

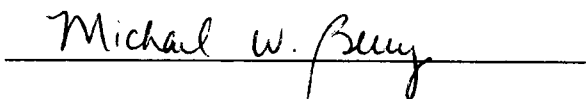
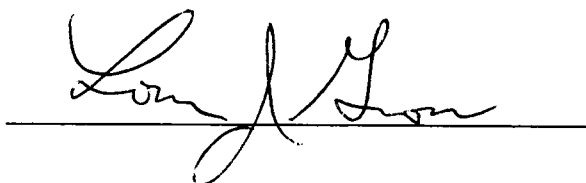
To the Graduate Council:

I am submitting herewith a dissertation written by Scott M. Sylvester entitled "Design and Analysis of a Parallel *Daphnia* Model." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Mathematics.



Thomas G. Hallam, Major Professor

We have read this dissertation
and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor

and Dean of the Graduate School

**Design and Analysis of a Parallel
Daphnia Model**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Scott M. Sylvester

August 1995

Abstract

The subject of this study is the parallelization of an existing serial implementation of an individual-based population model for *Daphnia* and the analysis of the subsequent model. To form the parallel model, the serial model was examined and the inherently parallel features identified. Several aspects of parallel programming were also considered as the form of the parallel model was resolved. Once the parallel algorithm was written, the resulting model was checked to confirm the accuracy of the results and to determine whether the parallel model offered any advantages over the serial model. The analysis revealed that the parallel model returned results consistent with those given by the serial model, and that the parallel model does offer advantages such as improved speed and model complexity over the serial model. The conclusion is that the parallel algorithm advances the state of the model, and reinforces parallelization as a useful tool for ecological modeling.

Contents

Introduction	1
1 Model Introduction	4
1.1 The Serial Model	4
1.2 The Parallel Model	18
2 Model analysis	44
Conclusion	102
Bibliography	105

List of Tables

1.1	List of variables for individual model.	9
2.1	Combination of populations, environmental effects, and nodes used. .	46
2.2	Values for structuring variables used for model analysis	47
2.3	Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 8.	50
2.4	The average distance between the time series data for a population with 1 ecotype, under no effects, run on 16 nodes, and a population with 27 ecotypes, under all effects, run in serial.	54
2.5	Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 8. Scaled.	57
2.6	The average distance between the scaled time series data for a population with 1 ecotype, under no effects, run on 16 nodes, and a population with 27 ecotypes, under all effects, run in serial.	57

2.7	Average Distances. Number of Ecotypes: 3. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.	67
2.8	Average Distances. Number of Ecotypes: 3. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 2.	67
2.9	Average Distances. Number of Ecotypes: 1. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.	68
2.10	Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 8.	68
2.11	Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 2.	69
2.12	Average Distances. Number of Ecotypes: 9. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 4.	69
2.13	Average Distances. Number of Ecotypes: 9. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.	70
2.14	Average Distances. Number of Ecotypes: 1. Effects: None Number of Nodes: 16.	70
2.15	Average Distances. Number of Ecotypes: 1. Effects: None Number of Nodes: 8.	71
2.16	Average Distances. Number of Ecotypes: 27. Effects: None. Number of Nodes: 2.	71

2.17 Average Distances. Number of Ecotypes: 3. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.	72
2.18 Average Distances. Number of Ecotypes: 3. Effects: None. Number of Nodes: 16.	72
2.19 Average Distances. Number of Ecotypes: 3. Effects: None. Number of Nodes: 8.	73
2.20 Average Distances. Number of Ecotypes: 9. Effects: None. Number of Nodes: 8.	73
2.21 Average Distances. Number of ecotypes: 1 Effects: Toxicant. Number of nodes: 2	74
2.22 Average Distances. Number of ecotypes: 1 Effects: Toxicant. Number of nodes: 8	74
2.23 Average Distances. Number of Ecotypes: 27. Effects: Toxicant. Number of Nodes: 8.	75
2.24 Average Distances. Number of Ecotypes: 27. Effects: Toxicant. Number of Nodes: 4.	75
2.25 Average Distances. Number of Ecotypes: 3. Effects: Toxicant. Number of Nodes: 2.	76
2.26 Average Distances. Number of Ecotypes: 3. Effects: Toxicant. Number of Nodes: 4.	76

2.27	Average Distances. Number of Ecotypes: 9. Effects: Toxicant. Number of Nodes: 8.	77
2.28	Average Distances. Number of Ecotypes: 9. Effects: Toxicant Number of Nodes: 2.	77
2.29	The average distance between the time series data for a population with 1 ecotype, under no effects, run on 16 nodes, and a population with 27 ecotypes, under all effects, run in serial.	80
2.30	Averages and variances of the time spent communicating.	88

List of Figures

1.1	Flowchart for the serial code.	17
1.2	Flynn's taxonomy.	20
1.3	Comparison of the total execution time for three different communications models. The line labeled <code>pvm_reduce()</code> shows the total time for execution using the PVM function <code>pvm_reduce()</code> to assess total biomass. The line labeled <code>pvm_mcast()</code> shows total model execution time vs. load balance time when the PVM function <code>pvm_mcast()</code> is used to assess total biomass, and the line labeled pass along shows the total model execution time when the home made communication algorithm is used to assess total biomass.	32
1.4	Time required to send various amounts of data to a single node, and to a local hard drive.	42

2.1	Graph of the total biomass as generated by the parallel and serial models for a population containing 27 ecotypes, under the effects of toxicant, oxygen and temperature. The parallel data was generated on 8 nodes.	51
2.2	Graph of the density of individuals as generated by the parallel and serial models. There are 27 ecotypes in the population, under toxicant, oxygen and temperature effects. The parallel data was generated on an 8 node machine.	52
2.3	Graphs of the total biomass for the reference population run in serial and a population containing one ecotype under no environmental stress run in parallel on 16 nodes.	55
2.4	Graph of the density of individuals for the reference population run in serial and a population consisting of one ecotype under no environmental stress run in parallel on 16 nodes.	56
2.5	Graph of population structure with respect to age for the population containing 27 ecotypes and exposed to all environmental effects. . . .	59
2.6	Plot showing the structure of the population with respect to the lipid content of individuals. The population for this plot contains 27 ecotypes, is exposed to toxicant, oxygen and temperature effects, and was run in serial.	60
2.7	Graphs of the average A_1 value for serial and parallel models.	61

2.8	Graphs of the average age for the serial and parallel models.	62
2.9	Graphs of the average mass of lipid for the serial and parallel models.	63
2.10	Graphs of the average mass of structure material for the serial and parallel models.	64
2.11	Graphs of the average plx value for the serial and parallel models. .	65
2.12	Graphs for the average resource for the serial and parallel models. .	66
2.13	Graph of the ratio and differences for the behaviour of a population as generated by the serial and parallel models.	79
2.14	Graph of the time series averages and standard deviations for A_1 for different populations.	81
2.15	Graph of the time series averages and standard deviations for age from different populations.	82
2.16	Graph of the time series averages and standard deviations for mass of lipid from different populations.	83
2.17	Graph of the time series averages and standard deviations for mass of structure material for different populations.	84
2.18	Graphs of the time series averages and standard deviations for plx levels for different populations.	85
2.19	Graphs of the time series averages and standard deviations for resource level for different populations.	86
2.20	Time scalings for a population with one ecotype.	89

2.21	Time scalings for a population with three ecotypes.	90
2.22	Time scalings for a population with nine ecotypes.	91
2.23	Time scalings for a population with twenty seven ecotypes.	92
2.24	Number of cohorts vs. time for a population containing 1 ecotype under toxicant stress, as generated by the serial model, 2 and 8 nodes.	94
2.25	Number of cohorts vs. time for a population containing 27 ecotypes, under toxicant stress, as generated by the serial model, 2 and 16 nodes.	95
2.26	Number of cohorts vs. time for a population containing 27 ecotypes, as generated on 16 nodes.	97
2.27	Processing time for a serial machine and a 11 nodes PVM network. .	99

Introduction

Ecological systems, such as a *Daphnia* population, are complex and involve many different types of interactions at many different levels. As with other natural systems, an essential tool in understanding how these systems work is mathematical modeling. That is, the quantification of the relevant features of the system and the derivation of equations or system of equations which describe the relationship between the relevant features of the system is an important tool. In the case of the *Daphnia* model, the equations describe the dynamics of a population structured by various classes of individuals. Because the interactions in a *Daphnia* population are complex, the system of equations used to describe the system, while describing only a limited range of features, are still complex. In fact, the system of equations for the *Daphnia* model involve terms which make a closed form solution unattainable. This does not render the model useless, since numerical methods can be applied to find approximations to the solution. Numerical methods require a large number of computations, and hence lead naturally to the use of computers. The more powerful the computer the greater

the accuracy and speed of the simulation. Parallel computers offer far more computational power than is available from serial computers. Therefore, parallelization is a natural step in the development of a model. This paper gives an account of a parallelization of a *Daphnia* model using Parallel Virtual Machine (PVM). The focus of the paper is twofold: the considerations needed to transform a serial model into a parallel model and an analysis of the resulting parallel model.

Parallelization presents an unique set of problems, changing the manner in which many processes are conceived and implemented. In addition, parallelization is only useful if the mathematical model has features which can be parallelized. That is, are there portions of the model which can be computed simultaneously, or does each step of the model depend on the preceding step? The *Daphnia* model is, in fact, parallelizable, and this paper addresses the difficulties arising from converting the existing serial model to a parallel model. A few features of the serial model were not completely parallelizable. For this reason, and because parallel computing with PVM is fundamentally different from serial computing, the parallel model will return results different from those given by the serial model. Thus, it is important to check the validity of the results generated by the parallel model. Checking these results is the other focus of the paper. The analysis of the results shows that the few features which did not translate well to the parallel environment did not seriously effect the results generated by the model. Implementing a parallel model, and using PVM as the vehicle for this task, implies the model should offer some advantage over the serial

model, lest time invested in parallelization be wasted. The principal advantage sought is a decrease in the length of time required to complete a simulation, as well as scalability and portability. To complete the analysis of the parallel model, the extent to which the parallel model satisfies these conditions is determined. In many important ways, the parallel algorithm written for this study offers advantages over the serial model, advances the state of the *Daphnia* model, and lends credence to parallelization as a tool for modeling ecological systems.

Chapter 1

Model Introduction

1.1 The Serial Model

The model used in this paper is an individual-based population model developed by Hallam et. al. [1] for *Daphnia*. *Daphnia* are a freshwater zooplankton commonly referred to as water fleas. An individual-based population model is a set of equations describing the dynamic behavior of a population based on the attributes of the individuals in the population. The system consists of an individual model, which incorporates individual parameters and governs individual behavior, and a population model which aggregates the individual behavior into population behavior and allows for the inclusion of population level effects. Because this approach is based on individual dynamics and tracks individual behavior, the structure, present in the population resulting from differences between individuals, is preserved. Population

structure is as important to describing and understanding how a population behaves as knowing classical features, such as average density or average birth rate. Having variability in the population model allows individuals to react differently when exposed to environmental stresses. This is an advantage over aggregated models, where all individuals are assumed to be equivalent and thus have identical reactions to stress. This modeling technique also facilitates the inclusion of environmental effects. An individual is directly affected by changes to its environment, so it makes sense to incorporate these effects at the individual level. Even when an effect is spoken of in terms of impact on a population, the response of the population is a consequence of individual behavior. Another argument for this type of model is that the effects at the individual level are better understood, and information detailing effects at the individual level is generally available. Thus, when a new factor is to be included, the effects need only be understood and incorporated at the individual level: the model allows for the extrapolation of these effects at the population level.

The design of a model should reflect those features of the population one wishes to investigate. The *Daphnia* model was designed to explore the effects of a lipophilic toxicant on the population. For this reason, an individual daphnid is represented in terms of acquisition, storage and use of lipid and structure material. Structure material is taken to be an aggregate of protein and carbohydrate. The acquisition of these resources is from food, where it is assumed only the lipid portion of the food contributes to the lipid of the individual, and only the structure of the food con-

tributes to the mass of structure; *i.e.* no lipid is synthesized from structure material by the individual. Loss of these resources is described in terms of the energetics of the individual. Lipid and structure material are converted into energy to fulfill the requirements of the individual for work, maintenance, and reproduction. Based on information taken from Hallam et. al. [2], the rate and manner of these losses and gains were estimated. The input from food is assumed to be a continuous process, and the individual is assumed to feed at a resource level which is constant through its lifetime. Likewise, losses due to work and maintenance are taken to be continuous throughout the individual's lifetime. Reproductive losses are considered to be discrete events, because they occur over a period of time which is short relative to the lifetime of a *Daphnid*. Initial time at which an individual gives birth, "birth time", is determined by a prescribed length of the individual. After the initial birth event, an individual will give birth every four days assuming it has sufficient reserves to produce an egg. Failure to give birth does not change the phase of the birth pattern. If an individual does not give birth at one time because it has failed to satisfy reproductive requirements, it will have to wait for the next birth time before it can possibly give birth again. Prior to each birth event, *Daphnia* shed and regenerate their carapace. Thus, losses from this sink will be assessed at birth times, and since carapace formation is short in comparison to other energetic processes, it will also be assessed as a discrete loss. When *Daphnia* shed their old carapace, the new individuals from the last birth event are released into the environment. The current brood of newborns

are held within the mother's pouch until the next birth event. While in the carapace, the offspring draw energy from neither their mother nor the environment, but survive on their individual stores of lipid and structure material. This is significant since it implies the dynamics of the individual while in the brood pouch are different from other *Daphnia*. After release, new individuals immediately assume the behavior of the other *Daphnia* in the population. Under normal circumstances, *Daphnia* will reproduce parthenogenetically, thus a newborn is a clone of its parent. The equations for the individual dynamics are:

$$\frac{dm_{l_i}}{dt} = g_{l_i} = \frac{A_{oL}x_{s_i}m_{s_i}}{A_{1_i}m_{s_i}^{1/3} + A_2x_i} - \begin{cases} A_3(m_{l_i} - \epsilon m_{ps_i}) & D > E \\ A_3(m_{l_i}l - \epsilon m_{ps_i})\frac{D}{E} & D \leq E \end{cases} \quad (1.1)$$

$$\frac{dm_{s_i}}{dt} = g_{s_i} = \frac{A_{oS}x_{s_i}m_{s_i}}{A_{1_i}m_{s_i}^{1/3} + A_2x_i} - \begin{cases} A_4(m_{s_i} - m_{ps_i}) & D > E \\ A_4(m_{s_i} - \epsilon m_{ps_i})\frac{D}{E} & D \leq E \end{cases} \quad (1.2)$$

$$D = A_5(m_{l_i} + m_{s_i})^{1/3} + A_6(m_{l_i} + m_{s_i})^{2/3} + A_7m_{l_i} + A_8m_{s_i} \quad (1.3)$$

$$E = 37.68A_3(m_{l_i} - \epsilon m_{ps_i}) + 16.75a_4(m_{s_i} - m_{ps_i}) \quad (1.4)$$

with the discrete losses due to births being given by:

$$\frac{(e_M - e_m)(m_{l_i} - \epsilon m_{ps_i})}{A_9E_s + (m_{l_i} - \epsilon m_{ps_i})} + e_m$$

$$\min \left\{ \max[0, \sigma(\sqrt{(1/3)m_{s_i}/\beta} - \eta)], (m_{s_i} - m_{ps_i}/\beta) \right\}$$

for losses from lipid and structure respectively. For individuals still in the brood pouch, the dynamics are given by:

$$\frac{dm_{l_i}}{dt} = g_{l_i} = -A_{14}m_{l_i} \quad (1.5)$$

$$\frac{dm_{s_i}}{dt} = g_{s_i} = -A_{15}m_{s_i} \quad (1.6)$$

The indices in each equation denotes the fact that a different equation is needed for each ecotype within the population. The differences between the equations arise from the different parameter values for the variables. The meaning, units, and parameters for each of the above variables are available in Table 1.1.

In practice, the individual model will be used to describe groups of individuals rather than single individuals. Individuals clustered in this way are cohorts of siblings and are assumed to be identical both physiologically and chronologically. This simplification will permit the model to have large numbers of individuals without having to store huge amounts of information. The cohorts are then aggregated using a population model, which in this case is given by a system of McKendrick-von Foerster equations of the form:

$$\rho_{i,a} + \rho_{i,t} + (g_{s,i}\rho_i)_{m_s} + (g_{l,i}\rho_i)_{m_l} = -\mu\left(\sum_{i=1}^N \rho_i\right) - \gamma(a, m_l, m_s) \text{ for } 1 \leq i \leq N \quad (1.7)$$

$$I_i(a) = \rho_i(a, 0) \quad (1.8)$$

$$B(t, m_{l_0}, m_{p_0}) = \int_0^{M_l} \int_0^{M_s} \int_0^{MaxAge} \beta(m_l, m_s, a) \rho_i dm_l dm_s da \quad (1.9)$$

Here ρ is the density of individuals in the population, $g_{l,i}$ and $g_{m,i}$ are the functions given in Eqs. 1.1 and 1.2, $\mu()$ and $\gamma()$ represent density-dependent and size/age class mortality respectively, and $\beta()$ describes birth rate as a function of size and age.

Table 1.1: List of variables for individual model.

Variable name	Units	Description
m_s	mg	mass of structure material.
m_l	mg	mass of lipid material.
x	mg	resource.
x_s	mg	portion of the resource which is structure material.
x_l	mg	portion of the resource which is lipid material.
m_{ps}	mg	portion of the structure which is considered to be nonlabile.
A_{0L}	...	Assimilation rate/ingestion rate.
A_{0S}	...	Assimilation rate/ingestion rate.
A_1	$mg^{2/3} \cdot d^{-1} \cdot mm^{-3}$	reciprocal of constant allometrically relating maximal filtering rate to organism length squared.
A_2	d	reciprocal of constant allometrically relating maximal ingestion rate to organism length squared.
E_s	egg	maximum number of eggs made from labile structure.
e_m	mg/egg	minimal lipid content per egg.
e_M	mg/egg	maximal lipid content per egg.
D	J/d	power demand.
E	J/d	power supply.
δ	mm^{-1}	slope of line relating organism length to maximum number of eggs per clutch.
β_1	mg/mm^3	isometric constant relating weight to length.
η	mm	reproductive reference length.
ρ	mg	structure content per egg.
A_3	d^{-1}	labile lipid mobilization rate.
A_4	d^{-1}	labile structure mobilization rate.
A_5	$J \cdot mg^{-1/3} \cdot d^{-1}$	multiplier representing energy cost for activity due to viscous forces.
A_6	$J \cdot mg^{-2/3} \cdot d^{-1}$	multiplier representing energy cost for activity due to inertial forces.
A_7	$J \cdot mg^{-1/3} \cdot d^{-1}$	energy required to maintain 1 mg lipid per day.
A_8	$J \cdot mg^{-1} \cdot d^{-1}$	energy required to maintain 1 mg structure per day.
A_9	mg/egg	half saturation constant modulated by E_s .
A_{10}	J/mg	energy required to mobilize labile structure for eggs.
A_{11}	J/mg	energy required to mobilize labile lipid for eggs.
A_{12}	$mg^{-1/2}$	allometric constant relating carapace to total weight of organism.
A_{13}	$J/mg^{3/2}$	energy required to mobilize structure for carapace formation.
A_{14}	d^{-1}	differentiation and utilization rate of lipid while in brood pouch.
A_{15}	d^{-1}	differentiation and utilization rate of structure while in brood pouch.

Source: Hallam et. al. [2] 1995.

Each equation, corresponding to a subscript i , models a subpopulation in terms of its age and size structure. Subpopulations consist of cohorts with identical physiological parameters but which may be of different ages. These subpopulations are referred to as ecotypes, and it is these ecotypes which provide structure within the overall population. It is important to note that the system of equations are linked through density-dependent mortality; without this term the equations could be solved independently. Through experimentation, changes in the level of resource (x), proportion of resource which is lipid (plx), and the parameter relating organismal length to filtering rate (A_1) were among those shown to have the greatest impact on individual behavior. Thus, an ecotype is defined by the levels of each of these three variables in order to generate a population with the greatest variability. The values for each of these parameters are set up in a regular fashion, with a baseline value and percentage chosen for each parameter. The allowable levels for each variable are the baseline value, a level depressed by the percentage and a level elevated by the percentage. In this way, three levels are provided for each structuring variable, and a total of twenty seven ecotypes. Other variables could have been used, extra variables added or more levels for each used. Each of these refinements would give a more diverse model population, more closely approximating natural variation seen in populations. The cost of increased variability is increased computation time. One of the key goals in parallelization is to increase population variability without time penalty.

The next step is to solve the system of partial differential equations (p.d.e.s). A

standard technique for doing this is the method of characteristics. Upon applying the method of characteristics, the system of p.d.e.s becomes the system of ordinary differential equations (o.d.e.s):

$$\frac{da}{dc} = 1 \quad (1.10)$$

$$\frac{dt}{dc} = 1 \quad (1.11)$$

$$\frac{dm_{l_i}}{dc} = g_{l_i} \quad (1.12)$$

$$\frac{dm_{s_i}}{dc} = g_{s_i} \quad (1.13)$$

$$\frac{d\rho_i}{dc} = -\mu\left(\sum_{i=1}^N \rho_i\right) - \gamma(a, m_{l_i}, m_{s_i}) \quad (1.14)$$

with the initial and boundary conditions remaining unchanged. In Eqs. (1.12) and (1.13) the terms g_{l_i} and g_{s_i} refer to the same variables used in Eq. (1.1) and (1.2) including the terms for discrete losses, or Eq. (1.5) and (1.6) if the individual is still in the mother's carapace. Note that the equations for the population model are linked through density dependent mortality, $\rho\left(\sum_{i=1}^N \rho_i\right)$. Without this term the subpopulations could be solved separately.

The method of characteristics also provides a nice link between the mathematics and the biology. With the method of characteristics, the solution surface is obtained by solving characteristic curves. Each curve starts at some point along either the initial or boundary condition and is governed by the five equations (1.10 - 1.14). Biologically, each point along the initial or boundary condition represents a cohort and the path of characteristic curve through the time-lipid-structure axes is governed by

equations (1.12) and (1.13) which are the same equations used to describe individual behavior. Thus, the path of a characteristic curve through the time-lipid-structure axes traces out the life history of a cohort. For this reason, the terms cohort and characteristic are used interchangeably throughout the remainder of this thesis.

In order to solve the system of o.d.e.s in Eqs. (1.10 - 1.13) a numerical method is applied. For this model a one step Runge-Kutta method is employed, commonly referred to as a backward Euler method. The full details of this numerical method are not germane to this thesis, and are not needed to understand the parallelization.

The model has been augmented in several ways over the description given here. The effect of a toxicant, dissolved oxygen and temperature have all been incorporated. For an accurate description of these additions see Koh et. al. [3] for dissolved oxygen and temperature effects and Hallam et. al. [4] for sublethal toxicant effects. Because the implementation of these effects takes place at the level of the individual, they do not seriously change how the model was parallelized, and will be omitted.

To better understand how the model was parallelized, a short description of the serial version is in order. This description will include file input/output and computational short cuts which have an impact on the nature and viability of the parallelization.

Initially, data is read from files specified by the user. The information in these files can roughly be broken down into two categories: general model parameters and cohort (characteristic) parameters. The general parameters describe features of the

model which are common to all cohorts: variables such as start/end times for the simulation, increment size for time, toxicant levels in the water, just to name a few. These common values are parameters which do not change in response to the *Daphnia* population and, with the exception of toxicant, will not change through time. The cohort parameters are the data needed to describe each cohort in the population.

For any given run, the cohort information is the initial condition for the system of p.d.e.s. Cohort information includes variables such as current mass of lipid, density of individuals in a cohort, resource level, lipid level in the resource, and other parameters which vary from one individual to another. These values are those which distinguish a cohort as belonging to a particular ecotype. The values of many of these variables will change through time and in response to the state of the population. The list of cohorts is stored in a stack which simplifies data management in several ways.

Next the program enters the outer of two loops, which controls the passage of time for the simulation. The current time is checked against the end time, and the stack checked to insure that it is non-empty. If either condition fails, the simulation ends, and the program proceeds to the end of the code to perform some house keeping tasks. If the conditions are met the simulation proceeds. The program next assesses the total biomass of the population. This entails summing the mass of individuals over all cohorts. This is one of two points in the algorithm where information about the entire population is needed; and, because of this, it will become a point of interest in the parallelization, where the population information is spread over many nodes.

The sum will be used later in the code to determine density dependent mortality.

Next, the program enters the second loop which will cycle through the cohorts in the stack, starting at the top and working down. Within this loop, the code passes a single cohort and the total biomass to a black box function, which performs three tasks: The black box will advance the cohort one time step, assess mortality within this cohort, including density dependent and other forms of mortality, and, finally, determine if the cohort has given birth to a new cohort. It is within the black box that the numerical method is used to solve Eq. (1.10 - 1.14) and is set into the code, using the FORTRAN programming language. For the most part, each step of the black box is dependent on the previous step, rendering this portion of the process difficult to parallelize. The black box will then return the cohort in its modified state. If the cohort has died, it is removed from the stack.

If the cohort has given birth, the program will do one of two things. If there are already new cohorts in this time step, the algorithm will seek for one which is in the same ecotype as the one in hand and merge the two into a single cohort. Failing this, the algorithm will make a new space atop the stack and the new cohort is placed in this space. There is, of course, the possibility that some other individual will give birth, and its children may be merged with the new cohort. Merging births is a computational trick which keeps the number of cohorts down, referred to hereafter as combining births. It is known from experimentation that if every new birth resulted in a distinct element, the stack has the potential to become extremely large, on the

order of tens of thousands. Cycling through such a number of elements for even a single time step consumes huge amounts of computational time. By combined births, the algorithm is guaranteed to add no more than 27 new elements to the stack for any given time step. This number is small enough that deaths from the population tends to stabilize the total number of characteristics at a relatively small number, on the order of thousands in the worse cases.

After dealing with births for the current cohort, the model advances to the next cohort and returns to the top of the second loop. Notice that the birth function places the new cohorts on top of the stack, and the advancement through the stack is performed downward. This means new cohorts are not processed during this time step, which is appropriate, because the only action a new cohort has in this time step is being born. After the program has advanced all cohorts by one time step, it may need to output information about the current state of the population. This is necessary to obtain a time lapsed picture of the population dynamics. In order to capture some of the transient behavior of the population, this information is dumped at least every two days. Finally, the program advances the time and returns to the top of the first loop. On this pass through the cohorts, the newly born individuals will count in the assessment of total biomass, and the second loop will start again at the top of the stack, which may have moved due to the addition of births. As mentioned before, if, at the top of the first loop, the stack is empty, indicating all the individuals have died, or the end of the simulation has been reached, the program exits the outer

loop.

At this point, the flow of the program proceeds to the end of the code. If the population has gone to extinction, then a message to this effect is printed and the program exits. If some individuals have survived to the end of the simulation the program will print all the relevant information about each cohort to a file and exits. This cohort information can be used to restart the model at a later date. Fig. 1.1 roughly summarizes this flow.

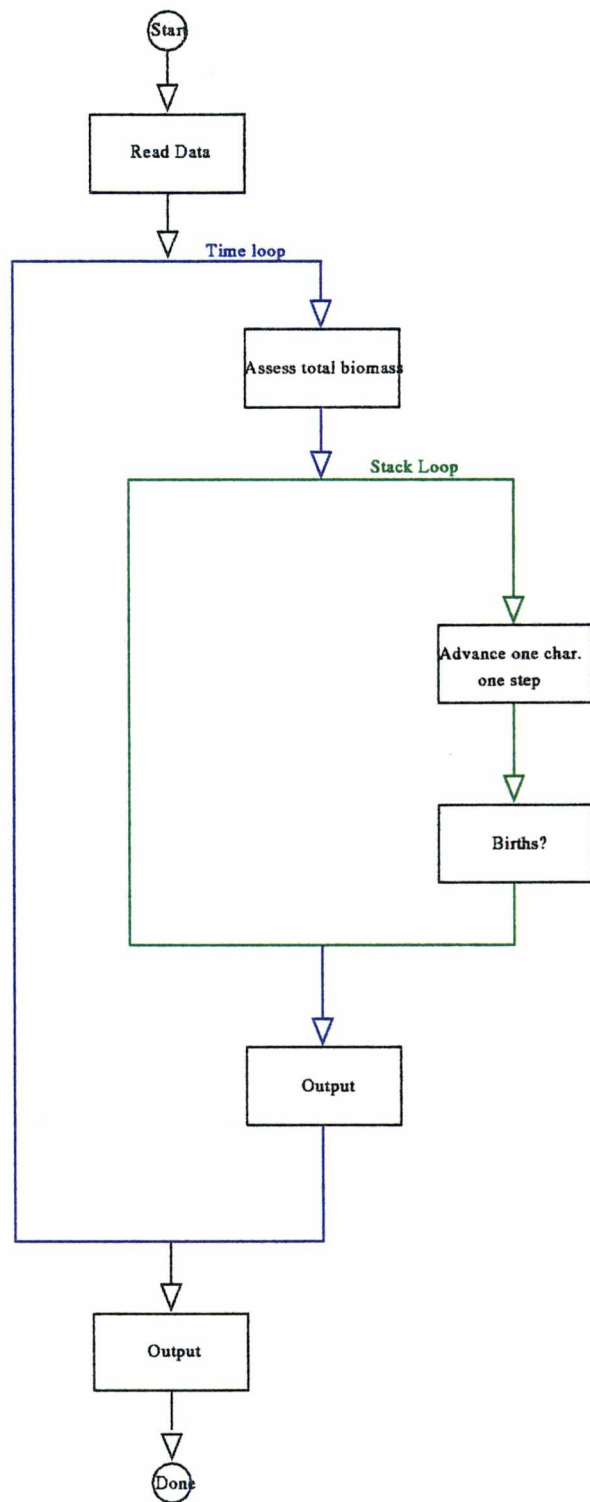


Figure 1.1: Flowchart for the serial code.

1.2 The Parallel Model

Writing a parallel algorithm involves blending a myriad of different considerations; some are features of parallelization in general, such as internodal communication, or the need and manner of node synchronization, while others are specific to the architecture of the parallel computer for which the parallel algorithm is written. In this study, the parallel algorithm was written using PVM [5] (Parallel Virtual Machine), a software package whose development began at Oak Ridge National Laboratory and the University of Tennessee. Through continuing development, PVM has become a useful tool for research into parallel programming. Simply put, PVM is a collection of message passing functions and software agents which unite several different computers, and allow both programmer and user to treat the group of computers as a single computational resource. The decision to use PVM will also affect many details of the parallelization. Other considerations are requisites for programming in general, such as reusing existing code to minimize effort and error, or portability to enhance the viability and longevity of the code. In addition to these considerations, the parallel model should offer an advantage over the serial model. This section describes how the serial model was parallelized, and how working in a parallel environment and using PVM shaped the outcome of the parallel model.

For the *Daphnia* model, and for many other models, the principal goal in parallelization is to increase model accuracy, without increasing the time needed to run

the model. Mention has been made about increasing model accuracy; because this prompts the writing of the parallel model, some care should be taken to clarify this point. The design of the *Daphnia* model offers many directions for refinement. One avenue is to increase the resolution used to simulate time. The model is based on a numerical method applied to a system of o.d.e.s. The numerical method makes the common assumption that, over a small enough interval, the solution can be approximated by a straight line. As this interval becomes smaller, the assumption becomes more accurate, yielding a solution closer to the true solution. Unfortunately, as the interval becomes smaller, the amount of work needed to find the solution increases. Another path for model refinement is increasing the diversity of the model population. Currently the model tracks at most twenty-seven ecotypes. By increasing the number of ecotypes, either through adding physiological levels, or through adding parameters about which the population is structured, the model more accurately approximates the composition of actual populations. Though not truly a model refinement, running the model for longer periods of time is desirable in some situations. Each of these refinements is useful in obtaining more realistic results from the *Daphnia* model. Each refinement also increases the work load, and thus the time needed to complete a simulation. The obvious solution is to run the model on more powerful computers. Parallel environments, such as PVM or true parallel machines, offer more computational power than serial machines are currently capable of providing, making parallelization a natural choice. For this reason, the future ability of the parallel

		Number of Data Streams	
		Single	Multiple
Number of Instruction Streams	Single	SISD	SIMD
	Multiple	MISD	MIMD

Figure 1.2: Flynn's taxonomy.

model to handle more complex simulations can be measured by the speed improvement the parallel model provides over the serial model. Decreasing the execution time of the model is a goal which describes the desired results of parallelization; other goals clarify the means to these ends.

Implementing the parallel model using PVM affects the type of parallel model used. In parallel computing, there are two basic types of parallel models: control parallel and data parallel. The distinction is based on the manner in which the parallel computer executes instructions and how these instructions operate on the data. Flynn [6] offers a classification for computers based on instruction execution and data manipulation which is illustrated in Fig. 1.2. Along the horizontal axis is data, either a single datum or multiple data. Along the vertical axis are instructions,

again either single or multiple. The resulting grid outlines four classes of computers. SISD computers are the most common computers, utilizing a single instruction stream and a single data stream. The true parallel machines are MIMD and SIMD computers. SIMD computers are machines which take many different pieces of data and perform the same instruction on each datum in one clock cycle. In this type of machine, the nodes are in lock step. Every node must execute the same instruction each cycle. Algorithms implemented on such machines are referred to as data parallel, since the data is processed in parallel. An example of this type of computer is the MasPar MP-2.

2. MIMD computers can also have different nodes working on different data. They are distinguished from SIMD because their nodes are not in lock step, each node may execute different instructions on data in a clock cycle. For this reason, algorithms implemented on such a machine are called control parallel, since the instructions are executed in parallel. Examples of this type of machine are the Thinking Machine CM-5 and the IBM SP-2. The last category of computer is the MISD machine describing a class of machines which can perform multiple instructions on a single data element in a single clock cycle; however, it is debatable if such a machine even exists. PVM does not actually fit into any of these categories since it is not truly a computer, but a program which simulates a parallel computer. Details aside, each node in a PVM machine is free to execute any instruction, independent of the other nodes with which it is joined, and each node can execute these instructions on different sets of data. Hence, PVM is referred to as MIMD paradigm, since it models the behavior of a

true MIMD machines. Using PVM, therefore, implies implementing a control parallel model. This, in turn, implies the model can best use PVM by converting a process involving many, mutually independent, steps into a few steps by giving each node a part of the process.

Another common objective in software design is portability, allowing existing code to remain useful as the underlying hardware changes. Portability is a strong argument for using PVM, because it supports many different platforms and networks, including true parallel computers such as the IBM SP-2 and the Intel Paragon. PVM adds another feature to portability, which is the ability to unite machines with different architectures into a single virtual computer. The unity extends to the process control level, where a user can start and stop parallel jobs and obtain information from the virtual machine about the status of jobs, without being logged into each machine. More importantly, PVM provides a set of message-passing routines which allow the various tasks running on the virtual machine to communicate with each other. PVM provides these functions at the C or FORTRAN level, making code easy to write, and, in the case of the *Daphnia* model, easy to integrate into existing code. In addition, the provided library functions closely mirror those used on a true parallel machine, so existing parallel models can be implemented without having to completely reevaluate their implementation.

As well as being portable to many different machines, the parallel algorithm should scale across different numbers of nodes. In this study, each of the machines comprising

the PVM network runs one copy of the model, and each machine is referred to as a node of the PVM network. This is important, since the number of nodes may change through time or by location. This indicates all the nodes should run the same program. Scalability refers to more than making the most use of available resources, but also to the best use of the available resources. For the *Daphnia* model, more nodes are not always better. As with many parallel implementations, there will be a need to communicate information between nodes. As the number of nodes is increased, more messages need to be passed. As the time spent communicating increases, performance declines. The question is: what number of nodes provides the greatest increase in speed? If t_1 is the time required for one node to complete the model, and t_n is the time required for n nodes to complete the same task, the speed improvement can be measured by the ratio of t_1 to t_n . A ratio larger than unity indicates the parallel model requires less time than the serial model to finish the simulation, a ratio less than one means the parallel model requires more time to finish. The ideal situation is to write an algorithm in which the ratio doubles as the number of nodes doubles. In practice, the ratio tends to fall off after a certain number of nodes is surpassed. Success in parallelization and applicability of parallel computation to a problem can be measured by how closely the ratio follows the ideal and how consistently the ratio increases with increasing nodes. The algorithm should be written to fit these criteria as best as possible.

Writing a parallel *Daphnia* model was not something which was done from scratch.

A serial version of the model already existed. In order to facilitate the creation of a parallel model, as much of the existing code as possible was reused. A completely new code could have been written for the parallel model, but this might miss the point of the exercise. The objective was to write a parallel algorithm, not to rewrite the program. Considering the fact that rewriting the program would have also required a new round of debugging, using existing code saved a huge amount of time, which was better spent addressing the problems of parallelization.

Improved accuracy, using PVM and, hence, implementing a control parallel algorithm, scalability and portability are all considerations which were weighed as the parallel model was written. These features are not unique to the *Daphnia* model, but are considerations for many biological models. For this reason, the answers to these problems have importance beyond this one model.

Of the various objectives for this model, the flow of the serial model is the best place to start, since reusing code is an objective and hence this flow is the structure which is to be parallelized. The flowchart in Fig.1.1 is intentionally simplified, presenting only those features which played an important role in the parallelization. The essence of parallelization is taking a process which requires many steps and reducing it to a few steps, at the very best, one step. In particular, these steps need to be mutually independent. Fig. 1.1 highlights just such a process in the model, the loop controlling advancement of cohorts to the next time step. This is also the loop which requires the bulk of the processing time within the serial model, requiring

one iteration for each characteristic. If each of M nodes of a parallel computer are given an equal portion of the population, then for each "step" the parallel machine takes through the inner loop, M characteristics will be advanced. Before completely accepting this approach, its effects on the rest of the model should be evaluated. The two principal concerns will be assessing total biomass and combining births. These features are important because they require information about the entire population, which is spread over the nodes of the parallel computer. Other aspects of the model are also of concern, such as tracking time within the parallel model, and file I/O, because these details are important to the correct functioning and usefulness of the model.

At the beginning of each time step, before the inner loop is entered, the model needs to assess the total biomass of the population in order to determine density-dependent mortality. Because the population is spread across several nodes, this presents a small problem. There are many possible solutions to this problem, the simplest is to have each node find the biomass of the cohorts it has locally and then have the nodes communicate this information amongst each other in such a way that the total biomass can be assessed. For many architectures summing over the nodes can be accomplished with a built-in library function specifically designed for this task. When a node calls such a function, it passes in a number which is its contribution to the sum. After all the nodes have called the function, it returns to each node the sum of the numbers passed to it by the nodes. These functions are

referred to as reducing functions, because they collapse many pieces of information into one. Typically the best approach is simply to use the vendor-provided functions, as they are optimized for the internal network of the supported machine and hence are likely to be faster than any home-made algorithm. PVM also has a reduction function called `pvm_reduce()`. In addition, PVM provides several other means by which information can be shared across nodes. As will be shown, a little effort before the code is written will reveal a more efficient mode for communication using one of these other functions.

Before describing the communication algorithms, there are two supporting PVM issues which should be addressed. PVM offers the programmer the option of dealing with nodes in terms of groups. Groups are sets of nodes which may comprise the entire virtual machine or some subset thereof. Groups allow for the restriction of certain types of operations to a subset of the virtual machine, freeing the programmer from having to devise an algorithm to complete the same task. In order for a virtual machine to take advantage of a group structure, some set of nodes must join a group by calling `pvm_joingroup()`, at which time a daemon is started to track which nodes are in which groups. Each time a group operation is called, the group server must be consulted to determine which nodes are in the group. Groups make the program simpler, however they add an extra layer of processing overhead to the program resulting from the group server. The other point of interest when using PVM message passing functions is direct routing. With the function `pvm_setopt()`, the programmer

can turn on direct routing, which gives a hint to the PVM daemon to keep a connection between two nodes open after a message has been sent. This will eliminate the need to reopen the connection as more messages are passed between nodes. According to the PVM manual [5], this hint can increase the rate at which data is transferred by as much as a factor of two. The price for the speed-up is that direct routing is only effective if the number of connections associated with a node is small. The PVM manual states that as the number of connections between a node and its neighbors increases, direct routing becomes less effective. The design of the communication algorithm should reflect these two features. In devising the communication algorithm, group operations should be avoided, as they add an extra layer of processing, and the fewest number of messages should be passed between machines to take best advantage of direct routing.

For the reasons given above, `pvm_reduce()` may not be the best option available. Use of `pvm_reduce()` is further discouraged because, unlike the reduction operation in other parallel languages, `pvm_reduce()` provides only one node in the virtual machine with the results of the reduction. After reduction, another function or set of functions needs to be called to redistribute the total biomass. These two reasons combined indicate another mode of communication may be profitable.

At the beginning of each time step, the model needs to evaluate the total biomass in the population. Each node in the virtual machine has the biomass for those cohorts in its local stack. To find the total biomass, the model will have to sum these numbers.

As has already been stated, limiting the number of messages is desirable. What is the smallest number of messages which can be passed and still find the total biomass? The problem can be solved with a theorem from graph theory. If each node of the virtual machine is considered to be a node in a graph, and each message passed to be a link from one node to some other node, the question becomes one of finding the fewest number of links needed to connect the graph. The theorem is as follows:

Proposition 1 *Given a graph with N nodes, there must be at least $N - 1$ links to connect the nodes.*

Proof: Clearly for 2 nodes we need only one link. So the statement is true for $N = 2$. Suppose the statement is true for $N - 1$ nodes. Then, given N nodes, any subset of $N - 1$ can be connected with $N - 2$ links. The N^{th} node can be connected to any of the already $N - 1$ connected nodes, with at most one link, as in the case $N = 2$. Thus the graph is connected with $N - 1$ links.

□

So any algorithm will have to use at least $N - 1$ communication events, with the best coming as close to this as is possible. Fortunately, PVM offers several different functions for communicating information between nodes, at least one of which will use exactly $N - 1$ messages.

What has been described so far is only the communication required to find the total biomass. In practice, when the total biomass is found, only one of the nodes

has the value. It still remains to send the value to the other nodes. This is easily taken care of using `pvm_mcast()`, which sends a message to a set of nodes. This is the approach taken in each of the communication algorithms. Other methods could be implemented; however, they would be more complex to write and would not offer any real speed advantage. Using `pvm_mcast()` to send the total biomass to the other $N - 1$ nodes will require $N - 1$ messages, resulting in a total of $2(N - 1)$ messages for the biomass to be found and distributed to all the nodes.

One way around distributing the biomass is to use `pvm_mcast()`. With this function each node can send its local biomass to all the other nodes, letting each machine sum the numbers individually. It may seem wasteful to do the same sum separately on each of the nodes, but remember summing the numbers on each of the nodes is done in parallel. Though this method avoids the need to redistribute the total biomass, there are several drawbacks. First, each node will send $N - 1$ messages, since, for each, there are $N - 1$ neighbors which need to receive the message. This results in a total of $N(N - 1)$ messages being sent. This is far more than the optimal value of $2(N - 1)$. Also, because there are a large number of messages being passed from machine to machine, direct routing will not be very effective.

Another model for communication, which follows naturally from the concept used in the proof of Prop.1, uses the function `pvm_send()`. This function sends a message from one host to another. The basic idea is to have a node pass its local biomass to another node, the receiver will add this number to its local biomass and pass the result

to another node. The subtotal is passed through every node exactly once, after which the last node will have the total biomass. This node will then undertake distribution of the final value. Given N nodes, there are several topologies which connect all the nodes with $N - 1$ links; however, the nature of Ethernet makes them all functionally identical. Ethernet only allows one machine per subnet to send a packet at any given time. So, independent of the geometry imposed on the network, the machines can only communicate in serial; *i.e.* one of them will talk first, then another, etc.

If different parts of the virtual machine are on different subnets then it is possible to parallelize the communication. Parallelizing the communication over subnets has two drawbacks. First, IP (Internet Protocol) is designed to hide the differences between subnets, so some amount of work needs to be put into discovering which machines are on which subnet. Secondly, and more importantly, an algorithm designed to take advantage of subnets will be very specific to the network being used, making the code non-scalable, and non-portable. Other networks are available which would allow high levels of parallelism in the communication, but again, the network may change with time or location so writing code for one network results in a non-scalable or non-portable program. Because Ethernet only allows serial communication, the simplest approach for communication, conceptually and programmatically, is to consider the machines to be arranged in a line and to control the manner in which the machines communicate. The control mechanism is simply to have the subtotal of local biomasses pass from left to right, leaving the last machine in the line with the total

biomass. For groups of machines which are close to each other, the sequence in which the subtotal travels need not reflect the geographical arrangement of the machines. It is assumed the machines comprising the virtual machine are close to each other and packets do not have to traverse a great deal of networking hardware, and hence the time for a packet to travel between any two machines is small. After passing the subtotal to the right, each machine will listen for a message from the right-most node, which possesses the total biomass. The right-most node, after receiving the subtotal and adding in its local biomass, uses a `pvm_mcast()` to send the total biomass to the other nodes. Programmatically, this method is slightly more complex, but also does not use groups, and uses the smallest number of messages, and therefore should be faster than either `pvm_reduce()` or the method involving `pvm_mcast()`. This is borne out by Fig. 1.3, which contains graphs of total execution time at different load balance times. The data was collected for a PVM network consisting for eleven machines, each machine ran one copy of the model. Each method was run several times, to assure the accuracy of the results. In testing the communications model a population with twenty seven ecotypes, under all environmental stresses was used.

Having to assess total biomass introduces a synchronization point within the program, since no node can process its cohort until it has the current value for the total biomass, and each node will contribute in part to this value. Synchronization raises a new question: "how will frequent synchronization affect the performance of the model?"

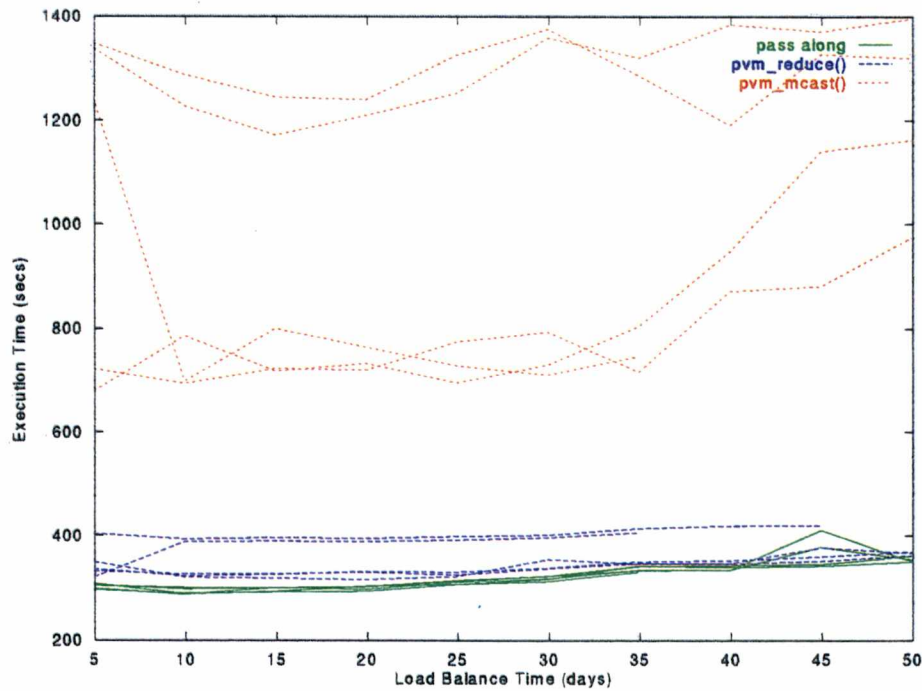


Figure 1.3: Comparison of the total execution time for three different communications models. The line labeled `pvm_reduce()` shows the total time for execution using the PVM function `pvm_reduce()` to assess total biomass. The line labeled `pvm_mcast()` shows total model execution time vs. load balance time when the PVM function `pvm_mcast()` is used to assess total biomass, and the line labeled `pass along` shows the total model execution time when the home made communication algorithm is used to assess total biomass.

Unfortunately, this question can only be fully answered after the program has been written and run on data. However, steps can be taken in anticipation of this problem to minimize its effects. The ideal situation is for all the nodes to arrive at the synchronization point at the same time, so that no node will sit idle while others finish their work. By giving each node an equal amount of work, the length of time any node spends waiting for others is minimized. This problem is further complicated by the fact that the *Daphnia* model tracks a population which is constantly changing in numbers of cohorts as individuals die or are born. As a result, after some length of time, the number of cohorts on each node could be drastically different from that with which it started. The consequences of load imbalance are varying lengths of time required by each node to advance its set of cohorts. Since the nodes need to synchronize every time step, this imbalance in loads will result in some nodes sitting idle while others finish their part of the task. This is very inefficient, since, at a point of imbalance, the algorithm is no longer taking full advantage of the parallel machine, which results in a waste of the resource and, more importantly, longer amounts of time for model solution. Thus, periodically, the load needs to be balanced across the available nodes. The algorithm which performs this task consists of two principal parts: a generic method for balancing the load which can be implemented on any control parallel computer and the second part which is specific to PVM, designed to compensate for the fact the PVM networks may consist of machines of differing computational abilities.

The general load balancing algorithm was designed by Siddharthan Ramachandramurthi for use in a parallelization similar to this one, implemented on a Thinking Machine CM-5. Its workings are very simple, though much thought was spent in its design. At load balance intervals, each node determines the local number of cohorts. Next, a list containing the number of cohorts held by each node needs to be distributed to the other nodes. The list is formed in much the same way as the total biomass is assessed. The left-most node places the local number of cohorts in the list and passes the list to the right. In turn, each node will add its local number of cohorts in the list and pass the results to its right-hand neighbor. The right-most node, after completing the list by adding its local value, will distribute the list to the other nodes. Next, each node sums the elements in the list to ascertain the total number of cohorts currently being processed by the model. The total number of cohorts is divided by the number of nodes being used using integer math, any remainder is distributed evenly over the first $N - 1$ nodes. This will give the number of cohorts each node should have in order for the load to be evenly balanced over the machine.

Each node then determines the surplus number of cohorts on each node, which is the current number of cohorts held by a node less the optimal number. Notice, at this point, all nodes have an identical copy of the surplus list. A positive surplus indicates too many cohorts, zero indicates the correct number, and a negative surplus indicates a deficit of cohorts. Every node must fall into one of these categories. If a node has a positive surplus, it will become a sender, and dispatch the excess cohort

to a node with too few cohorts. Nodes with a deficit of cohorts will become receivers and listen for messages containing cohorts from other nodes. If a node has a surplus of zero it will have very little to do for the remainder of the load balancing algorithm. As a check on the surplus levels, the sum of the levels should yield zero. If not, then some error has occurred, and the load balancing algorithm returns, informing the user of the problem, without balancing the load. All of the nodes start on the left-most element in the surplus list and proceed to the right, stopping at the left-most sender; *i.e.* the left-most node with a positive surplus. After the left-most sender is found, each node starts again at the left end of the list, this time stopping at the left-most receiver. When the left-most receiver is found, each node determines if it is the sender or receiver. If a node is the sender it will send a number of cohorts which will either bring itself or the receiver to the optimal number of cohorts. If a node is the receiver, the node will listen for messages containing cohorts from the sender. After the cohorts have been passed, all nodes update their local surplus lists to reflect the movement of cohorts. Finally, the next left most sender and receiver are located, starting from the current positions in the surplus list and the process starts over.

For PVM, this algorithm needs to be augmented. The change is at the beginning, where the surplus is found. Since each node in the PVM machine may be of different capabilities, it would be advantageous to give the strong machines more work than the weak machines. In fact, the cohorts should be distributed in such a way that the total time spent advancing a group of cohorts for a given time step is exactly

the same for all nodes. Again, this is done to prevent some node finishing before others, resulting in imbalance. The answer to this question is quite elementary. The condition to be satisfied is:

$$t_i N_i = t_{i+1} N_{i+1}, \text{ for } 1 \leq i \leq M - 1 \quad (1.15)$$

where M is the number of nodes in the virtual machine, t_i is the time needed by node i to advance one cohort, one time step, and N_i is the number of cohorts owned by node i . This is the condition which must be satisfied if all the nodes are to spend the same length of time processing their respective portions of the population. The problem is also constrained by:

$$\sum_{i=1}^M N_i = N_T \quad (1.16)$$

where N_T is the total number of cohorts currently in the population. Eqs. (1.15) and (1.16) together are nothing more than an N equation, N unknown problem. Solving the system for N_i we write:

$$\begin{aligned} t_{M-1} N_{M-1} &= t_M N_M \Rightarrow \\ N_{M-1} &= \frac{t_M}{t_{M-1}} N_M \\ t_{M-2} N_{M-2} &= t_{M-1} N_{M-1} \Rightarrow \end{aligned}$$

In general,

$$\begin{aligned} t_i N_i &= t_M N_M \\ N_i &= \frac{t_M}{t_i} N_M \end{aligned} \quad (1.17)$$

Using (1.16) gives:

$$\begin{aligned}
 \sum_{i=1}^M M \frac{t_M}{t_i} N_M &= N_T \Rightarrow \\
 t_M N_M \sum_{i=1}^M M \frac{1}{t_i} &= N_T \Rightarrow \\
 N_M &= \frac{N_T}{t_M \sum_{i=1}^M \frac{1}{t_i}} \quad (1.18)
 \end{aligned}$$

Substitution back into Eq. (1.17) gives:

$$N_i = \frac{N_T}{t_i \sum_{i=1}^M \frac{1}{t_i}}. \quad (1.19)$$

So the optimal load can be quickly computed if the t_i and N_T are known. Other than this addition, the load balancing algorithm functions as described above. The optimal load is given by the last equation, the surplus levels are found by taking the difference between the current level and the optimal level. From here, the algorithm proceeds as before.

Notice the solution to finding optimal loads for the PVM network involves knowledge of t_i , the time required for the i^{th} machine to advance one characteristic one time step. The question is, "where does this number come from?" Between load balance times, each node records how many cohorts it advanced, and how long it took to do this. By dividing the time spent advancing cohorts by the number of cohorts advanced, a node can estimate the time spent per cohort. The time spent per cohort is reevaluated each time in an attempt to compensate for the changing uses of each node. If for instance, another user starts working on a machine in the PVM network, that machine will effectively take more time to process it's characteristics, leading to

load imbalance. By finding the t_i between each load balance interval this imbalance can be offset by giving the node fewer cohorts to process. The t_i is then passed into the load balancing algorithm, which includes this number in its calculation for optimal load balance. The system of linear equations does not depend on having a fixed set of t_i , but each node will need the t_i from its neighbors. This will entail communicating the information from one node to the next, which is accomplished in much the same way as the current local loads are managed. An extra level of communication is justified, because it incurs a very small time penalty and ensures the load will always be balanced as well as can be managed, with little effort on the part of the user.

As mentioned in the description of the serial model, after a cohort has been advanced by the black box, the model checks for births, and, if there have been births, the model attempts to combine births to minimize the computational load. However, combination of births is not a feature of the model which is easily parallelizable. In order to function correctly, the combined births subroutine requires global information about all the births in a population. Recall from the description of the serial model, when two parents of the same ecotype give birth, the offspring are combined into a single cohort. In the parallel model, the parents may be on different nodes. In order for the offspring to be combined, they must be sent to a common node. One approach for global combination is to send one of the offspring to the node where the other is located. However, this requires tracking where matched parents are located within the parallel machine. This is complex, because there are possibly large numbers

of characteristics distributed over many nodes. Furthermore, this information will change due to births and deaths, or as cohorts travel between nodes due to load balancing. The large amount of information needed and the difficulty in keeping this information current rules out using such an approach. Other methods exist, but each significantly increases the complexity of the model, and will assuredly require more communication between nodes, and possibly another point of synchronization. The overall effect is to degrade the performance of the model. For this study, the decision was made not to perform global combined births. Instead, each node was allowed to combine births locally. Biologically speaking, this is not a problem because the combined births algorithm is a computational tool and not a feature inherent to the population. Omitting a global combined births guarantees the results of the parallel model will be different from those of the serial model. However, as the model analysis in Chapter 2 shows, this omission does not seriously affect the population behavior generated by the parallel model.

File input and output is typically a simple subject, but one which presents problems in a parallel environment. There are many options for handling this situation, but as with the other aspects of the parallel model, the method minimizing complexity is the best approach. When the serial model starts up, it reads the parameters it will need from data files. For the parallel model, each node could read the data from a single file. This would be easy to implement, however it would also generate a great deal of NFS (network file system) traffic, and the file server would be placed

under considerable load. Experience has shown that the level of NFS traffic generated by the nodes of a virtual machine can crash an NFS server. A simple fix for this problem is to have a copy of the data file located on each machine's hard drive, having the machines read from these files. This also has problems, as the user must be careful to keep all the copies identical, a tedious problem which becomes worse as the number of nodes is increased or different groups of machines are used. Any error in the data files results in either failure to execute or worse, erroneous results. To cure both these problems, a choice was made to let one machine read in all the data, and distribute the information to each node. Now only one copy of the data file is needed, making file management easy, and the process guarantees each machine in the network receives the same data. Division of the cohorts amongst the nodes also gives this approach other advantages. If each node read the file independently, they would then have to confer to determine which nodes should work on which subset of the data. A great deal of effort will have to be spent on the part of the programmer and the program to assure no cohorts are forgotten or included more than once. This complexity is a potential source of error. On the other hand, a single stack under the control of a single node program makes correct functionality much easier to assure.

From the flowchart for the serial model, Fig. 1.1, we see that we are going to have file output. It is desirable to keep the amount of time spent in supporting processes to a minimum, because it is performed while the model is running. Also, since file output takes place while the program is running, file output may affect node synchronization.

If nodes spend various amounts of time outputting data, some nodes will be idle while others finish. To simplify matters, this varying output time is not compensated for directly in the load balancing algorithm, but the fastest method for output is used to minimize any effects. With PVM, there are again two basic options. Send the information over the network to a single machine, or send the data to the local hard drive. First, it is clear (hopefully) that each node independently writing to a single file would be disastrous. Even if the nodes wrote to the file in turn, a good deal of care would have to be taken. For the machines to write in serial would also take a good deal of time, to assure file integrity and a great deal of information will have to be passed over Ethernet, which can result in unpredictable delays.

The second option is to have each node write to its local hard drive. In this situation, the output is done in parallel and the code is simple to write. Fig. 1.4 shows a comparison of PVM communication time needed to send various amounts of data to a single node and the time taken to write the same amount of data to the local hard drive. As this graph shows, when the number of nodes is increased, the time to output data to a single node over the network increases, while writing to the local hard drives takes a consistent amount of time, independent of the number of machines in the network. Also, by writing to the local hard drives output is performed in parallel, and the amount of time required for all the nodes to output all the information is the maximum of the write times, and not, as is the case with writing to a single node or file, the sum of the write times. This implies outputting to the local hard drives is

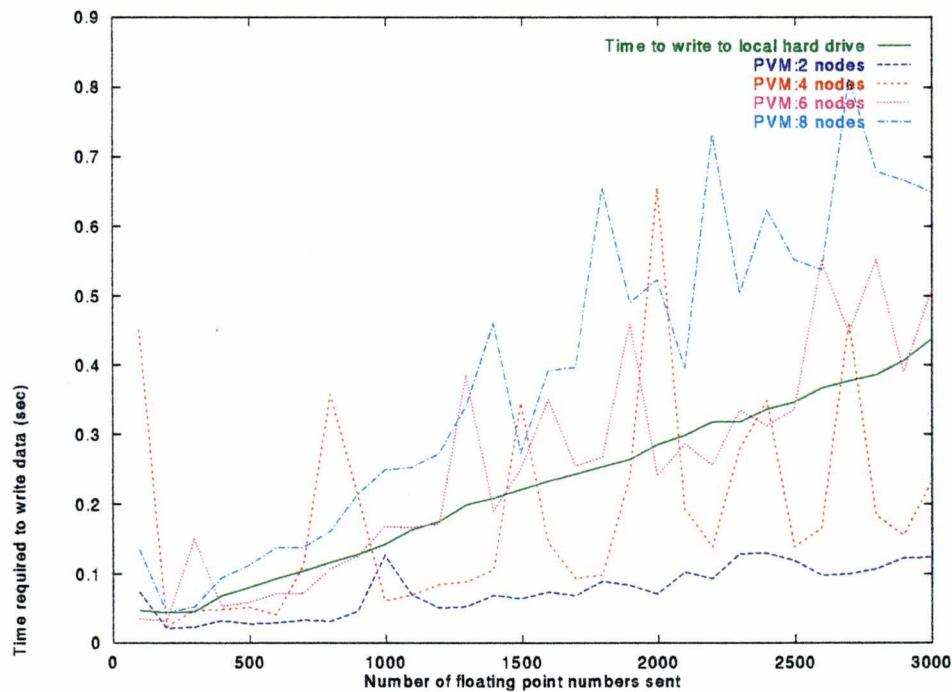


Figure 1.4: Time required to send various amounts of data to a single node, and to a local hard drive.

more scalable and hence more desirable.

The last important feature of the model which may be affected by parallelization is the flow of time within the simulation. However, this does not present very many problems. The method of file input guarantees every node receives the same set of parameters, including the start and stop times for the simulation, as well as the step size by which the model advances through time. Since each node runs the same program, they will always be at the same point in time throughout the course of program execution.

With the exception of combined births, parallelization by distributing cohorts

and processing them in parallel does not seriously affect the model in terms of the other processes in the algorithm. Features such as total biomass assessment and file input/output are all easily translated into the parallel environment. A non-global combined births is not sufficient reason to abandon the overall approach, because combined births is a computational short-cut and not a biological feature of the model.

Chapter 2

Model analysis

In this chapter the quality of the parallel model is gauged. This is measured by how well the parallel model fulfills the established goals of speed improvement over the serial model, scalability, the ability to run on many different platforms, and effective use of machines of different capabilities in conjunction. Pivotal to these goals is the ability of the model to return biologically reasonable results. By comparing the behavior of a population generated by the parallel model to the behavior generated by the serial model, the accuracy of the parallel model can be ascertained. The parallel code is measured against the serial code, because the basic algorithm for the parallel model was derived from the serial model and the behavior of the serial model is largely understood and accepted. The results of this comparison indicate the parallel model is able to return reasonable results. In addition, the parallel model was found to be scalable and offers a definite speed improvement over the serial model.

To gauge each of the above criteria fairly, the parallel model was compared to the serial model using a diverse set of data. It is impractical to check all possible configurations, but data sets were chosen in such a way as to test the important features of the model. One important feature is population structure, which is both a significant factor in population behavior and a special feature of this model. The amount of structure also affects the behavior of combined births, which has already been identified as a possible source of error in the parallel model. For these reasons, the parallel model was tested with populations consisting of one, three, nine and twenty seven ecotypes. The environmental parameters imposed on a population are also important and may affect population diversity by causing one or more ecotypes to go to extinction. To determine the impact parallelization has on the interaction between a population and its environment, three combinations of effects were used. In the first combination, the target population is exposed to a constant level of toxicant, and the effects of dissolved oxygen and temperature are included. The second combination imposes no environmental stresses on the population. The final combination places the population under a toxicant stress, but excludes the impact of dissolved oxygen and temperature. It is also important to understand how the level of parallelization affects the behavior of a population, hence runs were performed with two, four, eight, and sixteen nodes. These numbers were chosen to correspond to the number of nodes used in scaling analysis, in the hopes that the results would shed some additional light on the scaling results. Even with these restrictions, only a subset of combi-

Table 2.1: Combination of populations, environmental effects, and nodes used.

Number of Nodes	Number of ecotype in population	Effects		
		All	None	Toxicant Only
2	1			•
	3	•		•
	9			•
	27	•	•	
4	1	•		
	3			•
	9	•		
	27	•		•
8	1		•	•
	3		•	
	9		•	•
	27	•		•
16	1	•	•	
	3	•	•	
	9	•		
	27	•	•	

nations were used, to keep the amount of time needed to run the models within a reasonable limit, and to keep the amount of data which needed to be stored to a manageable level. A subset of combinations were employed to form a credible picture of the parallel model. Table 2.1 gives an overview of the combinations of population, environmental effects and nodes which were used. Many parameters, such as length of run and balance time, were fixed to keep the amount of data at a manageable level. The base values and percentages for the structuring variables, resource x , proportion of resource which is lipid plx , and the filtration coefficient A_1 are given in Table 2.2 for each population used.

The first issue addressed in this chapter is the comparison of the parallel model to the serial model. Does the parallel model generate population behavior equivalent

Table 2.2: Values for structuring variables used for model analysis

Number of Ecotypes in the Population	x levels (mg)	% for x	plx levels	% for plx	A_1 levels	% for A_1
1	0.28763e-02	0%	2.50000e-01	0%	1.00000e-06	0%
3	0.28763e-02	75%	2.50000e-01	0%	1.00000e-06	0%
9	0.28763e-02	0%	2.50000e-01	33%	1.00000e-06	25%
27	0.28763e-02	99.9%	2.50000e-01	15%	1.00000e-06	25%

to the behavior seen in the serial model? The answer to this question is crucial because it either validates other results and conclusions regarding timing and scaling, or indicates a careful scrutiny, of the approach taken in parallelization or the source code for the parallel model, is in order. Comparing the serial and parallel models raises two basic questions: What features of a population are important with respect to this modeling process, and how should these features be compared to obtain an accurate measure of correspondence between the two models? The method of comparison is largely determined by what is being compared, thus the relevant features of the model are identified first.

When examining a population as simulated by the *Daphnia* model, there are several quantities which reflect the behavior of the population through time. The most elementary of these are the total biomass of a population and the density of individuals in a population. These are typical population variables to examine and provide an overview of population dynamics. Additional parameters which reflect the behavior of the population can be identified by examining the system of p.d.e.s

upon which the model is based. Eq. (1.7) structures the population by age, the mass of lipid, and the mass of structure material. In addition, the system of equations is designed to allow for population structure based on the level of resource seen by the individual (x), proportion of lipid in the resource (plx), and the filtration coefficient (A_1). The time-dependent dynamics of these variables directly reflect the behavior of the population and provide a detailed picture of this behavior.

Several different methods exist for comparing the results of the parallel and serial models, with respect to the aforementioned variables. The large number of data sets used to test the results indicates a simple and concise method should be applied. What is needed is an overall measure of how the behavior generated by the parallel model differs from the behavior generated by the serial model. The approach taken here is to first reduce the population dynamics to a system of functions from S into $\mathcal{R}$ where S is the set:

$$S = \{100 + 2t(days) : 0 \leq t \leq 50\}. \quad (2.1)$$

This simply means considering the time series data for each variable to be a member of $\ell^1(S)$. Each combination of population, effects, and nodes were run for 150 days, so S is the last 50 days of each run. Using 50 days of output became a necessity, because data files generated for longer periods could become as large as 30 megabytes, which, combined with the number of runs used for this project, makes large file size a consideration. Using the last 50 days of a run as the basis for comparison does

not seriously impugn the results, for if the population dynamics generated by the parallel and serial models match well for the last 50 days, then it is reasonable to expect the first hundred days to display the same correspondence. With the data reduced to a system of $\ell^1(S)$ functions, the correspondence between the parallel and serial models is measured by the average distance between corresponding pairs of time series functions; *e.g.* to find the correspondence with respect to total biomass, the average distance between the total biomass functions for a population generated by the two models is determined. For the total biomass and density of individuals, this is straightforward, since each data set already maps from S into $\mathbb{R}$. In Figs. 2.1 and 2.2 are graphs of total biomass and density of individuals, as generated by the parallel and serial model for the same initial population. The population used, in this case, contained twenty seven ecotypes and is subject to all effects. This population will be used repeatedly to illustrate the application of the comparison technique used in this paper. Data for this population was collected from a serial machine and from a parallel machine consisting of eight nodes. As these graphs show, the results generated by the parallel model are very close to the serial model with respect to these variables. Table 2.3 gives the average distance between these data sets.

Table 2.3: Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 8.

Variable	Average Distance
Density of Individuals	3.714889e+01
Total Biomass	6.930777e-01

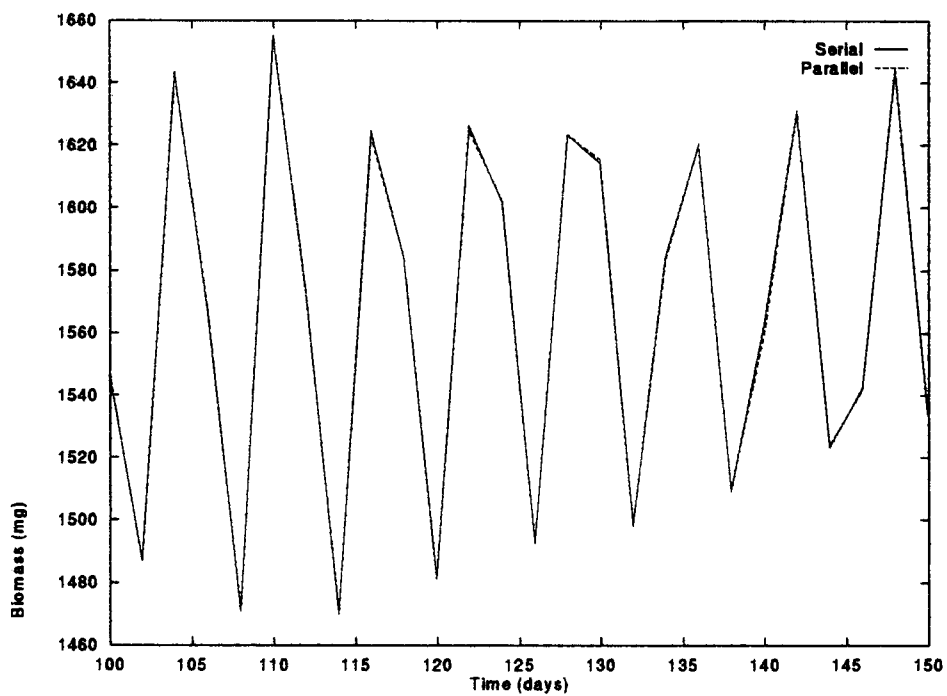


Figure 2.1: Graph of the total biomass as generated by the parallel and serial models for a population containing 27 ecotypes, under the effects of toxicant, oxygen and temperature. The parallel data was generated on 8 nodes.

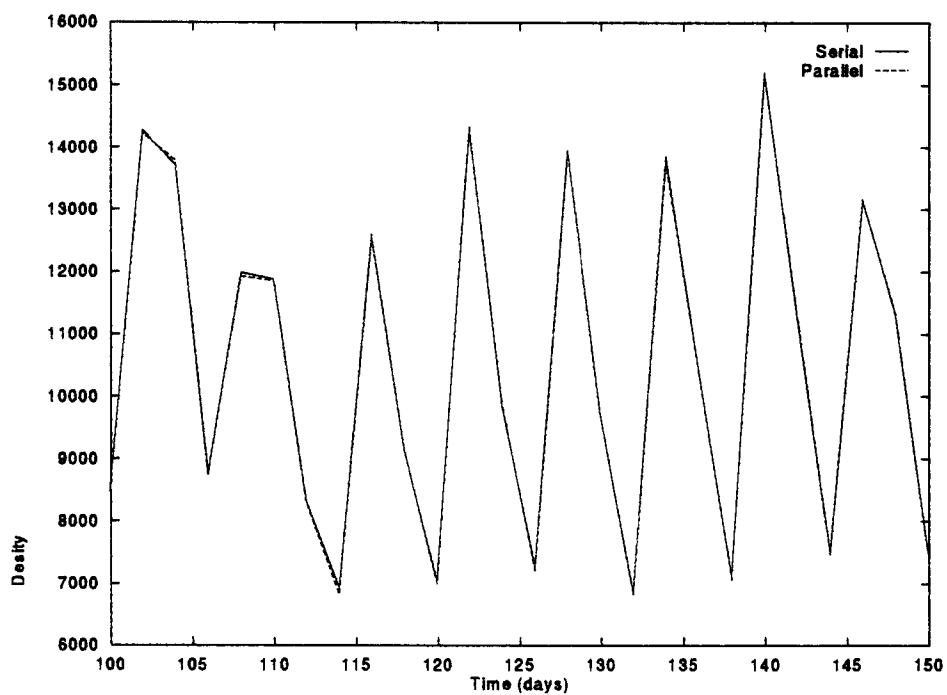


Figure 2.2: Graph of the density of individuals as generated by the parallel and serial models. There are 27 ecotypes in the population, under toxicant, oxygen and temperature effects. The parallel data was generated on an 8 node machine.

As can be seen from this table, the average distance between density and biomass are small in comparison to the scale of the data. The average distances are not zero, which means the parallel model does not generate results identical to those generated by the serial model. This was anticipated, since certain features of the serial model did not translate well to the parallel environment, most notably combined births. Thus, it is necessary to obtain an estimate for a reasonable level of error. Integral to this consideration is knowing whether or not the mode of comparison distinguishes between a pair of time series data displaying similar characteristics and a pair of time series data which display different behaviors. If the average distance are essentially the same for all pairs of time series data, then average distances will not be a very useful tool for comparison. To determine acceptable error and the validity of this method of comparison, a second population is compared to the reference population. The reason for comparing the reference population to a different population is to understand how to interpret the average distances. The second population contains a single ecotype, which is exposed to no environmental effects and was run in parallel on 16 nodes. The choice of this population for comparison to the reference population is largely arbitrary. The essential point is this population has different initial conditions, and hence displays different behavior. Figs. 2.3 and 2.4 show the graphs of total biomass and density of individuals respectively, for the reference population and the one ecotype population under no effects. Table 2.4 gives the average distances between the respective time series functions. As can be seen from the graphs, the

Table 2.4: The average distance between the time series data for a population with 1 ecotype, under no effects, run on 16 nodes, and a population with 27 ecotypes, under all effects, run in serial.

Variable	Average Distance
Density of Individuals	2.700868e+04
Total Biomass	2.344231e+02

dynamics of populations with twenty seven ecotypes under all effects and one ecotype under no effects are very different with respect to biomass and density, which is reflected in average distances on the order of 10% -25% of the scale of the data being compared and three orders of magnitude larger than the average distances in Table 2.3. On their own, the averages shown in Tables 2.3 and 2.4 can be very deceptive, since concepts of large and small average distance depends of the scale of the data being compared. To cure this problem, each time series function is scaled by the maximum value it obtains over S . The result is a new function which maps S into the unit interval, thus fixing the scale, and making the concept of large and small more objective. In addition, two time series functions may have the same qualitative behavior, but have a significant average distance if there is some constant separation. Thus, finding the average distance for scaled functions provides a qualitative measure, while the average distance of the unscaled data gives a quantitative measure, of the correspondence between the results generated by the two models.

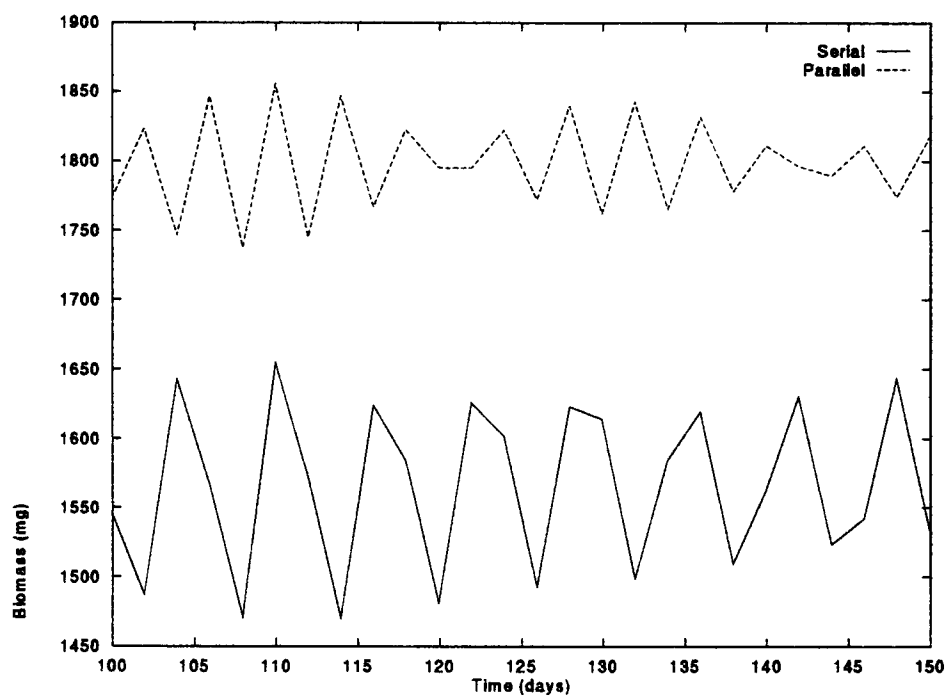


Figure 2.3: Graphs of the total biomass for the reference population run in serial and a population containing one ecotype under no environmental stress run in parallel on 16 nodes.

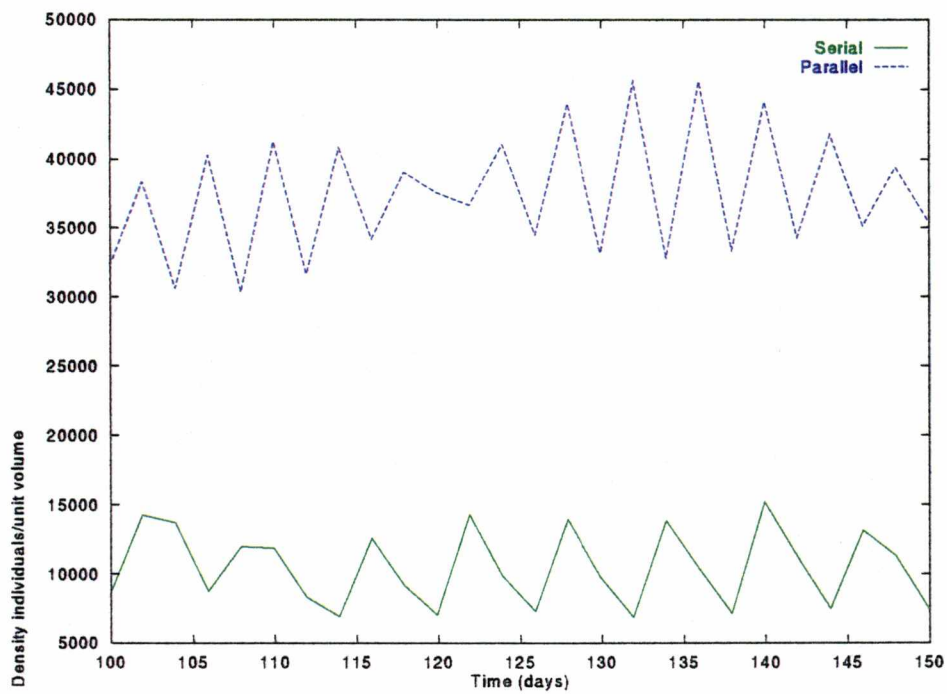


Figure 2.4: Graph of the density of individuals for the reference population run in serial and a population consisting of one ecotype under no environmental stress run in parallel on 16 nodes.

Table 2.5: Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 8. Scaled.

Variable	Average Distance
Density of Individuals	2.343043e-03
Total Biomass	4.543777e-04

Table 2.6: The average distance between the scaled time series data for a population with 1 ecotype, under no effects, run on 16 nodes, and a population with 27 ecotypes, under all effects, run in serial.

Variable	Average Distance
Density of Individuals	2.090034e-01
Total Biomass	3.801442e-02

The average distances for the comparison using scaled functions are shown in Tables 2.5 and 2.6.

Taken together, the scaled and unscaled differences give an indication of how well the results of the two runs compare. As the tables show, when the results of the parallel and serial codes match well, the average distances are small, on the order of 0.01% or less of the scale of the data. For dynamics which are very different the average distances are quite large compared to the scale of the data. In addition, the method of comparison distinguishes between model results which match well and those which display very different dynamics.

To apply this method to the dynamics of the structuring variables is only slightly more complex. Typically, data representing population structure have been represented in a three dimensional plot, with time along one axis, age or size class along a second, and density of individuals in a class along the third axis. Figs. 2.5 and 2.6 give examples of this type of graph for the reference population with respect to age and mass of lipid. This information can be reduced to $\ell^1(S)$ functions by finding the average and standard deviation of the population structure with respect to each variable at each time step. Figs. 2.7 through 2.12 show the averages and standard deviations of the structuring variables for the reference population as generated by the serial model and the parallel model run on eight nodes. The time series averages and variances can now be treated in the same fashion as the data for total biomass and density. A complete set of comparisons are given in Tables 2.7 through 2.28. As before, these numbers are subjective without knowing what level of error is acceptable, so the analysis performed for the total biomass and density is extended to the structuring variables. As the graphs in Figs. 2.7 through 2.12 show, there is a strong correlation between results generated by the two models for the reference population with respect to these variables. The scale of the graphs can conceal small, possibly important differences between the averages and standard deviation which can be revealed by plotting the ratios and differences between respective functions.

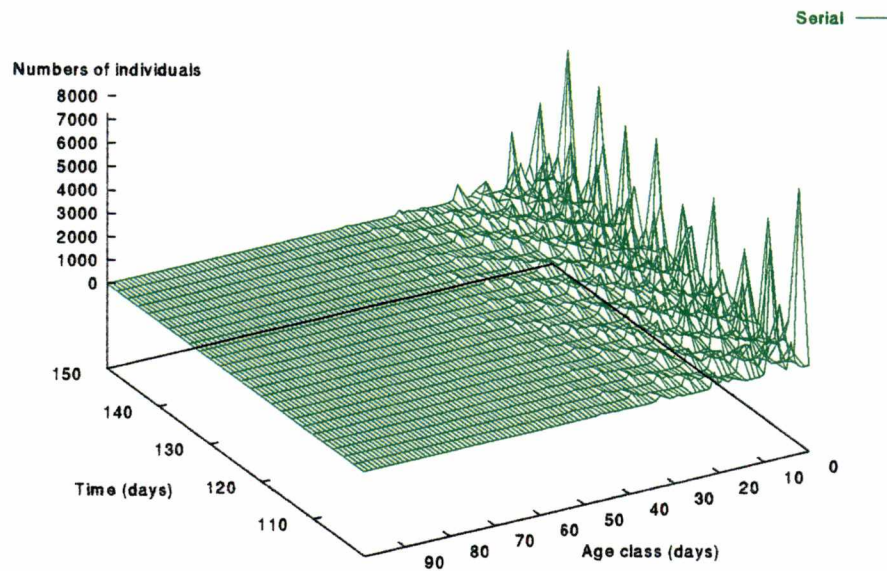


Figure 2.5: Graph of population structure with respect to age for the population containing 27 ecotypes and exposed to all environmental effects.

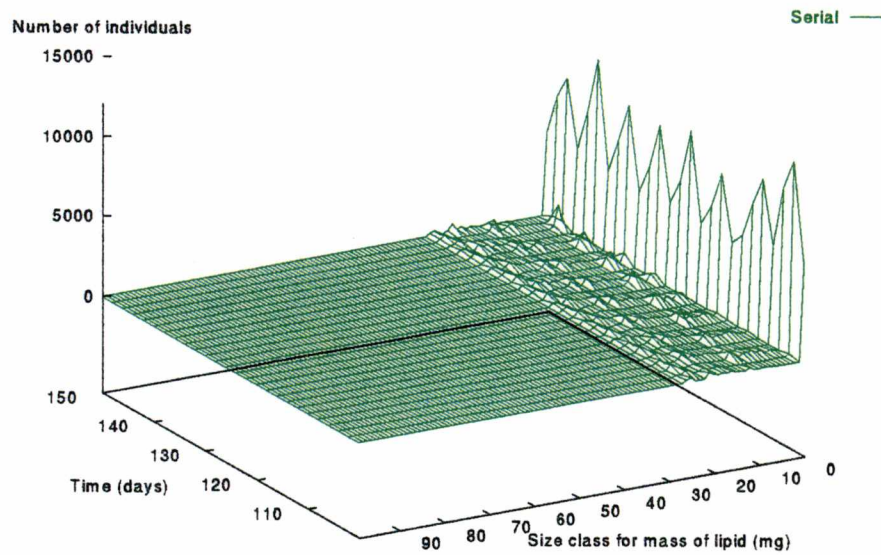


Figure 2.6: Plot showing the structure of the population with respect to the lipid content of individuals. The population for this plot contains 27 ecotypes, is exposed to toxicant, oxygen and temperature effects, and was run in serial.

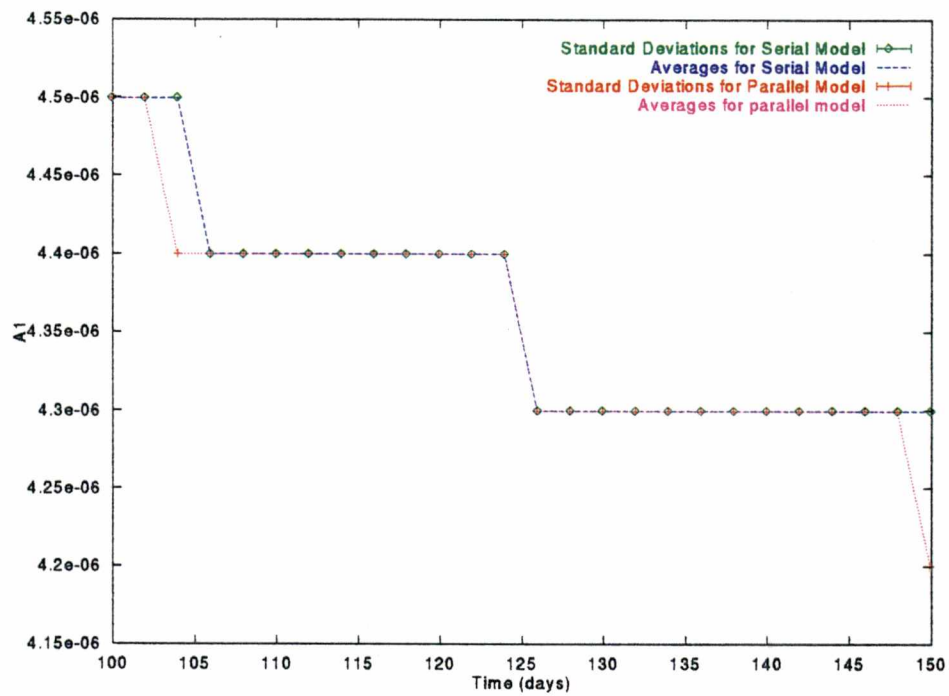


Figure 2.7: Graphs of the average A_1 value for serial and parallel models.

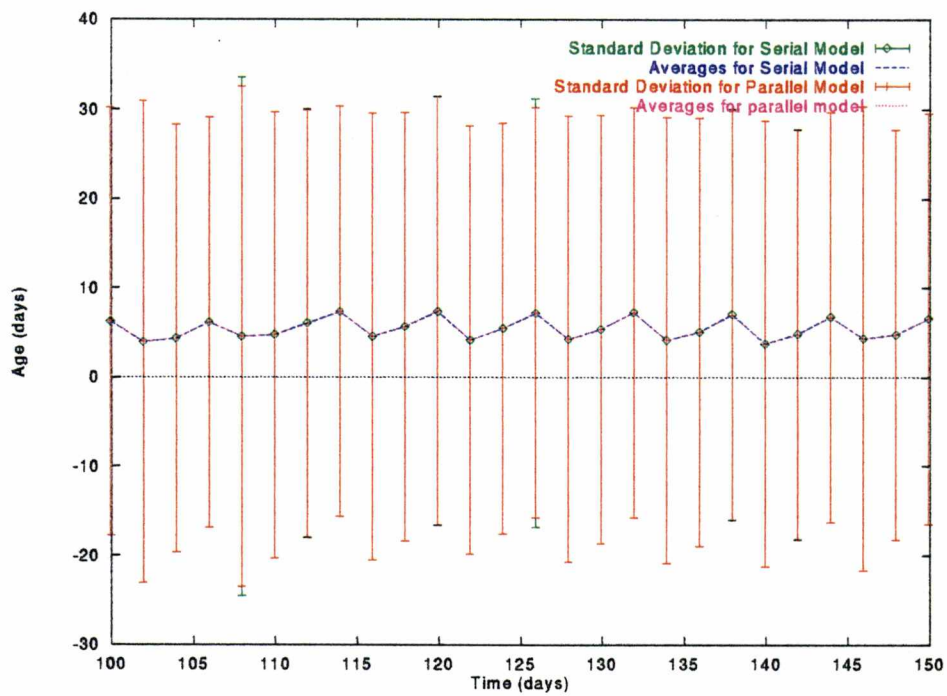


Figure 2.8: Graphs of the average age for the serial and parallel models.

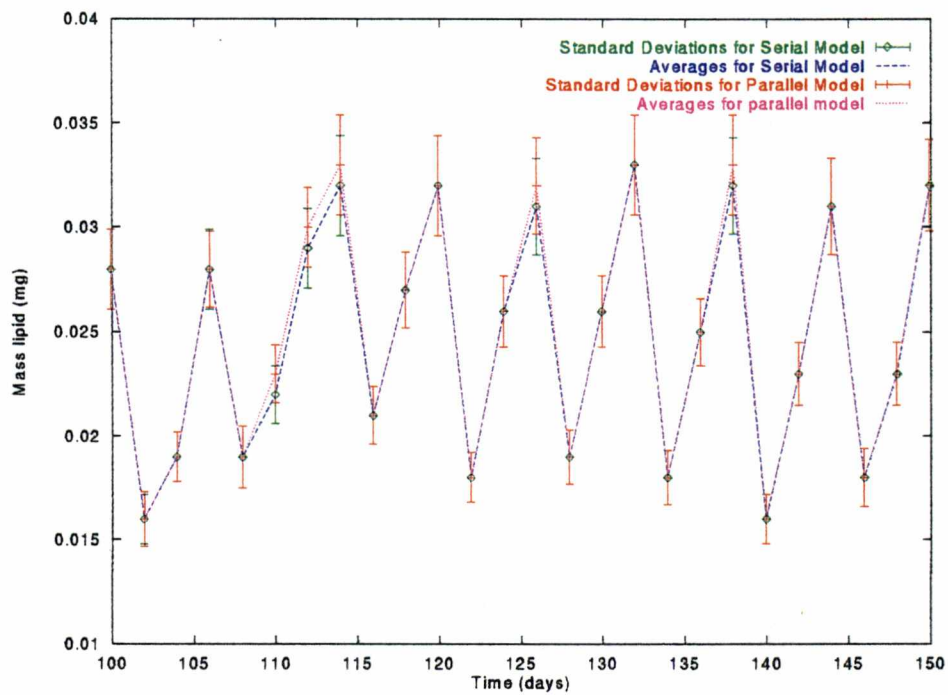


Figure 2.9: Graphs of the average mass of lipid for the serial and parallel models.

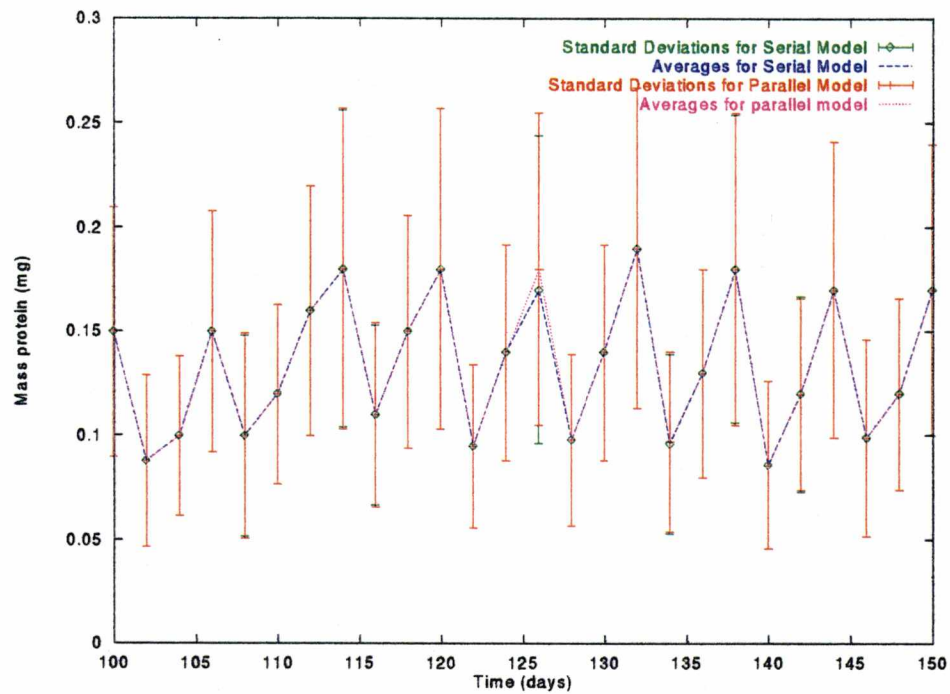


Figure 2.10: Graphs of the average mass of structure material for the serial and parallel models.

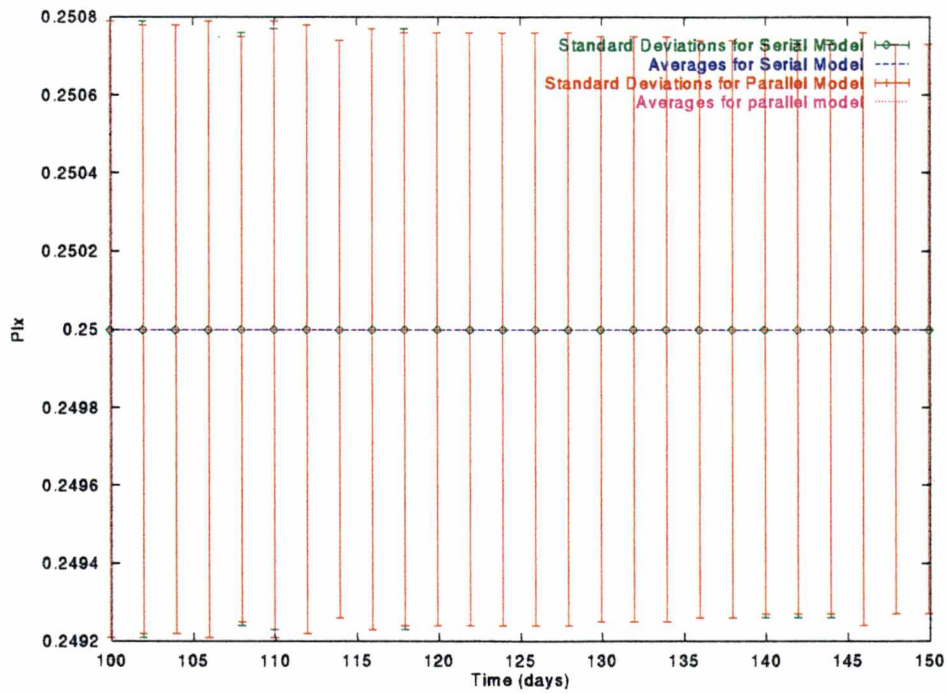


Figure 2.11: Graphs of the average plx value for the serial and parallel models.

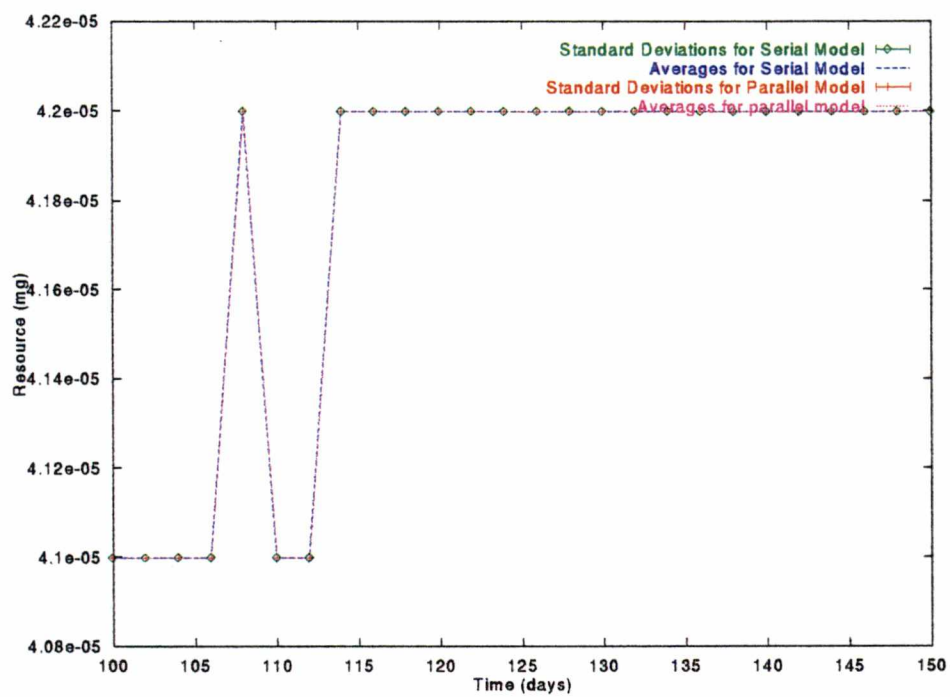


Figure 2.12: Graphs for the average resource for the serial and parallel models.

Table 2.7: Average Distances. Number of Ecotypes: 3. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	7.773586e+01	6.999228e-03
zmass	1.666171e+00	9.171940e-04
avg. age	5.384614e-02	1.344681e-02
var. age	3.076923e-01	2.827604e-02
avg. mass lipid	3.846152e-04	2.295850e-02
var. mass lipid	2.692308e-05	2.038016e-02
avg. mass structure	1.923077e-03	2.816573e-02
var. mass structure	3.076921e-04	1.709458e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	4.999998e-17	2.073904e-02
avg. resource	2.692309e-05	7.918610e-03
var. resource	2.692310e-08	9.615589e-03
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.8: Average Distances. Number of Ecotypes: 3. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 2.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	3.675503e+01	3.087384e-03
zmass	9.053861e-01	7.427725e-04
avg. age	1.153845e-02	1.748268e-03
var. age	2.307692e-01	1.003339e-02
avg. mass lipid	7.692302e-05	1.672231e-03
var. mass lipid	7.692313e-06	1.134069e-02
avg. mass structure	1.538461e-03	2.668646e-02
var. mass structure	4.615385e-04	2.564114e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	1.153846e-17	1.748268e-03
avg. resource	1.538463e-05	4.524921e-03
var. resource	1.153847e-08	4.120967e-03
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.9: Average Distances. Number of Ecotypes: 1. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	: 9.277629e+01	8.816044e-03
zmass	: 2.380000e+00	1.425493e-03
avg. age	: 3.076924e-02	1.425493e-03
var. age	: 1.923077e-01	1.068373e-02
avg. mass lipid	: 6.923078e-04	2.176449e-02
var. mass lipid	: 9.615386e-05	6.009614e-03
avg. mass structure	: 3.076922e-03	1.556358e-02
var. mass structure	: 3.076923e-03	2.304681e-02
avg. A1	: 0.000000e+00	0.000000e+00
var. A1	: 1.576924e-16	4.908773e-02
avg. resource	: 0.000000e+00	0.000000e+00
var. resource	: 2.695115e-18	2.118079e-01
avg. plx	: 0.000000e+00	0.000000e+00
var. plx	: 0.000000e+00	NaN

Table 2.10: Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 8.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	3.714889e+01	2.343043e-03
zmass	6.930777e-01	4.543777e-04
avg. age	1.923077e-02	1.045747e-02
var. age	7.692308e-02	2.756759e-02
avg. mass lipid	1.923077e-04	5.827500e-03
var. mass lipid	1.153846e-05	4.807729e-03
avg. mass structure	4.230772e-04	2.226694e-03
var. mass structure	2.307693e-04	2.997002e-03
avg. A1	7.692314e-09	1.709386e-03
var. A1	7.307696e-15	1.074658e-02
avg. resource	0.000000e+00	0.000000e+00
var. resource	4.230770e-13	2.082838e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	3.076920e-06	3.894808e-03

Table 2.11: Average Distances. Number of Ecotypes: 27. Effects: Toxicant, Oxygen,
Temperature. Number of Nodes: 2.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	1.750043e+01	1.175496e-03
zmass	3.180683e-01	2.044669e-04
avg. age	0.000000e+00	0.000000e+00
var. age	7.692308e-02	2.756759e-02
avg. mass lipid	1.153846e-04	3.496500e-03
var. mass lipid	7.692313e-06	3.205153e-03
avg. mass structure	7.692295e-05	4.048852e-04
var. mass structure	7.692309e-05	9.990014e-04
avg. A1	3.846148e-09	8.546939e-04
var. A1	1.538460e-15	1.030754e-02
avg. resource	0.000000e+00	0.000000e+00
var. resource	3.076923e-13	1.183434e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	1.538460e-06	1.947424e-03

Table 2.12: Average Distances. Number of Ecotypes: 9. Effects: Toxicant, Oxygen,
Temperature. Number of Nodes: 4.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	5.226501e+01	4.642037e-03
zmass	1.619230e+00	9.518082e-04
avg. age	3.461539e-02	5.244622e-03
var. age	1.923077e-01	8.361193e-03
avg. mass lipid	3.076924e-04	6.546731e-03
var. mass lipid	2.307690e-05	1.316819e-02
avg. mass structure	1.538464e-03	5.698002e-03
var. mass structure	9.615387e-04	4.807692e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	3.846153e-15	3.205116e-03
avg. resource	0.000000e+00	0.000000e+00
var. resource	5.710962e-18	1.795830e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	7.538460e-05	4.711538e-02

Table 2.13: Average Distances. Number of Ecotypes: 9. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	8.383428e+01	1.028292e-02
zmass	2.013475e+00	1.164656e-03
avg. age	2.307695e-02	3.496427e-03
var. age	2.307692e-01	1.003342e-02
avg. mass lipid	4.230771e-04	1.331507e-02
var. mass lipid	3.461539e-05	1.250550e-02
avg. mass structure	1.923078e-03	7.122502e-03
var. mass structure	1.076923e-03	3.250888e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	0.000000e+00	0.000000e+00
avg. resource	0.000000e+00	0.000000e+00
var. resource	8.586846e-18	1.936197e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	8.153845e-05	5.096154e-02

Table 2.14: Average Distances. Number of Ecotypes: 1. Effects: None Number of Nodes: 16.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	1.349014e+02	6.505808e-03
zmass	1.370005e+00	1.067535e-03
avg. age	0.000000e+00	0.000000e+00
var. age	4.153846e-01	7.837426e-02
avg. mass lipid	1.153850e-05	1.240730e-03
var. mass lipid	3.461537e-06	1.296962e-02
avg. mass structure	1.153844e-04	2.455001e-03
var. mass structure	1.192307e-04	9.171578e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	1.346308e-24	1.590055e-01
avg. resource	0.000000e+00	0.000000e+00
var. resource	2.588077e-18	2.943445e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.15: Average Distances. Number of Ecotypes: 1. Effects: None Number of

Nodes: 8.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	6.710690e+01	3.440926e-03
zmass	6.307702e-01	5.657283e-04
avg. age	0.000000e+00	0.000000e+00
var. age	3.923077e-01	7.402011e-02
avg. mass lipid	7.692348e-06	8.271543e-04
var. mass lipid	3.846153e-06	1.262889e-02
avg. mass structure	7.692295e-05	1.636693e-03
var. mass structure	1.346153e-04	1.035500e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	1.142462e-24	2.260805e-01
avg. resource	0.000000e+00	0.000000e+00
var. resource	1.659154e-18	3.138947e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.16: Average Distances. Number of Ecotypes: 27. Effects: None. Number of

Nodes: 2.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	1.162433e+02	3.990192e-03
zmass	1.392672e+00	1.064933e-03
avg. age	7.692300e-03	2.849001e-03
var. age	3.461542e-02	6.072890e-03
avg. mass lipid	1.153844e-05	1.341691e-03
var. mass lipid	7.692305e-07	2.403846e-03
avg. mass structure	7.692323e-05	1.709383e-03
var. mass structure	3.461536e-05	3.461537e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	1.538462e-15	9.049847e-03
avg. resource	1.538462e-07	2.054410e-02
var. resource	1.923077e-12	1.312604e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	1.192307e-05	1.954596e-02

Table 2.17: Average Distances. Number of Ecotypes: 3. Effects: Toxicant, Oxygen, Temperature. Number of Nodes: 16.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	3.116602e+02	1.088554e-02
zmass	3.581510e+00	3.112385e-03
avg. age	1.923077e-02	7.122503e-03
var. age	1.115385e-01	4.337815e-03
avg. mass lipid	4.615382e-05	1.067178e-02
var. mass lipid	4.615384e-06	1.442308e-02
avg. mass structure	3.846156e-04	8.546962e-03
var. mass structure	1.307692e-04	7.832166e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	5.000000e-15	3.040165e-02
avg. resource	2.692309e-07	1.772988e-02
var. resource	4.038461e-12	1.316230e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	2.500000e-05	2.655523e-02

Table 2.18: Average Distances. Number of Ecotypes: 3. Effects: None. Number of Nodes: 16.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	3.476321e+02	1.382408e-02
zmass	3.580773e+00	2.897423e-03
avg. age	1.538462e-02	5.698002e-03
var. age	1.230770e-01	3.166531e-02
avg. mass lipid	3.846154e-05	9.428494e-03
var. mass lipid	2.692309e-06	1.822982e-02
avg. mass structure	2.692309e-04	1.756011e-02
var. mass structure	5.769228e-05	4.437848e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	3.093808e-24	3.645248e-01
avg. resource	1.153846e-05	2.684292e-02
var. resource	1.307692e-07	4.670327e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.19: Average Distances. Number of Ecotypes: 3. Effects: None. Number of Nodes: 8.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	1.850778e+02	8.006735e-03
zmass	2.035776e+00	1.765198e-03
avg. age	0.000000e+00	0.000000e+00
var. age	1.192308e-01	2.249646e-02
avg. mass lipid	1.153846e-05	1.240730e-03
var. mass lipid	1.923078e-06	1.644090e-02
avg. mass structure	1.538462e-04	1.808861e-02
var. mass structure	2.307692e-05	1.775194e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	2.326115e-24	2.316846e-01
avg. resource	3.846147e-06	1.201923e-03
var. resource	1.269231e-07	4.532966e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.20: Average Distances. Number of Ecotypes: 9. Effects: None. Number of Nodes: 8.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	3.719095e+02	1.692789e-02
zmass	3.795762e+00	3.372236e-03
avg. age	1.538461e-02	3.733734e-02
var. age	8.846158e-02	1.669092e-02
avg. mass lipid	5.384613e-05	5.917155e-03
var. mass lipid	4.230770e-06	1.007331e-02
avg. mass structure	2.692310e-04	5.728346e-03
var. mass structure	1.615384e-04	1.242600e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	9.230765e-15	4.858289e-02
avg. resource	0.000000e+00	0.000000e+00
var. resource	7.803038e-18	2.914242e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	1.188462e-04	3.154866e-02

Table 2.21: Average Distances. Number of ecotypes: 1 Effects: Toxicant. Number of nodes: 2

Variable	Average Distance	
	Unscaled	Scaled
rho sum	1.018262e+01	4.863028e-04
zmass	1.076895e-01	8.110817e-05
avg. age	0.000000e+00	0.000000e+00
var. age	2.846154e-01	5.370088e-02
avg. mass lipid	0.000000e+00	0.000000e+00
var. mass lipid	2.307691e-06	5.494379e-03
avg. mass structure	3.846147e-05	8.183466e-04
var. mass structure	1.461538e-04	1.124258e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	6.895385e-25	2.859542e-01
avg. resource	0.000000e+00	0.000000e+00
var. resource	1.079885e-18	4.321292e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.22: Average Distances. Number of ecotypes: 1 Effects: Toxicant. Number of nodes: 8

Variable	Average Distance	
	Unscaled	Scaled
rho sum	6.991203e+01	3.446304e-03
zmass	6.434561e-01	4.840379e-04
avg. age	0.000000e+00	0.000000e+00
var. age	4.230769e-01	7.982564e-02
avg. mass lipid	7.692348e-06	8.271543e-04
var. mass lipid	3.846153e-06	1.262889e-02
avg. mass structure	7.692295e-05	1.636693e-03
var. mass structure	1.307692e-04	1.005912e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	8.208846e-25	9.905173e-02
avg. resource	0.000000e+00	0.000000e+00
var. resource	1.950577e-18	2.621242e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.23: Average Distances. Number of Ecotypes: 27. Effects: Toxicant. Number of Nodes: 8.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	5.514378e+01	2.182147e-03
zmass	7.657705e-01	5.700038e-04
avg. age	3.846150e-03	1.424498e-03
var. age	6.538455e-02	5.725420e-03
avg. mass lipid	3.846165e-06	4.995007e-04
var. mass lipid	0.000000e+00	0.000000e+00
avg. mass structure	7.692309e-05	1.788923e-03
var. mass structure	1.923074e-05	2.289426e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	7.692310e-16	4.807692e-03
avg. resource	1.153847e-07	2.814229e-03
var. resource	1.346154e-12	1.489062e-02
avg. plx	0.000000e+00	0.000000e+00
var. plx	5.769230e-06	1.477054e-02

Table 2.24: Average Distances. Number of Ecotypes: 27. Effects: Toxicant. Number of Nodes: 4.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	3.505183e+01	1.350955e-03
zmass	4.711492e-01	4.331584e-04
avg. age	3.846150e-03	1.424498e-03
var. age	3.846150e-02	6.203459e-03
avg. mass lipid	3.846165e-06	4.995007e-04
var. mass lipid	0.000000e+00	0.000000e+00
avg. mass structure	3.846147e-05	8.944617e-04
var. mass structure	7.692295e-06	9.157703e-04
avg. A1	0.000000e+00	0.000000e+00
var. A1	7.692310e-16	4.807692e-03
avg. resource	3.846155e-08	9.380762e-04
var. resource	9.615383e-13	7.494234e-03
avg. plx	0.000000e+00	0.000000e+00
var. plx	4.999999e-06	1.651581e-02

Table 2.25: Average Distances. Number of Ecotypes: 3. Effects: Toxicant. Number of Nodes: 2.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	4.015370e+01	1.703423e-03
zmass	3.038518e-01	1.810376e-04
avg. age	3.846150e-03	1.424498e-03
var. age	4.230767e-02	6.508772e-03
avg. mass lipid	0.000000e+00	0.000000e+00
var. mass lipid	0.000000e+00	0.000000e+00
avg. mass structure	3.846162e-05	1.098922e-03
var. mass structure	7.692295e-06	1.569847e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	5.167731e-24	1.862550e-01
avg. resource	0.000000e+00	0.000000e+00
var. resource	7.692314e-09	2.564077e-03
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.26: Average Distances. Number of Ecotypes: 3. Effects: Toxicant. Number of Nodes: 4.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	7.022581e+01	2.979148e-03
zmass	6.123282e-01	4.146168e-04
avg. age	7.692310e-03	2.848999e-03
var. age	6.538459e-02	1.005897e-02
avg. mass lipid	7.692295e-06	1.098885e-03
var. mass lipid	0.000000e+00	0.000000e+00
avg. mass structure	3.846162e-05	1.098922e-03
var. mass structure	1.153844e-05	2.354771e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	7.100385e-24	3.065932e-01
avg. resource	0.000000e+00	0.000000e+00
var. resource	1.923078e-08	6.410193e-03
avg. plx	0.000000e+00	0.000000e+00
var. plx	0.000000e+00	NaN

Table 2.27: Average Distances. Number of Ecotypes: 9. Effects: Toxicant. Number of Nodes: 8.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	2.961415e+02	1.107135e-02
zmass	3.208069e+00	2.225571e-03
avg. age	2.307693e-02	8.546997e-03
var. age	1.153846e-01	8.764852e-03
avg. mass lipid	4.615386e-05	1.149696e-02
var. mass lipid	2.692308e-06	1.076923e-02
avg. mass structure	4.230772e-04	8.461540e-03
var. mass structure	1.653846e-04	1.272192e-02
avg. A1	0.000000e+00	0.000000e+00
var. A1	7.692300e-16	4.048540e-03
avg. body burden	1.153845e-06	6.410308e-03
var. body burden	3.076926e-10	2.564041e-03
avg. resource	0.000000e+00	0.000000e+00
var. resource	1.173515e-17	2.099943e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	1.153847e-05	6.787385e-03

Table 2.28: Average Distances. Number of Ecotypes: 9. Effects: Toxicant Number of Nodes: 2.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	8.413717e+01	3.388655e-03
zmass	7.500047e-01	7.104530e-04
avg. age	7.692310e-03	2.849001e-03
var. age	5.000003e-02	1.004996e-02
avg. mass lipid	1.153848e-05	1.091527e-02
var. mass lipid	1.153847e-06	4.615385e-03
avg. mass structure	1.538463e-04	3.076922e-03
var. mass structure	8.846153e-05	6.804693e-03
avg. A1	0.000000e+00	0.000000e+00
var. A1	7.692300e-16	4.048540e-03
avg. resource	0.000000e+00	0.000000e+00
var. resource	1.001731e-17	3.042178e-01
avg. plx	3.846150e-04	1.831499e-03
var. plx	7.692313e-06	4.524923e-03

The graphs of the ratios and differences for the time series averages are shown in Fig. 2.13. As this shows, the differences between the results are minor. The average distances between the time series results for the reference population are given in Table 2.10. As before, the dynamics for the population containing one ecotype under no effects is compared to the dynamics of the reference population; Figs. 2.14 through 2.19 show the graphs for the averages and standard deviation for the structuring variables of the two populations. As with total biomass and density, the behavior of one ecotype under no effects is very different from that of the reference population with respect to the structuring variables. This is reflected in Table 2.29, which shows the average distance between the time series for the populations. From this, it can be seen that populations with different behaviors have average distances on the order of one to three times larger than those in Table 2.9. As before, average distances on the order of 0.01% or less of the scale of the data indicates population behavior which are similar, while large average distances reflect population dynamics which are different. Not every variable in every comparison displays a small error, however, the variables in a comparison should be considered collectively, not independently. From a collective stand point, Tables 2.7 through 2.27 provide the support needed to conclude that the parallel model has introduced only small errors into the observed behavior of a population. In particular, omitting a parallelized combined births has not harmed the results of the model.

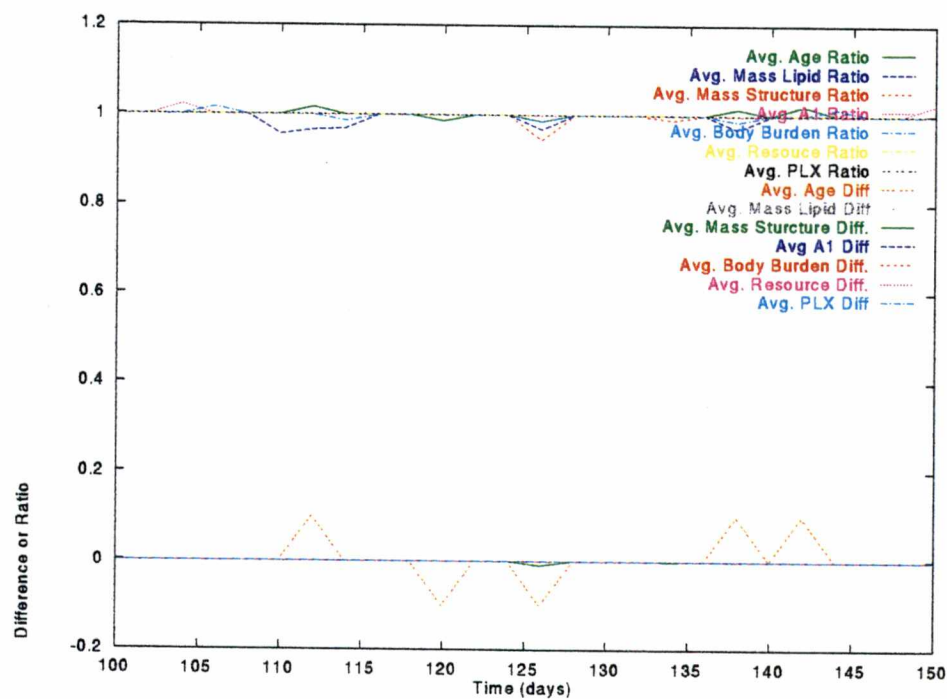


Figure 2.13: Graph of the ratio and differences for the behaviour of a population as generated by the serial and parallel models.

Table 2.29: The average distance between the time series data for a population with 1 ecotype, under no effects, run on 16 nodes, and a population with 27 ecotypes, under all effects, run in serial.

Variable	Average Distance	
	Unscaled	Scaled
rho sum	2.700868e+04	2.090034e-01
zmass	2.344231e+02	3.801442e-02
avg. age	3.234615e+00	1.775813e-01
var. age	1.931923e+01	9.966973e-02
avg. mass lipid	1.664231e-02	1.843371e-01
var. mass lipid	1.386154e-03	1.949538e-01
avg. mass structure	9.380769e-02	2.049530e-01
var. mass structure	4.447692e-02	1.978328e-01
avg. A1	2.261538e-06	3.076941e-02
var. A1	5.073077e-13	6.480562e-01
avg. resource	2.858231e-03	5.494622e-03
var. resource	1.861538e-11	5.479384e-01
avg. plx	0.000000e+00	0.000000e+00
var. plx	7.580769e-04	NaN

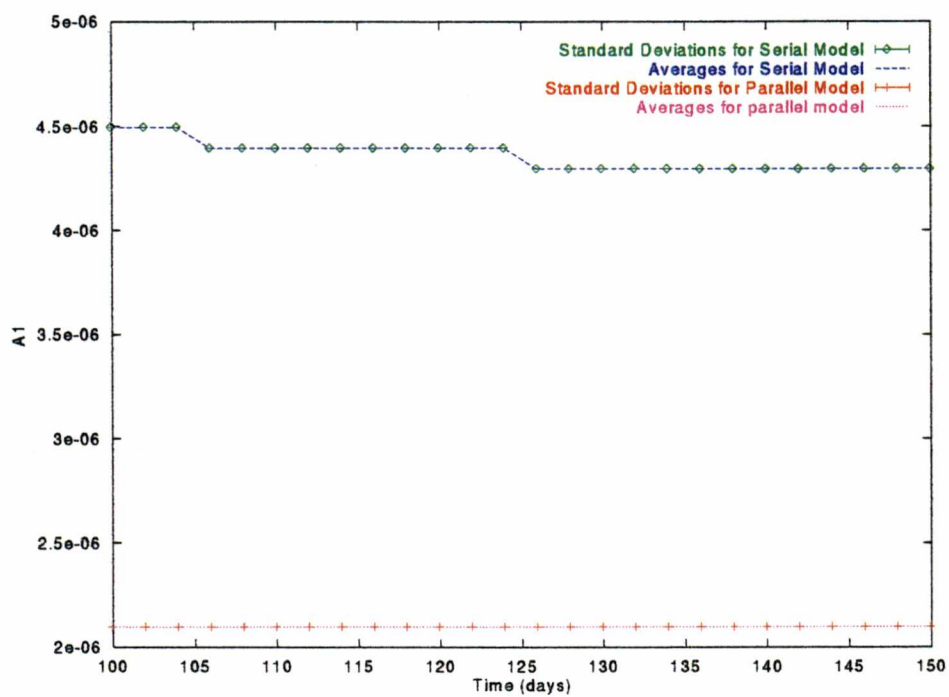


Figure 2.14: Graph of the time series averages and standard deviations for A_1 for different populations.

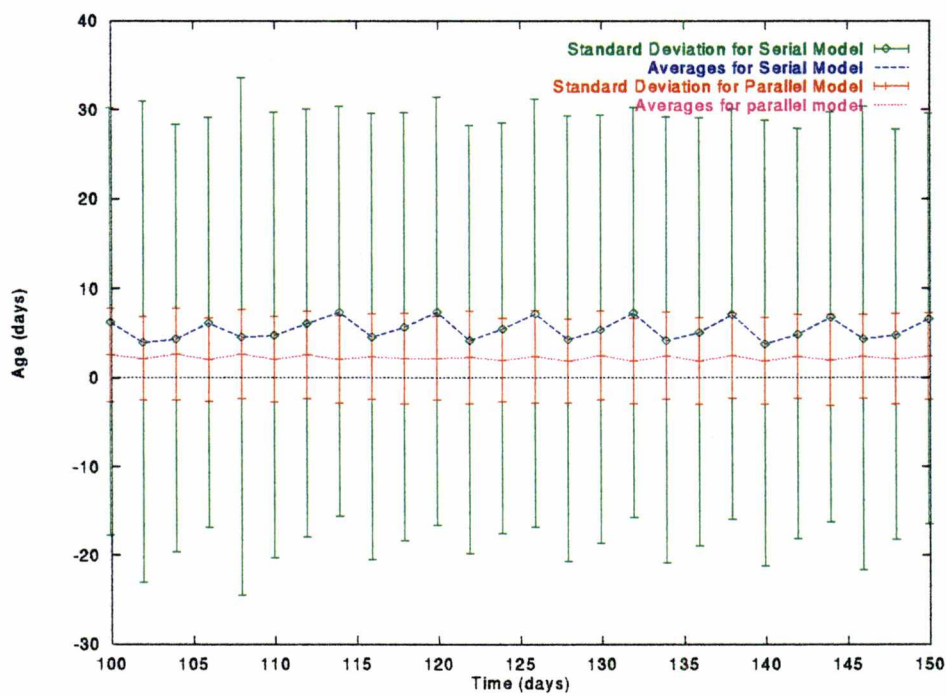


Figure 2.15: Graph of the time series averages and standard deviations for age from different populations.

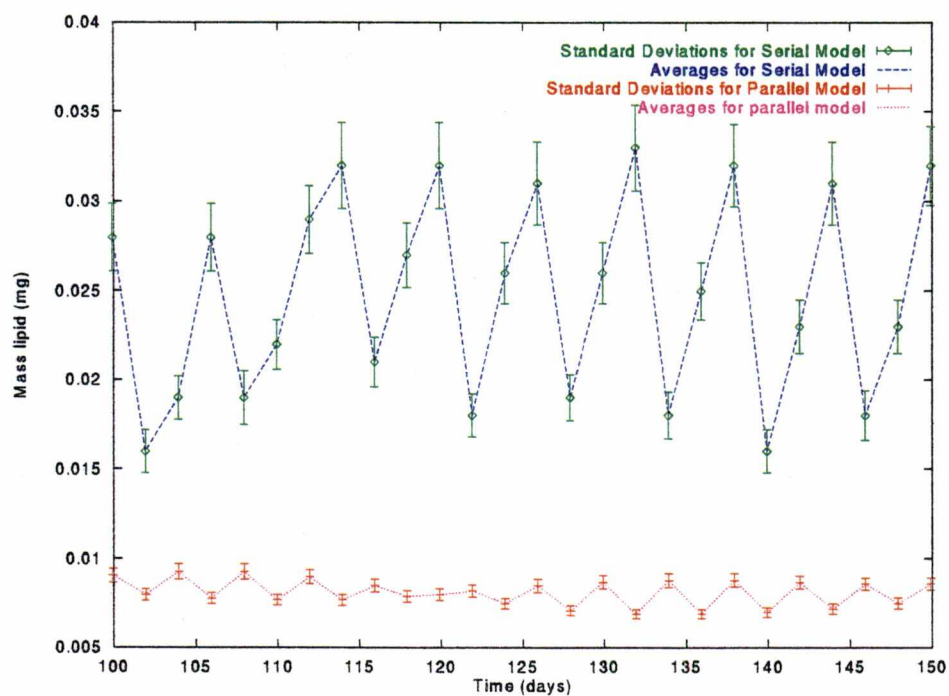


Figure 2.16: Graph of the time series averages and standard deviations for mass of lipid from different populations.

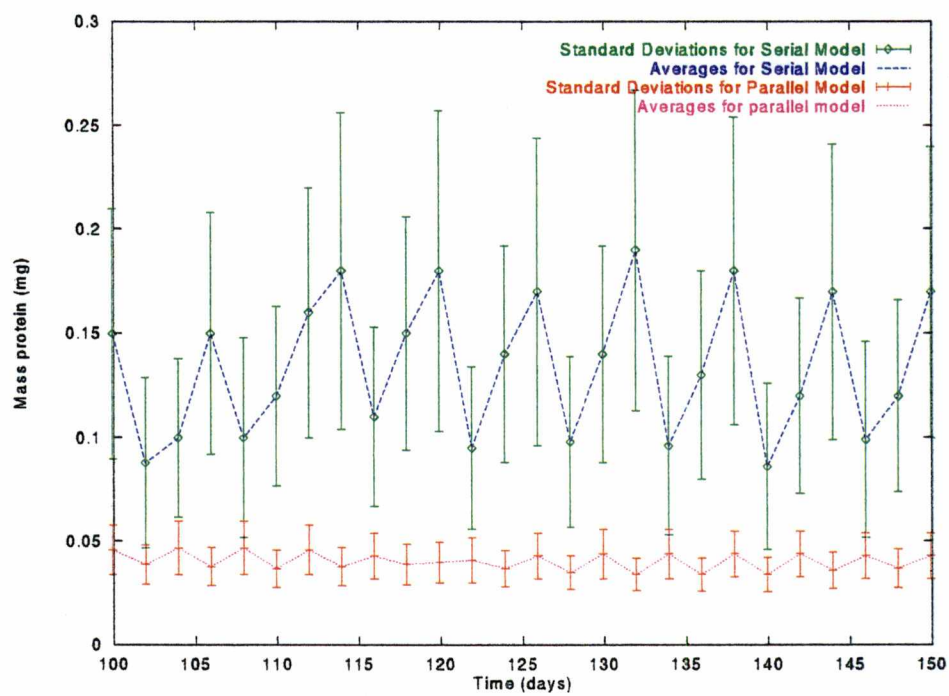


Figure 2.17: Graph of the time series averages and standard deviations for mass of structure material for different populations.

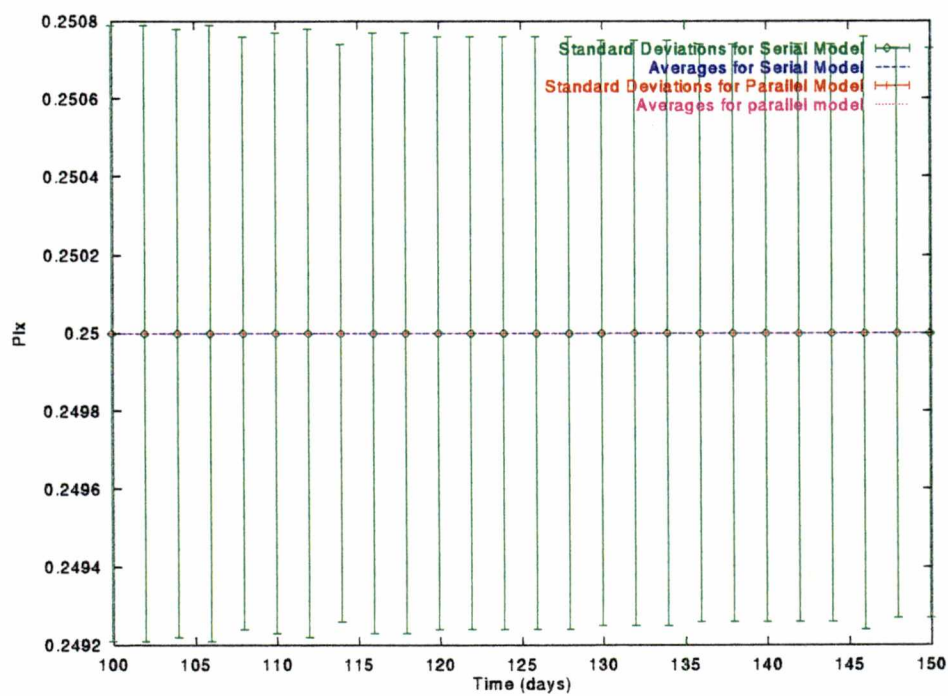


Figure 2.18: Graphs of the time series averages and standard deviations for plx levels for different populations.

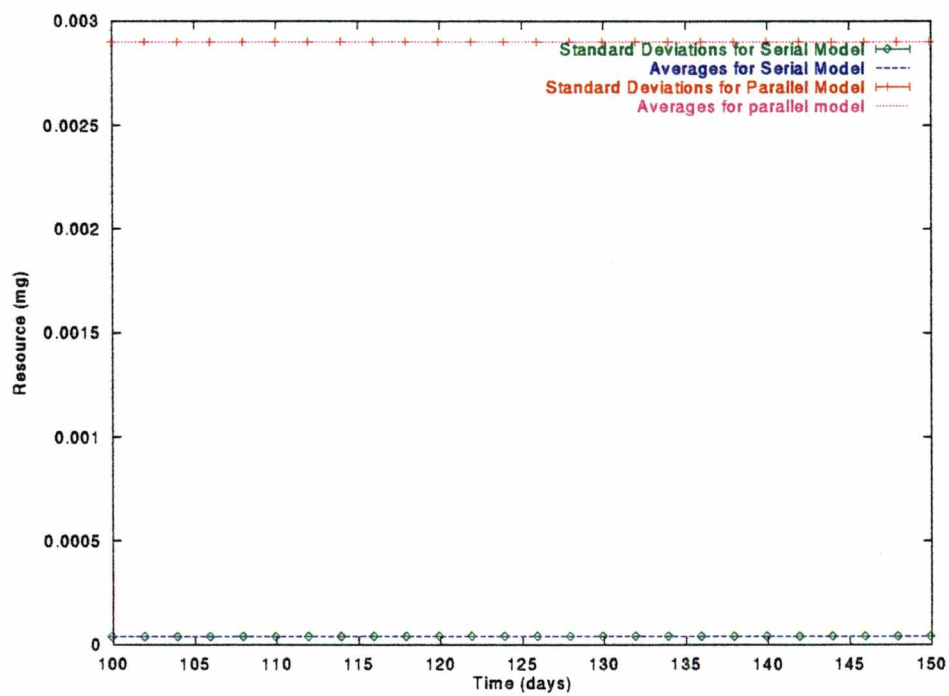


Figure 2.19: Graphs of the time series averages and standard deviations for resource level for different populations.

Having established accuracy of the model, fulfillment of the other goals can be assessed. The other goals of the model were to increase the speed of the model, have the model scale well over different numbers of nodes and to make the code portable.

The source code is a combination of C and Fortran, both of which are standardized languages and are supported on many computing platforms. The code written in each language is designed with portability in mind, and standard FORTRAN 77 or ANSI C constructs were implemented wherever possible. For the features of the C code which are not covered by ANSI C standard, calls common to all Unix platforms where used. The use of the Unix operating system calls does not seriously detract from the model as the majority of platforms supported by PVM are Unix based systems. The PVM functions present no portability problems, since all the the calls used are supported by PVM on all platforms. The code was successfully compiled on Sun Sparc work stations, and an IBM RS/6000 (the nodes of the IBM SP-2). The code was successfully run on different numbers of nodes from a single node to as many as 25. The model was also run on PVM networks consisting of machines of greatly differing speeds. In each situation the load balancer was able to keep the work evenly distributed over the nodes. Table 2.30 shows the average and variance of the time each node spent communicating for some of the test populations. The data was collected for a groups of machines which ranged in computational power. Of particular interest is the small size of the variance in each case, indicating each node in a virtual machine spent roughly the same amount of time communicating,

Table 2.30: Averages and variances of the time spent communicating.

Number of Ecotypes	Environment	Number of Computers Used	Communication Times Averages	Variance
1	All	7	74.13	2.18
1	None	7	75.13	2.27
1	Toxicant	7	76.07	2.28
3	All	7	86.94	4.9
9	None	7	81.12	3.19
9	Toxicant	7	93.3	8.99
27	All	11	374.85	4.8

which means no node was left idle for a prolonged period of time.

An examination of the times required by the parallel model to simulate various populations on different numbers of nodes reveals some unexpected results. In particular the model does not handle populations with small numbers of ecotypes well. As Fig 2.20 shows, when a population containing a single ecotype is simulated, the use of even two nodes requires more time than does the serial model. The parallel model is also affected by the environmental stresses which are imposed on the population. As each of the graphs in Figs. 2.20 through 2.23 show, when a population is placed under more environmental stresses, better performance is obtained from the parallel model.

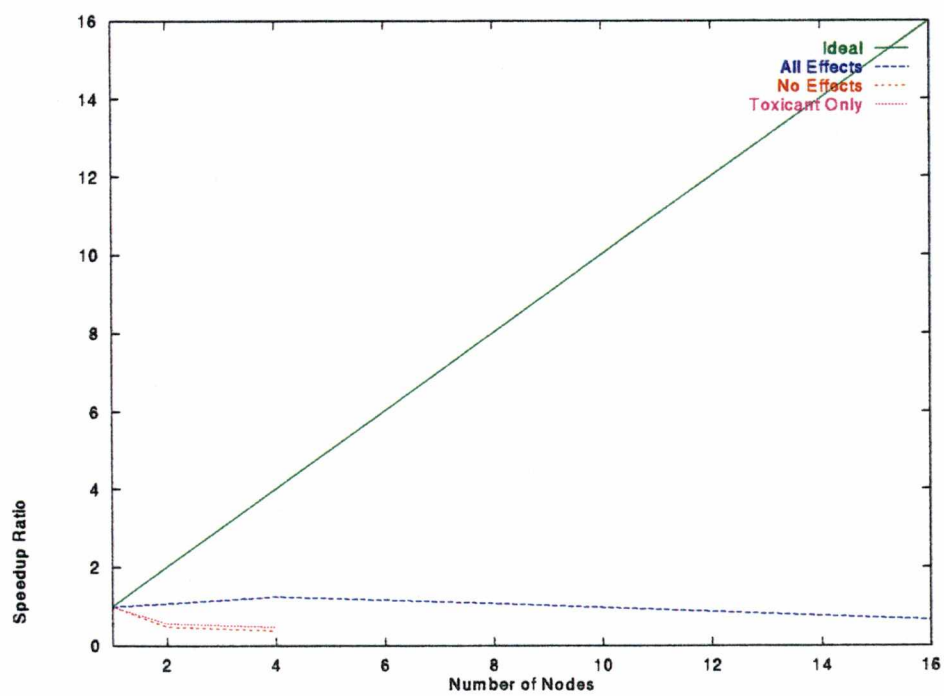


Figure 2.20: Time scalings for a population with one ecotype.

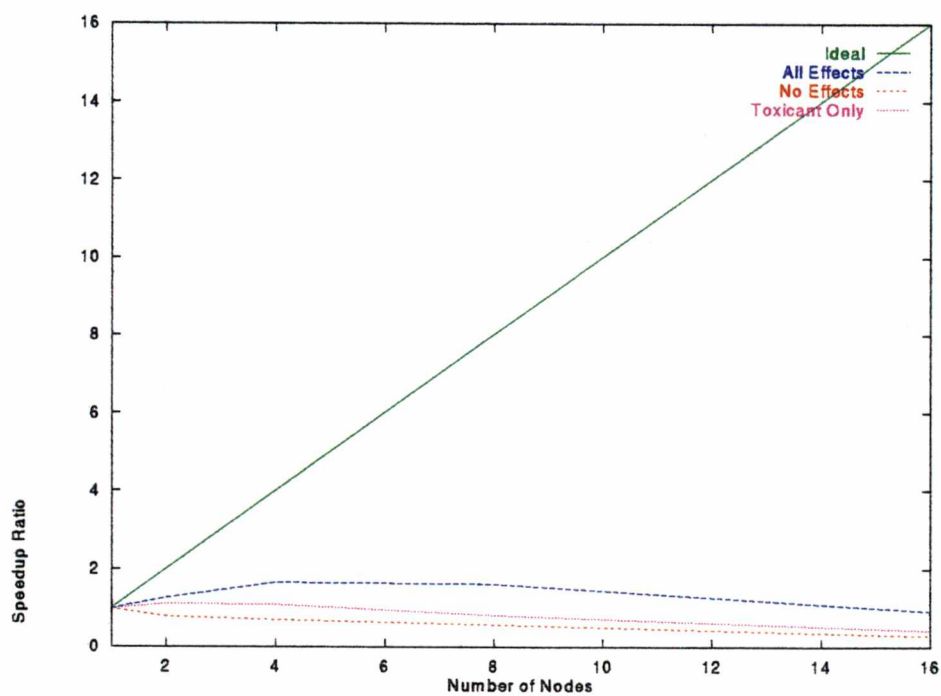


Figure 2.21: Time scalings for a population with three ecotypes.

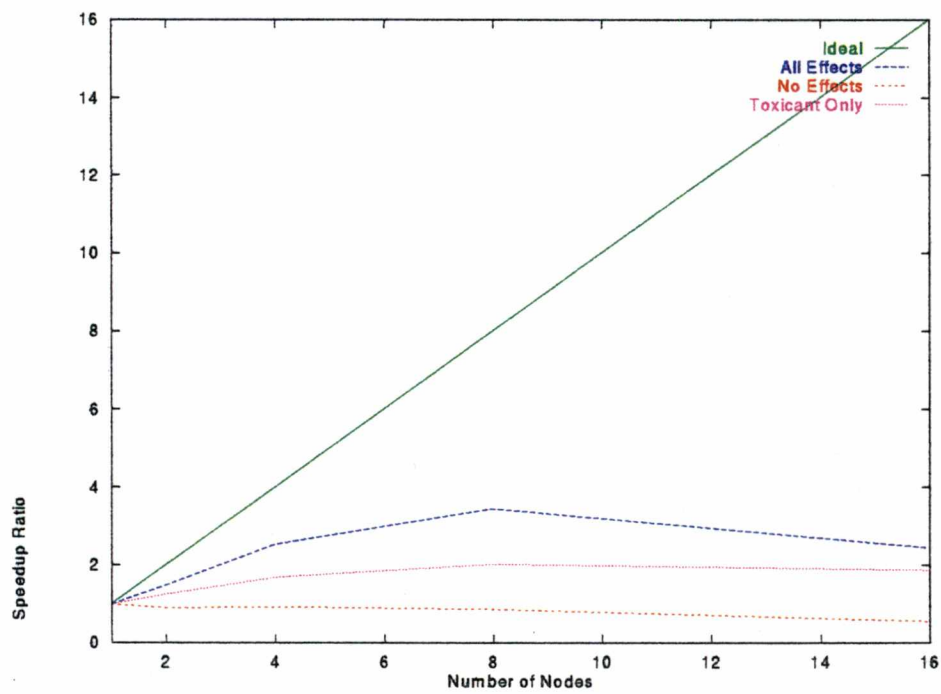


Figure 2.22: Time scalings for a population with nine ecotypes.

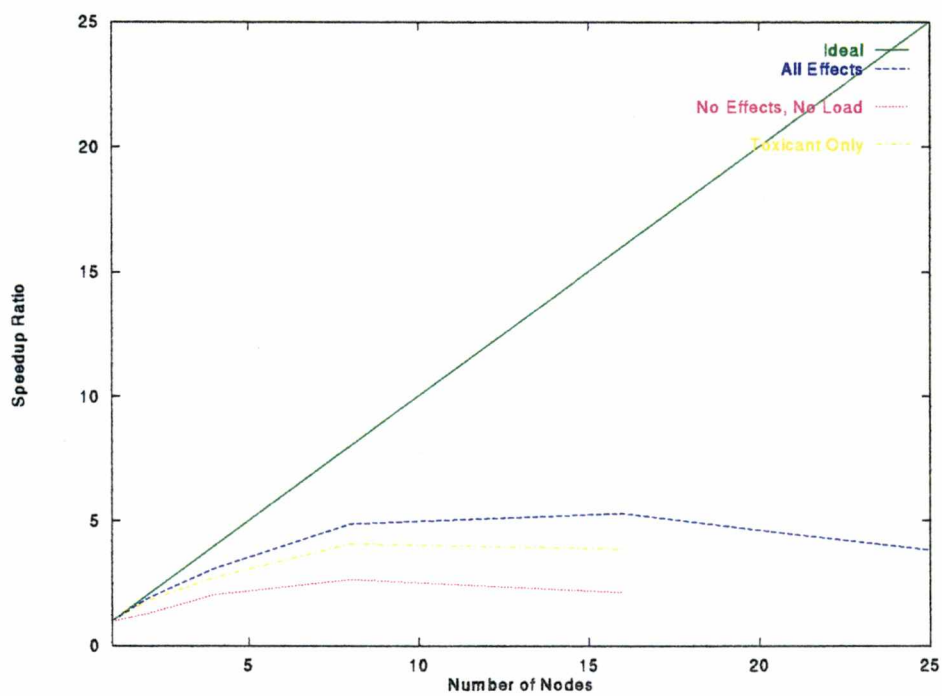


Figure 2.23: Time scalings for a population with twenty seven ecotypes.

Currently, the reason the number of ecotypes and environmental stresses affects the performance of the parallel model is not completely understood. It is undoubtedly related to the non-global combined births, and the fact that more complex populations tend to have larger numbers of individuals. However, the exact details of the interaction between combined births and the parallel environment have not yet been resolved. Some understanding as to why population diversity affects the parallel model can be obtained by examining a population which contains a single ecotype. When run in serial, combined births guarantees that for each time step the model will only generate one new cohort. If the population is run in parallel, on N nodes, combined births is not effective because the births on different nodes are not combined. Thus, there is the possibility of adding as many as N new characteristics on each time step. Over a relatively short period of time, this results in the parallel model processing at least as many cohorts per node as the serial machine. Fig. 2.24 illustrates this with a graph of the number of cohorts processed through time, by the serial algorithm and two nodes of a parallel machine, for a population with one ecotype under toxicant stress only. From this graph it can be seen that after about 50 days, the parallel model is processing twice as many cohorts as the serial model, and this ratio remains close to this level for the rest of the simulation. Combined with the communication time, the result is degraded performance from the parallel model. As the number of ecotypes is increased, the effects of not having global combined births are diminished. The likely cause is that populations with large numbers of ecotypes

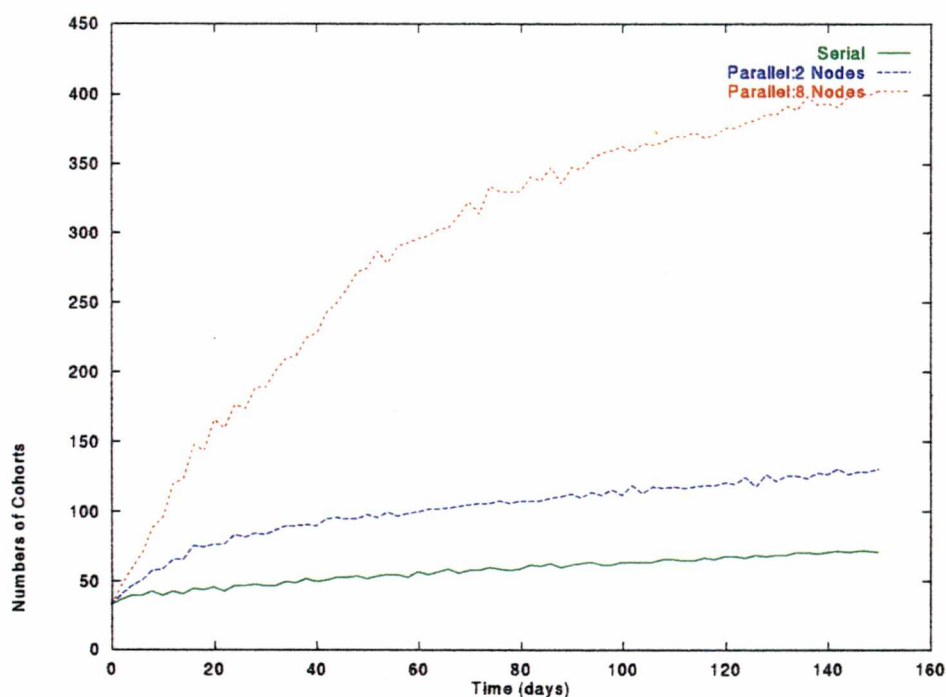


Figure 2.24: Number of cohorts vs. time for a population containing 1 ecotype under toxicant stress, as generated by the serial model, 2 and 8 nodes.

have larger numbers of cohorts, since even in serial, new cohorts of different ecotypes are not combined. Fig. 2.25 shows the graphs of numbers of cohorts in the population through time for a population containing 27 ecotypes under toxicant stress only. One line gives the number of cohorts generated on a serial machine, the second the number of cohorts generated on 2 nodes and the third on 16 nodes. The best speculative answer at this time is when the number of ecotypes is large relative to the number of nodes, there is a larger chance of two parents of the same ecotype, giving birth at the same time, being on the same node. Since births are combined locally, the number of new cohorts added to the population is kept to a minimum. The exact mechanism

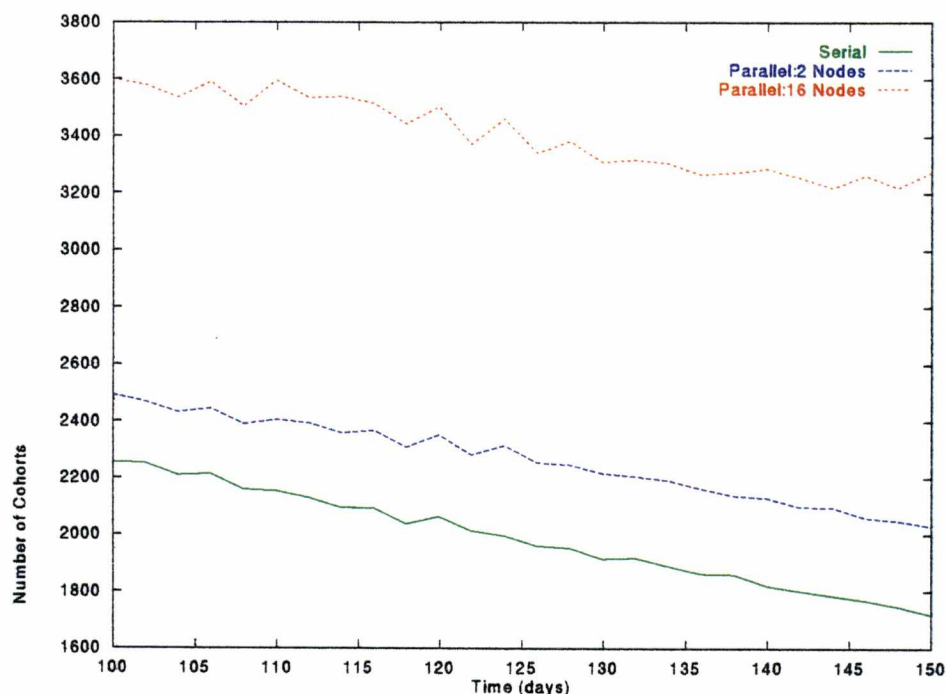


Figure 2.25: Number of cohorts vs. time for a population containing 27 ecotypes, under toxicant stress, as generated by the serial model, 2 and 16 nodes.

behind this process is not understood in general, though this scenario seems likely, and is supported by 2.25. This graph shows three lines, one for the number of cohorts for the serial model, one for the parallel model using 2 nodes, and the last for the parallel model using 16 nodes. In this graph, the number of cohorts generated by 2 nodes is almost identical to the number of cohorts generated when the model is run in serial. This may be due to the increased number of ecotypes, but may be more closely related to the number of cohorts per node. The population with 27 ecotypes, starts with 267 cohorts in the population, while the one ecotype population has 36. When these populations are distributed over two nodes, there are very different numbers of

cohorts per node. By distributing the twenty seven ecotype population over sixteen nodes, there are approximately 17 cohorts per node, comparable to the distribution of the one-ecotype population on two nodes. For the one-ecotype population, with 18 cohorts per nodes, the result was a doubling in the number of cohorts held by the population. In fact, as the third line in Fig. 2.24 shows, using 8 nodes resulted in approximately 8 times as many cohorts. If number of cohorts per node is the sole indicator of the rate of combined births, and using N nodes multiplies the number of cohorts by N , then using 16 nodes on the twenty seven ecotype population should result in 16 times as many cohorts. This is not the case, as Fig. 2.25 shows; when the population with twenty seven ecotypes is run on 16 nodes, the number of cohorts is only increased by approximately $3/2$. So in some other way, the number of ecotypes is affecting the performance of the model.

The reason environmental effects change the performance of the model seems straightforward. As each population is placed under greater environmental stress the performance of the parallel model is improved. This is reflected in the heights of the scaling curves in Figs. 2.20 through 2.21. This is explained by Fig. 2.26, which shows the number of cohorts generated for the reference population on 16 nodes. As this graph shows, when the environmental stresses are decreased, the number of cohorts in the population increases. The greater number of cohorts reflected in this graph is the result of greater life span and fecundity under lighter environmental stresses.

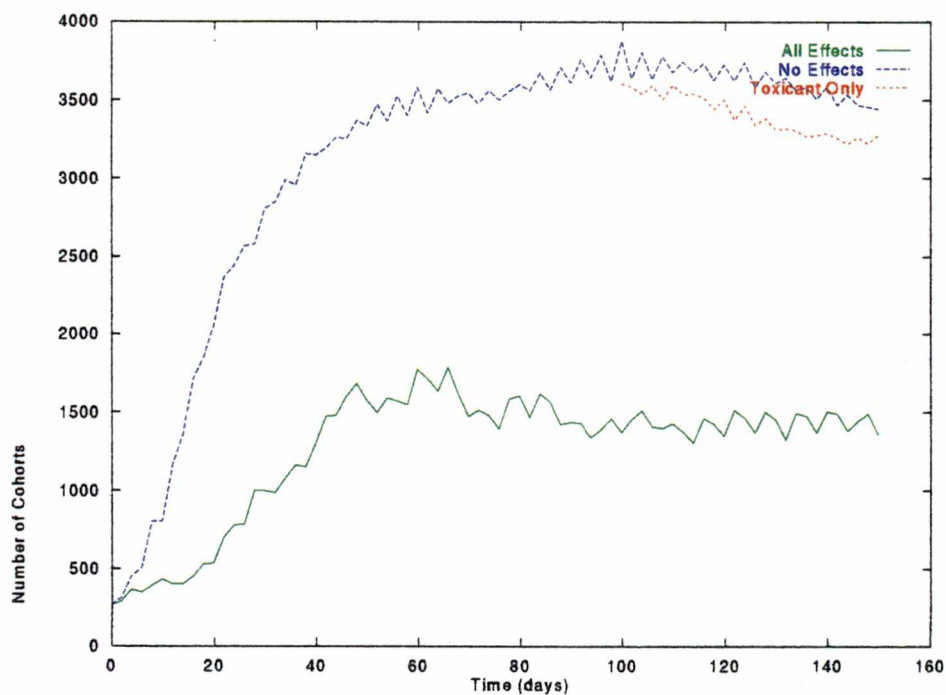


Figure 2.26: Number of cohorts vs. time for a population containing 27 ecotypes, as generated on 16 nodes.

The larger number of individuals means more work of the parallel model, and hence longer amounts of time for execution.

The information obtained from the scaling results are not all negative. On the contrary, they show the parallel model is quite capable of handling complexity, both in the form of environmental factors and population diversity, and offers better overall

performance than the serial model. This is supported by the graphs appearing in Figs 2.20 through 2.23. Notice in each graph, as the environmental conditions go from no stress, to the inclusion of toxicant to the addition of dissolved oxygen and temperature effects, the height of the scaling curves increases. This indicates the parallel model is running for shorter periods of time relative to the serial model, as each condition is added. This is most noticeable in Fig. 2.23. By tracing the scaling curves, for a particular set of environmental effects, from one graph to the next, the ability of the parallel model to better handle more complex populations becomes clear. Not only does the parallel model run faster than the serial model as the population becomes more complex, but as the number of ecotypes is increased, the extrema of a set of scaling curves moves out toward larger numbers of nodes. This means higher levels of diversity in the population lead to higher levels of parallelism in the model. In a sense, the parallel model responds to greater complexity by becoming more efficient. Clearly the parallel model has fulfilled the goal of offering a speed advantage over the serial model, as seen in Fig. 2.27, which shows the time required to process the reference population on an 11 node PVM machine, and a serial machine with more computational power than any of the 11 nodes. The serial machine in this case was a Sun Sparc 20 model 61 and the PVM network consisted of four Sun Sparc 10s, two Sparc 5s, a Sparc LX, a Sparc 1, and three Sparc IPCs. In addition, as a population and its environment are made more complex, and thereby more realistic, the parallel model provides a greater advantage over the serial model.

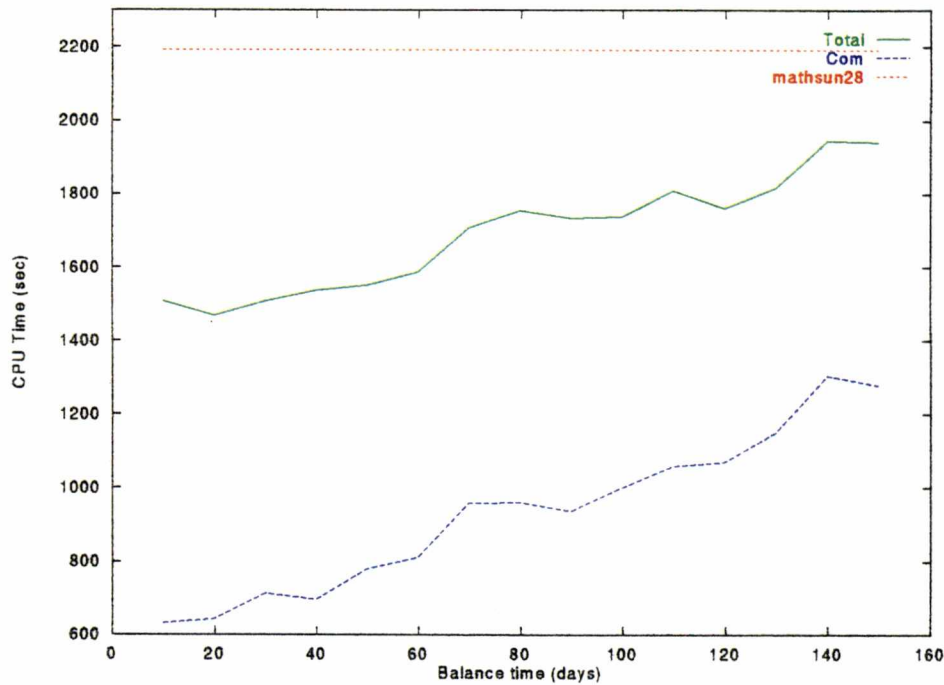


Figure 2.27: Processing time for a serial machine and a 11 nodes PVM network.

As each graph shows, as the number, and hence complexity, of environmental stresses are imposed on each population, the performance of the parallel model improves, both in terms of the optimal number of nodes applicable to the problem, and the speed improvement over the serial model. Furthermore as the number of ecotypes increases, the model delivers better performance.

Thus, the parallel model fulfills the goal of providing better performance over the serial model. This performance can be extended to compensate for the weaknesses of the model. Model populations which are now thought of as containing a single ecotype can be refined into populations with many closely related ecotypes. This will exploit the advantages of the parallel algorithm and provide a more detailed picture

of the behavior of population with very similar traits.

The discrepancies and shortcomings of this model indicate the need and potential for several avenues of future research. The most obvious inadequacy is the lack of a global combined birth algorithm. Some investigation into this problem has already been made by Siddharthan Ramachandramurthi. His approach does allow for births to be combined even if they occur on different nodes. However, it is only effective under certain environmental conditions. There is still a need to find an approach which will allow for births to be combined under any combination of effects. Related to this topic is the investigation as to why population diversity, and environmental effects alter the performance of the model. An understanding of how these factors affect the parallel model may lead to an approach which will cure the problems arising from combined births.

The load balancing algorithm also has some drawbacks. Currently load balancing is performed at regular intervals. The optimal value for the frequency of load balancing currently can not be determined before running the model. Finding a method for determining this value *a priori* or determining this value during run time would greatly enhance the performance and functionality of the model. In the course of this study it was discovered that cohorts are dropped by the model when the number of individuals within that cohort fall below a certain level. Again, because combined births is not carried out on a global scale within the parallel model, this changes the way in which the parallel model behaves as compared to the serial model. Cohorts

are dropped by killing them from within the black box, and it is done to keep the computational load down by eliminating those cohorts which have so few individuals as to have an insignificant effect on the population. An alternative, which makes more sense for the model in general and may improve the accuracy of the parallel model, is to dynamically set the cut-off level based on the current state of the population. For instance, the cut-off point could be set as a percentage of the average number of individuals per cohort. This still eliminates insignificant portions of the population, but allows the model to compensate for the variability in the number of individuals per cohort resulting from the parallel model. Even if a global combined births is found, this addition will make the model more dynamic and realistic, eliminating cohorts based on their size relative to the population in which they exist, and not by a global standard for relevance. In addition to refining the algorithm, the model is also ready now to investigate greater levels of population resolution. Some of the possible improvements which can now be made were outlined in an earlier section and include increasing population resolution and decreasing the time step.

Conclusion

The parallelization of the *Daphnia* model is successful on many different levels. First, as was shown in Chapter 2, the parallel model generates population behaviors consistent with those generated by the serial model, and hence can be relied upon to return ecologically meaningful results. As is shown in Fig. 2.27, the parallel model also offers a speed advantage over the serial model, even when the serial model is run on a machine which is faster than any of the nodes of the PVM machine. The ability of the parallel algorithm to complete simulations quickly can be used to offset the increased computational load resulting from refining the model. In addition, Figs. 2.20 through 2.23 show that the model is scalable, meaning increased computational power is obtainable by increasing the number of nodes in the parallel machine, and that the number of nodes which provide the best speed improvement is increased by increasing the population complexity. Thus, the parallel model is better suited to dealing with complex populations than is the serial model. During the course of this study, the parallel model was only ported to a few architectures, and there

were no major obstacles encountered when the underlying hardware was changed. In addition the code was written with portability in mind and only standard functions were employed which should make for easy porting of the code to other platforms. Finally, the algorithm was able to take advantage of PVM networks, consisting of machines with varying levels of processing power, and provided this feature with a minimum of effort on the part of the user. The parallel model written for this study satisfies the established goals and most importantly, offers an improvement over the serial model. These improvements will allow for the simulation of model populations which more accurately reflect actual populations, and thus enhances the usefulness of the model as a tool for understanding *Daphnia* populations. The success of this model also lends credence to parallelization as a tool for modeling ecological systems in general.

•

Bibliography

Bibliography

- [1] Thomas G. Hallam, Ray R. Lassiter, Jia Li, and William McKinney. *Individual-Based Models and Approaches in Ecology*, chapter 15: An Approach for Modeling Populations with Continuous Structured Models, pages 312–337. Chapman & Hill, 1990.

- [2] Thomas G. Hallam, Ray R. Lassiter, Jia Li, and Luis A. Suarez. Modeling individuals employing an intergrated energy response: Application to *Daphnia*. *Ecology*, pages 938–954, 1990.

- [3] H. L. Koh, Thomas G. Hallam, and H. L. Lee. Combined effects of environmental and chemical stressors on a model *daphnia* population. Technical report, Universiti Sains Malaysia, 1994.

- [4] Thomas G. Hallam, Graciela Ana Conziani, and Ray R. Lassiter. Sublethal narcosis and population persistence: A modeling study on growth effects. *Environmental Toxicology and Chemistry*, 12:947–954, 1993.

- [5] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM 3 user's guide and reference manual. Technical report, Oak Ridge National Laboratory and University of Tennessee and Carnegie Mellon University and Pittsburgh Supercomputing Center and Emory University, 1993.
- [6] Flynn. *Some Computer Organizations and Their Effectiveness*. IEEE Transactions on Computers, 1972.
- [7] W. Richard Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley, 1992.

Vita

Scott Sylvester was born on April 16, 1969 in Chaplehill North Carolina. He spent the majority of his life in Teaneck New Jersey where he suffered through the Teaneck Public School System. He graduated from Teaneck Public High School in May of 1987. The following August he started college at The University of North Carolina at Greensboro. He started his studies in physics, but changed over to mathematics because of frustration with the incomplete mathematics used in the physics courses he took. In 1991 he finished his Bachelors of Science in mathematics. In the fall of that year he started graduate school at The University of Tennessee at Knoxville. In 1995 he graduated from the University of Tennessee with a Masters degree in mathematics. After graduate he started work with the University of Tennessee as a computer programmer/analyst with the Everglades research project. Future avenues of research include Ecology, Computer Science and Computer Modeling.