any processor. In order to utilize message passing and enable deer movement among processors, extensive changes to the sequential model are required.

3.3.1. Program Flow

Due to an unavoidable timing problem introduced by parallelization, a reordering of the stages of subroutine **`drepro`** was necessary. In the sequential model, each male fawn disperses from the natal range immediately before its mother gives birth to new fawns. Adhering to the same structure in the parallel model could possibly reduce the deer population size over the simulation, since there exists a small window of time during which the dispersing male is unavailable for mating with females located on other processors. To eliminate this timing error, dispersing males are placed on a queue for later dispersal after all processors have completed the reproduction phase.

Since the number of deer residing on each processor is inevitably unequal due to deer movement and reproduction, as well as the timing of different processing nodes, synchronization is necessary at certain points in the parallel program in order ensure that all processors have completed one component phase before starting the next. Explicit synchronization occurs after **`dage`**, after **`drepro`**, and again after the dispersal of queued male deer. Since these routines may require message passing, all processors waiting in the synchronization loop must continually poll for incoming messages (see [1] for the C code used in the synchronization loop). An additional implicit synchronization occurs at the end of each day as daily results are accumulated from each node. With the entire map divided among the the 32 processors, message passing becomes necessary for deer dispersal, mating, and foraging activities if these routines require knowledge of data located in *remote* grid cells (i.e., a grid located on another processor).

3.3.2. Deer Data Structure

All data pertaining to an individual deer are contained in a data structure. The deer data structure used in the parallel SIMPDEL model requires several additional fields (Figure 7), as compared to the structure used in the sequential model. The fields related to the deer foraging process: **`daily_trav`**, **`save_trav`**, **`intake`**, **`graze_time`**, **`graze_dist`**, and **`energy_gain`**, of the parallel structure are global variables in the sequential model. The floating-point values stored in these global variables always correspond to the current foraging deer, since no other deer can begin the foraging sequence until the current deer's sequence has ended. A deer may graze on several grid cells before it's daily intake is fulfilled, requiring these variables to be updated for each grazing step. However, in the parallel model, a deer may be moved to a new processor during the foraging sequence (to be explained in Section 3.3.5) and its foraging continued at the new location. By containing these variables as fields in the deer data structure, the copying and recopying of deer data each time a deer is moved to a new processor is avoided.

```
typedef struct
{
   int days_preg;
   int mating_day;
   int fawns;
   int births;
   struct deer *fawn_ptr[2];
} doetype;


typedef struct
{
   int days_rutting;
   int trav_mate;
} bucktype;


typedef union
{
   doe_type doe;
   buck_type buck;
} deergen;
```

```
struct deer
{
   int ID;
   int sex;
   int age;
   int age_days;
   int row;
   int col;
   float body_wt;
   float max_body_wt;
   float daily_trav;
   float save_trav;
   float intake;
   float graze_time;
   float graze_dist;
   float energy_gain
   int num_moves;
   int graze;
   int removed;
   struct deer *mother;
   deergen gender;
};
```

Figure 7. Parallel deer data structure (new fields are shown in *italics*).

The data structure fields **num_moves**, **graze**, and **removed** are specific to the parallel model. The integer **num_moves** is used to examine the number of times a deer moves across processor boundaries, and is incremented for each move. The integer **graze** is assigned the value of the class of forage for which the deer is searching (**FQ_HIGH**, **FQ_MEDIUM**, or **FQ_LOW**) before being moved to a location on a new processor. The integer value stored in **removed** is used to determine if a deer has been *logically* removed from the deer array, due to mortality or movement between processors. A deer is first logically removed, by setting its **removed** field to 1, before being physically removed (by deallocating memory) at the end of the day, in order to avoid constant compressions of the deer array.

3.3.3. Deer Dispersal

Dispersal of male deer occurs either when the deer turns 18 months of age or when the mother gives birth to new fawns (see Section 2). As in the sequential model, potential dispersal locations are continually generated up to a maximum number or until a suitable dispersal location is found. If processor PN_i generates a local grid location, PN_i will determine if the location is suitable for deer existence. However, message passing is required if the randomly generated grid cell is not local to the current processor. If PN_i generates a grid location that is owned by processor PN_j, the entire deer structure is sent as a message to PN_j, after updating the deer's row and column to the new grid position.

Upon receipt of the deer message from PN_i, PN_j will determine whether the deer's row and column location is suitable. If so, the deer is inserted into the deer array owned by PN_j. Whether or not the deer remains on PN_j, a message is sent back to PN_i indicating suitability of the dispersal location.

If the response message from PN_j indicates suitable dispersal habitat, the deer is removed from PN_i's deer array and the dispersal process is complete. However, if the response indicates unsuitability, another grid location is generated and the process is repeated. If, after generating a maximum number of potential dispersal locations, no suitable grid cell is found, the deer's row and column fields are reset to the original values and the deer will remain at its initial grid position. A flowchart illustrating the dispersal process is shown in Figure 8.

3.3.4. Reproduction

To achieve population sizes similar to those in the sequential SIMPDEL model, message passing is necessary in order to locate suitable mates in areas owned by other processors. In the sequential model, any male deer within mating distance of a receptive female that has not already mated within the year is a potential mate. To locate a mate, a search is performed on the entire deer array. In the parallel model however, the maximum radius of 20 $500m$ grid cells (*i.e.*, $10km$) from the female often spans several processors. In order to reduce or eliminate message passing, a restriction could be imposed on male deer, allowing only those located on the same processor as the female, or only those deer located on the north or south processors, to be potential mates. However, any restriction

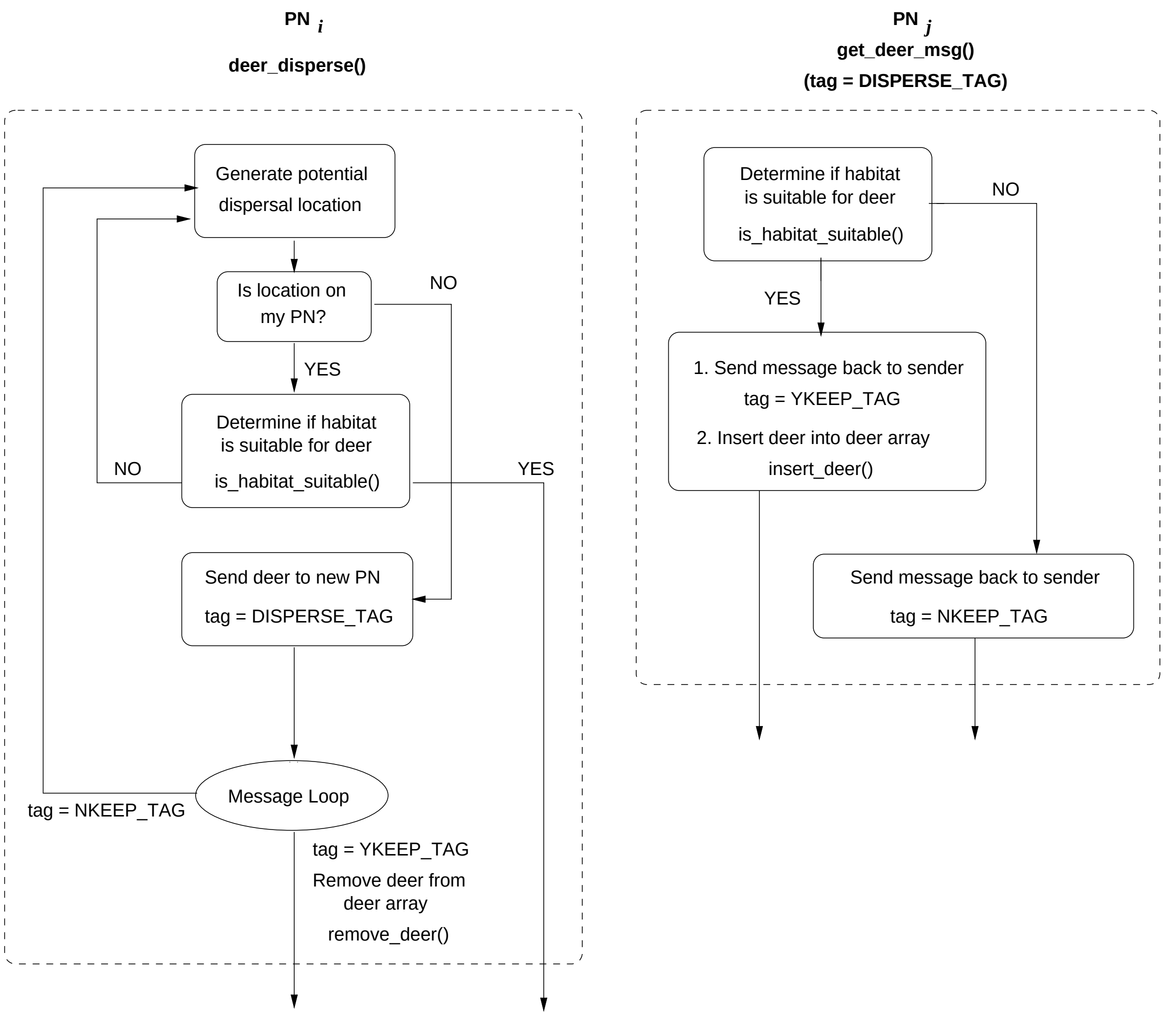


Figure 8. Deer dispersal process on processors PN_i and PN_j.

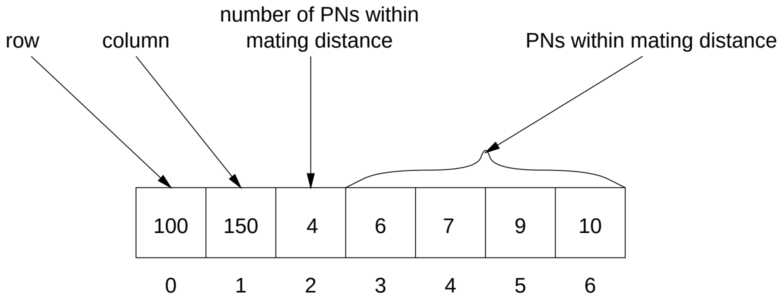


Figure 9. Example **`mate_msg`** array located on processor PN_8

drastically affects the deer population over time, in comparison to that of the sequential program, and would produce different results for different partition sizes. Thus, the entire mating distance is made available to each deer, regardless of the number of processors this distance spans.

During the initialization phase of the parallel simulation, an array, **`mate_msg`**, is created on each processor (see Figure 9) and is used to store the set of processors within the maximum mating distance. Processor PN_i's set includes all processors, except PN_i, containing one or more map rows which lie within the interval

[**`(map.firstrow-radius+1)`**, **`(map.firstrow+map.rows+radius-2)`**].

The processor numbers within the set are stored beginning at index 3 in **`mate_msg`**. The value stored in **`mate_msg[2]`** is the total number of processors within the set, and **`mate_msg[0]`** and **`mate_msg[1]`** are reserved for storage of a female deer's row and column locations.

If no available male deer is located on PN_i, the female's row and column locations are loaded into **`mate_msg[0]`** and **`mate_msg[1]`**, respectively, and a new processor, PN_j, is selected at random from those stored in the array. PN_i then sends the entire array to PN_j and awaits a response. The response message indicates whether or not an available male was located and may be sent by any processor within the set.

Upon receipt of the message from PN_i, PN_j will search its deer array for an available male deer within mating distance of the location specified in the message buffer. If a male is located, PN_j will send a confirmation message to PN$_i$. Otherwise, PN$_j$ sets the message index corresponding to its PN number to -1, and chooses a new processor, PN$_k$, at random from the remaining values in the message buffer. PN$_j$ will then send the updated message buffer to PN$_k$, and so on until either an available male is found, or messages have been sent to all processors in the message buffer. A reply message indicating that no male was located will be sent to PN$_i$ only if no male deer are available on any processor in the set, and will only be sent by the last processor to receive the message buffer. See [1] for more details of the parallel mating process.

3.3.5. Foraging

Since the landscape is divided among the 32 processing nodes, a deer's search area may encompass more than one processor. Without knowledge of data across processor boundaries, it becomes impossible for the animal to select a new grid location similar to the one it would have chosen in the sequential model.

Similar parallel ecological models developed in the past have determined whether an animal should move to a new processor by using a decision method referred to as PMI (Preferential Moving Index) averaging. With this method, a PMI value is computed at the beginning of each simulation day, using data from all or portions of a processor's grid cells, and then broadcast to every other processor. If the animal cannot locate enough forage on its own processor, the nearest neighbor processor with the greatest PMI average above a threshold is chosen and the animal is sent to that processor to continue its foraging sequence [2].

Although PMI averaging may result in decreased parallel execution time for some ecological applications, this method was not used in the parallel SIMPDEL model for several reasons:

1. The per processor area is too large and may increase in future implementations.

2. Forage amounts decrease after each deer grazes. With simulation test populations from 2,000 to 20,000, the PMI averages would quickly become invalid, and frequent updating of the PMI averages would most likely offset any gain realized by their use.

3. Available forage amounts in the same grid cell differ for bucks, does, and does with fawns, since these amounts are computed using the maximum water depths that each class of deer can withstand.

4. Maintenance of PMI averages for both high *and* medium quality forage classes would be required.

5. A processor's PMI average is dependent upon area, thus different partition sizes would produce different results.

Therefore it was necessary to develop a different parallel method for determining deer movement locations based on the most recent forage levels. The parallel method for the deer forage search relies on all processors containing grid locations within a deer's search area to examine the necessary grid cells in parallel.

High/Medium Quality Forage Search

The parallel forage search algorithm follows the same general movement rules as in the sequential SIMPDEL model. Each day of the simulation, deer forage sequentially on each processor. Order is determined randomly each day. Each deer's daily forage search begins with an attempt to graze on the current grid location. If the cell does not contain enough available high quality forage, a search for a new grid cell position is initiated (as explained in Section 2). In order for the forage search to be performed in parallel by multiple processors, subroutine **`dmove`** was separated into the two subroutines: **`dmove`** and **`find_forage_cell`**. Subroutine **`dmove`** is invoked only by the processor on which the deer is located, however **`find_forage_cell`** may be called by all processors containing grid locations within a deer's search area. Subroutine **`dmove`** is responsible for calling **`find_forage_cell`**, waiting for message responses, and later updating the deer's position and travel distance after a new grid cell is found. Locating the new grid cell is the responsibility of **`find_forage_cell`**, which initiates the forage search among concentric squares centered at the deer's current location, and sends messages to neighboring processors if the search area expands past the current processor's boundaries. The subroutine flow of the forage search and deer growth process is discussed in [1].

If either the top or bottom row of processor PN_i's search area is owned by another processor, PN_i will send a *search message* to the processor containing that row. If the top row of PN_i's search area is not located on PN_i, it will always be the last row on PN_{i-1}, and if the bottom row of PN_i's search area is not located on PN_i, it will always be the first row on PN_{i+1}. The search message contains the following information: the deer's grid position, maximum water depth, maximum moving distance, the class of forage being searched (**`FQ_HIGH`** or **`FQ_MEDIUM`**), and the current search radius.

Whether or not processor PN_i sends a search message to another processor, the local search will continue until either a $500m$ cell with available forage is found or the deer's maximum moving distance is reached. At this point, if a message has been sent to a neighboring processor, PN_i must wait for one or more message responses, referred to as *location messages*. A location message contains the row and column positions of the $500m$ grid cell with a maximum of available forage, the available forage level (or 0 if no grid cell is found), the search radius (the number of grid cells between the selected forage cell and the center of the search area), and an integer value indicating whether or not the search area expanded past a processor boundary.

As each location message is received, PN_i will compare the current minimum search radius with the search radius from the message buffer. As in the sequential model (explained in Section 2), the nearest grid cell with a maximum level of available forage above the threshold is chosen as the new grid cell. For example, in Figure 10, a $500m$ grid cell with a maximum level of available forage above the threshold has been located on each of the three processors. The maximum forage cell located on processor PN_{i-1} has the smallest radius and therefore will be chosen as the deer's new grid position, regardless of the amount of available forage in the selected grid cells on PN_i and PN_{i+1}. If multiple grid cells along the same concentric square perimeter have the same maximum, one will be chosen at random.

If a grid cell with available forage above the threshold is located after all message responses have been received, the deer's position is updated, as well as the positions of any fawns. If the selected grid cell is owned by processor PN_j, the deer's **`graze`** field is set to the value of the forage class (**`FQ_HIGH`** or **`FQ_MEDIUM`**), so that the deer will continue grazing on the same class of forage on PN_j. Processor PN_i will then send the deer structure as a message to PN_j. However, if the deer

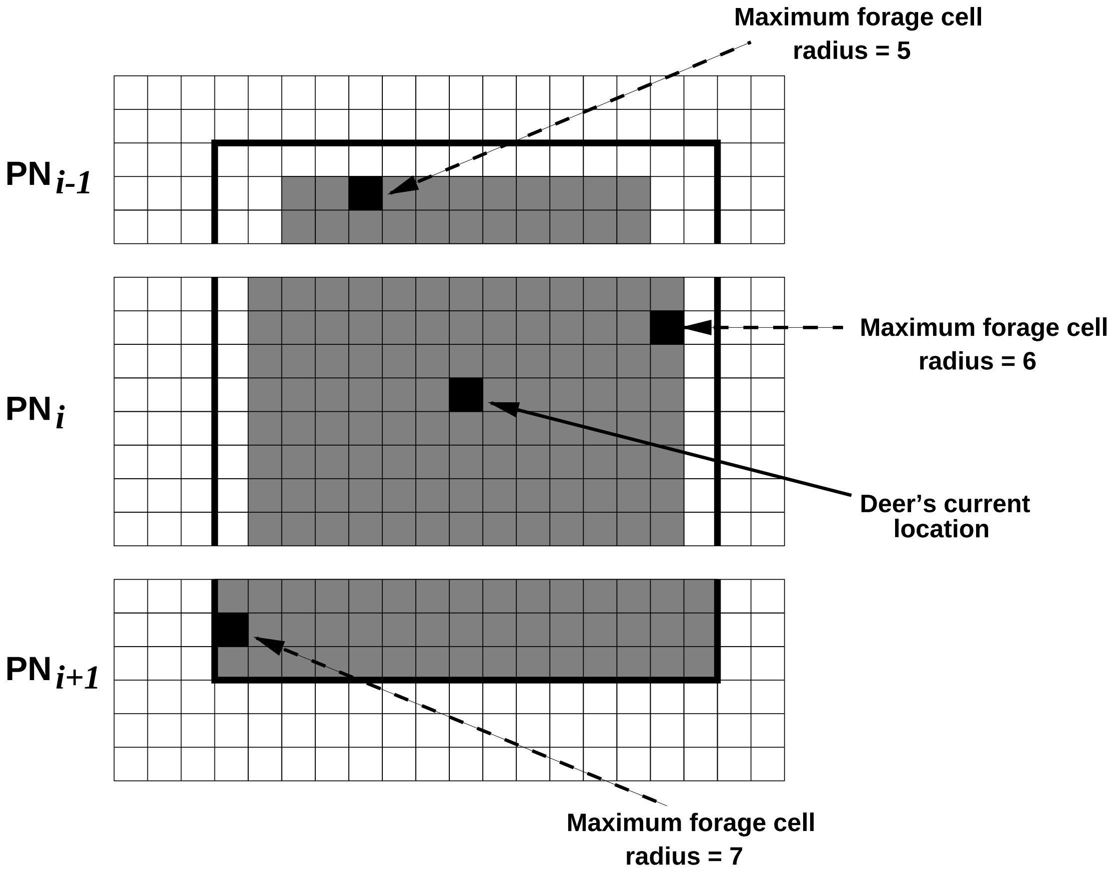


Figure 10. Search area for a deer located on processor PN_i. Searched grid cells are shaded gray and the maximum travel distance perimeter is outlined with a thick black line.

is a female with fawns, the deer data for the mother and fawns is first copied into an array of deer structures, and the entire array is then sent as a message to PN_j.

Upon receipt of the deer message from PN_i, PN_j must first determine the size of the message buffer, since a deer message may contain data for one, two, or three deer. To determine the size of the message buffer, the CMMD function **`CMMD_bytes_sent`** is used. The integer value returned by this function divided by the size (in bytes) of the deer data structure equals the number of deer structures in the message buffer. The data for each deer is then copied into a separate structure and inserted into PN_j's deer array. All adult deer (or independent fawns) will be placed on a queue. When the current deer has completed foraging for the day, the queue is checked. If the queue is not empty, the deer that has been waiting for the longest time will be the next to forage. The entire queue must be emptied before any deer originally residing on the processor can begin the foraging sequence. The reason for this constraint is that all deer on the deer queue have already begun the foraging sequence, but did not find enough forage on their original processors to satisfy their maximum intake. Those on the queue have been assigned new grid locations with available forage levels. In order to increase the chance that the forage level on a deer's grid cell will be the same as when it was chosen, those on the queue must forage before any other. It is still possible however, that a foraging deer could deplete the resources on a grid cell upon which a queued deer is waiting to graze, forcing the queued deer to search for a new grid cell. The intensive checking required to avoid this potential problem would be unnecessary in all but a few simulation years, and therefore has not been implemented.

Low Quality Forage Search

If no available high or medium quality forage is found within a deer's maximum travel distance, the deer will be placed randomly at a suitable grid location along the maximum travel distance perimeter, as explained in Section 2. Since this area may be distributed across several processors, a message is sent to each processor containing grid locations along the maximum travel distance perimeter. For the parallel model, **`dmovelow`** was separated into two subroutines. The first subroutine (of the same name), which is called only by the processor on which the deer is located, determines the search area perimeter, sends a message to each processor containing a portion of the perimeter, and waits for message responses before determining a deer's new location. The second subroutine, **find_forage_cell_low**, is called from within **dmovelow**, and also from **get_deer_msg** for all other processors containing grid cells on the maximum travel distance perimeter.

This process of sending messages and waiting for responses is different from that of the high/medium quality forage search because of the fixed search area perimeter. The message, referred to as a *low quality search message*, contains the left and right column positions of the search area, the deer's maximum water level, and a *row limit* value. The row limit is set to the first row of the search area for all processors north of the sending processor, and is set to the last row of the search area for all processors south of the sending processor. Thus any processor containing a row limit boundary will search the entire top or bottom row of the search perimeter in addition to the two left and right columns. For example, in Figure 11, processors PN_{i-2} and PN_{i+1} both contain row limit values, and therefore must search the entire row of gray colored grid cells.

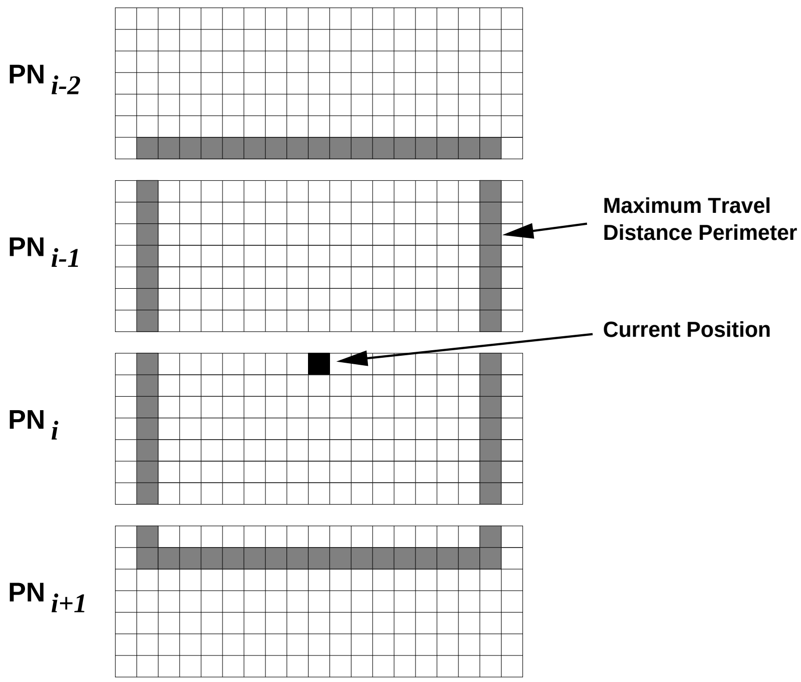


Figure 11. Low quality search area.

After sending a message to each processor containing locations within the search perimeter, PN_i will attempt to locate a suitable grid cell along the local portion of the search area perimeter, according to the rules described in Section 2. The processor must then wait for message responses from all other processors to which messages were sent. The message response to a low quality search message, referred to as a *low quality location message*, contains the randomly selected grid cell position and a value indicating whether the grid position is above or below the deer's maximum water depth.

After all message responses are received, the final grid cell position will be selected based on water level, as described in Section 2. If a suitable grid position is found, the deer's row and column locations will be updated. If this location is owned by another processor, the deer's graze field is set to the value **FQ_LOW**, indicating that the deer has completed the foraging sequence, and the deer structure is then sent as a message to the new processor.

4. Verification and Performance

In order to verify the correctness of the parallel model, outputs of both the sequential and parallel models were compared and yearly deer distribution maps were created. This section presents the selected output comparisons and speed improvements in all model components.

4.1. Comparison of Selected Outputs

Although several outputs were produced and analyzed, three main statistics were chosen for comparison: *average daily travel distance* of deer, *deaths due to weight loss*, and *year-end population size*. Since daily travel distance and weight loss deaths are dependent upon available forage and water levels, these statistics are used to verify the hydrology and vegetation components, as well as the foraging phase of the deer component. Year-end population sizes are used to verify the reproduction phase of the deer component. Statistics are plotted for an initial deer population size of $10,000$. Statistics are also plotted for $2,000$ and $20,000$ deer population sizes and are presented in Appendices A and B respectively. Random initial deer locations and other characteristics were generated separately for the sequential and parallel programs. The comparisons shown are for single simulation runs for each of the serial and parallel models for each initial population size.

Figure 12 illustrates the average daily travel distance per year for an initial deer population of 10,000, with discrepancies ranging from 0% to 8% between the sequential and parallel models. A deer's travel distance is related to the amount of available forage. Larger available forage quantities result in smaller travel distances while smaller available forage quantities result in larger travel distances. The slight increase in travel distance in the parallel model can be explained by the fact that deer waiting on the queue to forage are required to travel further if the grid cells to which they are assigned have no available forage by the time they are allowed to graze. Since the total number of weight loss deaths is large and very similar in both models, a graph showing the log of these numbers is presented in Figure 13. A difference of only 3% was noted between the two models over the entire 23 year simulation. This slight difference can be attributed to the increase in travel distance in the parallel model, since increased travel distance results in increased energy expenditures and therefore less weight gain. The graph of year-end population sizes shown in Figure 14 illustrates a difference of no more than 5% between the two models.

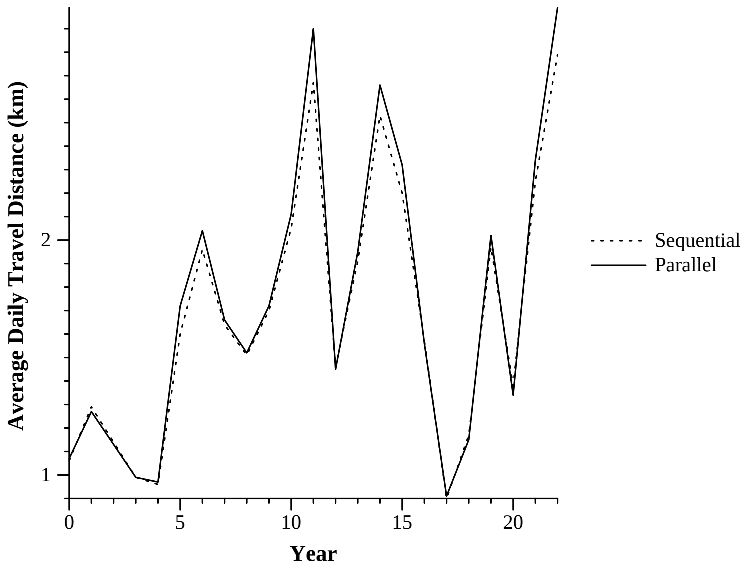


Figure 12. Average daily travel distance per year for an initial deer population of 10,000.

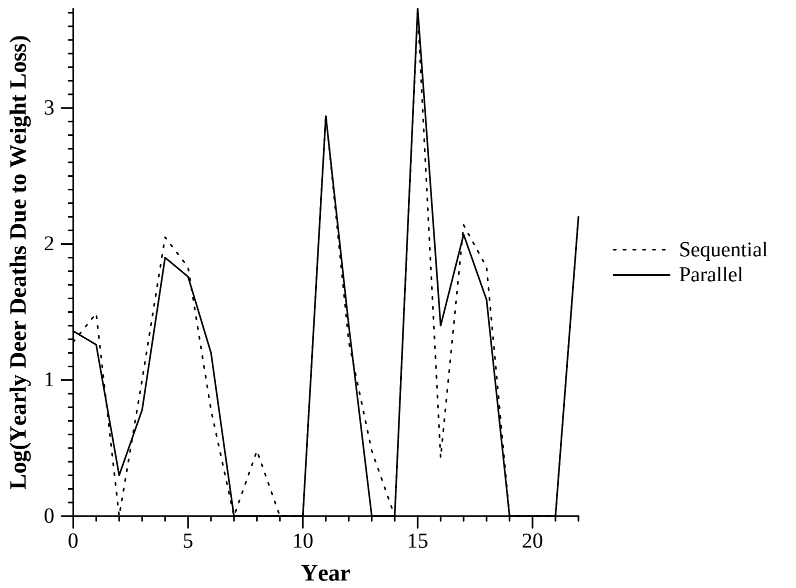


Figure 13. Weight loss deaths per year for an initial deer population of 10,000.

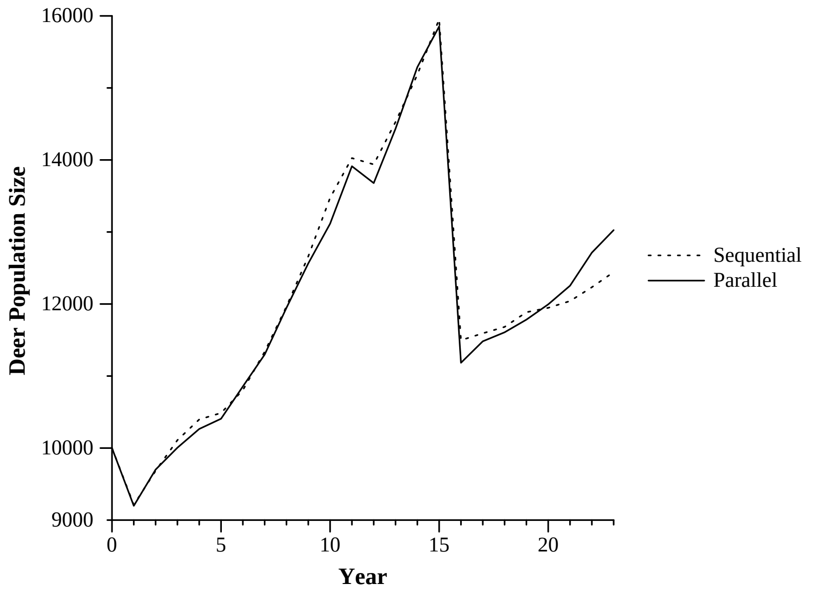
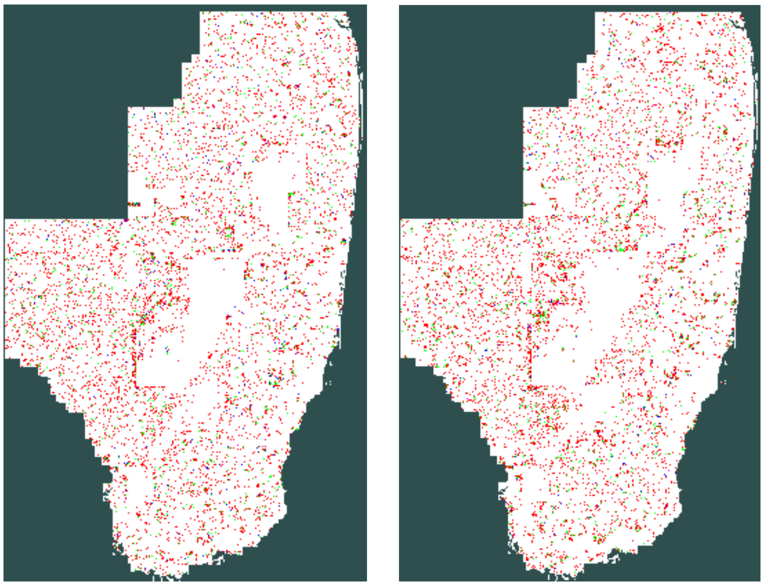


Figure 14. Year-end population sizes for an initial deer population of 10,000.

In order to examine the distribution and abundance of deer across the landscape, yearly distribution maps were created. Figures 15 through 19 illustrate the actual grid positions of every member of the deer population in selected simulation years. These figures represent those years showing the greatest change from previous years. Each dot represents a single $500m$ grid cell, which may contain more than one deer, thus different colors are used to represent the number of deer located on each grid cell. A legend describing the colors used for the distribution maps is provided in Figure 20. Figure 15 shows the initial random deer distribution in both models on the first day of the simulation. Figure 16 shows an almost exact distribution of deer in the two models on the first day of year 4. Results become less exact in the beginning of year 15, as shown in Figure 17. This difference is due to the random placement of deer on their outer search perimeters since mostly low quality forage is available during the previous simulation year. Year 18 (Figure 18) shows a distribution similar to year 4, but with a larger deer concentration in the upper left portion of the map. Finally, Figure 19 again shows very similar distribution patterns.

(a) Sequential Model (b) Parallel Model

Figure 15. Deer distribution at the beginning of **Year 0** in (a) the sequential model and (b) the parallel model.



(a) Sequential Model (b) Parallel Model

Figure 16. Deer distribution at the beginning of **Year 4** in (a) the sequential model and (b) the parallel model.

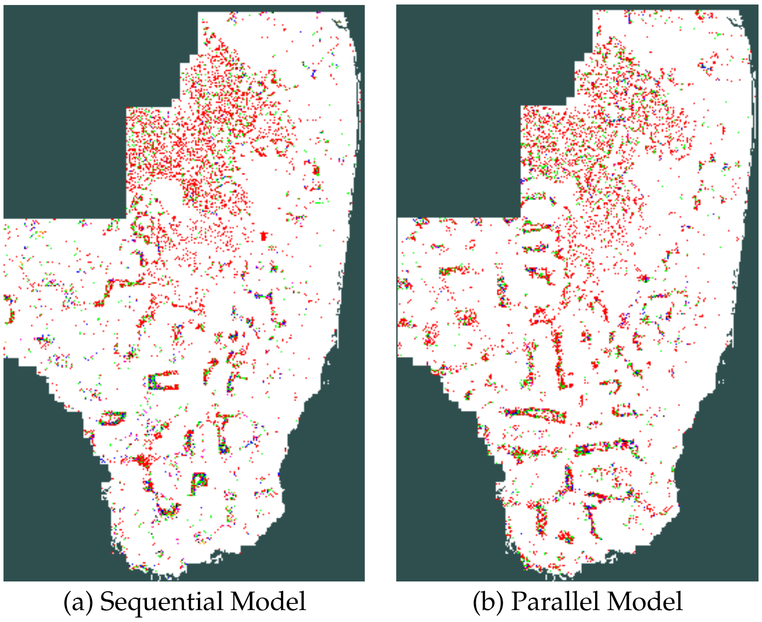


Figure 17. Deer distribution at the beginning of **Year 15** in (a) the sequential model and (b) the parallel model.

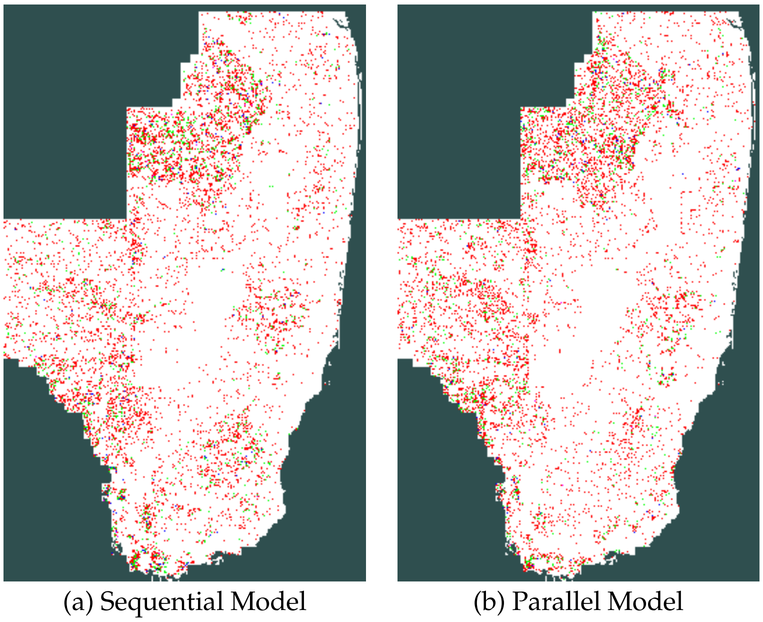


Figure 18. Deer distribution at the beginning of **Year 18** in (a) the sequential model and (b) the parallel model.

(a) Sequential Model (b) Parallel Model

Figure 19. Deer distribution at the beginning of **Year 20** in (a) the sequential model and (b) the parallel model.

Figure 20. Deer distribution map legend.

4.2. Parallel Results

A comparison between the sequential and parallel SIMPDEL models was made based on a 23 year simulation for each initial population size. Execution times and speed improvements are given in Table 1. The sequential program was executed on a Sun SPARCstation 5, with 32 Mbytes of memory and 640 Mbytes of disk space. The parallel program was executed on a 32 processor Thinking Machines CM-5 with 32 Mbytes of memory on each SPARC 2 processor. Speed improvements were

Table 1. Wall-clock times (in seconds) for the sequential and parallel SIMPDEL models for each model component with varying population sizes.

Population Size		2,000	
Component	Sequential	Parallel	Speed Improvements
Deer	16071	807	19.9
Hydrology	21345	609	35.1
Vegetation	20713	394	52.6
Total	59371	2195	27.0
	(16.5 hrs)	(.61 hrs)	

Population Size		10,000	
Component	Sequential	Parallel	Speed Improvements
Deer	55482	6635	8.4
Hydrology	21354	609	35.1
Vegetation	20713	394	52.6
Total	98057	8248	11.9
	(27.2)	(2.3 hrs)	

Population Size		20,000	
Component	Sequential	Parallel	Speed Improvements
Deer	93235	14400	6.5
Hydrology	21354	609	35.1
Vegetation	20713	394	52.6
Total	140434	15757	8.9
	(39.0 hrs)	(4.4 hrs)	

Table 2. Cpu times (in seconds) for the sequential SIMPDEL model for each model component with varying population sizes.

Population Size	2,000	10,000	20,000
Deer	7754	41398	75413
Hydrology	9761	9761	9761
Vegetation	7012	7012	7012
Total	26724	56135	95593
	(7.4 hrs)	(15.6 hrs)	(26.6 hrs)

calculated using wall-clock times; however, cpu times for the sequential model are also presented (Table 2).

Execution times were recorded for the initial deer population sizes: 2,000, 10,000, and 20,000, with a peak speed improvement of 27 for the parallel model over the sequential model for the population size of 2,000. Speed improvements for the hydrology and vegetation components are larger than the number of processors, due to the greater memory limitations of the sequential computing environment. For example, execution of the sequential model produces approximately 100,000 page faults per simulation year for an initial population size of 10,000, however with 32 Mbytes of memory on each processor of the CM-5, all data for the parallel model fits in main memory. In addition, memory requirements are greater in the sequential model since about 1/3 more map data is stored than in the parallel model.

As the sequential model is updated and improved, speed improvements in the hydrology and vegetation components will most likely decrease. By simply replacing the arrays in the sequential model with a map data structure similar to that of the parallel model, execution times should decrease somewhat. The parallel map data structure has two main advantages over the various arrays used to store map data in the sequential model. First, the data structure enables the storage of only the valid grid cells (those representing grid cells within the study area) and provides a means of mapping each grid cell onto its actual position in the landscape. Second, all map data corresponding to the same $500m$ grid cell is encapsulated into one structure and stored in contiguous memory. Since most computations require the use of data values located in the same array index, but from several different arrays, the parallel map data structure provides faster access to data values.

4.3. Performance of Processing Nodes

Although asynchronous communication was used, the explicit synchronization in the parallel model resulted in similar execution times per processor. If the synchronization points were removed, similar execution times would still result due to the use of the CMMD reduction functions at the end of each simulation day. In order to determine actual computation time per processor (Figure 21), the CMMD timing functions were used to measure *idle time*. Idle time is that portion of time during which a processor is performing no useful computation. The total number of deer processed during the 23 year simulation for each processor is shown in Figure 22.

The difference in computation time per processor is dependent not only on the number of deer residing on a processor, but also on the proximity of the deer to the processor boundary. In addition, this difference is dependent upon the grid positions of deer located on *neighboring* processors. The closer a deer is to a processor boundary, the more likely it will be that the neighboring processor is required to participate in the deer's forage search. For example, in Figures 21 and 22, PN_0 processed more deer during the simulation than PN_9, PN_{10}, and PN_{11}, but has a smaller computation time. Similarly, PN_{22} processed more deer than PN_{23}, however PN_{23} has a greater computation time.

5. Summary and Future Work

A parallel model of the hydrology, vegetation, and deer components of the current sequential SIMPDEL model has been presented. Results were very similar in both models and excellent speed improvements were obtained. In addition to promising results, the parallel model proved worthwhile in verifying the outputs produced by the sequential model. The original SIMPDEL model contained an extremely large number of bugs and was restructured and improved during parallel model development. Thus the parallel model provided a means for result comparison which aided in locating bugs in the sequential model that may not otherwise have been noticed. The results also provide strong evidence that grid-based parallelization schemes can be highly effective for individual-based ecological models with explicit spatial structure. Earlier concerns on parallelization for individual-based ecological models involving movement argued that distributing individuals over processors would be more efficient than dividing space over processors [10]. A

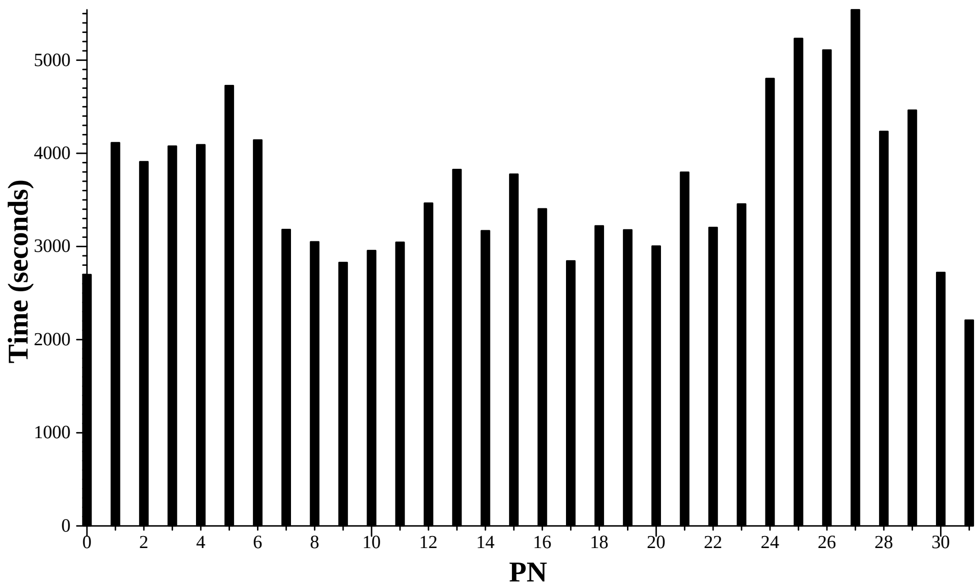


Figure 21. Total computation time per processor for an initial deer population of 10,000.

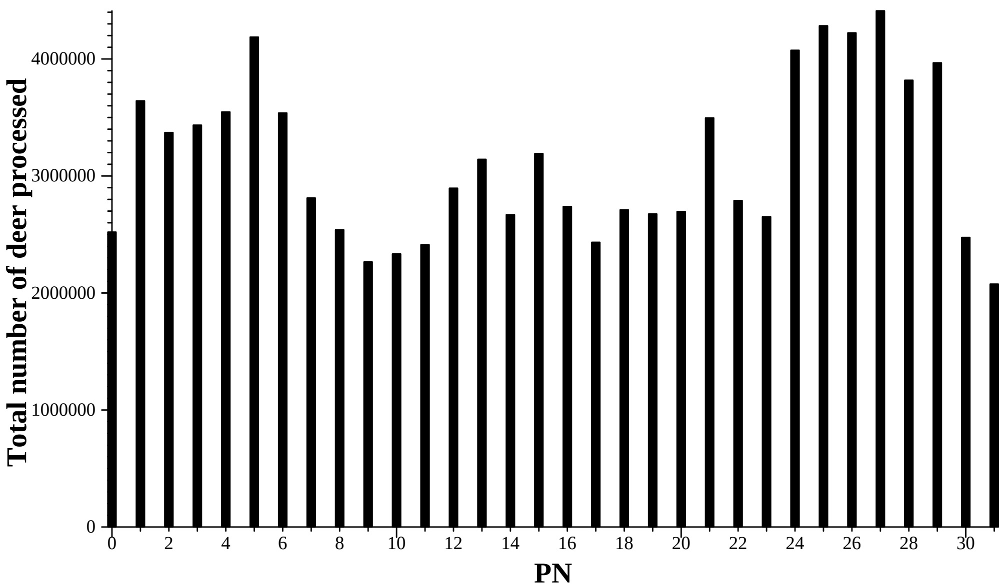


Figure 22. Total number of deer processed on each processor for an initial deer population of 10,000.

parallel implementation of SIMPDEL in which processors handle specified individuals was not developed since initial attempts indicated that it would be extremely difficult and inefficient to do so, in part due to the message passing required to update the underlying forage maps.

Future work on the parallel SIMPDEL model consists of porting the existing parallel code to a network of workstations using PVM [9]. Since the landscape is initially partitioned dynamically according to area, the number of processors used can be scaled up or down, with little or no change to the code structure beyond the replacement of CMMD function calls with the corresponding function calls from the PVM library. In addition, the panther component will be parallelized and incorporated into the model. Future plans for the sequential model include additional water level inputs, map data layers, and catastrophic events, which may also be included into the parallel SIMPDEL model.

References

1. ABBOTT, C. A. 1995. Master's Thesis, A Parallel Individual-Based Model of White-Tailed Deer in the Florida Everglades, University of Tennessee, Knoxville.
2. BERRY, M. AND UZIEL, E. 1995. Parallel Models of Animal Migration in Northern Yellowstone National Park. *International Journal of Supercomputer Applications and High Performance Computing*. In press.
3. COMISKEY, E., GROSS, L., FLEMING, D., HUSTON, M., BASS, O., LUH, H.-K., AND WU, Y. 1995. A Spatially-explicit Individual-based Simulation Model for Florida Panther and White-Tailed Deer in the Everglades and Big Cypress Landscapes. Department of Mathematics, University of Tennessee.
4. DAVIS, S. AND OGDEN, J. 1994. *Everglades: The Ecosystem and Its Restoration*. St. Lucie Press, Delray Beach, Florida.
5. DEANGELIS, D. AND GROSS, L. 1992. *Individual-Based Models and Approaches in Ecology*. Routledge, Chapman and Hall, New York.
6. DUEVER, M., MEEDER, J., MEEDER, L., AND MCCOLLOM, J. 1994. The Climate of South Florida and Its Role in Shaping the Everglades Ecosystem. In *Everglades: The Ecosystem and Its Restoration*, pp. 225–248. Delray Beach, Florida: St. Lucie Press.
7. FENNEMA, R., NEIDRAUER, C., JOHNSON, R., PERKINS, W., AND MACVICAR, T. 1994. A Computer Model to Simulate Everglades Hydrology. In *Everglades: The Ecosystem and Its Restoration*, pp. 249–289. Delray Beach, Florida: St. Lucie Press.
8. FLEMING, D., DEANGELIS, D., GROSS, L., ULANOWICZ, R., WOLFF, W., LOFTUS, W., AND HUSTON, M. 1994. ATLSS: Across-Trophic-Level System Simulation for the Freshwater Wetlands of the Everglades and Big Cypress Swamp. In *Proceedings of the ATLSS Workshop, National Biological Survey*, Homestead, Florida.
9. GEIST, A. ET AL. 1994. *PVM: Parallel Virtual Machine A User's Gui and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA.
10. HAEFNER, J. 1992. Parallel Computers and Individual-Based Models: An Overview. In *Individual-Based Models and Approaches in Ecology*, pp. 126–164. New York: Routledge, Chapman and Hall.
11. HWANG, K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, Inc., New York, New York.
12. KUMAR, V., GRAMA, A., GUPTA, A., AND KARYPIS, G. 1994. *Introduction to Parallel Computing: Design and Analysis of Anlgorithms*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, California.

13. Thinking Machines Corporation 1993b. *CMMD Reference Manual Version 3.0*. Cambridge, Massachusetts: Thinking Machines Corporation.

14. Thinking Machines Corporation 1993a. *CMMD User's Guide Version 3.0*. Cambridge, Massachusetts: Thinking Machines Corporation.

15. WILLIAM B. ROBERTSON, J. 1989. *Everglades: The Park Story*. Florida National Parks and Monuments Association, Inc., Homestead, Florida.

Appendices

A. Plotted Statistics for the Initial Population Size of 2,000 Deer

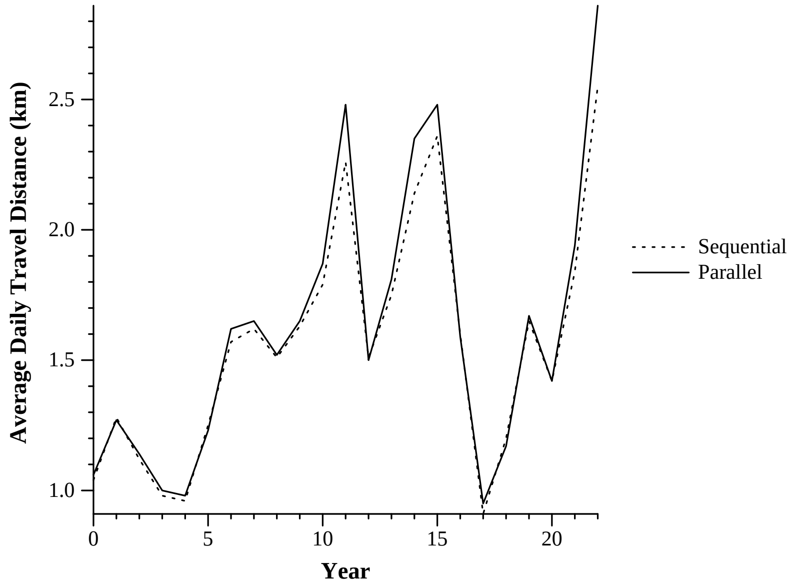


Figure 23. Average daily travel distance per year for an initial deer population of 2,000.

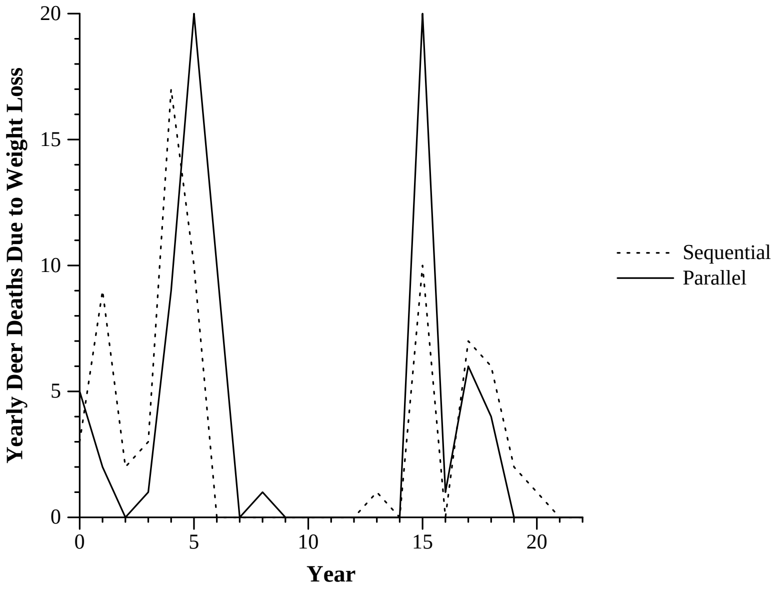


Figure 24. Weight loss deaths per year for an initial deer population of 2,000.

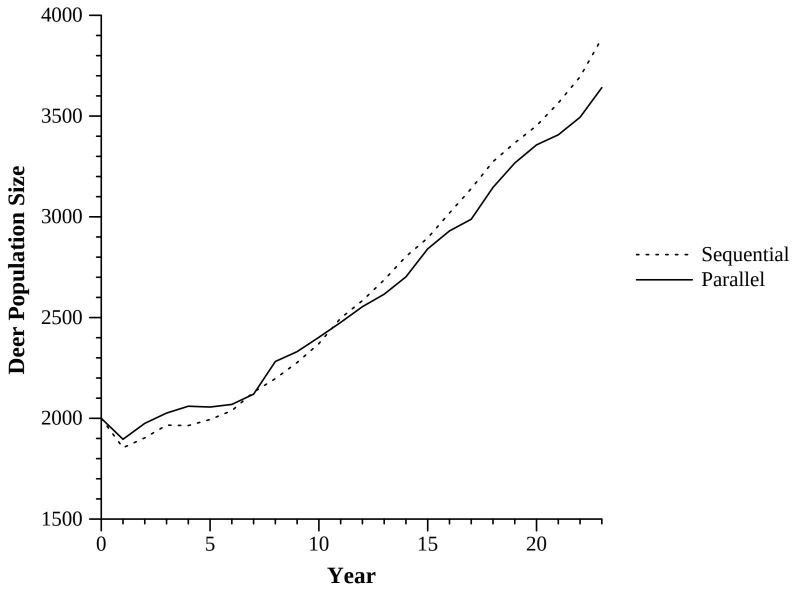


Figure 25. Year-end population sizes for an initial deer population of 2,000.

B. Plotted Statistics for the Initial Population Size of 20,000 Deer

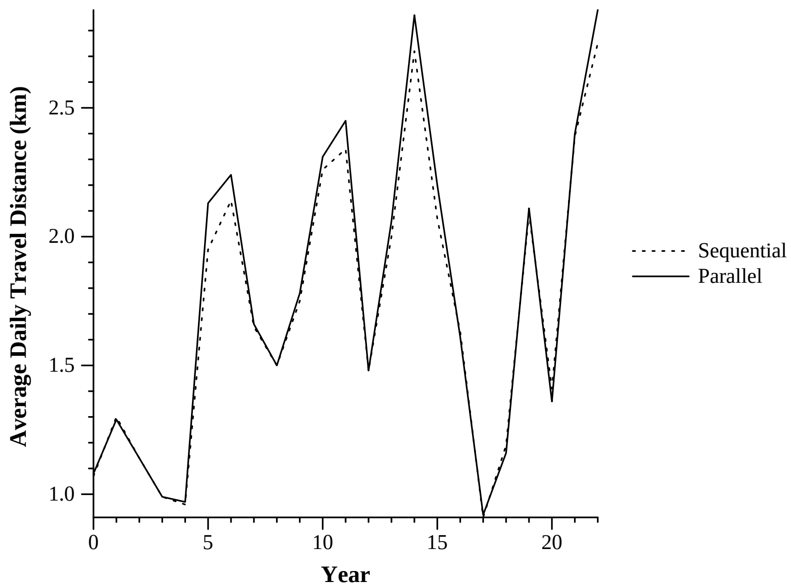


Figure 26. Average daily travel distance per year for an initial deer population of 20,000.

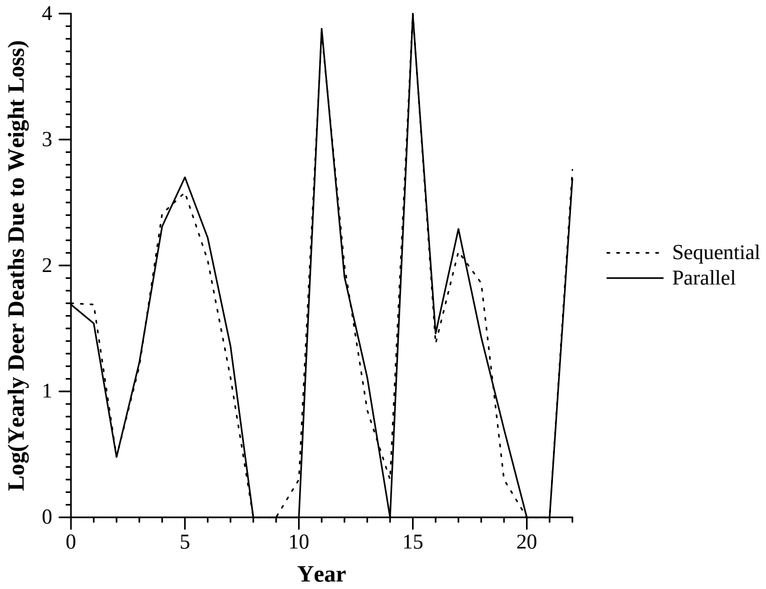


Figure 27. Weight loss deaths per year for an initial deer population of 20,000.

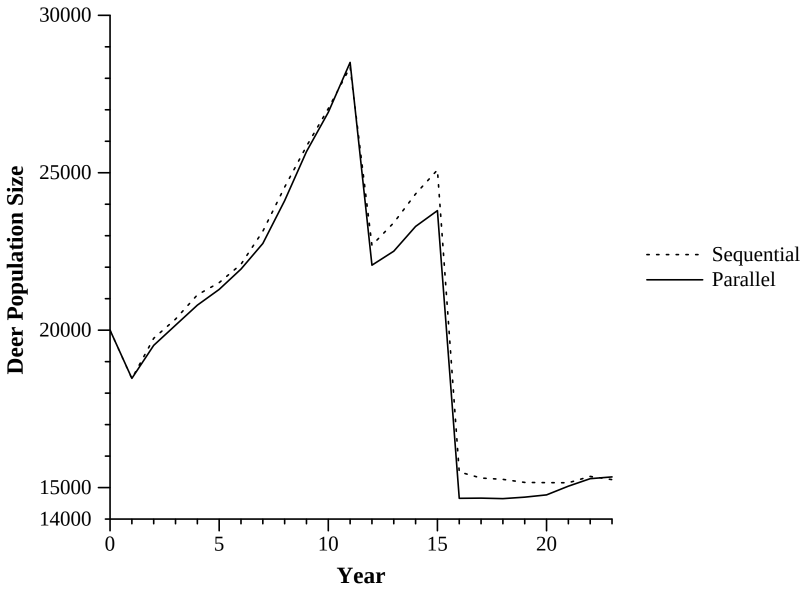


Figure 28. Year-end population sizes for an initial deer population of 20,000.