5-2008

Parallel Simulation of Individual-Based, Physiologically-Structured Population and Predator-Prey Ecology Models

Jeffrey A. Nichols
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Jeffrey A. Nichols entitled "Parallel Simulation of Individual-Based, Physiologically-Structured Population and Predator-Prey Ecology Models." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Mathematics.

Thomas G. Hallam, Major Professor

We have read this dissertation and recommend its acceptance:

Don Hinton, Lou Gross, Charles Collins

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Jeffrey A. Nichols entitled "Parallel Simulation of Individual-Based, Physiologically-Structured Population and Predator-Prey Ecology Models." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Mathematics.

Thomas G. Hallam, Major Professor

We have read this dissertation
and recommend its acceptance:

Don Hinton

Lou Gross

Charles Collins

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the
Graduate School

(Original signatures are on file with official student records.)

Parallel Simulation of Individual-Based, Physiologically-Structured Population and Predator-Prey Ecology Models

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Jeffrey A. Nichols

May 2008

Dedication

This dissertation is dedicated to:

My Saviour who has given me life

and

My wife who has given me love

Acknowledgments

Being somewhat of a polymath, I tend to be distracted by new challenges and problems. This work culminates my time in graduate school begun eighteen years ago in August 1990. I left graduate school for an extended period to pursue other challenges. Over these years I have tested the patience of many and I thank you for your patience.

In the years between my first and second attempts at completing this dissertation, I also learned that no one is an expert in everything, and that we all depend on each other. To that end, there are many to whom I owe thanks and want to acknowledge for their part in this dissertation:

First to my dear wife, Sherry, who has made so many sacrifices for this dissertation. Thank you.

To my children, especially Rebecca for her offer to help me with that "math thing I was working on" after helping her with her science fair project on the mathematics of Sudoku puzzles. And to Philip, Karla, and Rachel for helping me keep it real and know that life goes on, even when I did not see any hope of finishing this dissertation.

To my parents and parents-in-law: Thank you for demonstrating life-long love and dedication.

To the Members of my Committee:

Dr. Tom Hallam for so many things. I still remember the moment I received the e-mail to see if I wanted to come back. I am so glad we were able to wrap up this "loose end" before your retirement (barely!).

Dr. Don Hinton who visited UAB several times when I was an undergrad and was my advisor when I was in the Research Experience for Undergraduates (REU) program at UT in 1989. I certainly cannot leave out the several hikes in the mountains. And for giving my ODE exam (again) when I returned to grad school.

Dr. Charles Collins for my Numerical Analysis class, excellence in teaching, and a most helpful discussion under a shade tree when I was so frustrated and did not think I had done anything worth mentioning.

Dr. Lou Gross for teaching REU, for being another night owl, and for having the ideas and drive that got this whole project started.

To Dr. Stephanie Smullen for encouragement, guidance, and advise.

To Dr. Bill Holls for delivering our children and for his support through several years.

To my current colleagues for your help and discussions: Dobri, Haritha, and Paula. To my former colleagues in graduate school who so often helped: Kenny, Renee, Nathan, and Eric. Special thanks to

Suzanne for copying (twice!) her notes from Applied Linear Analysis to replace my lost set, so that I could take the exam when I returned to graduate school. And finally and especially to Shandelle. It seemed that when I was most disquieted and was failing to understand something that I found in one of her articles the exact idea or derivation I was missing.

To Bob, Bill, and Jerry at ACN for their financial support enabling me to return. I enjoyed the opportunities you gave to "change everything" more than once.

To Paul Izbicki for seeing in me something special and showing me how a switcher works. I will never forget it!

To Dr. Michael B. Johnson whose dissertation provided impetus to return to my own and for his exhortation at WWDC to "Go out and do great things."

To Apple Computer for building the machines and the environment on which I could create great things. Special thanks for the Student Scholarship to WWDC07 at which I had the extreme pleasure of meeting George Warner, Ken Wilson, and the great crew from Intel. I will never forget the thrill of using OS X for the first time. It started me dreaming and I have not been disappointed.

And to NCSA who provided supercomputer time in support of the research in this dissertation. (I finally got to put this acknowledgment in a published work!)

Abstract

Utilizing as testbeds physiologically-structured, individual-based models for fish and *Daphnia* populations, techniques for the parallelization of the simulation are developed and analyzed. The techniques developed are generally applicable to individual-based models. For rapidly reproducing populations like *Daphnia* which are load balanced, then global birth combining is required. Super-scalar speedup was observed in simulations on multi-core desktop computers.

The two populations are combined via a size-structured predation module into a predator-prey system with sharing of resource weighted by relative mass. The individual-based structure requires multiple stages to complete predation.

Two different styles of parallelization are presented. The first distributes both populations. It decouples the populations for parallel simulation by compiling, at each stage, tables of information for each of the distributed predators. Predation is completed for all fish at one time. This method is found to be generally applicable, has near perfect scaling with increasing processors, and improves performance as the workload to communications ratio improves with increasing numbers of predator cohorts. But it does not take best advantage of our testbed models.

The second design decouples the workload for parallel simulation by duplicating the predator population on all nodes. This reduces communications to simple parallel reductions similar to the population models, but increases the number of cycles required for predation. The performance of the population models is mimicked.

Finally, the extinction and persistence behaviors of the predator-prey model are analyzed. The roles of the predation parameters, individual models, and initial populations are determined. In the presence of density-dependent mortality moderating the prey population, competition via resource of the larger fish versus the smaller is found to be a vital control to prevent extinction of prey population. If unconstrained, the juvenile fish classes can — through their rapid initial growth and predation upon the juvenile prey classes — push the prey population to extinction. Persistence of the predator-prey community is thus threatened when the fish population is dominated by juveniles. Conversely, the presence of larger fish moderates the juveniles and stabilizes the community via competition for shared resource.

Contents

List of Tables

List of Figures

Overview

There are two purposes behind the research presented in this dissertation. The first purpose is to explore the applicability of parallel computers for the simulation of individual-based, physiologically-structured population and community ecology models. We utilize as testbeds for our techniques extant population models for *Daphnia* (parameterized for *D. magna*) and trout (parameterized for rainbow trout, *Oncorhynchus mykiss*). The second purpose is to understand for an individual-based, predator-prey model composed of these two populations, the interactions between the parameters, populations, and individual models.

Chapter 1 documents our testbed models and introduces our terminology and general solution techniques. A decade ago, when we began our parallelization efforts, the size of our computational problems was very large compared to the desktop and workstation-level computers available at the time. The execution of these models took a long time, especially the original predator-prey models, which could take several days to complete. Chapter 2 describes the parallelization of individual-based population models. Chapter 3 describes the testbed community model and its parallelization.

Our testbed population models had been previously developed and their important behaviors were well-understood. A preliminary version of the predator-prey model we took as our testbed for parallelization had been developed, but the primary focus had been on toxicant flow through the community and its effects. There had been no focus on the dynamics, important behaviors, and features. In particular, the roles of the predation parameters, competition for resource, initial populations, population structures were not known. Our discoveries for this model are described in Chapter 4.

Chapter 1

Introduction and Overview of Parallelization of Individual-based Models

1.1 Introduction

This chapter has two purposes. First, we introduce the models on which the parallelization efforts are tested. Second, we provide an overview of our parallelization efforts introducing the terminology and techniques we have applied during our research.

The first part of this chapter describes both the base individual models and the population models which envelope these individual models. Important physiological aspects of the organisms that affect our efforts are also described. The individual models play significant roles through both the parallelization and analysis of these models, so this chapter is referred to throughout this dissertation.

The *Daphnia* and fish models have been developed over a number of years. The references to the development and analysis of these models are consolidated herein. The population models add population density, births, initial populations, and per capita mortality to the individual models which are each described in this chapter. Typical output and behavior of the individual models and the population models are described and illustrated. Density-dependent mortality is introduced especially because it is significant to our analysis of the predator-prey model.

The individual fish model differs significantly from the *Daphnia* model in that it utilizes a size-structured resource. The predator-prey model introduced in Chapter 3 makes fundamental use of the size-structure of the prey population. The functional response for the fish and its dependencies on the particular prey items is important for both the parallelization and analysis efforts, so special attention is focused on it in this chapter.

The second half of this chapter provides an overview of our parallelization efforts. The original simulation codes on which we built our testbeds were not designed in a way that was conducive to parallelization. The redesign required to draw out the parallel structure innate to individual-based models is described. Further, parallelization efforts must always consider load balancing issues. Our solution to these issues is illustrated in this chapter. For individual-based models, birth combining is a unique aspect that must also be considered. The research directions that these efforts led to are described. These led to the discovery of synchronizations that are inherent in individual-based simulations that lead to oscillations in the populations. These oscillations are observed in natural populations, but are often exaggerated in simulations. Even just small oscillations can lead to large effects when combined with the threshold behavior of density-dependent mortality, a consequence which is observed in later chapters. These synchronizations also provide an alternative way to multiply the workload required for a particular simulation. We utilize this additional workload to naturally scale our population models further into the realm where they benefit from parallel execution.

1.2 Population Models

We applied our initial efforts to existing population models of *Daphnia* and fish. The particulars concerning parallelization of these models are discussed in Chapter 2. In Chapter 3 the combination of these models into a predator-prey system which we parallelized is discussed. In this section, the basic equations and ideas are described on which these two population models are constructed. During our discussion of the parallelization efforts, especially of the predation model in Chapter 3, several of these topics will be revisited.

1.2.1 Individual-based Model for *Daphnia* Populations

Daphnia are a basic food resource for many aquatic populations. They are widely used as an indicator species for water-quality management. An NSF-funded site devoted to the order Cladocera to which *Daphnia* belong is **waterflea.org**. A picture of single *Daphnia magna* is reproduced in Figure 1.1.

Figure 1.1: Single Daphnid with Brood

Note the brood apparent inside the parent's body. *Daphnia* reproduce parthenogenetically, except under conditions of stress when males can be produced. The parent carries the brood in a pouch until they hatch and are born live. After release of her brood, she moults, a new, larger carapace is formed, and new eggs are deposited into the brood pouch. Adults spend as much as 80% of assimulated energy on reproduction (Dudycha and Tessier, 1999; Tessier et al., 1983). Because of their wide use in ecology studies and availability of data, they also are a common target for modeling studies; see for instance, Peters and De Bernardi (1987); Kooijman (2000); Hart and Gill (1993).

Individual Model

The model for *Daphnia* developed by T. Hallam's group at the University of Tennessee's Institute for Ecological Modeling (TIEM) for environmental toxicant risk assessment studies has been described in several publications. The fundamental paper describing the individual model for *Daphnia* appeared in *Ecology* (Hallam et al., 1990b). The primary development and application of a population model encompassing this individual model are described in Hallam et al. (1992b) and Hallam and Lassiter (1994). Application of these models towards demonstrating the moderating influence of lipid on lipophilic toxicants is in Lassiter and Hallam (1990); Hallam et al. (1990a). There have also been several extensions of the *Daphnia* population model. For instance, sublethal toxicity effects were added in Hallam et al. (1993). The effects of temperature and dissolved oxygen were added in Koh et al. (1997). In this section, an overview of the mathematical basis of the individual *Daphnia* model is given.

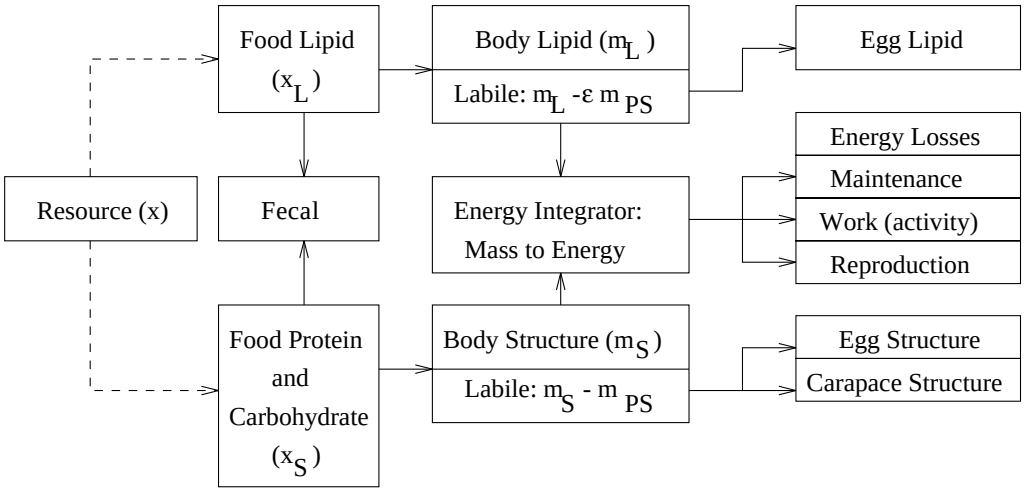


Figure 1.2: The model of an individual organism

Being developed for application to ecological risk assessment, these models incorporate submodels for testing the effects of lipophilic toxicants on populations of *Daphnia*. In order to accurately model the effects of such chemicals, the dynamics of lipid in an individual must be followed. Let m_L denote the mass of lipid (mg) in an individual. Lipid serves as a source of energy for the individual and plays a role in reproduction. During acute toxicant exposure it can also act as a buffer by diluting the effects of the toxicant on the individual. Protein and carbohydrate are also important sources of energy for an individual. These are combined into one variable called *structure*, with the justification that the energy equivalents per unit mass of protein and carbohydrate are nearly equal. Let m_S denote the mass of structure (mg) in an individual. The lipid and structure compartments are further considered to have a labile and a nonlabile portion. The final variable tracked for an individual is age. Although age is not a physiological variable, it is useful for timing several life history events for an individual. Let a denote the age of an individual.

Ordinary differential equations describing the growth with respect to time of m_L and m_S in an individual are developed in Hallam et al. (1990b) through analysis of the structure and lipid flows into and out of an individual daphnid. The flow diagram is pictured in Figure 1.2. Lipid and structure are assimilated from the individual daphnid's resource — lipid is assumed to compose some fixed percentage of this resource with the remainder being structure. Both lipid and structure can be used to provide energy for the individual. Or they may be used in bulk allocations to, for instance, form components of the eggs for reproduction, or the carapace of the daphnid.

When modeling, a common requirement is determining the length of the individual. In our models this is done through the concept of *Protected Structure* where m_{PS} represents the nonlabile amount of

the structure compartment. The concept is that protected structure is structural components of the body that are not available for energy even in the event of starvation. Protected structure is modelled as a non-decreasing fraction of m_S. It is related to the length through an allometric relationship that converts from the mass units to an expression of length; length $= \sqrt[3]{m_{PS}/\text{allometric constant}}$. Labile structure is thus $m_S - m_{PS}$. Nonlabile lipid is assumed to be proportional to m_{PS} by a constant factor ϵ.

The end of the juvenile period is assumed to be size dependent, in that *Daphnia magna* females are assumed to deposit their first brood when they reach a certain, fixed length. This will be termed the *first birth age*, b_1, later in this chapter. Bulk allocations, especially of lipid, are made from the storage compartments for egg formation. The number of eggs is constrained by the labile resources of the parent. There is a constant amount of structure allocated to each egg, with a variable amount of lipid. The lipid per egg can vary within a fixed range. (This variance of initial egg stores will also appear later in this dissertation.) A significant modeling assumption made is that after the first birth time, the individual daphnid reproduces with a fixed period. A consequence of this assumption significantly affects the population dynamics and is described in Section 1.3.6.

The differential equations for the growth of the mass of lipid and structure are:

$$\frac{dm_L}{dt} = G_L(t,a,m_L,m_S) - L_L(t,a,m_L,m_S) \tag{1.1}$$

$$= \frac{A_{0L}x_L m_S}{A_1 m_S^{1/3} + A_2 x} - \begin{cases} A_3(m_L - \epsilon m_{PS}) & for \quad D > E \\ A_3(m_L - \epsilon m_{PS})D/E & for \quad D \le E \end{cases}, \tag{1.2}$$

$$\frac{dm_S}{dt} = G_S(t,a,m_L,m_S) - L_S(t,a,m_L,m_S) \tag{1.3}$$

$$= \frac{A_{0S}x_S m_S}{A_1 m_S^{1/3} + A_2 x} - \begin{cases} A_4(m_S - m_{PS}) & for \quad D > E \\ A_4(m_S - m_{PS})D/E & for \quad D \le E \end{cases}, \tag{1.4}$$

where A_{0L} and A_{0S} are assimilation efficiencies for lipid and structure; x_L and x_S are the amount of lipid and structure, respectively, in the resource; A_1 is a parameter inversely related to maximal filtering rate; A_2 is a function $k_3 m_{PS}^{-k_1}$ (no longer a fixed parameter as in the *Ecology* paper) fitted to published data with $k_1 < 1$ that is inversely related to maximal ingestion rate; ϵ is the fraction of lipid associated with nonlabile structure; m_{PS} is the portion of structure unavailable for use (nonlabile); A_3 and A_4 are maximum mobilization rates for stored lipid and structure; D is the energy demand of the individual from movement, maintenance, and feeding; and E is the energy available from the lipid and structure compartments. D, E, and A_2 can be expressed in terms of

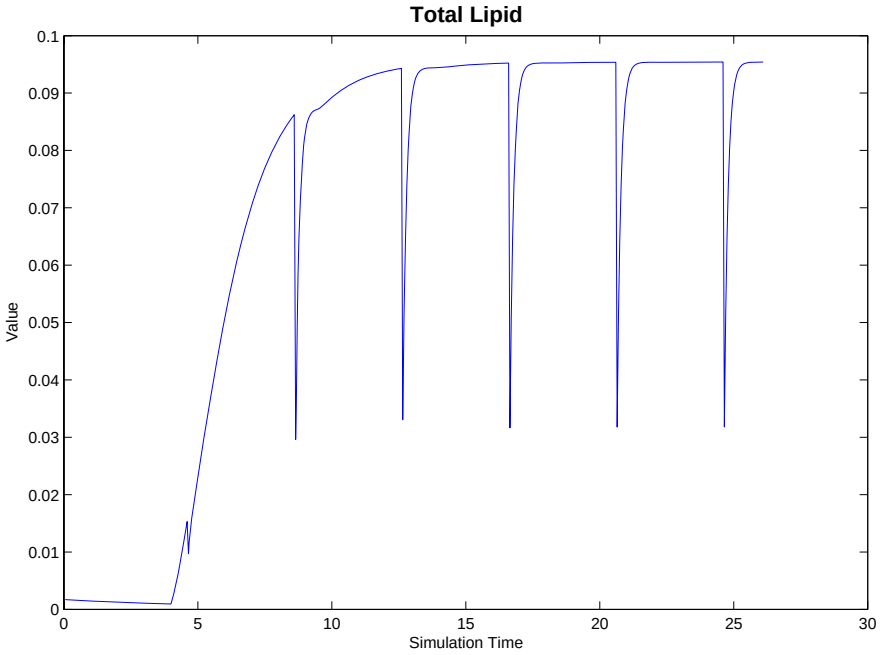


Figure 1.3: Individual Model Lipid Profile

m_L and m_S. These equations express the dynamics of m_L and m_S on a continuous time scale — allocation of m_L and m_S for reproduction and carapace formation are carried out at discrete times, but are not described here. Because of these bulk allocations for reproduction, our output functions for m_L and m_S are not continuous, but our expression for m_{PS} is continuous since it is taken to be a non-decreasing fraction of m_S. Others do model reproductive allocations continuously; see, Kooijman (1986). Energy requirements for movement for zooplankton are expressed using the work of Gerritsen (1984); it is noted that most energy for *Daphnia* is spent on reproduction with relatively little spent on movement. The units and values used for the parameters are given in Appendix A.

Typical outputs from the individual model are given in Figures 1.3 to 1.5. Notice the first birth time around 5 days and the initial, explosive growth that is followed by relatively small increases. The saturation of growth from the individual model will trace its way through all of the population models and will be an important factor in the predator-prey model because of its effects on the size-class distributions. Also notice the discrete allocations from stores for births and the non-decreasing nature of the protected structure.

The function G_x describes the uptake of lipid from resource and the function L_x describes the use of lipid and structure for energy and maintenance. As long as energy demand, D, does not exceed the available energy, E, then only the fraction required for energy is used; otherwise the maximum

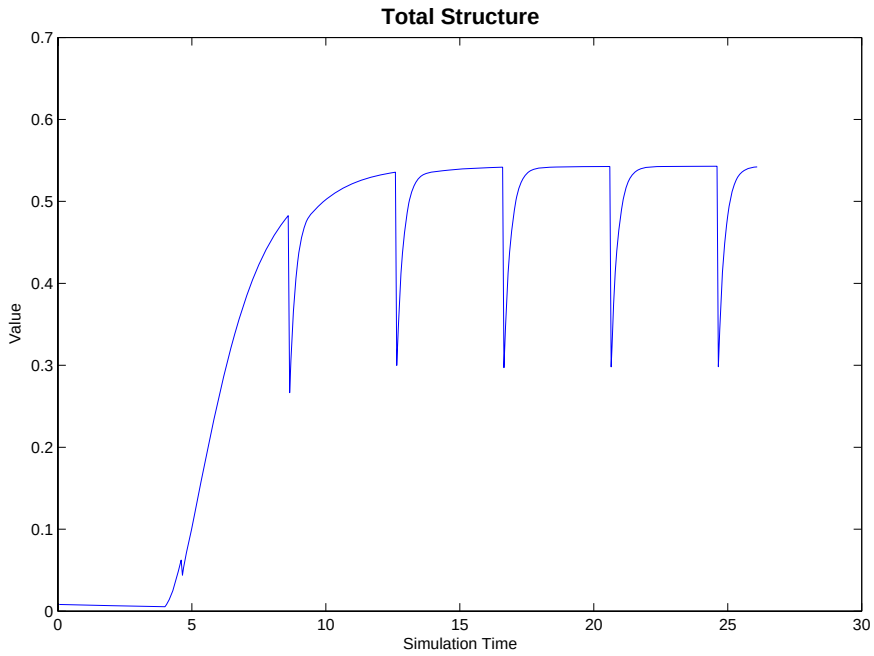


Figure 1.4: Individual Model Structure Profile

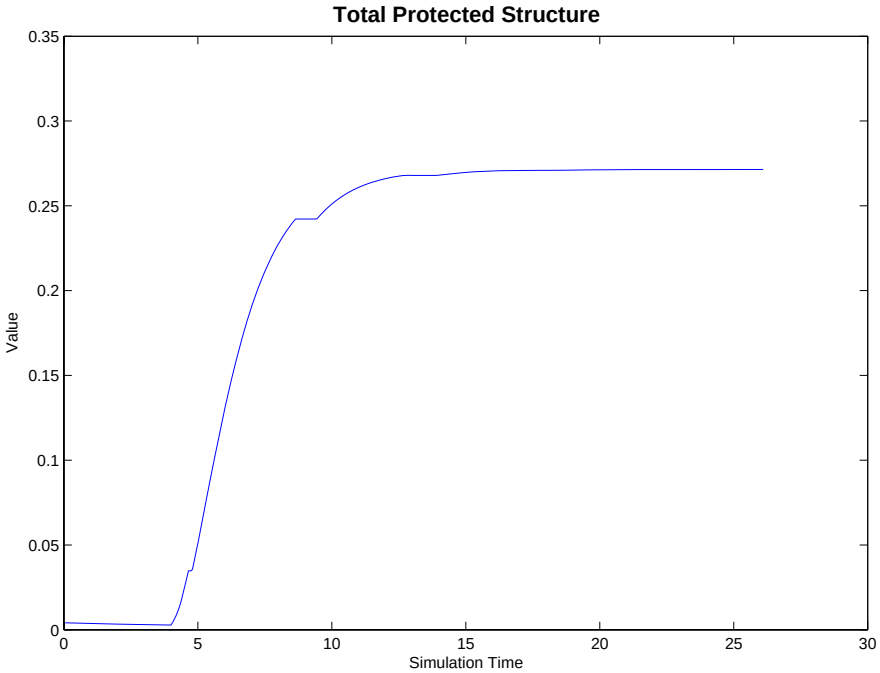


Figure 1.5: Individual Model Protected Structure Profile

mobilizable portion is used. This is the role of the switching function in the right-hand side of the equations. The change of A_2 to a function proportional to surface area rather than volume was made to prevent daphnids from growing arbitrarily large. It is clear from the equations, since the loss terms, L_x, are proportional to mass, then the feeding terms, G_x, must be proportional to mass to some power less than one at high resource levels or else they will grow without bound when presented with increasing resource levels (Hallam et al., 2000). The switching function's effect on the derivatives of these functions is carefully worked out in Appendix A.

A fundamental idea behind the development of this individual model for *Daphnia* is tracking energetics. The energy required for various life requirements underlie the equations. S.A.L.M. Kooijman is well-known for his development and application of Dynamic Energy Budget (DEB) models to populations of individuals across different trophic levels (Kooijman, 1986, 2000). In his book he explores many roles of storage materials and concludes that it is not possible to understand the dynamics of populations without a storage compartment at the individual level, just as we have incorporated into our models. The change of A_2 to an expression rather than a constant also bring our models into line with the derivations of Kooijman. This refactoring did cause a large change in the simulation programs because it affected several important derivatives. Further, with it in the denominator of the uptake term, several of the equations became more complicated because of the quotient rule. All of these are detailed in the Appendix.

The form of the uptake terms for lipid and structure moderates between different feeding constraints. It is a hyperbolic uptake expression in x. A canonical form is shown in Figure 1.6. Note that it crosses the x-axis at zero and that it has a horizontal asymptote which is approached for large resource levels. When the functional response value nears the horizontal asymptote, then we describe the organism's uptake as saturated. We also describe an organism's saturation level by the percentage of the horizontal asymptote attained. When resource is limited, then the filter rate is dominant; when resource is abundant, then maximal ingestion rate is dominant. The role of the *functional response* in the uptake term is to determine how much of the resource is consumed when the organism at a particular resource level. The uptake can be affected by filtering, ingestion and similar rates, the amount of time required to find prey, the satiation level, availability of preferred prey items, etc. See pages 130 forward in Kot (2001) for further discussion of the different potential response curves. Understanding the uptake process as a sequence of steps in the feeding process that have to be accomplished in sequence or parallel is an alternate way to derive the uptake terms. The chapter *Constraints to Feeding: A Mechanistic Approach* in Henson (1994) and published in Henson and Hallam (1995) are references to this alternate derivation. This process-oriented approach will

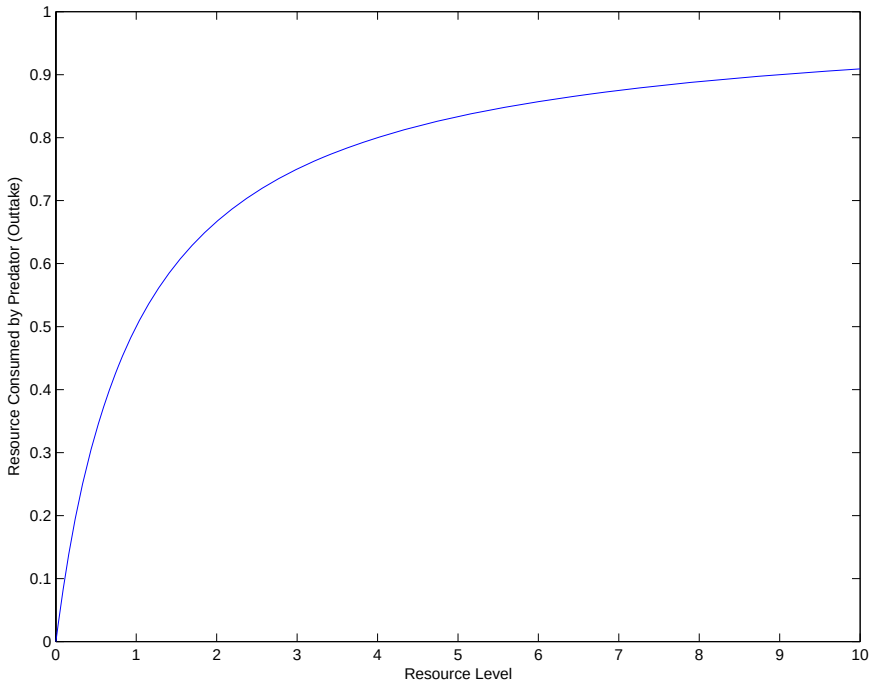


Figure 1.6: Typical Hyperbolic Functional Response

be especially helpful to understand the development and parallelization of the predation process in the predator-prey model.

Population Models

Individuals are incorporated into a population by a system of extended McKendrick-von Foerster partial differential equations which are described in this section. These equations give a mathematical framework for studying population effects proving results such as asymptotics. Having the individual model, it is not strictly necessary to incorporate a differential equation-based population model; individuals could be tracked through simple counting which is the case in modern *agent-based* models. A recent individual-based model from TIEM for bat populations follows in this vein (Federico, 2007). The use of a mathematically-based population model allows proofs and derivations to be made of observed model behaviors. A derivation from first principles of the McKendrick-von Foerster equation for age and size structured models, as well as its relationship with other standard models of populations, can be found in Sinko and Streifer (1967). An early application of this equation to a population of *Daphnia* can be found in Sinko and Streifer (1969). Work performed at TIEM on these equations include Li and Hallam (1988) and Henson (1994).

These equations are termed "extended" in the sense that each parameterization of the individual model is described its own McKendrick-von Foerster equation; i.e., populations built on different parameterizations of 1.1 and 1.3 result in a separate PDE. Let r_i be a vector of parameter values chosen from a fixed set $r_1, r_2, ..., r_n$. This vector is a set of parameter values fixed for the individual model. Each vector r_i is said to define an *ecotype*. For a fixed individual model, structured by r_i, its McKendrick-von Foerster equation is:

$$\rho_t + \rho_a + (g_L\rho)_{m_L} + (g_S\rho)_{m_S} = -\mu(t, a, m_L, m_S, \rho; r_i)\rho, \tag{1.5}$$

where $\rho(t, a, m_L, m_S; r_i)$ is a density function which represents the number (or density) per unit age per unit mass of lipid per unit mass of structure of individuals which are age a, and have masses m_L and m_S at time t. The functions g_L and g_S are the functions expressing the growth of the variables m_L and m_S with respect to time given by equations 1.1 and 1.3, respectively, with parameter vector r_i. The mortality function μ expresses the per capita rate of mortality for individuals of age a, and masses m_L and m_S; this function can also depend on the density. In general, we assume that $\mu = \mu_A + \mu_S + \mu_D$ where μ_A, μ_S, and μ_D are the age, size, and density dependent mortality rates. The density-dependent mortality function is described later in this chapter. Equation 1.5 requires

initial and boundary conditions in order to be well-posed. The initial population distribution is specified by

$$\rho(0, a, m_L, m_S) = \phi(a, m_L, m_S). \tag{1.6}$$

The method we used to generate initial populations will be discussed later in this chapter. The boundary condition is an expression for the newborn individuals:

$$\rho(t, 0, m_{L0}, m_{S0}) = \int_0^\infty \int_0^\infty \int_0^\infty \beta(t, a, m_{L0}, m_{S0}, m_L, m_S, \rho)\rho(t, a, m_L, m_S) da dm_L dm_S, \tag{1.7}$$

where m_{L0} and m_{S0} are the initial sizes of the newborn, determined from the parent, and β is the birth rate at time t by individuals of masses m_L and m_S for newborns of masses m_{L0} and m_{S0}. Note that β can depend also depend on ρ. Further note that for our models, m_{L0} varies over a fixed range $[m_{L0}^{min}, m_{L0}^{max}]$ determined from the parent's available lipid stores and based on the number of eggs to be produced; but the structure value m_{S0} is a fixed allocation per egg produced. (This variance comes up several times.) This boundary condition is sometimes referred to as the *birth* or *renewal equation*. Together equations 1.5, 1.6, and 1.7 form a well-posed model of a population of individuals whose growth is described by equations 1.1 and 1.3.

There being n ecotype vectors of parameters, then there are n distinct subpopulations. The parents are assumed to convey their ecotype to their offspring, so the populations do not intermix through the ecotypes. The subpopulations in the metapopulation are only coupled through the density-dependent mortality term μ_D. The parameter values r_i which we vary in the individual model for *Daphnia* are

1. Percent of lipid in the resource (PLX),
2. A1, a parameter inversely related to filtering rate, and
3. x, the resource level.

These parameters were found to be the most sensitive for the individual model; see Hallam et al. (1990b). Ecotypes introduce a type of diversity to the metapopulation. We vary these parameters within a specified percentage. We can vary them any odd number of levels. For most studies, each is varied 3 levels resulting in 27 ecotypes, but we can produce hundreds of ecotypes and thus magnify our workload to any level desired (at least initially, because density-dependent mortality drives out

ecotypic diversity over time). This fact is used to create problems of sufficient size to benefit from parallel computation.

As an aside: I repeated this sensitivity analysis in preparation for generating differently-sized prey populations for an experiment with multiple prey populations in the predator-prey model. I varied the four "sensitive" parameters: resource, A1, percent of lipid, and the gut clearance parameter k3. I found I could increase resource level by 100 times, which resulted in a weight increase of only 0.0014 mg. Varying A1 through orders of magnitude starting at $8 \times 10^{-}8$ resulted in decreases in size up to 0.5 mg after increasing A1 by 10^4. PLX was a reasonably sensitive parameter; varying over 10% from nominal of 15% yielded expected changes in lipid stores. K3 turns out to be the sensitive parameter. It sets the gut clearance rate which is the major determinant in how much food is processed by the individual daphnid. In the presense of sufficient resource, then it is the primary constraint on uptake.

Solution

This PDE system can be solved analytically in certain cases such as when the population is structured only by age; see Chapter 23 in Kot (2001) and Metz and Diekmann (1986). For age and size structured populations, like ours, analytical solutions are not possible. To produce solutions numerically, the method of characteristics is employed to reduce this PDE to a system of ODE's. The characteristic equations for equation 1.5 are

$$\begin{aligned}
\frac{dt}{ds} &= 1 && (1.8)\\
\frac{da}{ds} &= 1 && (1.9)\\
\frac{dm_L}{ds} &= g_L && (1.10)\\
\frac{dm_S}{ds} &= g_S && (1.11)\\
\frac{d\rho}{ds} &= -(\mu + (g_L)_{m_L} + (g_S)_{m_S})\rho. && (1.12)
\end{aligned}$$

The method of characteristics has the biological interpretation of tracing cohorts of identical individuals through their complete life cycle. Because of this interpretation, we will use the terms 'cohort' and 'characteristic' interchangeably. Although not precisely true — because each cohort represents ρ-many identical individuals — in our interpretation and visualizations we consider each cohort to represent an individual organism. The description of a new characteristic is defined by the renewal equation 1.7. This equation gives the initial value of ρ (the number of individuals represented by

the cohort). The initial values for m_L and m_S are determined from the parent's available resources. The remaining parameters represented by r_i are copied from the parent's values. The numerical simulation thus involves the management of a set of cohorts, which, individually are formed from the state variables, m_L, m_S, and a, a parameter vector, r_i, and several intermediate variables generated in the course of numerically solving the ODEs. An initial population for study is likewise given as a set of parameters and a set of cohorts specifying the population frozen at a fixed point in time. These form the initial cohorts used to start the population simulation and the parameters specify the values under which the population will be simulated.

Figure 1.7 shows typical output for the density function ρ for a fixed simulation time. The egg classes ($age < 4$) tend to dominate, so we often use log-lin graphs to see the other classes; see Figure 1.8 and note that there are actually older members of the population, but they are not apparent when compared in numbers to the egg classes. These figures are generated by summing the values of ρ across all characteristics at a particular simulation time for a set of size, age, and lipid classes. These classes are determined by taking equal-sized steps from zero up to a given, maximum value. For instance, take the maximum age, say 50 days. Now, fix a number of classes, say 100, and divide this by the maximum age. (The choice of 100 is nice, because it has easy interpretation as a percentage, but it is an arbitrary number. I use 1000 in some situations, because, with a time step of 0.05d, this guarantees that each cohort moves up an age class with each iteration of the model.) Then, in this case, there are two classes per day of age. If an individual characteristic had age = 20 days, then it would be in age class 40, or it has reached 40% of its maximal age. Similar interpretation can be applied using maximal lipid and structure values. The class interpretation makes calculating and recording output easier, because a common class value can be recorded once and the sum of the ρ for each age, lipid, and structure class can be given rather than recording age, lipid, and structure values separately. These graphs can be plotted through time in order to visualize the surface traced out by the population models over the course of simulation as shown in Figure 1.9. In Figure 1.10 the population density for lengths for population for the same simulation are shown. The oblique angle was chosen to emphasize how the lengths saturate and that there is a maximal size. Also notice that there are no daphnids in the smallest lengths because the minimal size is around 1 mm.

Generally, for equation 1.12, $(g_L)_{m_L}$ and $(g_S)_{m_S}$ could be sufficiently negative as to cause the $\mu + (g_L)_{m_L} + (g_S)_{m_S}$ to become negative, which would result in the equation describing growth in ρ rather than simple mortality. See (Metz and Diekmann, 1986, p. 15) for an interpretation of this in terms of an "elastic conveyor belt" that can contract and the mass class represented to stack up and

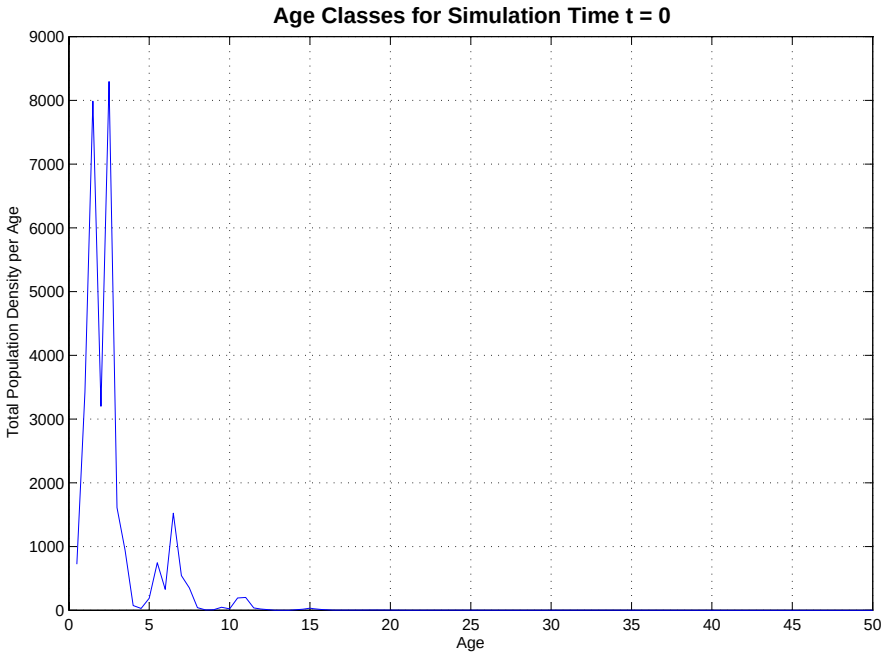


Figure 1.7: Graph of ρ Over Age Classes for Simulation Time=0

Figure 1.8: Log Graph of ρ Over Age Classes for Simulation Time=0

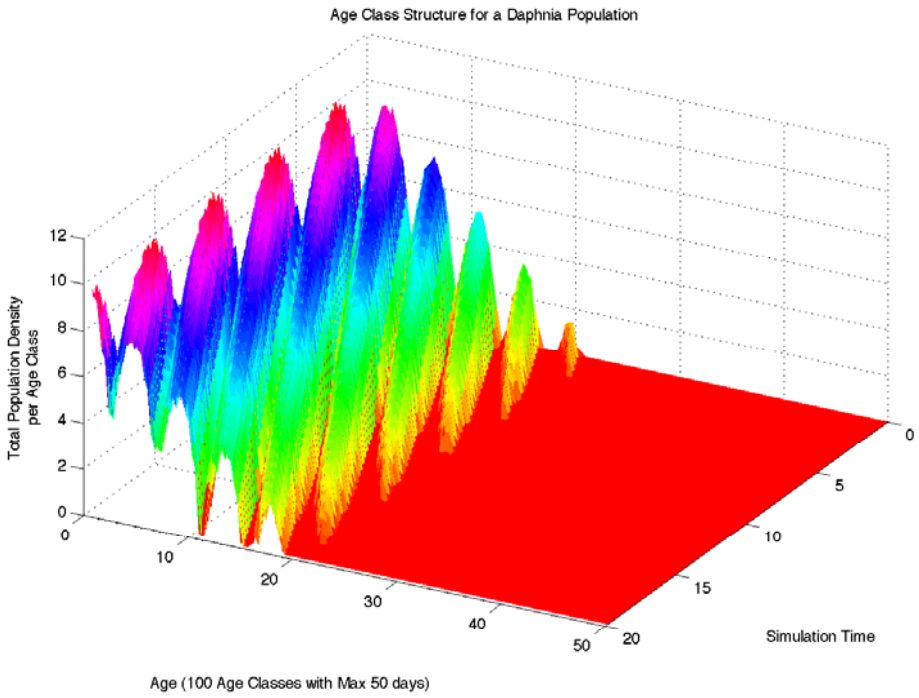


Figure 1.9: 3D-Plot of $\ln(\rho)$ Over Classes for First 20 Days of Simulation

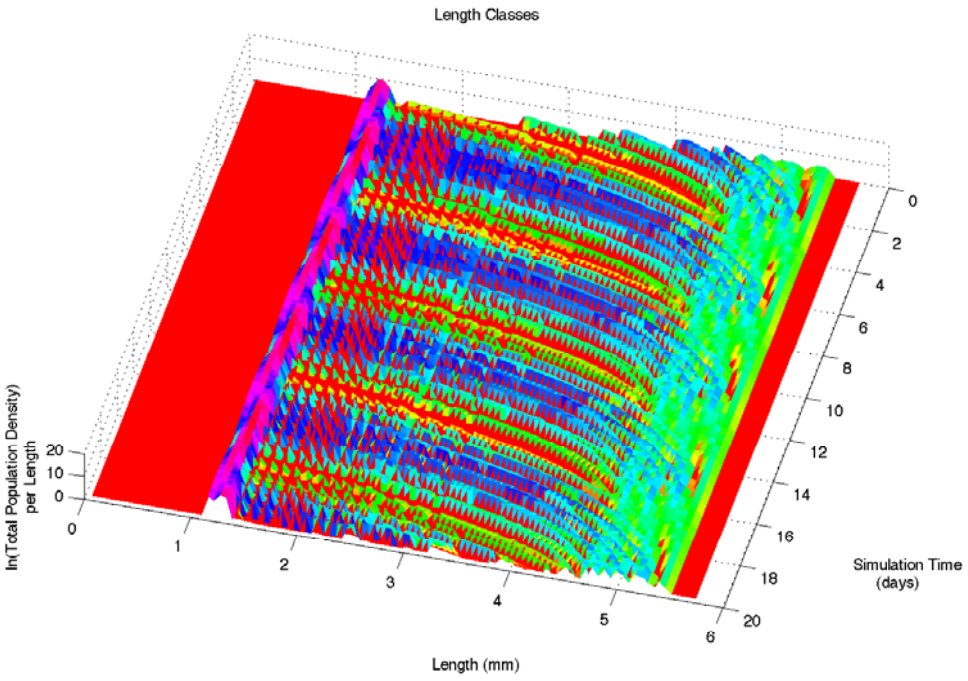


Figure 1.10: 3D-Plot of $\ln(\rho)$ Over Lengths for First 20 Days of Simulation. Oblique Angle to Emphasize Saturating in Length.

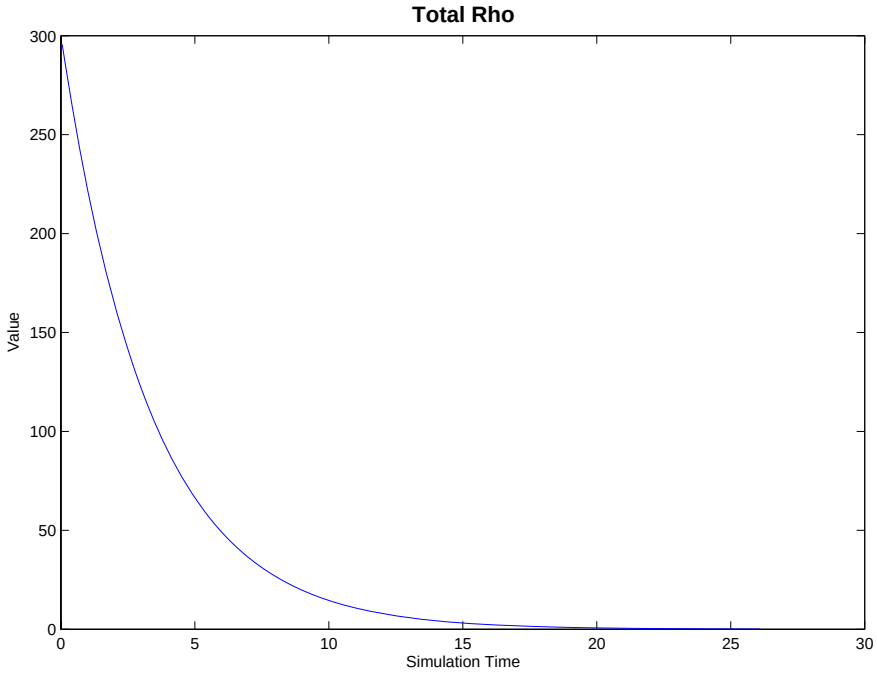


Figure 1.11: Profile for Population Density Function Along a Cohort

increase. Because we want to interpret this equation simply as representing mortality for a cohort, a transformation of ρ is performed in order to get a density function $n(t, a, m_L, m_S)$ such that only mortality is acting on the characteristics. The equation which replaces equation 1.12 is

$$\frac{dn}{dt} = -\mu(t, a, m_L, m_S, n)n. \tag{1.13}$$

See Hallam et al. (1992b) for an explanation of this transformation. This function now describes a decreasing exponential function with variable μ.

For a given cohort, equations 1.8 through 1.11 and 1.13 are simulated numerically by the population model. Equations 1.8 and 1.9 are trivial to solve. Equations 1.10 and 1.11 are solved simultaneously using a Runge-Kutta method described in Appendix A. These updated values of m_L and m_S are then used to evaluate the various components of the mortality function μ, and the final equation is solved. In Figure 1.11 the form of ρ is shown for an individual. The other individual model outputs were shown earlier. A flowchart of this simulation is given in Figure 1.12.

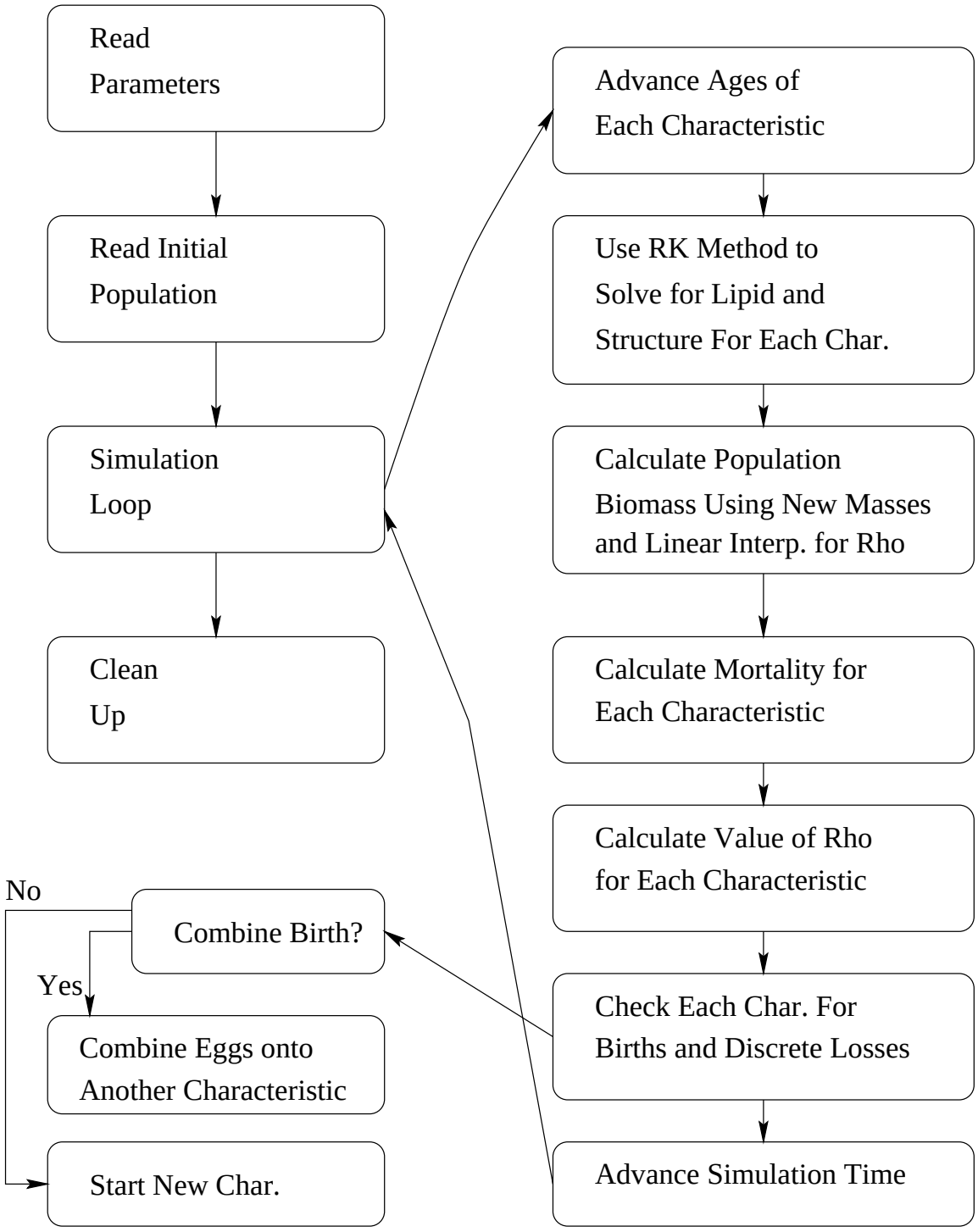


Figure 1.12: Flowchart of Simulation of Original Population Model

Mortality Assessment

The only direct control we have over a population is through the mortality function. The growth of lipid and structure follow the same path as all of the predecessors. This has the result that for constant resource, the physical growth will be identical for a particular age from cohort to cohort within an ecotype. As already mentioned, we assume a form for the mortality function of $\mu = \mu_A + \mu_S + \mu_D$ where μ_A, μ_S, and μ_D are the age, size, and density-dependent mortality rates. Another form of mortality imposed discontinuously is maximum age. Continuous age mortality is useful to tune out large changes in biomass due to large cohorts reaching maximum age and suddenly being removed from the population. Density-dependent mortality can serve many modeling functions such as crowding and competition for resource. For our *Daphnia* populations which have constant resource values, density-dependent mortality serves to limit the otherwise exponential growth of the population.

The main forms of mortality with which we will be concerned are density-dependent and maximum age mortalities. As also previously mentioned, density-dependent mortality couples the metapopulation models. It has the consequence of inducing the asymptotic dominance by a single ecotype which was initially observed by simulation studies, then proven in Henson (1994).

The form of the density-dependent mortality we use is shown in Figure 1.13. There is an optimal biomass, with a well around it. When above the optimal biomass, then the biomass of the population is driven downward through mortality. When below the optimal level, there is an increasing level of mortality imposed with decreasing biomass which almost certainly forces the population to extinction. The natural growth of the biomass for the population pushes upward, so the biomass levels for a population tend to oscillate inside the well above the optimal value. The lower threshold value is not normally exercised by any healthy population, but in our predator-prey dynamics studies this does have significant effect.

Short-term and long-term oscillations in biomass are an often studied and observed feature of natural and modeling studies which we will encounter later in this work. In combination with the minimum threshold for biomass, oscillations can cause the population biomass to dip just a little too far which leads to almost immediate collapse of the population. This is observed in our predator-prey models in Chapter 4.3.

Note that the optimal biomass can be used along with a value for density in order to determine an effective volume. Although the population models do not have a requirement for a volume, we will need to introduce control volumes for the populations when we get to the predator-prey model.

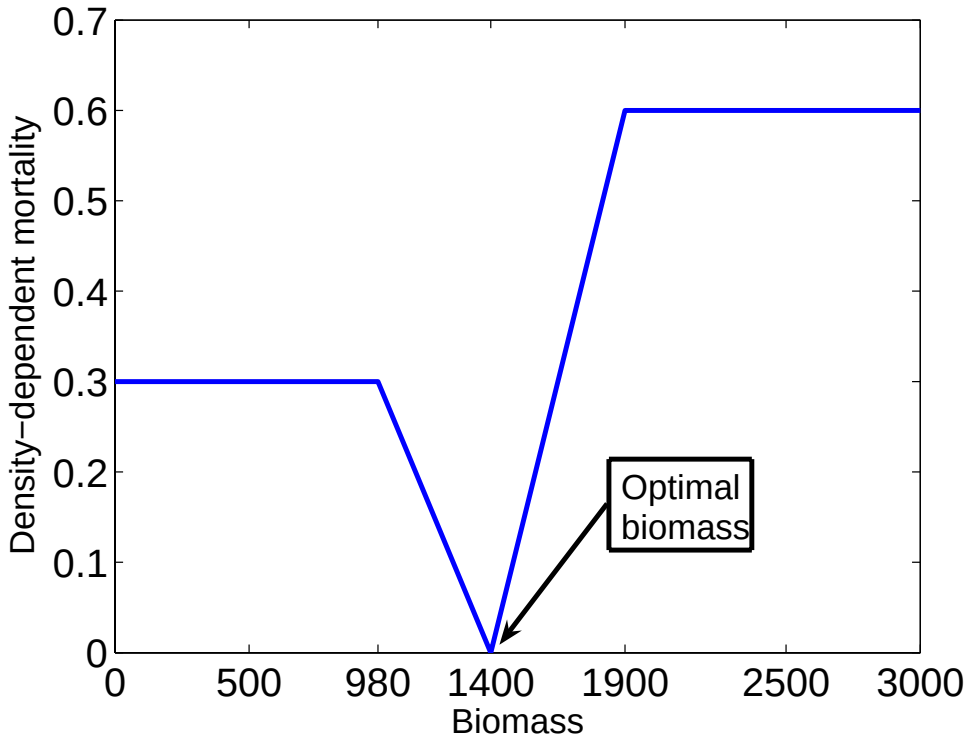


Figure 1.13: Density Dependent Mortality Graph

1.2.2 Individual-based Model for Fish Populations

The model for fish populations was also developed for ecological risk assessment purposes and follows similar lines of development to the *Daphnia* model. In fact, in the final code base, they share the same code except for the functions specific to the individual fish model. In particular, they share the submodels for testing the effects of lipophilic toxicants and they share the same cohort tracking population model. In this section the individual model for fish is presented, with emphasis on the aspects of the model that present significant challenges when developing and parallelizing the fish-*Daphnia* predator-prey model. The model we used is parameterized for rainbow trout (*Oncorhynchus mykiss*). The big differences in this model arise from size-structured resource. Size-structured resource for the fish is a requirement of our predator-prey models.

See Hallam et al. (2000, *Modeling Fish Population Dynamics)* and Henson (1994) for a complete description of the fish model from the individual level through to the population level including the parameters used. The *Fish Dynamics* paper is in the style of the *Ecology* paper for *Daphnia*, tracing the energy and mass flows throughout the individual. The energy flow diagram for fish is identical to Figure 1.2, with the exception of the flow into carapace formation. All of the functions and derivatives used in simulation codes are contained in the appendix.

For the fish we also model the partitioning into lipid and structure components. Using F to denote the feeding function, then the differential equations for the growth of the mass of lipid and structure are:

$$\frac{dm_L}{dt} = \frac{A_{0L}x_L}{x}F - \begin{cases} A_3(m_L - \epsilon m_{PS}) & for \quad D > E \\ A_3(m_L - \epsilon m_{PS})D/E & for \quad D \leq E \end{cases}, \tag{1.14}$$

$$\frac{dm_S}{dt} = \frac{A_{0S}x_S}{x}F - \begin{cases} A_4(m_S - m_{PS}) & for \quad D > E \\ A_4(m_S - m_{PS})D/E & for \quad D \leq E \end{cases}, \tag{1.15}$$

with parameter descriptions matching those already made for *Daphnia* with the exception that units of volume and mass are cm^3 and g respectively. Also, the energy requirements for movement are a large proportion of the energy requirements of the individual fish. Again, we apply the work of Gerritsen (1984) to express the energy requirements of movement. See the Appendix to this dissertation and the appendix to Hallam et al. (2000) where it is shown that the energy requirements for movement grow by the fourth power of length. Thus larger fish will have much greater energetic costs for movement. This has significance to us in our predator-prey models. Since the times spent encountering and pursuing prey will vary with the resource level, the energy requirements to support

these activities will vary in the predator-prey model. These critical times are now developed in terms of prey availability.

The derivation for the expression for the uptake function F is the difficult part of this model because of the structured resource. The role of F is to moderate consumption of resource; it is an expression of satiation level. See Henson and Hallam (1995) and Hallam et al. (2000) for complete derivations for our fish model; the first reference describes optimal feeding in general as a sequence of sequential and/or parallel steps in the feeding processes. For fish we consider three steps in the feeding process that take place sequentially: encounter, pursuit, and gut clearance. Let T_e, T_p, and T_g denote the times required to complete each of these steps, respectively, for a single food item. Thus the total time budget (Holling, 1959) for a single particle to pass through the system is $T_e + T_p + T_g$. Under the premise that all organisms feed optimally subject to environmental and biological constraints, then

$$F = \frac{1}{T_e + T_p + T_g} \tag{1.16}$$

$$= \frac{1}{R_e^{-1} + R_p^{-1} + R_g^{-1}} \tag{1.17}$$

where R_e, R_p, and R_g the optimal task rates associated with each step. The reader is also referred to Lassiter (1986) for another description of pursuit feeding (around equation (64) in his paper) on which the process-oriented feeding derivation is based and to Hart and Gill (1993) for a comparison of the application of foraging models to fish-*Daphnia* systems.

Because each of these rates involves values that depend on the prey, in anticipation of the predator-prey model, each of these rates is now carefully expressed. The dependence for each rate on values from the prey is summarized at the end of each of these short sections. They are also described in Appendix A in the context of the population codes.

Encounter Rate

The encounter rate is expressed as $R_e = a_d N_p$ (numbers day^{-1}) where N_p is the density of the prey (numbers volume^{-1}) and the encounter rate coefficient a_d (volume day^{-1}) is a function of the reaction distance of the fish, the velocities of the fish and prey, and represents the volume swept per unit time by the foraging fish. The encounter rate we used is developed in Gerritsen and Strickler (1977) and has units cm^3/d. The encounter rate is further used to produce a weighted average in the predator-prey model in order to equitably assess mortality back onto the individual prey cohorts that form a given fish's resource.

The expression we use for foraging fish is

$$a_d = \frac{\pi sd(m_S)^2 spd(vp^2 + 3vh(m_S)^2)}{3vh(m_S)} \tag{1.18}$$

where $sd(m_S)$ (cm) is the reaction distance for the fish; $spd = 86400$ (seconds per day); vp (cm/s) is the velocity of the prey item; and $vh(m_S)$ (cm/s) is the velocity of the fish while hunting, which is calculated by multiplying a constant body-lengths per second while hunting by the length of the fish which is determined from the structural mass m_s. The reaction distance of the fish (Breck and Gitter, 1983) is a function that depends on the lengths of the prey item and fish.

$$s_d = (a \cdot lp + b)\sqrt{lf(m_S)} \tag{1.19}$$

where a ($\text{cm}^{-0.5}$) and b ($\text{cm}^{0.5}$) are constants; lp (cm) is the length of the prey item; and $lf(m_S)$ (cm) is the length of the fish determined allometrically from m_{PS} (g).

When computing the population model for fish, most of these are constants. But, when we look at the predator-prey model, then N_p, vp, and lp for each prey item consumed will be determined from the actual prey items.

Pursuit Rate

The pursuit rate (numbers day^{-1}) describes the capture event after spotting a prey item.

$$R_p = \delta_v N_p / s_d \tag{1.20}$$

There is still a dependence on the prey items since $\delta_v = spd|vc(m_S) - vp|$ where $vc(m_S)$ is the (scalar) capture velocity of the fish, which generally is different than the encounter velocity. When computing the predator-prey model, the velocity of the prey vp will vary so δ_v must be calculated per prey item.

Gut Clearance Rate

The last rate concerns the time it takes to clear the gut of the consumed prey item. The rate

$$R_g = \frac{kM_g N_p}{M_p} \tag{1.21}$$

where $M_g = volgut(m_{PS})bdensp$, the mass capacity of the gut; $bdensp$ (g cm^{-3}) is the dry weight body density of prey; $volgut(m_{PS})$ is the volume of the gut which is related to the protected structure mass of the fish; M_p is the total mass of the prey item (lipid plus structure); and k is an expression for the gut emptying rate. k is not a constant but depends is the clearance time fitted to data $k = k_3 m_{PS}^{-k_1}$; see Hallam et al. (2000). k_3 is one of the ecotypic parameters for fish.

The dependence for this rate on the prey population is through $bdensp$ and mp. This means that we will have to calculate a mass density using each prey item consumed by the fish in the predator-prey model.

Putting these all for an expression for F one obtains Hallam et al. (2000, Equation (15)).

$$FM_p = \frac{N_p}{[a_d M_p]^{-1} + \left[\frac{s_d}{M_p \delta_v} + [kM_g]^{-1}\right] N_p} \tag{1.22}$$

The interpretation of F as one over the sum of the three process step times T_e, T_p, and T_g is the overall view taken in the code, because the energetic demands are also in terms of time. In other words, we must determined how much time is spent foraging, capturing, and clearing food items. Equation 1.22 can be seen to have a hyperbolic functional response structure, which is not apparent from the time-budget form.

Since the remaining description for the fish population model mirrors that already given for the *Daphnia*, we omit it. The derivation of each of the expressions used in the numerical calculations for the fish population model can be found in the Appendix.

Ecotype Parameters

The ecotypic parameter values r_i which we vary in the individual model for fish are

1. Percent of lipid in the resource (PLX),
2. $k3$, the parameter related to gut clearance rate, and
3. x, the resource level.

These parameters were found to be the most sensitive for the individual model. Since PLX and x will vary with the prey population in the predator-prey mode, only $k3$ will remain as a direct ecotype parameter. The size of the fish is used to determine a gape size through which the prey population will be filtered, so individual model effects will remain.

Young-of-Year Mortality

In recognition that the mortality rate is much higher for fish fry, an additional Young-of-Year (YOY) mortality is assessed for fish in a certain fixed age range. This mortality function is similar to density-dependent mortality in the sense that there is an optimal YOY number towards which the YOY are driven. The mortality is assessed based on population density, not biomass.

1.3 Parallelization Efforts Overview

1.3.1 Development History

When we originally started our parallelization efforts, the size of our computational problem was very large compared to the desktop and workstation-level computers available at the time. The computational programs in use prior to our project had been written in the Fortran-77 language. The execution of these models took a long time, especially the predator-prey models which took several days. Parallel computation was in its early days of design, development, and standardization. Could our models and similar ecological models benefit from these new technologies? Specifically we were looking to decrease the execution times. In Haefner (1992), an introduction is given to the state of the art for parallel computers at the time, with specific application to individual-based models. An alternative benefit of parallel computation is to allow more detailed simulations (hence larger) than those feasible on a single machine (Hwang, 1993).

Almost by definition, to say that we wanted to test our models in parallel execution, meant that we had to utilize supercomputers, because supercomputers were the only parallel machines available. An exception to this statement was PVM (Geist et al., 1994) which let one combine networks of workstations (NoWs) into a single computational cluster. Scott Sylvester, in his thesis (Sylvester, 1995), describes our use of NoWs in application to the *Daphnia* population models. The programming of supercomputers was unique to each machine and was a tedious process. Often libraries were provided that allowed programs written in PVM to be executed on supercomputers. Since then, the rise of grid-computing (joining individual supercomputers themselves into clusters) and the need for portability across all types of parallel computers has driven the development and near universal adoption of the Message-Passing Interface (MPI) as the method used to express parallel constructs. We have adopted MPI. The specification of MPI-2 is fairly recent (Gropp et al., 1998) and its adoption continues to present. We experimented with some MPI-2 specific features

and some advanced MPI constructs for the predator-prey parallelization, but otherwise only require standard constructs.

Much time has passed between when we started and when we are finishing these results (mainly because I became a computer consultant and IT Director based on the knowledge I gained in this project). The final results are being computed on my own personal, dual-processor, 8-core desktop computer containing two 3.0 GHz quad-core Intel X5365 CPUs which are incorporated by Apple into a machine running Mac OS X — a version of Unix. (Currently these CPUs are special-run and only available in this computer.) As our previous designs and results were reviewed for relevance, it was interesting to find that several of the same problems we observed before for parallel execution are still of significant concern. In some sense they are exacerbated by the advances in CPU cycle speed. (Compare the 133 Mhz CPU clock speed of our fastest machines at the time to the 3000 Mhz of a single core in my desktop computer.) Core memory has also significantly increased in size from 32 MB or less to 3 GB in my current desktop. At the desktop level, the 'MHz Race' — the move towards ever faster CPU clocks speeds — is over. There has not been significant speed increases for several years. The number of cores (computational units) is now being increased in order to continue to advance desktop speeds. In a sense we are being forced to transition from sequential to parallel computers in our own desktop machines. How to make use of this power in desktop applications is a cause of current concern; see for instance Sutter (2005). These new resources are not as readily available to programmers, so the speed increases they could count on have ceased for the time being and foreseeable future. How we can take advantage of these resources and apply them effectively to ecology models is the main thrust of the first part of this dissertation.

The size of supercomputers has also grown to tens and even hundreds of thousands of processors in the effort to solve Grand Challenge problems; see **top500.org** for a current list of the top supercomputers. The CPUs in these computers have become commoditized and are often the same CPUs in our desktop machines. The investment is in now in the interconnect network. The largest supercomputer on which we ran our codes originally was as the first public code run on a 256-processor Cray T3D at the National Center for Supercomputer Applications (NCSA). The execution of the computational core of the population model completed in about 5 seconds, but took 45 seconds to initialize and shut down and cost about 4 hours of CPU time. In an experiment on my current desktop, the same 500 day calculation took 5 seconds total on one processor and 6 seconds for two. Clearly our base problems that we started with are now too small to tell us anything interesting.

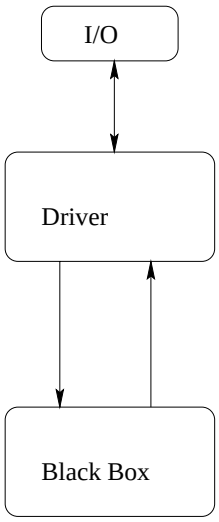
In this section, several concepts are introduced to which we will refer later. They summarize our decisions and solutions to the problems inherent to any parallelization efforts of these types of models.

1.3.2 Population Models Redesign

We begin our parallelization efforts with the population models, because the individual-based food web models are formed from coupling two population models together via a feeding mechanism; thus an understanding of how to parallelize the population models is necessary for the successful parallelization of the individual-based community models.

Looking at Figure 1.12, which shows the overall flow of our population models, one can see that there are many steps which can be performed in parallel; e.g., the update of the ages on each characteristic, the calculation of lipid and structure on each characteristic, etc. The only point where the characteristics are coupled is when the total population biomass needs to be calculated. Once the biomass is known, the calculation of mortality and all subsequent computations can continue in parallel. Each of these are examples of *data parallelism* — the same computations can be applied to different data independent of each other. Such a flow could be well-suited to a Single-Instruction/Multiple-Data (SIMD)-type parallel architecture (also called vector processing). The MasPar MP-2 was an example of such a machine to which we had access to at the start of this project; it had 256 execution units lock-stepped together. We tried unsuccessfully to map our models onto the MasPar. Such machines no longer are built as standalone computers, but several vector-units are typically built into our CPUs in order to accelerate repeated mathematical operations like those in matrix computation and image processing. Furthermore, the Graphics Processing Unit (GPU) in our video cards provides another such an example with dozens to hundreds of vector execution cores.

This flowchart analysis shows that the model is a good candidate for parallelization. But all standalone, parallel computers are now of the Multiple-Instruction, Multiple-Data (MIMD) design for which the parallel execution design focus is different. Even the lone, common example of SIMD execution, GPUs, are moving towards the MIMD-design with independent branching, conditional execution, etc., with firmware programming through shaders (Rost, 2006). For the MIMD computational model, the effort is to draw out *task parallelism*. Towards this end, we reorder the population model program into two parts. One is a computation portion which would perform all of the calculations necessary to advance a given characteristic one time step which we call the *blackbox*. The



1. Communication with other nodes.
2. Maintain list of characteristics.
3. Read and write data files
4. Maintain global information, such as simulation time, and population biomass.
5. Call blackbox to advance each characteristic.

1. Compute every step necessary to advance a characteristic: calculate lipid and structure; assess mortality; calculate density; check for births.

Figure 1.14: Diagram of Program Redesign

other is a driver portion which would concern itself with holding the list of characteristics in memory, calling the computation code for each characteristic, and performing any necessary communications with the other nodes. This is pictured in Figure 1.14.

Because of this design change we also anticipated several other benefits, including:

- The driver can be written in another language, more suited for maintaining lists, communications, and report generation.
- The blackbox can be optimized and simplified so that it can calculate as fast as possible.
- The design of more complicated programs, such as predator-prey or higher food webs, is simplified, because the computational portion for each species does not have to be changed at all; only the driver requires modification in order to link the different populations together. (This was found not to be entirely true; see Chapter 3.)

We have tried to capture each of these benefits into our serial and parallel versions of this model. We chose C as the language in which to implement the driver. While C++ now would be the natural choice for the base language, C++ support and its runtime environment requirements were not universally available, especially on supercomputers. Likewise, while extensions had been made to the Fortran language (Metcalf and Reid, 1993) that allow list management, etc., these were not readily available.

The overall flow of the new program is changed from the original. Figure 1.15 illustrates the flow of the redesigned program. The only algorithm change was the movement of the computation of the population total biomass from the middle of the simulation loop to outside of the loop. Total biomass is required for the computation of the density-dependent mortality effects. Without this, the mathematical equations would be decoupled. Moving the biomass calculation outside of the computational loop proved convenient for two reasons. One is that the calculation of biomass is located in the driver portion which has access to the entire local population, and can communicate with the other nodes as necessary to compute the total population biomass. The other is that it removes the only synchronization point of the algorithm from the middle of the computation loop.

The recoding of the various components of the *Daphnia* population model to conform to the black box model was performed by myself. The *Daphnia* population model translated is the one modified by H. L. Lee to include the effects of temperature and dissolved oxygen on the *Daphnia* populations (Koh et al., 1997); which is itself an extension of the modified model by G. Canziani which includes sublethal effects of toxicants (Hallam et al., 1993).

Our first translation of the population models had the driver portion in C with the computational routines in F77. We then implemented parallel versions of the program on the Thinking Machines CM-5 (32 nodes, 4.0 GFlops peak), and the Intel iPSC/860 (128 nodes, 5.1 GFlops peak). (These numbers and descriptions were kept for comparison.) The desktop computer on which we ran the final models is capable of approximately 90 GFlops using its eight processors. We also ran the population and predator-prey codes on several other supercomputers including the Cray T3D using MPI. These times are now matched by my desktop. As such, most of these timings have been removed except for when they motivate an idea more clearly than new runs would.

There were a number of versions of the codes which, in their final versions, have been combined into a single code base. The final code base is written entirely in C including the black box portion which comprised a vast majority of the code base and was the least understood. We tried to avoid translating the blackbox, but to gain optimizations and access to better tools it had to be translated. The final code base uses MPI for parallel message passing. The code base between the population and predator-prey models shares the same routines except for the predation routines. We also unified the sequential and parallel codebases, so either version can be generated. This eliminates the concern and problem we had being certain that the same versions of the routines are executed. This also greatly simplifies maintenance in that updates to one codebase are easily applied to the other; I found this useful several times, because I only had to solve problems and fix common bugs once. We have retained all of the versions including the originals in a Concurrent Version System (CVS)

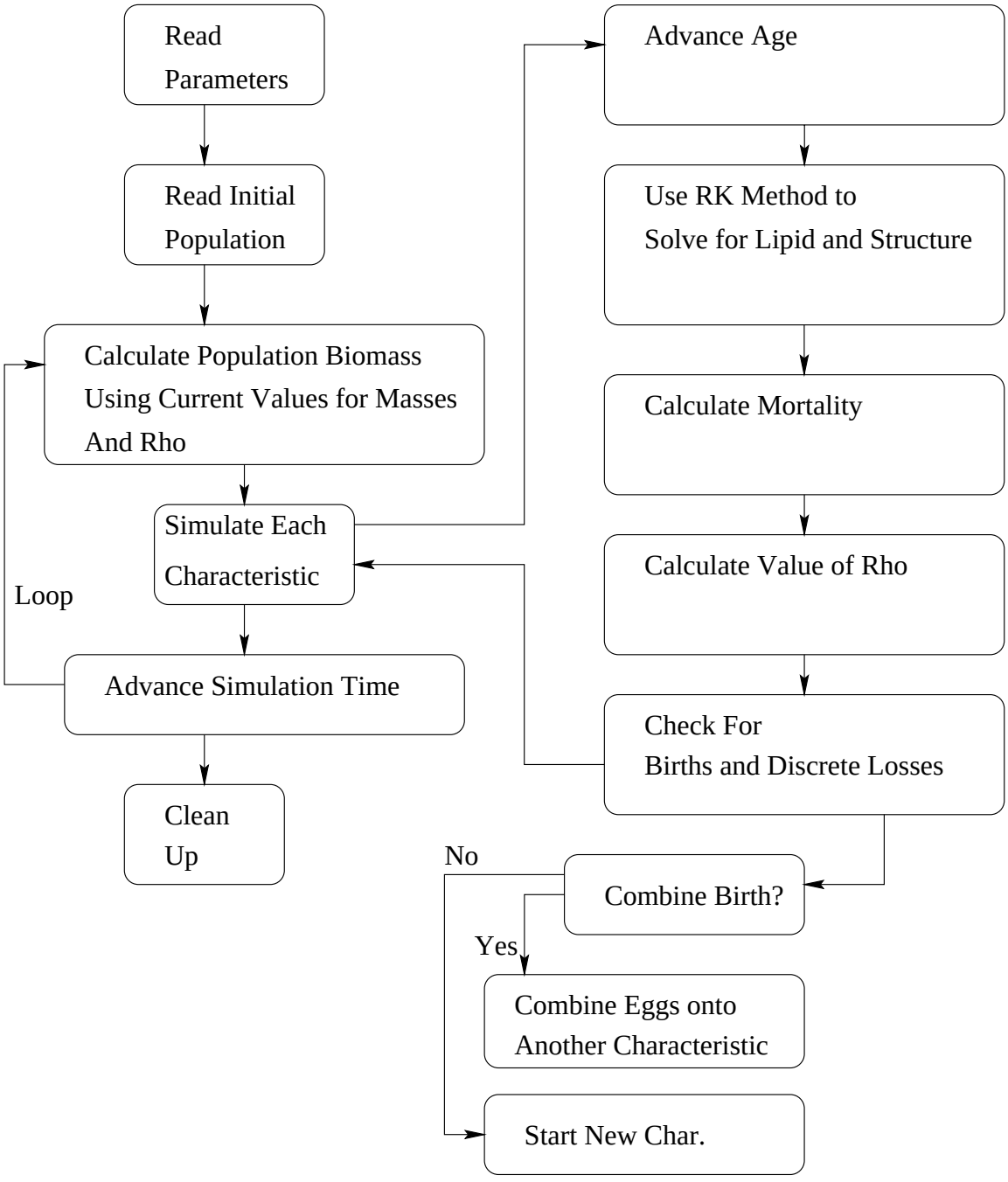


Figure 1.15: Flowchart of New Population Model

archive which has been useful several times. For instance, I used it to pull the original predator-prey program and initial populations used by Henson for Extinction Thresholds discussed in Chapter 4.1.

While minor, an addition made early in development deserves mention. As we translated from Fortran's array-based indexing, we found that an efficient system for managing ID numbers was required. We created our own system based on the balanced-tree algorithm (Aho et al., 1985). With FORTRAN, array index position was used to maintain parent-offspring relationships. These relationships have to be maintained, stored into output files, and restored for *Daphnia*, because the parent carries her brood in a brood pouch until they are released. This becomes especially important for predator-prey systems where predation mortality assessed on the parent also results in mortality on her brood. During program execution these relationships are maintained through pointers, but for initial populations and output data files we needed a management system. It also let us grow the problems to machine limits without concern for running out of space and made our predation visualization in Chapter 4 possible.

1.3.3 Cohort (Birth) Combining

In the pure mathematical expression of characteristics, the renewal equation 1.7 specifies that a new characteristic be generated and tracked for any difference in m_{L0} or m_{S0} no matter how small. Since our initial values for m_{L0} and m_{S0} vary according to each parent's resources, we would quickly generate a geometrically-growing number of characteristics that would overwhelm any computer. This is neither realistic nor feasible. We therefore *combine births* in the sense that if two parents belonging to the same ecotype give birth at the same time step, then the value of ρ taken for the combined characteristic is the sum of the two newborn characteristics. The parameter vector r_i will match because they are of the same ecotype. The value for m_{L0} is the only variable that is not conserved. Birth combining will be revisited in several times throughout this thesis.

1.3.4 Node Rebalancing

In our models, since there is hardly any difference in execution time for different cohorts, and because all cohorts must be advanced each time step, we calculate our work requirements in terms of number of cohorts. As our parallel algorithms execute, different nodes accumulate different levels of work. This occurs because of deaths or births that occur on the node. As the nodes become out of balance, then inefficiencies increase as some nodes have to wait at a synchronization point while other nodes complete their tasks.

A rebalancing algorithm for efficiently balancing the number of characteristics on each processor is required for efficient, scalable execution of the model on parallel computers. We did not consider this in our original ideas for the parallelization of the population model, but we found it to be a fundamental requirement for efficient use of parallel resources. Even ten years later, managing the balance of workload remains a fundamental problem for any parallel effort.

The rebalancing algorithm implemented in our parallel codes takes a fixed rebalancing period as a parameter; denote it by R. Every time the simulation time advances R steps, then the rebalancing routine is called. This routine is executed on each of the processing nodes simultaneously. The following steps compose this algorithm which is more carefully described in Chapter 2. A particular challenge with any such parallel algorithm when working with MPI is exactly pairing SENDS and RECEIVES. If they do not match, then parallel execution will deadlock.

For a simulation with n-nodes.

1. Each node computes how many characteristics it has and stores this value into its respective position of a vector of length n. This vector is globally merged so that each processor knows how much characteristics every other processor has. Each processor can now compute the same average and also compute how much each processor differs from the average.

2. Each processor computes the left-most (or lowest numbered) node which is a sink (lower than average) and the left-most that is a source (higher than average).

3. If the sink processor number equals the local node number, then it sets up to receive additional characteristics. Likewise, if the source processor number equals the local node number, then it sends a number of characteristics to the leftmost sink. The number of characteristics sent or received is chosen so that either the source or the sink will have exactly the average number of characteristics after the transaction is completed. If the local node number is not equal to the source or sink, then it updates its local vector to reflect the changes, but otherwise it does nothing.

4. Steps 2 and 3 are repeated until all nodes are balanced. Only one global communication is necessary; all other communications are node-to-node.

An example for a four processor computer is pictured in Figure 1.16. The entire algorithm is described precisely in Chapter 2.

For supercomputers, and multi-core machines it can be assumed that each node in a simulation is equally capable and completely available to the program (homogeneous-case). For a network of

Number of Characteristics per Node Prior to Rebalancing:

5	15	9	7

Average equals 9.

Number over or under average:

-4	6	0	-2

Node 0 = Left-most sink.

Node 1 = Left-most source.

Node 1 sends 4 characteristics to node 0.

0	2	0	-2

Node 3 = Left-most sink.

Node 1 = Left-most source.

Node 1 sends 2 characteristics to node 0.

Figure 1.16: Illustration of The Rebalancing Algorithm for Four Nodes

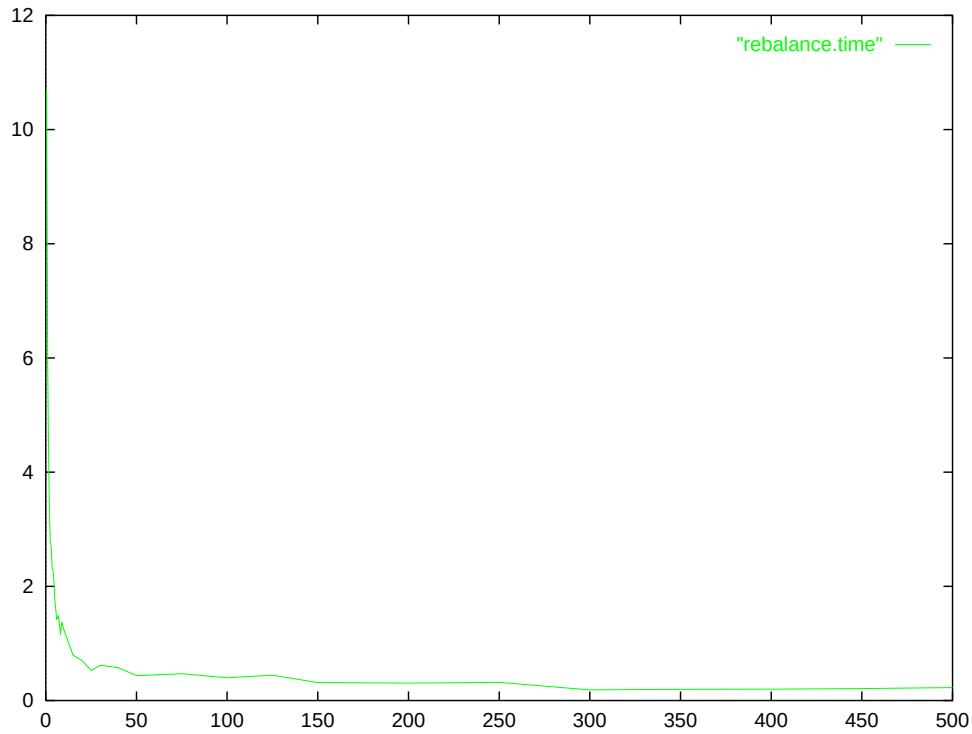


Figure 1.17: Typical Graph of Total Rebalance Time with respect to Various Rebalancing Periods

workstations (NoWs) this is not the case. Different nodes can have different levels of load and can have different processing speeds (heterogeneous-case). Sylvester in his thesis (Sylvester, 1995) addresses this problem. It amounts to exchanging the total computation time required by each node to advance its local population. This information is used to scale the rebalancing process appropriately so that each node is balanced in terms of execution time. A NoWs also has the problem of network contention which is also addressed in this thesis. We did not incorporate the scaled load balancing into our MPI code because we assume uniform capabilities of each node. This analysis utilizes graphs obtained from older machines, because current machines mask the effects of imbalance. One of the adaptive rebalance methodologies tested is analogous in design to rebalancing for the heterogeneous-case.

A typical graph of the total time required for rebalancing is pictured in Figure 1.17. It has a hyperbolic shape because there are twice as many balancing steps when balancing every timestep, as opposed to every two timesteps; and twice as many steps when balancing every two timesteps, as opposed to every four, etc.

As an illustration of the benefit of rebalancing for various periods, consider Figure 1.18 which shows the execution times of the same simulation run with varying rebalance times. This graph was retained from our original data sets, because modern CPUs almost completely mask the effects of

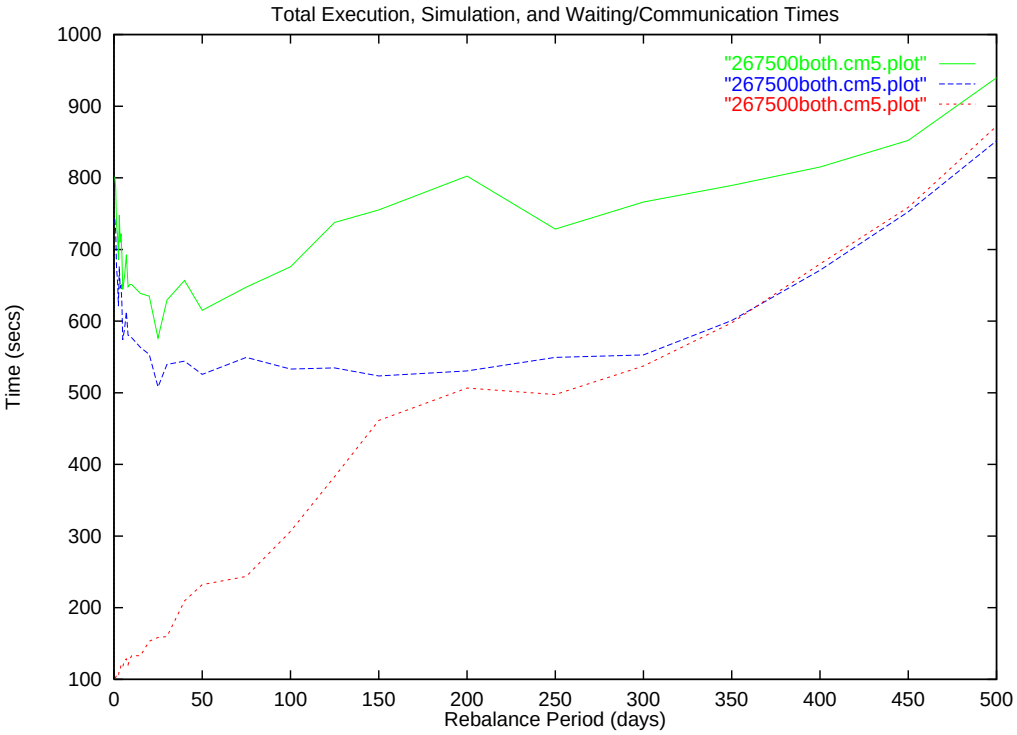


Figure 1.18: Execution Time on CM-5 for 500 days for DAPH267 Initial Population with Environmental Effects and Toxicant

imbalance. The initial population is distributed evenly to each of the nodes. The top curve shows that there is a significant decrease in execution time when the rebalancing period is chosen properly. The next curve is the maximum amount of time any one processor spent in computation. The final curve illustrates the maximum amount of time any one processor spent in communication waiting at the synchronization point for the population biomass calculation. The two bottom curves do not sum to the top curve because they are times from two different processors. The large amount of time spent waiting by one node as the rebalancing period increases — actually surpassing the maximum computation time at some point — illustrates the effect of imbalance. The final rebalance time essentially shows how the program would behave if no balancing were performed.

One of our original ideas for parallelization was to have one ecotype per node. The reason that we planned to do it this way was in order to minimize the amount of communication that had to be performed in order to combine births. If characteristics of the same ecotype are on several different nodes, then, in order to combine their births together exactly like in the serial code, a lot of communication would be required. We quickly moved away from one ecotype per node paradigm for two reasons. The first reason is that because of density-dependent effects one ecotype will dominate the population after some period of time. This effect was first reported as an observation in Hallam et al. (1990a). Henson in her dissertation (Henson, 1994) proved the asymptotic dominance of an ecotype for models of our type. With this in mind, we knew beforehand that with only one ecotype per node, then one processor would eventually end up doing a majority of the work, while the other nodes remain idle. The second reason we moved away from one ecotype per node is that it failed to use the full power of the parallel computer. For example, if there were only nine ecotypes, then only nine processors would be used, even if there were 100 available.

The method we have chosen is to distribute the initial characteristics evenly among all of the available nodes, and then use load balancing to keep the numbers per node even. This eventually distributes the dominant ecotype evenly upon all of the nodes. In order to simplify the restoration of the parent-offspring relationships for *Daphnia* we initially distribute the populations by ecotype.

These same issues come up again when designing the parallel predator-prey models. Since the eggs are carried in a brood pouch, the mortality of the parent also applies to the offspring cohort. If the offspring cohort is located on a separate node from the parent, then notifications would be necessary to properly update the offspring. This is addressed in Chapter 3.

1.3.5 Optimal Rebalance Strategy

I retained the old Figure 1.18 to show how the choice of the rebalance period had a significant effect in the execution time of the algorithm on the old computers. The benefit of keeping the nodes in balance was so high, and the cost (in terms of execution time) of rebalancing so low, that we eventually decided to just rebalance every timestep. This eliminated the need to come up with some choice for the rebalance period. With modern CPUs we have a similar problem of choosing a good period for rebalancing for an opposite reason. If we rebalance too often, then we spend more time in global communications keeping the workload balanced, than we would have spent just computing the few extra characteristics. Rebalancing too often now can cause much higher execution times because the computational abilities of the nodes so exceeds the communication bandwidth.

An interesting occurrence though is what happens when the workload exceeds CPU or core cache size (typically 8 MB shared between two cores). This forces a much slower trip to main memory to retrieve data for a cohort, which leaves the CPU idle while waiting. This is further exacerbated by there being only one shared path to memory, so if several of the cores are faulting to main memory, then the access times are even slower. We demonstrate superscalar speed-up for our models in this thesis; i.e., execution times reduced by a greater factor than the number of processors added. Such effects are caused by caching. If one core, because of imbalance starts faulting to main memory, then all of the cores are idled waiting for the one at the next synchronization point. We use the load balancing to add processors to large calculations in order dynamically expand parallel resources in order to maintain the super-scalar speedup achieved.

Much analysis went into trying to mathematically determine an optimal rebalancing period *a priori*. The effect that was determined to be dominant was that of birth combining. This led to the concept of birth classes described in the next section in order to be able to compute an estimate of the probability of births combining. With this information, the design was that the work and waiting/communication costs generated by a run using only one initial characteristic of the dominant ecotype could be scaled in order to yield an estimate for the optimal rebalancing period for a more diverse population.

1.3.6 Birth Classes

When we first started examining the likelihood of births being combined, in the search for good, *a priori* rebalancing period choice, we assumed that birth events will even out and be equally likely at any particular simulation time interval. But further investigation revealed that this is not the

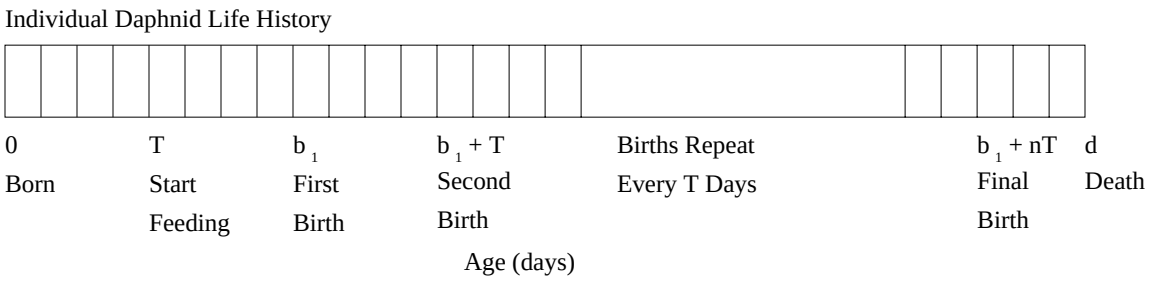


Figure 1.19: Illustration of a Daphnid's Life History

case. Consider the life history of an individual daphnid of some fixed ecotype in the absence of any mortality other than that of old age; this is illustrated in Figure 1.19. The daphnid is deposited as an egg at age zero, begins to consume food and grow when it is released at age equal to the fixed reproductive period, T, reaches the size necessary to reproduce at age b_1, and continues to reproduce (assuming sufficient growth) on a fixed cycle with period T until it reaches the maximum age. In our model T equals 4 days, but could be chosen to be any reasonable value. The important thing is that it is fixed. In the absence of mortality other than age, all characteristics of a fixed ecotype will follow this same cycle. Density-dependent mortality affects the age of death of characteristics; but the age of first birth for an ecotype is a constant since the resource level experienced by the daphnids is fixed over the course of the entire simulation. The value b_1 is fixed for an ecotype r_i unless other effects like sublethal effects or environmental effects cause the growth of the individuals to be advanced or retarded over time. The age at first birth is one of only a few *size-dependent* life history events in our models. Such events become more significant in the predator-prey model. As shown in Figure 1.5, birth events cause temporary pauses in the change in length of the daphnid, thus potentially causing them to spend more time in a prey-size window.

This shows why birth combining is such a successful technique with these types of models. Consider a population initialized by one characteristic of ecotype r initialized as an egg at time zero. Suppose for this ecotype, the age at first birth is b_1, the reproductive period is fixed at T days, and only maximum age mortality is assessed. Maximum age limits each characteristic to reproduce at only a fixed number of times, say n. This first characteristic reproduces at the following simulation times:

$$b_1, b_1 + T, b_1 + 2T, \ldots, b_1 + nT.$$

The offspring characteristic generated at age b_1 will start reproducing b_1 days later, and will reproduce at the following times:

$$2b_1, 2b_1 + T, 2b_1 + 2T, \ldots, 2b_1 + nT.$$

Finally, the set of times when the characteristic, which was generated at age $b_1 + T$ by the initial characteristic, will reproduce is:

$$2b_1 + T, 2b_1 + 2T, 2b_1 + 3T, \ldots, 2b_1 + (n+1)T.$$

Thus by the third generation from the first characteristic, there is a large amount of overlap in the births; and this overlapping of generations continues through the whole simulation.

One can use these sets to define the *Birth Class* of a characteristic for a fixed simulation step size. Let dt denote the time step used in the simulation. If one takes each birth time from the first set modulo T, then one will always get $b_1 \bmod T$. Let B the integer from 0 to $\frac{T}{dt} - 1$ (T/dt is assumed to be an integer), defined such that $b_1 \bmod T \in [Bdt, (B+1)dt)$; B is defined to be the *birth class* of this characteristic. The second and third sets both belong to the birth class C where $2b_1 \bmod T \in [Cdt, (C+1)dt)$. It is apparent that this definition is very dependent on the value of the time step. There is a total of T/dt possible birth classes. If one looks at the times of the birth events modulo T, then one will find that the characteristics in a birth class will always give birth at the same time modulo T. Analysis of the asymptotic dynamic behavior of the model was examined by Funasaki in his dissertation. In Funasaki (1997, Chapter 5), he used these concepts of birth classes as predictors for ultimate dynamical outcomes. For instance, when he runs the model for long periods of time and for various levels of toxicant in the water, cw, then he has found various dynamical behaviors exhibited by the model. For example, for $cw = 0.0$, $dt = 1/20$, and $T = 4$ days, he gets a five point attractor as his ultimate dynamic behavior. For this case, there are sixteen classes that contain birth events; the remaining classes are empty. Dividing the 80 possible classes by 16 yields five — the same number of attracting points. This generally holds true, because the population can be in only a fixed number of states between new cohorts being added.

This same sort of analysis also shows why we have had such a problem comparing models with different step sizes. One is generally inclined to think that decreasing the step size should make the model more accurate in some sense. Thus, if one had a five point attractor when the step size was equal to twenty steps per day, then one would expect that by decreasing the step size, then one

should be able to get a better estimate of the values of the attractor. This is not the case, because if dt is decreased to 40 steps per day, then one gets a quite different behavior from the model, aperiodic in fact.

Birth classes are an additional source of oscillations in these types of models. Enserink (a former student of Kooijman) in her dissertation carefully compared laboratory populations of *Daphnia* with predictions from DEB models (Enserink, 1995). She found that the density oscillations predicted were more pronounced than actual oscillations. She attributed this to the "convergence of body sizes and synchronization of life cycles, being the main cause of unrealistic oscillations in population simulations." The periodic birth assumption of our model is one such source of synchronization. But, because of birth combining, the periodic decision is a requirement, because the offspring cohorts are disassociated from their parents. Events on an individual parent that may advance or retard the release of its brood cannot be easily be modeled. We introduce later in this thesis a concept called Brood Pouch that was found to be a solution for a similar problem introduced by parallel execution of predator-prey models. This concept could be used to model delayed brood release or similar individual parent effects on her brood if required by the study.

Another way that Birth Classes are used in this thesis was as a method to increase the work requirements. By changing how the initial populations are generated, then populations can be created that have cohorts in each potential birth class. By populating all possible birth classes, we are in a sense creating several non-overlapping populations. This effectively lets us increase the amount of workload by some factor. This was a non-trivial method I utilized to to increase the simulation workload in order to study different parallelization designs, algorithms, and machines.

1.4 Creating Populations - "Egg-hatching"

A necessary process when studying these population models is that of creating new populations "from scratch." We call this process *egg-hatching* because both of our populations hatch from eggs. There used to be a separate code that carried out the process, but, because it was separate, then it was difficult to guarantee that the populations generated by the program as initial populations for the general population and predator-prey models had experienced the same growth and mortality calculations as they would experience in the general models. Thus starting populations were difficult to generate, so starting populations tended to be taken as given values, which is apparent in our use of "pop267" throughout one of our research projects.

The process of generating a new, initial population is:

1. Generate a complete file of defining parameters including the number of varying levels for each ecotypic parameter. These values can be retrieved from published works or from the initial population files. Any initial population defined in the file can be deleted, because it will be replaced.

2. Set the egg-hatching cycles to 1 or greater.

3. Run with the standard, sequential population model with this as an initial population for some period of time. The standard model knows how to generate populations from eggs. At the completion of this run, the output restart file will be the newly-generated initial population. Typically, Daphnia populations are only run out to 50 days. Fish are run several years. Both periods of time are chosen in order to get a certain number of birth cycles.

The process of starting a population from scratch can be summarized as make a bunch of eggs of the right type and let them go. A few complications like generating the ecotype information for each type of egg are necessary to this process. In order to guarantee that initial populations were compatible, we combined this process into the population model, so that it could also be used as an egg-hatching system to generate initial populations. (Egg-hatching is not defined for the parallel versions of the code and will cause the program to abort.) This capability to easily generate new initial populations helped when we found that we needed to change the cohort combining process for the predator-prey model.

In order to create populations consisting of many more characteristics with which we could test our computers in "overload", we complicated the populations via additional ecotypes. Initially this will multiply the number of characteristics that must be modelled until competitive exclusion eventually drives out diversity. An additional way that we could multiply the populations was through birth classes. We tested some populations with up to 729 ecotypes and all birth classes filled with over 50,000 characteristics.

An minor item that is necessitated by starting populations this way is that they possess a certain "age" which is the simulation time at which they are recorded as populations to be used as initial populations for the population models. The population's timed cycles such as births are centered around this time. A TRUEAGE parameter is embedded in order to allow the populations to be recentered at arbitrary start times for new simulations.

1.5 Chapter Summary

In this introductory chapter we described the mathematical models on which we based our development efforts. Of particular significance for later developments are the rapid initial growth exhibited by the juvenile classes once they begin feeding and the approach towards a maximum size as the organisms age. A maximum size is not assumed, but is a consequence of the functional response which describes the uptake rate for an individual as a function of the resource level. The method of characteristics used to solve the hyperbolic McKendrick-von Foerster population equations directs us towards tracking cohorts through their life cycles. The calculation of the boundary condition describing births will arise several times since births pose a parallelization challenge that is like *Daphnia*, because the workload for local birth combining scales directly with the number of parallel processes. to individual-based ecology models. Maximum age mortality assessed at the cohort-level and density-dependent mortality assessed at the population level are the two main types of mortality to which we will refer in the remaining chapters.

The cohort solution structure is what we mainly exploit for our parallel and analysis efforts, but the role of the individual model and individual physiology described in this chapter is not completely suppressed and had to be considered throughout our work.

A few subjects that arose during our parallelization efforts, such as Birth Classes, Initial Population Generation, and Optimal Rebalancing Period are described in this chapter. They play subordinate roles in our work, so they appear only in passing reference later in this dissertation. The development of a method for dynamic determination of the Optimal Rebalance Period was completed based on timing calculations which we added during the course of evaluating performance of our designs. The method was not implemented, because the potential performance improvement on modern multi-core processors was found to be minimal, but in Chapter 3 we will describe a new design (hard-coded for now) in which these calculations will play a role in fitting the size of the parallel calculation dynamically to the problem size.

Chapter 2

Parallel Simulation of Population Models

2.1 Introduction

This chapter covers a period of almost ten years. It begins with an edited version of a paper we published in 1997 which summarizes progress to that point with our understanding of how to parallelize individual-based ecology models. In this paper, we present a general scheme for parallel simulation of structured population models. The critical development then was an efficient method of balancing the load among the available processors, so that they all reached the global synchronization points at the same time and no processors were left waiting. We introduce in this paper: how we compare to sequential versions of the same model; the level at which we drew out parallelism; the designs of parallelization by ecotype and by cohort; how we measure workload; two different ways of handling births (local and global); the effects and necessity of load balancing; and the design of the load balancing algorithm. Through the combination of parallelization by cohort, frequent rebalancing, and global birth combining, then we were able to demonstrate good scaling and speedup on 32-processors. These results were later extended to 64, 128, and 256-processor machines.

Revisiting this project in late-2006 presented a unique perspective and opportunity for retrospectively comparing to our earlier work. Technology had advanced an incredible amount in the intervening years giving me exclusive access to my own 8-core, 3GHz, 90GFlop desktop computer. Further, many software technologies had matured from their nascent forms in the mid-90's. This chapter describes the new choices made for our parallel hardware and software platforms. With the

modern platforms and tools, we find is that in several cases the original challenges for which we had to develop solutions had evaporated or were inverted, but the solutions became important for new reasons. This is particularly true for load balancing.

In the course of development for the parallel predator-prey model, the population models were completely redeveloped and many tools and capabilities added. The source-level details of both the population and predator-prey testbed models and the significant programming structures are detailed in this chapter.

Global versus Local Birth Combining was introduced in our original paper as two near equals. Global combining was found to be superior because it maintained the workload to be the same level as that required by sequential execution. The Local Birth Combining caused some additional workload to be incurred that lessened the scaling performance, but did not eliminate parallel benefit. Elimination of a global synchronization point and several parallel communications was its benefit, because it was simpler and more time was spent in computation. It is hypothesized that with the core execution speed being so high compared to communication bandwidth that Local Birth Combining would be the superior choice now.

Finally, execution times for our parallel population algorithms under various workloads and for varying numbers of processors are presented.

2.2 Parallel Simulation of Individual-Based, Physiologically Structured Population Models

(**Note:** I removed many tables, references, and overlaps from this paper and parenthetically annotated it as needed to integrate into this presentation. Original paper was published as "*Parallel Simulation of Individual-Based, Physiologically Structured Population Models*" in *Mathematical and Computer Modelling* (Ramachandramurthi et al., 1997) on which I was a co-author.)

Ecology, traditionally an empirical science, is becoming increasingly simulation based. Current applications of high performance computers in the ecological sciences are generally limited to global or regional environmental problems with little effort directed towards population or community ecology. We investigate the modeling and simulation of a class of individual-based population models that have the common foundation of physiology and energetics. Parallel computation has been proposed for individual-based models (Haefner, 1992; DeAngelis et al., 1995) but, as far as we know,

*This work was funded in part by the National Science Foundation under the grant NSF-BIR-9318160 and by the U. S. Environmental Protection Agency through the Cooperative Agreement EPA-XE-819569.

very little effort has been directed toward development of the parallel computational methodology needed for successful implementation. We develop a modular approach to design sequential and parallel simulation algorithms for these models while attempting to meet three objectives: efficiency, portability, and extensibility.

Modeling and simulation are invaluable tools for ecological risk assessments, especially those that tend to focus on chemically stressed populations or communities (Suter, 1993; Bartell et al., 1992). Two facts provide motivation to employ individual-based approaches in population and community risk assessment. First, chemicals impact individuals directly and the organismal effect is transferred through the higher levels of ecological organization. Second, because of composition and physiology, different individual organisms can react quite differently to the same concentration of chemical.

Individual-based models are often able to capture the inherent complexity of many ecosystems (DeAngelis and Gross, 1992). While the ability to model the behavior and physiology of individuals in greater detail is beneficial, this technique also generally increases model detail and complexity, creating a need for better algorithms and high performance computers to simulate these more sophisticated models. Individual-based ecological models are conceptually ideal for parallel computation because there are many similar structures throughout the ecological hierarchy of individuals, ecotypes, populations, and communities. For example, population dynamics are composed of interacting dynamics of a multitude of individuals who progress temporally in similar stages through their lifetime employing analogous processes of growth, reproduction, and death. It is the interactions between individuals, such as competition for resources or reproductive advantage, that constrain parallelization efficiency. These interactions are often density dependent phenomena whose representation requires integrative information about many individuals in the population.

First, we describe a generic approach for employing parallel computers in the simulation of physiologically structured populations. The generic model, a system of hyperbolic partial differential equations, is described in biological and mathematical detail in Hallam et al. (1992b). Then, as a prototype for developments, we describe our efforts to simulate *Daphnia* populations on parallel computers. Sequential simulation of the *Daphnia* population model is described in Hallam et al. (1990b).

Individual models are dynamic representations of the life history of the organisms that compose the population. For n individual structural attributes, m_i, such as size, protein mass, or lipid mass, a form of the dynamic is

$$\frac{dm_i}{dt} = G_i(a, m_1, ..., m_n; \alpha) = F_i(a, m_1, ..., m_n; \alpha) - L_i(a, m_1, ..., m_n; \alpha)$$

where G_i is the growth rate, F_i is the uptake term for growth, L_i is the loss of m_i for energetic demands and reproduction and $i = 1, 2, ..., n$. α is a parameter vector defining rate process parameters and environmental properties such as resource level and resource quality; a fixed α defines a specific type of individual, called an *ecotype*. The individual model is incorporated into an extended McKendrick-von Foerster model to study population dynamics (Hallam et al., 1992b). The hyperbolic partial differential equation

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho}{\partial a} + \sum_{i=1}^{n} \frac{\partial(\rho G_i)}{\partial m_i} = -\mu\rho$$

describes the population in terms of a density function measured in terms of the numbers per age, per structural attributes m_i, for $i = 1, 2, ..., n$, while G_i is the growth rate of the individual attribute i as above. (This format is a more general expression of the models presented in the previous chapter for fish and *Daphnia*.) Variation among individuals in this model occurs only in age, and subsequently, as age changes, so do the physiological variables m_i. The advantage of this population model is that it is a hyperbolic equation that can be solved by the method of characteristics; that is, ordinary differential equation methods may be used to simulate the cohorts of individuals that follow the dynamics of the characteristic. Because births are determined at the individual level, the boundary condition for the birth of new organisms and the formation of new cohorts can be computed simultaneously with the cohort dynamics. Our sequential numerical scheme employs the method of characteristics that fully utilizes the individual model and follows cohorts in the population; this procedure differs substantially from the generic escalator boxcar train approach of DeRoos (1988), where moments of the population density are featured computational objects.

The purpose of the population model is to investigate effects reflected in the population dynamics caused by the physiology of individuals. The spatial environment is assumed to be homogeneous. While effects of both chemical and environmental stressors (temperature and dissolved oxygen) are present in the model we employed, their presence does not require additional efforts for the parallelization so discussion of these processes and their representations are omitted. Since they potentially affect the rate of growth, hence the first birth times, these do have effects (sometimes large effects) on the workload generated during simulation, but they do not have any direct effect on the design choices we made.

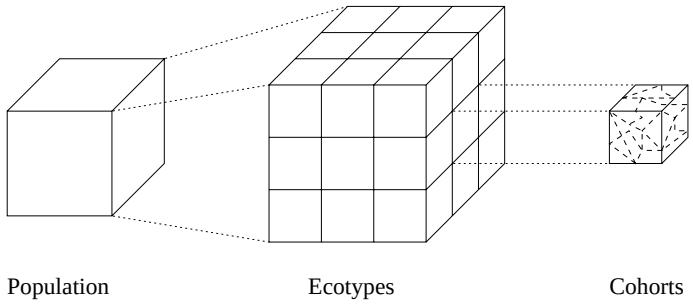


Figure 2.1: The decomposition of a population into ecotypes and cohorts.

2.2.1 The Sequential Algorithm

Two benefits can accrue from a carefully designed sequential algorithm for simulating a structured population model. First, the sequential algorithm would serve as a benchmark against which to compare any parallel implementation. Second, by elucidating the structure inherent in the model, the sequential algorithm can also serve as a convenient starting point for developing parallel algorithms with minimum effort. In this section, we present some techniques to design efficient sequential algorithms.

The Concept of a Cohort

The Method of Characteristics mathematically directs us toward the concept of tracking identical individuals as a single cohort. Overall, this concept is common in many ecological models as a specification of the work that needs to be performed in order to simulate a population. (Because individuals of the same age and the same ecotype share identical growth characteristics, then for simulation purposes it is convenient to collect such individuals into a common *cohort*. An extreme version of this concept is a pure Individual-Based model, where each cohort represents a single member of the population (Grimm and Railsback, 2005). Such model designs still fit into our conceptual framework.) Figure 2.1 shows the structure of a population studied here in terms of ecotypes and cohorts.

For our simulations, in order to maintain the work required for simulation at a reasonable level, if two or more cohorts of the same ecotype give birth at the same time, then the two offspring cohorts are combined into a single cohort; i.e., we combined births. Combining births helps to minimize the total number of cohorts in the population and hence the amount of computation required for the simulation, without significantly compromising the accuracy of the results.

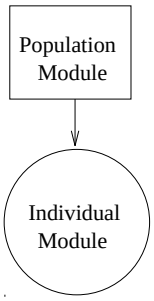


Figure 2.2: Structure of the sequential algorithm

A Modular Approach

We propose a modular approach to the design of sequential algorithms in order to elicit the inherent parallelism in individual-based models. We define an elementary unit of simulation to be the simulation of one cohort for one time step. It is the function of the *Individual Module* to perform this elementary unit of simulation. The task of the *Population Module* is to simulate a structured population through time by invoking the *Individual Module* for each cohort for every time step. Figure 2.2 shows the structure of such a modular sequential algorithm.

Typically, the *Individual Module* executes a numerical method to solve the ordinary differential equations representing the individual. In the example below, we use higher order Runge-Kutta methods to solve the individual models. In this design, the *Population Module* maintains all necessary population level information including the computed values for all the cohorts in the population (e.g., biomasses of lipid and structure, age, reproductive state). It also calculates the summary statistics of the population (e.g., average age, average biomass of lipid or structure or total biomass, total numbers) in order to assess the population level effects on individuals (e.g., density dependent mortality regulation).

For the individual-based population model of *Daphnia* described, a sequential program was developed by Hallam et al. (1990a) and utilized for analyses of the effects of chemical stressors on the population. We redesigned this program into the modular structure described above. We do not discuss the *Individual Module* nor the *Population Module* in detail here because, except for organization, they are quite similar to the approach utilized in the original sequential program. Our purpose is to address the issues associated with parallelization. Later, we provide experimental results to compare the performance of the sequential and parallel algorithms.

2.2.2 Parallel Algorithms

We set three goals for our parallelization effort. First, we must achieve efficiency in terms of speedup over the sequential algorithm. Second, our parallel programs must be easily portable to other architectures that are based on similar programming paradigms. Third, the algorithms developed for the population model should be easily extensible to more complex community and food-web models.

The target architecture for our parallel algorithms was the multiple instruction multiple data (MIMD), distributed memory parallel computer. Typically, such a computer consists of a collection of processors each with private memory that pairwise have no shared memory. They have facilities for interprocessor communication and synchronization. Such computers are usually programmed using the single program multiple data (SPMD) paradigm where a copy of the same program executes on each processor but on different data. The wide availability and popularity of distributed memory MIMD parallel computing both through dedicated hardware and through heterogeneous computer networks, means that our algorithms would be easily accessible.

Methodology Overview

Our parallel algorithm design is also based on a modular approach that can be viewed as an extension of the sequential algorithm design. First we present a temporal overview of the parallel simulation. The simulation is initialized with each of the processors assigned a portion of the initial population. At the start of each time step, all the processors must cooperate to compute the statistics needed for computing the population level effects. This cooperation is a point of synchronization of all the processors at the start of each time step. Once the population statistics have been computed, the processors work in parallel, independently advancing their local populations through the next time step.

Figure 2.3 presents an overview of the structure of the parallel algorithm. Typically, a parallel algorithm for an individual-based population model would consist of three different modules: *the Individual Module* and the *Population Module* as in the sequential algorithm, and the *Load Balancing Module*. A copy of all three modules resides in each processor. We further utilize node (or process) zero as both a compute node and as the host whose job includes the handling of input and output of data, as well as distributing the initial population among all the processors in the system. The other compute nodes are idle during these host operations. This is potentially a significant bottleneck for any parallel computation.

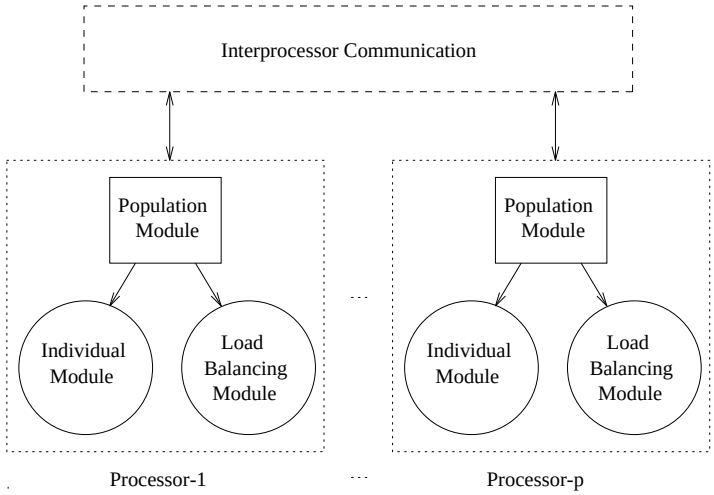


Figure 2.3: Structure of the parallel algorithms

Each processor executes the *Population Module*, which employs the *Individual Module*, to simulate its local population. This simulation is accomplished in much the same way as the sequential program, namely, by repeated invocations of the *Individual Module* at each time step for each cohort. When all the cohorts in the local population have been advanced one time step, the total biomass (or some other measure of total statistics) of the population across all the processors is calculated in order to assess the global effects of individual behavior. This entails synchronization of all the processors and interprocessor communication. Similarly, additional communications can be required in order to collect output data for analysis of the simulated population.

We compare two different ways to distribute the population among the processors for parallel simulations. First, we present the coarse-grained approach of parallelization by ecotype. Then, we describe the finer-grained strategy of parallelization by cohort.

Parallelization by Ecotype

The organization of a population into ecotypes suggests a simple way to distribute the population across the processors of a parallel computer, namely, using only one processor per ecotype. If there are n ecotypes and p processors, then each processor is assigned n/p ecotypes. If a parent and its offspring are assumed to belong to the same ecotype always — as is the case in our models — then this method has the advantage that once the simulation is started, then there is no need to move cohorts across processors during the simulation in order to maintain the "only one processor per ecotype" paradigm. Furthermore, combining births is also simplified because each ecotype is

localized to one processor. Therefore, this method of parallelization can significantly reduce the amount of interprocessor communication.

One limitation of this method is that the amount of parallelism cannot exceed the number of ecotypes. Another limitation of this algorithm arises from the *survival of the fittest* principle (Henson and Hallam, 1994) which states that as the simulation progresses, a small number of ecotypes will dominate the population while the others are doomed to die out gradually. This principle translates into an uneven usage of the processors because the simulation would eventually become localized to a few processors while the rest are idle.

Since all the processors must synchronize before the start of each time-step, any imbalance in the distribution of cohorts can mean longer waits for synchronization. While it can be tempting to enlist multiple processors to work on a given ecotype, such attempts would detract from the main advantage of this method, namely, minimum interprocessor communication. It is conceivable that under certain circumstances, gradually easing the restriction of "only one processor per ecotype" into a more liberal "one or more processors per ecotype" would speed up the simulation sufficiently to outweigh the additional communication cost. However, it is difficult to determine *a priori* when such an approach would be beneficial.

The performance of this algorithm for the *Daphnia* population model will be compared with other algorithms in a later section.

Parallelization by Cohort

In this method, the cohorts that constitute the initial population are distributed evenly across all the processors without regard to their ecotype. This represents a finer granularity of parallelism than the previous method with the amount of parallelism here limited only by the number of cohorts.

In a sequential computation, birth combining can be performed perfectly without additional effort, completing the simulation with the minimal amount of work. A problem arises when distributing the population across several nodes that do not have access to each other's memory. How can births be combined? As a part of this study, we will investigate the efficiency of combining births locally on a single processor (called the *Local Combine Algorithm*) or globally across all processors (called the *Global Combine Algorithm*). In the *Local Combine* algorithm, where combination is restricted to the population local to each processor, two or more new-born cohorts of the same ecotype on the same processor are combined into a single new cohort. This is easily accomplished and involves no interprocessor communication. However, it does create more cohorts in the population than the sequential algorithm and thus leads to additional computation, which can translate into

longer execution times. An alternative is the *Global Combine* algorithm, where births are combined across the entire global population; i.e. combination of births is made across all the processors in the system. The benefit of this approach is that the number of cohorts in the total population, and hence, the amount of computation are kept to the absolute minimum, namely, identical to the procedure of the sequential algorithm. However, this can only be accomplished by means of additional costs in communication. Note that when parallelization is by ecotype, use of the *Local Combine* algorithm is sufficient to minimize the amount of computation.

2.2.3 Load Balancing

Because local populations (number of cohorts) on different processors can grow at different rates, an even initial distribution of cohorts across the processors can become unbalanced during the simulation. We have devised an efficient parallel load balancing algorithm to remedy such a situation. When necessary, all the processors execute the *Load Balancing Module* in parallel in order to redistribute the population evenly.

This algorithm is designed specifically for parallelization by cohorts. While imbalances also arise when the model is parallelized by ecotype, they cannot be remedied without compromising the "only one processor per ecotype" paradigm.

We define the terms balance and imbalance more precisely before presenting the load balancing algorithm.

Determining the Imbalance

Let p be the number of processors and for $1 \leq i \leq p$, let l_i denote the load located at processor i. Here we assume that l_i is simply the number of cohorts at processor i. The total load L on the system is $\sum_{i=1}^{p} l_i$ and the average load per processor, l, is $\frac{L}{p}$.

We say that the load on the system is *balanced* if the load at each processor satisfies the condition $|l - l_i| < 1$.

The *excess load at processor* i, x_i, is determined as follows:

$$x_i = \begin{cases} l_i - [\frac{L}{p}] - 1 & \text{if } 1 \leq i \leq L \bmod p; \\ l_i - [\frac{L}{p}] & \text{otherwise.} \end{cases}$$

where [y] denotes the greatest integer in y. If $x_i > 0$ then we say that processor-i is overloaded. If $x_i < 0$ then processor-i is underutilized. The reason this function is split into two is that the total system load will usually not be evenly divisible by the number of processors.

The following lemma shows that our definition of *excess load* is consistent.

Lemma 2.2.1 *The net excess in load is zero i.e.,* $\sum_{i=1}^{p} x_i = 0$

Proof

$$\begin{aligned}
\textstyle\sum_{i=1}^{p} x_i &= \textstyle\sum_{i=1}^{L \bmod p}(l_i - [\frac{L}{p}] - 1) + \sum_{i=L \bmod p+1}^{p}(l_i - [\frac{L}{p}]) \\
&= \textstyle\sum_{i=1}^{p}(l_i - [\frac{L}{p}]) - (L \bmod p) \\
&= L - \textstyle\sum_{i=1}^{p}[\frac{L}{p}] - (L \bmod p) \\
&= L - p[\frac{L}{p}] - (L \bmod p) \\
&= (L \bmod p) - (L \bmod p) \\
&= 0
\end{aligned}$$

■

If any overloading is detected on a processor, then we can move some cohorts from that processor to an underutilized processor in an attempt to achieve balance. The following lemma shows that this strategy will indeed result in balanced load across the system.

Lemma 2.2.2 *Redistributing the excess load results in a balanced load i.e., for each* $1 \leq i \leq p$, $|l - (l_i - x_i)| < 1$.

Proof If $i > L \bmod p$, then $x_i = l_i - \frac{L}{p}$ and the proof is trivial. Suppose $i \leq L \bmod p$. Since $x_i = l_i - [\frac{L}{p}] - 1$, we have $l_i - x_i - 1 = [\frac{L}{p}]$ and $l - (l_i - x_i - 1) = \frac{L}{p} - [\frac{L}{p}]$. Since $i \geq 1$, we know that $L \bmod p \geq 1$ or $[\frac{L}{p}] < \frac{L}{p}$. Therefore, $0 < \frac{L}{p} - [\frac{L}{p}] < 1$. Hence, $0 < l - (l_i - x_i) + 1 < 1$ or $-1 < l - (l_i - x_i) < 0$ or $|l - (l_i - x_i)| < 1$. ■

A Parallel Load Balancing Algorithm

Our algorithm for load balancing consists of two components. The first component, called algorithm BALANCE, computes the amount of imbalance at every processor. The second component, called algorithm REDISTRIBUTE, redistributes the load in order to achieve a balanced load. These two algorithms are sketched below.

Algorithm BALANCE

begin

0. Initialize the data.

   $i \leftarrow$ processor-id;

   $p \leftarrow$ number of processors;

   Let $l[1...p]$ and $x[1...p]$ be arrays of p integers.

1. Determine the load on each processor in the system.

   for ($j \leftarrow 1$ to p)

      $l[j] \leftarrow$ the number of cohorts on processor-j;

2. Compute the average load per processor.

   $L \leftarrow \sum_{j=1}^{p} l[j]$;

   $average \leftarrow [\frac{L}{p}]$;

   $remainder \leftarrow L \bmod p$;

3. Compute the imbalance levels.

   for ($j \leftarrow 1$ to p) $x[j] = l[j] - average$;

   for ($j \leftarrow 1$ to $remainder$) $x[j] = x[j] - 1$;

4. Call Algorithm REDISTRIBUTE to achieve balance.

end

Algorithm REDISTRIBUTE

begin

0. Initialize the variables.

   $source \leftarrow 1$;

   $destination \leftarrow 1$;

1. Find the source with lowest id.

   while (($x[source] \leq 0$) and ($source \leq p$))

      $source \leftarrow source + 1$;

2. Find the destination with lowest id.

   while (($x[destination] >= 0$) and ($destination \leq p$))

      $destination \leftarrow destination + 1$;

3. Check for termination.

   if (($source > p$) or ($destination > p$)) **then** exit;

4. Determine the number of cohorts to be transferred.

   $number \leftarrow \min(x[source], x[destination])$;

5. The source processor sends the cohorts to the destination.

   if $(i = source)$ **then** send$(destination, number, cohorts)$;

6. The destination processor receives the cohorts from the source.

   if $(i = destination)$ **then** receive$(source, number, cohorts)$;

7. Update the imbalance levels.

   $x[source] \leftarrow x[source] - number$;

   $x[destination] \leftarrow x[destination] + number$;

8. Go to step 1.

end

Analysis of the Load Balancing Algorithm

The first non-trivial feature of algorithm BALANCE is the interprocessor communication required by Step 1 in order for each processor to determine the loads at all the other processors in the system. This is a point of synchronization of all the processors in the system. Steps 2 and 3 are straightforward. In algorithm REDISTRIBUTE, Steps 1 through 4 are also straightforward computations. Only Steps 5 and 6 of algorithm REDISTRIBUTE involve interprocessor communication.

The computation of the number of cohorts in Step 4 of REDISTRIBUTE ensures that the load at either the source or the destination processor will be balanced after the transfer. If there are p processors in the system, then it is easy to see that algorithm REDISTRIBUTE terminates after at most p iterations. In fact, since the last step must saturate both the source and the destination, the number of iterations cannot exceed $p - 1$.

In the following section, we examine the performance of the parallel algorithms presented thus far.

2.2.4 Performance of the Parallel Algorithms

Recall that we defined an elementary unit of simulation to be the simulation of one cohort for one unit of time. We can measure the amount of computation during a simulation in terms of the total number of units of simulation performed during the entire period of the simulation i.e., in cohort-steps.

Table 2.1: A comparison of the sequential and parallel algorithms

Algorithm	Computer	Execution time in seconds	Max. time for communication
Sequential	SPARC 20/61 (1 processor)	1026	–
Parallel by ecotype	CM-5 (32 processors)	851	99%
Parallel by cohort with local combine	CM-5 (32 processors)	489	15%
Parallel by cohort with global combine	CM-5 (32 processors)	250	19%

Experimental Results

We used a 32-processor Thinking Machines CM-5 as our parallel program development platform. The CM-5 consists of a collection of SPARC processors interconnected by a fat-tree network (Hwang, 1993). Each processor has its own private memory, which in our case was 32 megabytes, and there is no shared memory in the system. We used the CMMD message-passing library for interprocessor communication. Most of our program development was done using the C language.

Table 2.1 can be used to compare the performance of our sequential algorithm and three different parallel algorithms by indicating the total time required for the same experiment and the maximum percentage of the time required for communication as contrasted with time required for computation. The sequential simulation was performed on a SPARC 20/61 workstation. The data represents a 500-day simulation of an established *Daphnia* population initially consisting of 267 cohorts. For this simulation, the size of a time step was fixed at 0.05 days, and the load-balancing algorithm was invoked at the end of every day i.e. after every 20 time steps. The effects of toxicants and temperature variation on the *Daphnia* population were also assessed in this simulation.

From the table, it is clear that the *Global Combine* algorithm is the fastest of the three parallel algorithms. The difference between the *Local Combine* and *Global Combine* algorithms stems from the fact that while the former simulated 23 million cohort-steps the latter had to simulate only about 9 million cohort-steps for the same population. In this case, a little additional communication resulted in huge savings in computation.

Before we can compare the parallel and sequential algorithms, we need a measure of the relative difference in computational power between a SPARC 20/61 processor and one processor of the CM-5. The SPARC 20/61 took 76 seconds to simulate the above data for 50 days whereas one processor of the CM-5 alone took 522 seconds. Both systems simulated an identical number of cohort-steps.

Therefore, we may say that for this application, the SPARC 20/61 is about 6.8 times faster than each processor of the CM-5. Thus, in principle, we may expect that a 32-processor CM-5 would be 4.7 times as fast as the SPARC 20/61. However, when two or more processors cooperate, interprocessor communication comes into play.

Table 2.1 shows that in reality, the *Global Combine* algorithm on a 32-processor CM-5 is only about 4 times faster than a SPARC 20/61. The difference can be accounted for by interprocessor communication. In the *Global Combine* algorithm, each processor spent only 210 seconds at the most for actual simulation i.e., in the *Individual Module*. The rest of the time was spent on communication, synchronization, and other overhead.

Performance of the Load Balancing Algorithm

Load balancing represents a trade-off between computation and communication. The amount of computation involved in the load balancing algorithm itself is insignificant. Load balancing does not decrease the total number of cohorts in any way. By evenly distributing the cohorts across all the processors, it simply helps minimize the maximum number of cohorts on any one processor. Thus, depending on the degree of imbalance, load balancing can save us a considerable amount of computing time.

(This is the point where the relationships have flipped. The communication time relative to computation time is now large. Thus having a small number of timesteps between rebalancing typically increases the simulation execution times. For the basic population models, the cost of maintaining perfect balance, now exceeds the potential gain. There is a potential for 5-10% performance gain in the predator-prey model, but not in our population models. Since we are aiming to be more generally applicable than to just our simulation models, then reviewing and including the rebalance algorithm is merited.)

The communication involved in load balancing can be broken up into two parts. The first is the global communication required to exchange load information among all the processors. If any imbalance is detected, some cohorts must be transferred between individual processors in order to achieve balance. Hence, the amount of communication required for load balancing depends partly on the degree of imbalance at the time of invocation. Therefore, we are faced with a choice of frequencies with which we use the load balancing algorithm.

Figure 2.4 shows the time taken by our load balancing algorithm for various balancing frequencies. The population data used was the same as that used for Table 2.1. Recall that each time step is 0.05 days. From the figure, it is clear that load balancing is extremely fast both in absolute terms

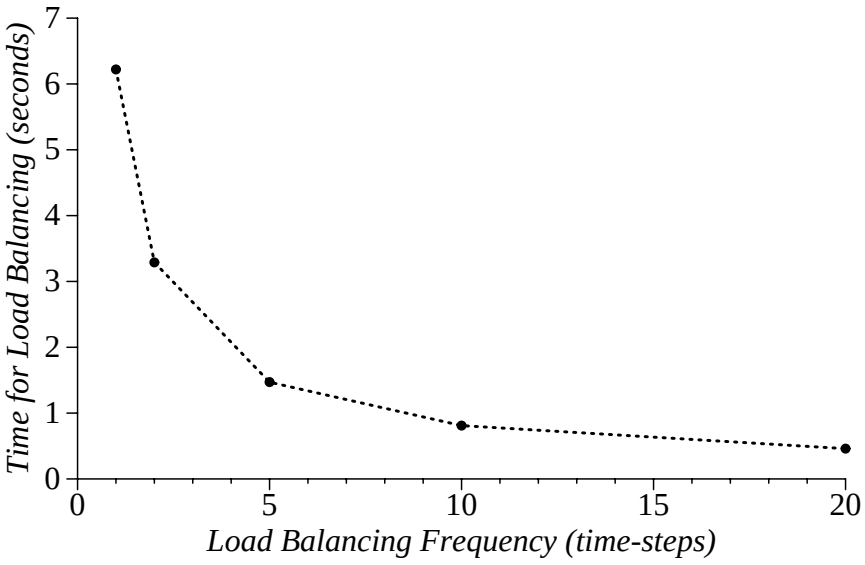


Figure 2.4: The cost of load balancing

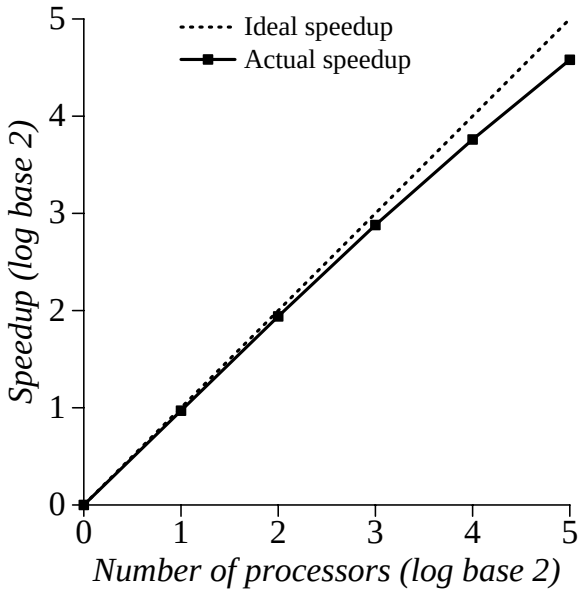


Figure 2.5: Speed-up achieved by doubling the number of processors

and also when compared to the total time taken by a 500-day simulation. (For these reasons, we later decided to just rebalance every time-step, because the computational benefits far exceeded the few seconds of costs.)

Speedup

For a given algorithm, the *speedup* attained by using p processors to solve a problem is the ratio of the time taken by one processor to the time taken by p processors to solve the same problem (Almasi and Gottlieb, 1989).

Figure 2.5 shows that doubling the number of processors used on the CM-5 nearly halves the execution time of the global combine parallel algorithm. From the figure we can conclude that this algorithm scales very well, sustaining a speedup very close to the ideal for up to 32 processors on the CM-5.

2.2.5 Initial Conclusions

We have proposed a modular approach to design parallel simulation algorithms for physiologically structured individual-based population models in ecology. The objectives of our approach were three-fold: efficiency in terms of speedup, portability to a variety of computational platforms, and extensibility to more complex ecological models. A modular design for the sequential simulation

algorithm was also presented with a view to easing the development of parallel algorithms. It was shown how such sequential algorithms lend themselves to parallelization easily.

Our parallel algorithms were targeted toward the MIMD parallel computing paradigm. Issues such as granularity, communication and synchronization, and load balancing were addressed in the effort to obtain maximum speedup. Two different parallelization strategies of different granularity, each based on the structure of the simulated population were presented. We showed two contrasting schemes to handle discrete events such as births in the population during a parallel simulation. An efficient and general load-balancing algorithm was also presented and analyzed.

Using the modular approach, concrete algorithms were developed to simulate a specific individual-based, physiologically structured model for *Daphnia* populations. These algorithms were implemented on a Thinking Machines CM-5 parallel computer. Using a fine-grained parallelization strategy together with global combining of births, and frequent use of the load balancing algorithm, we were able to obtain nearly ideal speedup. Thus we have demonstrated that individual-based population models can be efficiently simulated and analyzed by using parallel computational techniques. Moreover, owing to their modular design, we were able to port these programs easily for execution on a network of heterogeneous workstations using the PVM (Parallel Virtual Machine) software. The wide availability of PVM, even in homogeneous parallel computing platforms, makes this approach very attractive.

We consider our efforts to parallelize a generic structured population model to be part of a larger mission, namely, the determination of efficient algorithms to simulate an integrated food-web model. The next specific step in this direction is the development of parallel algorithms to implement an individual-based predator-prey model where, for example, *Daphnia* are the prey and fish are the predators. The modular approach that we have used to construct algorithms for the individual-based population model ensures that these algorithms will serve as valuable components in food-web analyses.

2.3 Transition to Modern Systems

As a way of transitioning from the paper presented in the last section to the final design, I record major developments and applications of both the population and predator-prey models in Table 2.2. The predator-prey model is described in the next chapter.

Between the time that the paper was published and when I left the project, we did have a computer time grant from NCSA which had several types of machines at the time include Cray,

Table 2.2: Major Development Milestones

Code Description	Code Type	Parallel Library	Computers Utilized
Population	hybrid F77/C	MPI	NCSA: Cray, SGI, and HP Convex
Predator-Prey	hybrid F77/C	MPI	NCSA: Cray, SGI, and HP Convex
Predator-Prey Sequential and Pure Parallel	C	MPI	NCSA, Network of two dual-processor Linux PCs, and Suns
Population	C	MPI	NCSA, Network of two dual-processor Linux PCs, and Suns
Population and PPrey	C	Home-Grown Shared Memory	Network of two dual-processor Linux PCs

SGI, and HP Convex machines. (Ours was the first public code run on a 256-node Cray T3D.) An MPI version was developed from our earlier CM-5 codes for these machines. As part of working with these machines I finally completed the translation of the remainder of the program to C. This made porting between machines easier as well as debugging. A "fish-on-all" predator-prey version was mostly implemented when I left, but was not completely debugged. The pure parallel version was tested on several of the NCSA machines, but it too was not completely debugged. On the HP Convex we did exhibit superscalar speed up for the first time. Almost always these are caused by taking advantage of caching and hardware. The HP Convex was a shared-memory design versus multi-processor. None of these NCSA timings are directly reported, but the knowledge gained does make appearances throughout this thesis, such as superscalar speed-up, shared memory, large problem sizes, and MPI.

To give these machines any challenge, especially the ones with 128 or more nodes, we ramped up the workload by increasing the number of ecotypes and by birth class multiplication as described in Section 1.4. We would utilize populations with 125 ecotypes and variable numbers of birth classes filled. Each machine was still somewhat unique. Even though all supported MPI, there were various I/O, compiling, debugging, and timing and profiling options. There still remain flags and references in the final code base referring to these machines and the CM-5.

In 1997 and 1998, MPI-1 was still being adopted and its early incarnations did not necessarily take full-advantage of machine resources. For networks of workstations, the presumed base communication method was TCP, even on multi-processors. I developed my own shared-memory, message-passing library in order to avoid this overhead and was able to handedly beat the MPI

libraries at the time. I also learned a lot about dead-locking, semaphores, cache lines, and other shared memory and parallel problems and concepts. One of the first things I did after picking this project back up was to test my shared memory version against the newest MPI implementations. Mine was close, but they all bested my execution times indicating that they are taking advantage of the faster communication channels.

2.4 Modern Systems

Picking this project back up late 2006, I had to reorient myself to all of the technology choices available. I initially built a small network of two dual-core machines in my house with a gigabit connection between them to act as a testbed.

2.4.1 Parallel Methodologies

Threads

Message-passing is not the only parallel methodology. The other common methodology is that of threads. Threads are separate execution paths inside of a single program which are scheduled and executed independently and simultaneously by the operating system. (Since threads are all in the same process, they share the same process ID.) They can communicate information through various methods including inter-process calls, method calls on the thread, shared memory, and similar mechanisms. Threads are the preferred method for single programs to gain benefit from multi-processor/multi-core machines because they can be set up and utilized within a program without any external files or other setup. For instance, worker threads can be spawned by a program to carry out a long-running task, while not disrupting the responsiveness of the user-interface. POSIX Threads (pthreads) are the current standard; see Butenhof (1997). Threads can require significantly more design and implementation effort and extending the concept across disjoint computers or supercomputer nodes is impractical. Because of the rise of multi-core processors and the need to be able to efficiently multi-thread applications in order to get better performance for applications from these machines, there are many current efforts. Applying the techniques to these models in which we have already invested so heavily in MPI did not make sense.

Shared Memory

Another concept is shared-memory, with all of the information sitting in a common memory space and processors all having access to this information. Earlier, I did implement a shared-memory, message-passing library based on SYSV shared memory (Stevens, 1993). This was especially beneficial in the early days of MPI, because MPI was encumbered by network protocols even if it was running on a single machine, so I was able to achieve significant speed improvements from early dual-processor PC computers. The possibility of using a full shared memory space with the work located in the middle with the processors around each grabbing a new task as it completed the previous was considered, but would also have required significant reconceptualization and redesign, so I did not pursue it.

Computational Grids

Another parallel methodology I considered was that of computational grids and these are addressed briefly. Originally this term applied to the concept of joining entire supercomputer centers into clusters with high-speed network connections. The supercomputers would then each become a node in a larger calculation. The term has evolved to include a single computational tasks carried out in small segments by entirely separate computers which are not necessarily aware of or in contact with each other. This allows problems of immense size to be tackled that could not be otherwise be performed. The utilization of several computers to apply to tasks like compiling, video rendering, and similar, is behind the concept of the XGrid system built into all Macs.

Computational resources joined via XGrid can be utilized for research computations. For example, the Xgrid@Stanford project utilizes XGrid to couple computers on the Stanford campus and all over the world to work on a pharmacology problem which allows them to "run a calculation in 1 week instead of 1 year" (Parnot, 2007). Another particularly interesting application of the Xgrid feature is the Kentucky Data Seam project (**kydataseam.com**) which utilizes the statewide network joining all of the schools and the Macs placed in each school as a large grid. It is "dedicated to advancing research and promoting education to support economic growth."

I explored grids after my return because I initially thought they were the next, natural step beyond clustered computers. Further, XGrid is advertised as MPI-compatible. I eventually determined that the problems which work best on this parallel methodology are uncoupled problems which split out into self-contained units that can be completed in any order, at any time, without common communication. This description does not fit our ecology simulation problems.

MPI

I tested several MPI versions that claimed XGrid-compatibility, but found them lacking, especially the advanced structures required by the predator-prey parallel model. I initially returned to MPICH2 (MPICH2, 2007), because I had used MPICH1 in my earlier development. The MPI-2 implementation is now mainly Python-based including the daemon-processes. (Python is freely-available scripting language supported on many platforms (www.python.org).) Because it is Python-based, then it is easy to diagnose and fix problems. But performance and usage was often clumsy and very dependent on configuration file formats that were not well-documented. Also, in a PVM-like manner, it requires the user to start and maintain daemon processes on all of the machines that will be participating in the computation, which adds another layer of maintenance problems.

OpenMPI, **www.open-mpi.org**, is the implementation of MPI that I use exclusively in this presentation. I was introduced to it at the WWDC2007 conference by some researchers after months of trying to get MPI and XGrid to function together as advertised. OpenMPI can spawn MPI jobs onto various grids including XGrid. It is developed and maintained by consortium of "academic, research, and industry partners" including the University of Tennessee, so it is well-supported.

2.4.2 Modern Parallel Hardware

The final results in this report are being made on an eight-core Apple Mac OS X with each core running at 3.0 GHz. There are two quad-core CPUs. Each of these quad-core CPUs is effectively two dual-core CPUs on a single die. The picture that the reader should have in mind of the computational core and memory layout of this system is in Figure 2.6 (Wilson, 2007). In particular note that there is a shared cache of only 8MB for each pair of cores. This machine mostly functions as four dual-core CPUs, except that there is a shared path to the northbridge (memory and video access) for the cores on a single CPU. Further, all eight cores have to contend for access to Main Memory.

Cache contention and CPU idling while accessing main memory are some significant causes of lessened performance which we encountered. Many real-world applications do not benefit over the otherwise identical dual dual-core Mac Pro, which is a common desktop machine for graphics and media applications. Even the base operating system, Mac OS X 10.4 "Tiger" does not handle the eight cores as efficiently as it could, because it will often move a long-running task to a different CPU core. This requires cache to be flushed, the program's state to be reloaded, and other time consuming tasks. The next version of OS X called "Leopard" does have better *core affinity* in order to perform better on these processors. We have observed a 10% or more gain in performance entirely

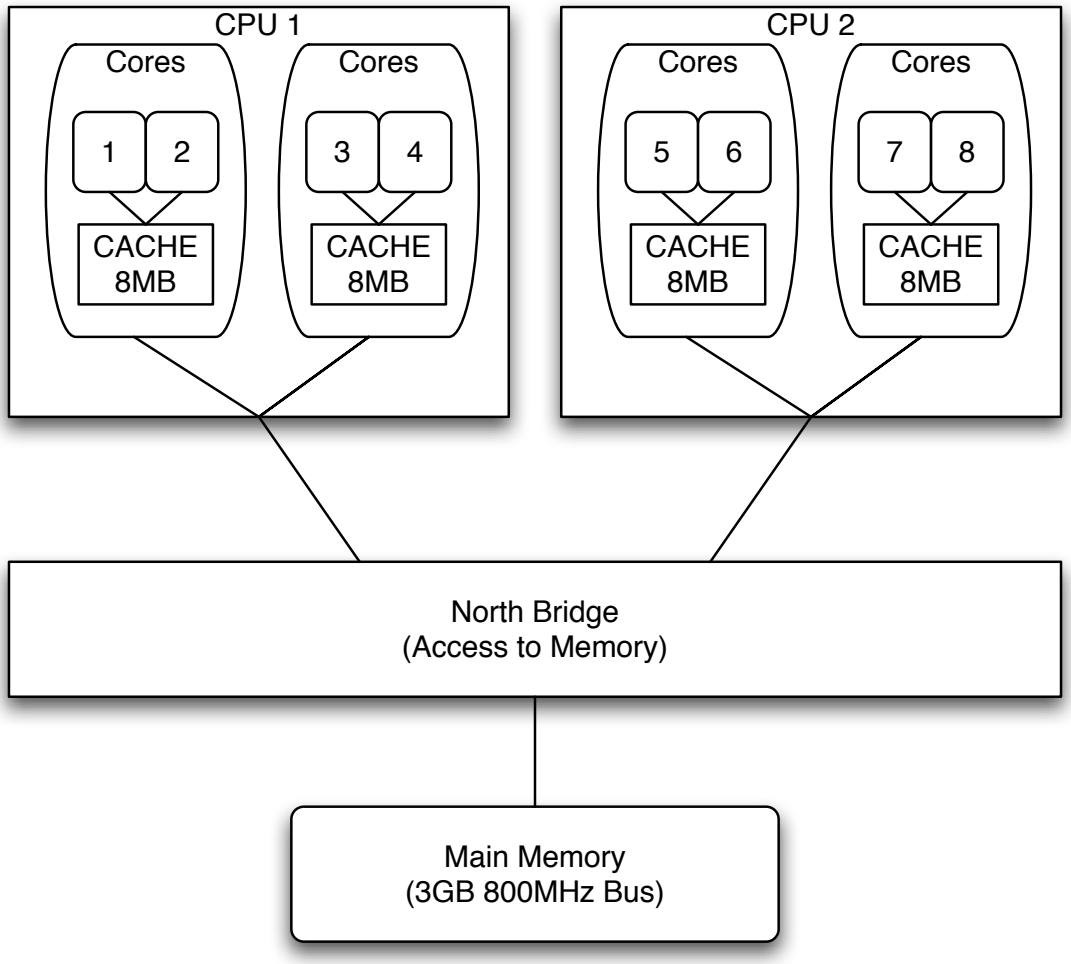


Figure 2.6: CPU and Memory Layout for Test Computer

due to better affinity when testing on a pre-release version of Leopard. All timings in this report are from running under Tiger. It cite this to show that parallel performance is still an evolving field of research. As new hardware is created, then new bottlenecks can appear that have to be worked around by new techniques. In this case, the task-scheduling at the OS level is the culprit.

2.5 Program Design

The overall program design for individual-based ecology models is now described more fully. We did settle on the individual cohorts as the basic unit of work. This design is pictured in Figure 2.7. We do not try to go inside the *Individual Model* as it was termed in the paper (or the black box as we informally called it). That is where a bulk of the computations are done, but the parallelism available inside is hard to extract. Machines like the MasPar and streaming processors like GPUs depend on doing simple computations on vector streams of data, but there are too many branches and individual behaviors for ecology models to map well into this paradigm. (For our *Daphnia* models, the initial population is distributed by ecotype in order to re-establish parent-offspring relationships, but this is not generally required.)

With a population of typically thousands of cohorts, but each independent except for a few population-level calculations, then the parallel advantage is from distributing them onto individual compute nodes and letting each node advance its local population to get it done faster than one node doing all of the work. Obviously, parallel programs require work at least equal to the compute resources to be of any potential benefit. In other words, parallel problems require a minimum amount of work in order to be justified. Further, the *granularity*, the total size of work per processor, needs to be chosen well or else too much time is spent in communication shuttling the workload around and not enough time is spent doing useful calculations. One can choose how many processors are involved in different stages of the work and this may be appropriate in order to balance workload against communication load. This idea is discussed in the next chapter.

For the population model we focused on the *Daphnia* version because of its short reproductive period and resulting large numbers of cohorts. A fish population has an annual reproductive period and a lifetime of a little over eight years. The fish lay thousands of eggs each, but these all combined into a single cohort, so they induce only the load of one more cohort.

The run loop is diagrammed in Figure 2.8. Note that the local population on a processor can be zero. It has to continue to participate in the collective operations or else the parallel program fails.

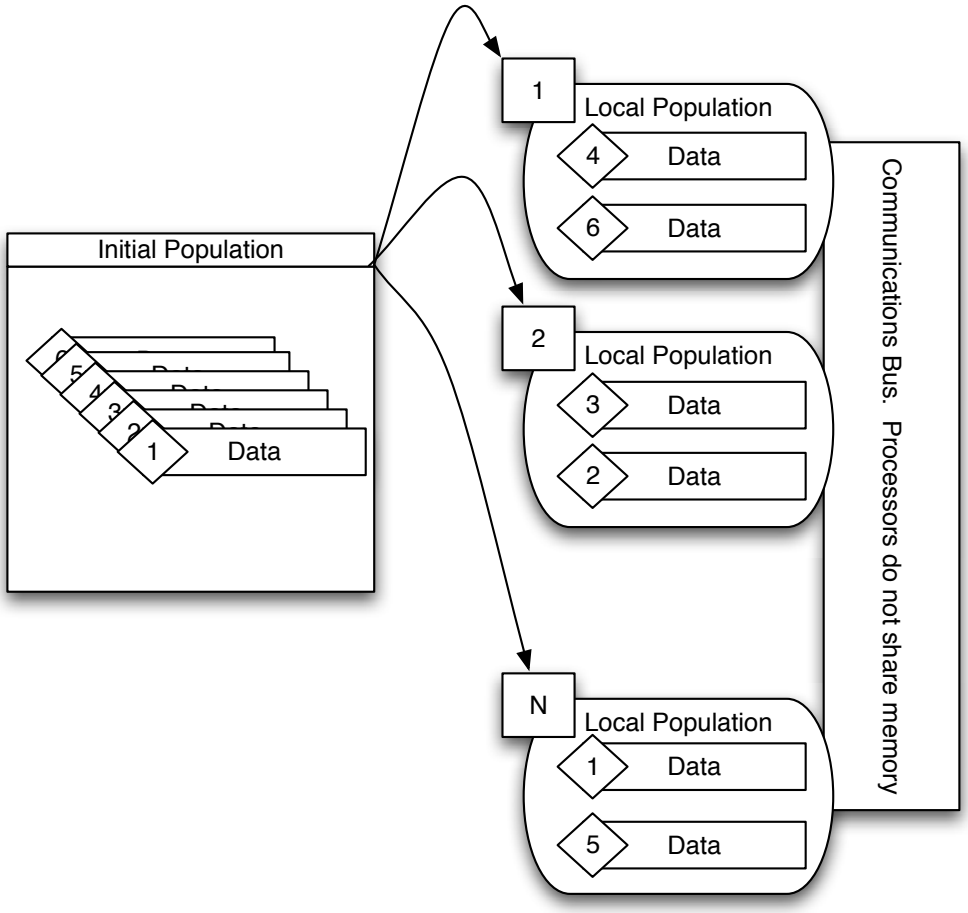


Figure 2.7: Parallel Program Design

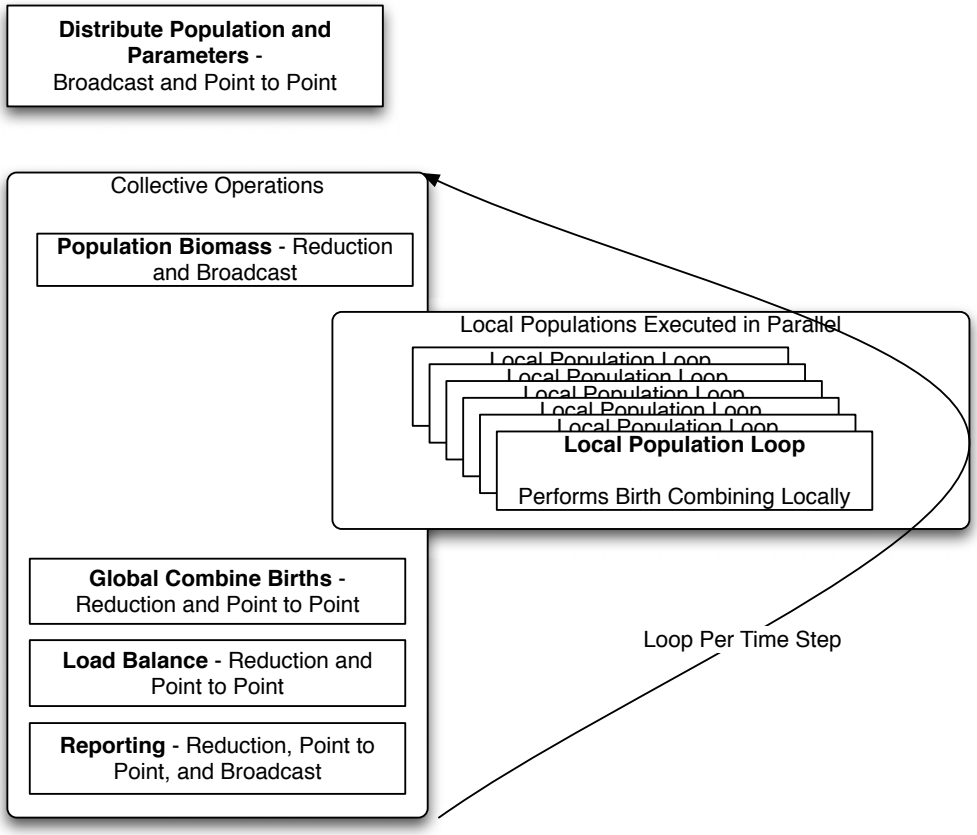


Figure 2.8: Program Run Loops

It just does not have any real work to do, so it proceeds to the next parallel call and waits for the others.

Even though my final MPI-programs were all run inside the same machine, each node participating in the parallel execution program is a completely disjoint process which can only communicate with other members via messages. Each node in the computation has its own process ID and is independently scheduled by the OS onto the available processors. Messages copy information from one node's memory space into another's memory space. We heavily use these types of parallel communication patterns:

1. Point-to-Point: One node directly to another
2. Reduction: All nodes performing some common operation like sum or max on their local data, with a single node compiling the final result. For example, summing biomass across all nodes.
3. Broadcast: One node transmitting data to all other nodes.

These are the main methods of communication, but there are several others; see Snir et al. (1998). For the fully-parallel predator-prey simulation we had to utilize more advanced communication techniques in order to distribute predation tables. The techniques for predator-prey are described in the next chapter.

2.5.1 Source Code Descriptions

Table 2.3 lists how each of the source code components is used by the various versions of models we have. If the source file is identical and shared between both population and predator-prey models, then it is marked as Shared. Otherwise it is marked as Population or Predator-Prey Only. It can be marked both Population-only or Predator-Prey Only if it is changed uniquely for both, like constants.h. If a file is needed only for the parallel versions, then it is marked Parallel-Only. Only header (*.h) files that define significant structures are listed. This table demonstrates the amount of unification we were able to achieve between the two types of models.

These sources are now briefly described as an orientation towards the functions that need to be carried out during the course of simulation. This gives an idea of where to find the routines related to different portions of the simulation.

balance.c, balance.h Cohort imbalance and rebalancing routines for the parallel simulations.

Table 2.3: Source Code Usage

Source File	Population-Only	Predator-Prey Only	Shared	Parallel-only
balance.c			X	X
blck_box_cmn.c			X	
check_pops.c			X	
constants.h	X	X		
dbsqlout.c			X	
deck.c			X	
deck.h			X	
driver.c	X			
error.c			X	
generic_stack.c		X		X
id_decls.h			X	
id_manager.c			X	
inithost.c			X	
initnode.c			X	
mysqlout.c			X	
parallel.c			X	X
parpred.c		X		X
pgsqlout.c			X	
popbirth.c			X	
popmort.c			X	
popsim.c			X	
ppmain.c		X		
pred.c		X		
reports.c			X	
shared_memory.c			X	
shared_memory_clear.c			X	
timing.c			X	

blck_box_cmn.c, blck_box_cmn.h Individual-model numerical integration routines. Note that for fish in predator-prey model their numerical routines because of the feeding are so different that none of the population routines are used.

check_pops.c, check_pops.h Debugging - Performs advanced sanity checks on the memory structures and computed values. This is to locate exactly when an aberrant value shows up during debugging. Tests are embedded throughout the loops so that we can determine exactly what procedures caused any detected errors.

constants.h The most important configuration file. This has all of the main structure variables and defines so that the various models can be built from the same source. Heavily documented.

dbsqlout.c, dbsqlout.h The general routines called for SQL database output of computed data and populations.

deck.c, deck.h The population memory structures and routines.

driver.c, driver.h The main loops for the population models.

error.c, error.h Error reporting and careful shutdown.

generic_stack.c, generic_stack.h Used to hold the affected prey items for each predator as predator feeding is calculated.

id_decls.h, id_manager.c, id_manager.h ID Management system.

inithost.c, inithost.h Initialization and shutdown routines for the sequential program and host node in a parallel program. Mainly initialization and restart file I/O.

initnode.c, initnode.h Initialization and shutdown routines for compute nodes in a parallel computation.

mysqlout.c, mysqlout.h MySQL database specific routines

parallel.c, parallel.h Parallel functions and types generalized, so that we can compile against and use any parallel communication libraries. This includes all of the point-to-point, broadcast, etc. functions, and types, including global_combine.

parpred.c, parpred.h Predation routines for distributed fish and prey populations.

pgsqlout.c, pgsqlout.h Postgresql database-specific routines

popbirth.c, popbirth.h Birth

popmort.c, popmort.h Scans populations removing any individual cohorts that have died due to toxicant exposure, too low of a density remaining (minimum population cutoff), starvation, or age. It also checks for and carries out births in the case that DELAYBIRTHS option is set. (Because the number of eggs per individual is multiplied by ρ and used to set the population density on the new characteristic, then delaying births until after mortality assessment can have a large effect. DELAYBIRTHS is required for predator-prey simulations.)

popsim.c, popsim.h Routines to calculate total population biomass or, in the case of fish, the total biomass and YOY biomass. Also routines to make projections forward of the lipid and structure in order to provide starting values for Newton's Method.

ppmain.c, ppmain.h Main loops for the predator-prey model

pred.c, pred.h Sequential and Fish_On_All predation routines for predator-prey

reports.c, reports.h Reporting functions

shared_memory.c, shared_memory.h Shared memory message-passing library which I developed based on SYSV shared-memory structures.

shared_memory_clear.c, shared_memory_clear.h Frees shared-memory locations if incomplete shutdown.

timing.c, timing.h Wall and processor timing routines and elapsed timing structures.

This may seem a complicated base of code. It does combine several models and can produce all of the different versions we utilize include the predator-prey models. (See the Appendix for additional development history.) This is much preferred over having several just slightly different versions of the same overall program. Further, being able to generate good sequential and parallel versions from the same codebase demonstrates that parallel coding does not have to be different, nor does one have to start over. As we added and tried different machines, features, scenarios, ideas, reporting mechanisms, etc., then the code base for our testbed examples was also expanded, so it would be smaller if we redeveloped just aiming at a single design.

Program Heritage and Improvements

As an aside, when we began this project there were several programs that were handed down through my predecessors. The basic program design ideas and implementation mostly had to be derived from

the code, papers, and folklore. In particular, I was concerned about the mathematical computations for both of the populations, hence the results presented in this thesis in the Appendix. These derivations have all been incorporated into the source code as comments, so that one can read through the source and know what is being done at each step.

These codes were not particularly handed down as sets and they were written in several languages. For instance, we have already talked about the difficulties encountered just trying to generate initial populations. I gathered together many sources and have stored them together as a CVS archive for comparison and archiving. For the most part we have produced another set of code in a different language, C, but we have unified all of the required functions and variations into a single codebase. As described earlier, for parallelization, having a good sequential version to start from and test against is helpful. The sequential and parallel versions of all of our programs are built from the same source. Similarly, knowing the differences and similarities between the population and predator-prey models helps in their mutual development.

2.5.2 Memory Structures

The development of the population models has kind of a strange loop. I actually wrote the predator-prey as C-only and then extracted the population models from it. In doing so, several of the developments I had made for the predator-prey model got carried back to the population models. Once the hybrid C/Fortran population codes were replaced, then we could allow the compilers to fully-optimize. (Optimization of mixed language programs often causes problems with linking.) This also allowed the more effective use of profiling tools among other benefits.

One of the bigger problems for the food web models was the management of the information for each of the populations. First, I developed file formats that allow us to specify multiple populations in the initialization files including all of their parameters and initial cohorts. The basic structure is diagrammed in Figure 2.9. This file structure is shared by the "restart" file which is compiled from all of the populations (unified across the nodes if in parallel) and output at the end of a simulation, so that the simulation can be continued if desired. The files are self-documenting in the sense that each parameter is commented and comments are preserved. Second, I developed structures to manage the populations in memory. These structures have the benefit that they can be stepped through by generic code that is the same no matter what the specific population with which it is working actually is. Each characteristic carries with it an identifier, so that the correct individual model routines are applied to it in order to advance it forward in time. The main memory structures for

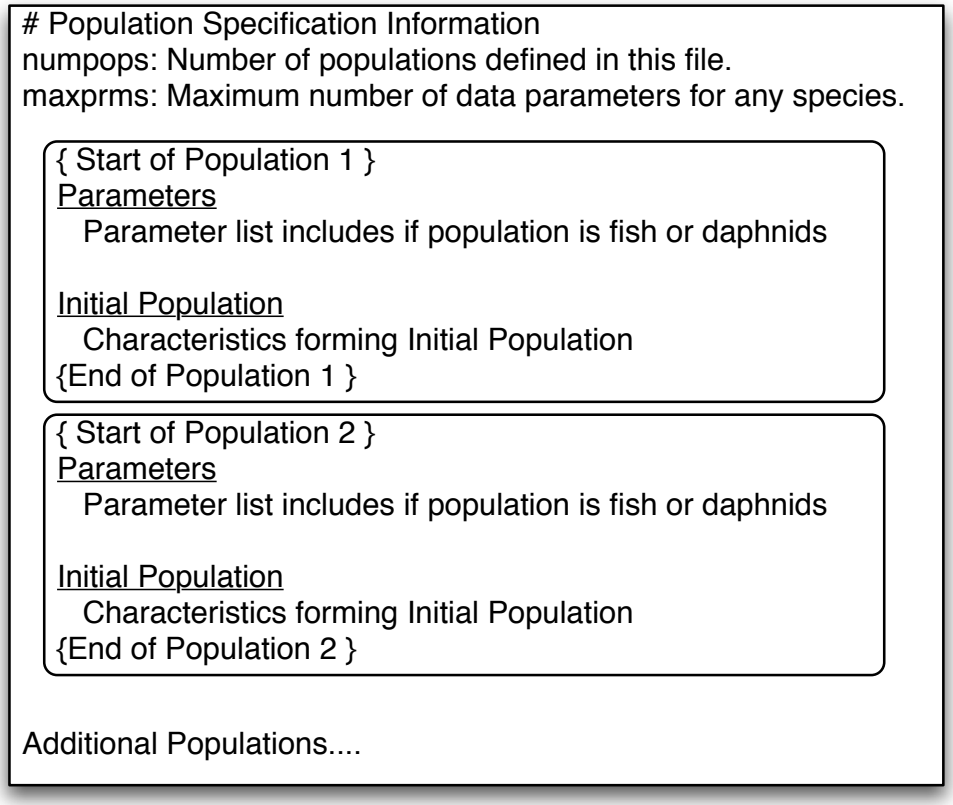


Figure 2.9: Initialization and Restart File Format

the multi-populations are diagrammed in Figure 2.10. Note that the characteristics are actually stored into stacks split out by ecotype. (They started out stacks, but predation calculations could be simplified by changing them to double-linked stacks which are termed dequeues.) So if there are 125 ecotypes for a population, then there would be 125 "ecostacks" allocated and the initial population would be sorted into these bins. As new cohorts are created, then they are pushed onto the top of the appropriate ecostack and nearly so by size. (Variation of initial lipid per egg prevents equivalence between sorting by age and size.)

The ability to simulate multiple populations simultaneously in the population model may seem esoteric. But one could conceive of two different species of *Daphnia*, parameterized to different size ranges, run simultaneously and coupled through density-dependent effects. We later propose just such an extension for the predator-prey model to make a simple food-web. It does introduce the requirement that all of the major routines, especially those for output, are multi-population aware in the sense that separate outputs are compiled and written for each of the populations. Also, the memory structures being unified between the population and predator-prey models helped with consistent handling and debugging.

2.6 Load Balancing

On our modern computer system, small imbalances of load for our style of simulations has become of no significance. An imbalance of 1000 cohorts adds maybe 5 seconds to the execution time. It is much better to let the processors work through their local populations without requiring them to pause, count, and shuttle a few cohorts around. Equally significant to consider is that as the number of processors increases, then the amount of time to load balance also increases.

Although Load Balancing has diminished in its original role, its occupies a useful position and function in the run loop. If for some reason we want to add additional computational resources as a simulation progresses, then the load balance module could be invoked in order to populate the new processor and reduce the load on the other processors. By reducing the load per processor, then perhaps they can operate out of cache rather than main memory. This is considered in the next chapter. Also, as the execution time per time step of one cohort has decreased, then the amount of imbalance that can be tolerated has increased. Thus we can be less strict about the definition of balance. This is also considered in the next chapter.

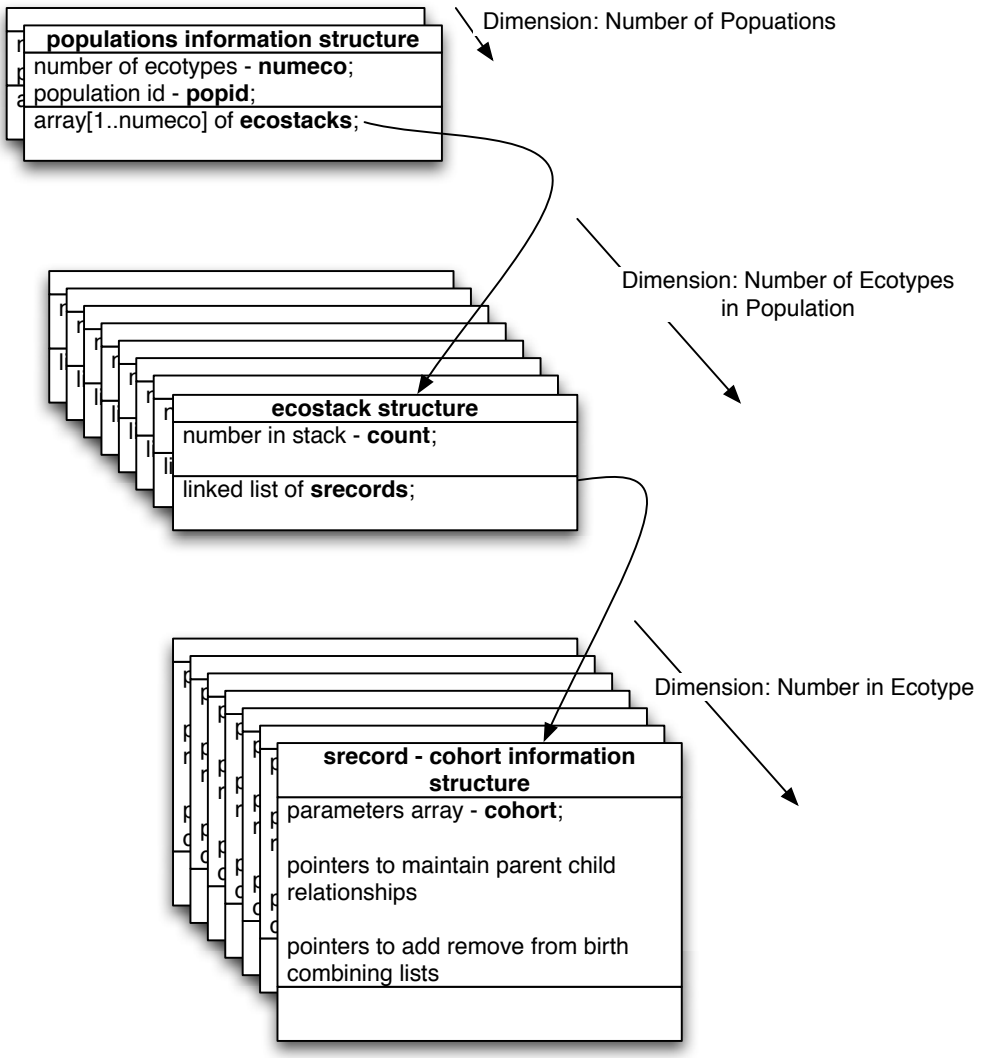


Figure 2.10: Population Memory Structures

2.7 Global and Local Birth Combining

The subject of births and the combining of the populations (i.e. summing ρ) of newborn characteristics has already arisen several times and is a unique aspect of individual-based ecology models. Birth combining was our biggest and most persistent challenge when parallelizing these models. It seemed that no matter what direction we took toward parallelization, we ran into birth combining issues. This is a point where the individual model definitely affected our choices. The easiest way around was for us to "parallelize by ecotype" as we did initially. For a sequential model where all of the characteristics are listed in a single array, then birth combining can be performed exactly and without much additional effort. It was done in order to keep the size of the problem tractable for regular computers.

Combining does compromise the pure mathematical concept of the method of characteristics, so we considered that maybe combining was not something we should do in any case. It can be further argued that this will produce a "more detailed" study that is not compromised by random losses or gains of biomass at the time of births. The reason for the random gains and losses of biomass at birth times is because the lipid allocated per egg varies in the individual model depending on the particular parent's lipid stores at the time of reproduction. When the newborns are combined, then the *first* lipid value is the one carried forward, so potentially some lipid biomass may be gained or lost from the population, thus violating conservation of biomass. Further, *first* is not well-defined; it is just which parent was processed first. We had to handle this problem, because in a parallel simulation, there are now potentially several *first* parents; one on each processor.

We thought perhaps not combining births was a pathway and an opportunity for parallelization. Since we have large, parallel computers at our disposal, then maybe we should throw out any such compromises made merely to enable of sequential computation. It is also closer to the mathematical concepts and conserves biomass. But, as we will demonstrate in this section, not combining births causes many more problems. We also further demonstrate that global combining is a requirement for effective parallelization in our ecology models, since we model a rapidly reproducing populations.

Birth Combining's main effect is on the value of ρ, the population density associated with a characteristic. Recall the renewal equation (Equation 1.7), which determines the value of ρ on a newborn characteristic (age = 0) with given initial lipid and structure values:

$$\rho(t, 0, m_{L0}, m_{S0}) = \int_0^\infty \int_0^\infty \int_0^\infty \beta(t, a, m_{L0}, m_{S0}, m_L, m_S, \rho)\rho(t, a, m_L, m_S) da dm_L dm_S$$

Table 2.4: Execution Time and Workloads for 4-Processor Simulation with No Birth Combining or Minimum Threshold

Simulation Output Time	Execution Time (s)	Maximum Cohorts Per Processor
9.95	4	18,090
19.95	26	86,495
29.95	132	420,605
39.95	652	2,130,980

Further recall, that along characteristic curves, equation 1.13 describes the dynamics of ρ as a decreasing exponential function with exponent determined by the mortality function μ:

$$\frac{dn}{dt} = -\mu(t, a, m_L, m_S, n)n. \tag{2.1}$$

Mathematically, the linearity of equation 2.1 allows us to split and combine characteristics without consequence. The problem arises in the numerical simulation of these equations. When the value of ρ decreases below a fixed threshold value ρ_{min}, then it is considered to be insignificant and is eliminated from the population. This is done in order to keep the population culled and to prevent the simulation from becoming burdened by lots of insignificant characteristics. Since this is a decreasing exponential, all characteristics will eventually fall below any fixed, positive threshold. But, as shown in Figure 2.11 with the threshold effect it does cause the problem of advancing the time of removal of the two uncombined characteristics. This is pictured for two characteristics, but can be extended to any number of characteristics. This is not a problem by itself, because the value of ρ_{min} is chosen so that only insignificant characteristics are removed, but it can become a problem if large segments of the simulated population somehow becomes "insignificant" by this criteria.

2.7.1 No Birth Combining

If one eliminates the threshold, then the problem quickly becomes intractable even by our modern, parallel computers. The timing and workload data from such a simulation is given in Table 2.4. This data is for a 4-processor simulation, so the cohort-steps workload is over 8M within 800 time steps. I had to stop the simulation at this point, because each process was consuming over 600M of memory which was approaching the limits of my machine. This clearly demonstrates that some method of culling characteristics is required.

Now, examining the case of no birth combining, but retaining some fixed, positive value for the threshold ρ_{min}, what happens? Do the total biomasses of the combined and no-combined simulations

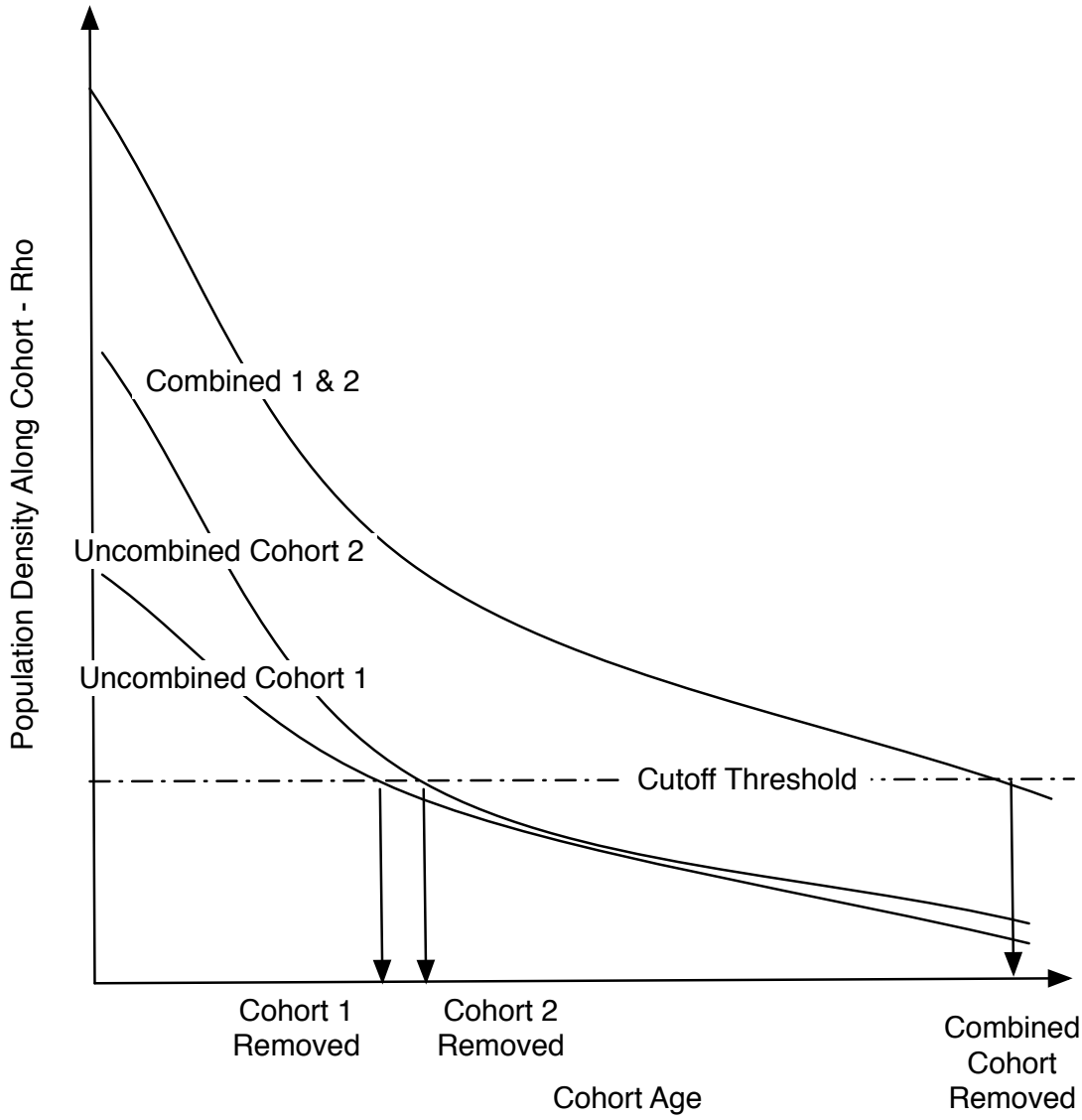


Figure 2.11: Removal Times for Two Uncombined Characteristics versus One Combined

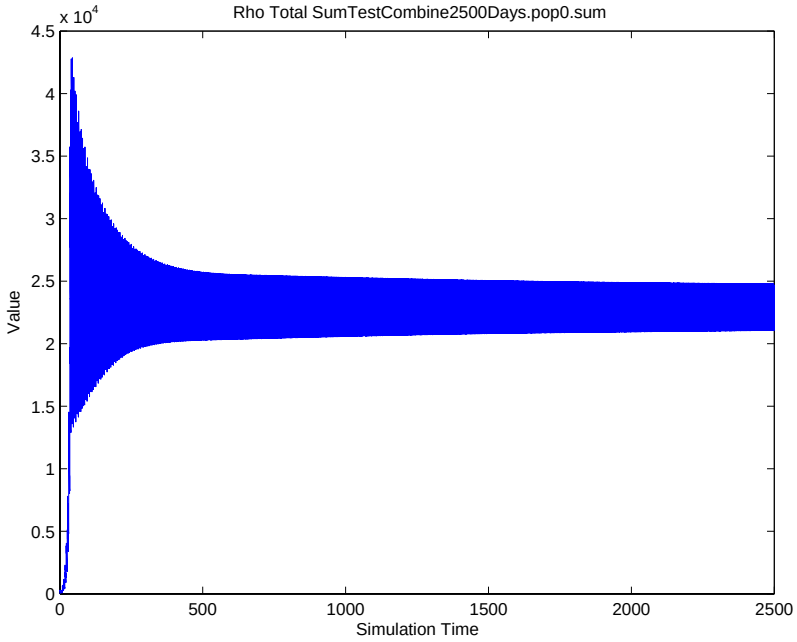


Figure 2.12: Total Population Density for a 2500-Day Run - Normal Birth Combining

match? Figure 2.12 gives a graph of the total biomass for a simple, single ecotype population started from eggs utilizing birth combining. This 2500-day simulation runs in about 20 seconds. In contrast, Figure 2.13 gives the same graph for the same population, with the only difference being that birth combining is not utilized — every newborn generates a new characteristic. They hardly look alike, but they are in fact identical for about the first 33 days of simulation. That is the point that deaths start to occur in the uncombined population that are not matched in the combined population. One should also note the instability exhibited in Figure 2.13. The values of ρ are much more erratic compared to the combined case. What is occurring in this case is that large portions of the population are being eliminated shortly after birth because of dropping below the threshold as evidenced in Figure 2.14.

2.7.2 Local versus Global Birth Combining

With the necessity of combining established, what further did we find with regards to birth combining? Table 2.1 in our paper at the start of this chapter compares global and local combining in parallel simulation. We started with the local combine case because it was a much simpler path to parallelization, eliminates another synchronization point, and allowed us to more closely match the sequential simulation's behavior. We concluded in the paper that global combine was the fastest

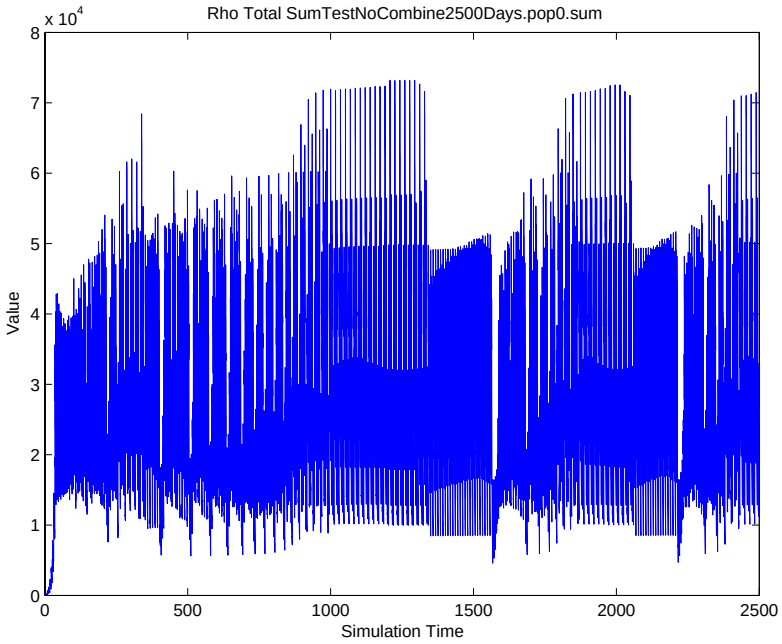


Figure 2.13: Total Population Density for a 2500-Day Run - No Birth Combining

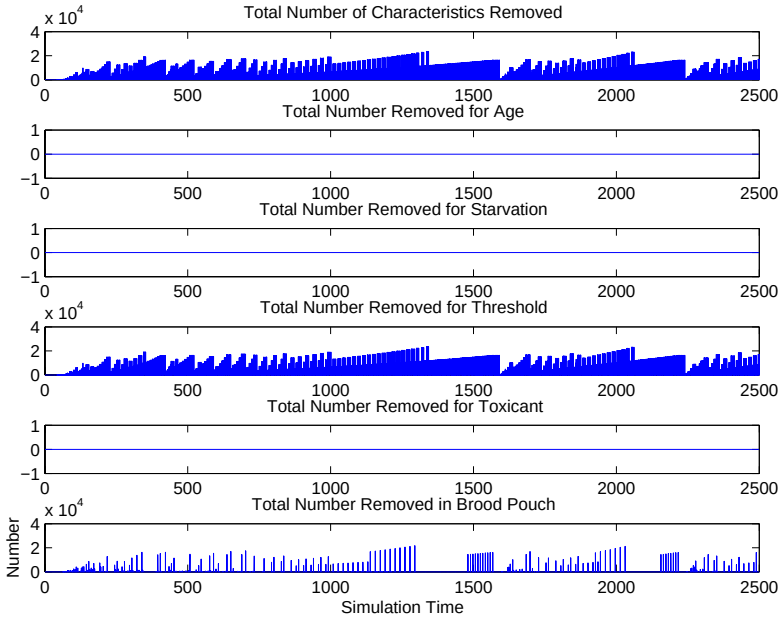


Figure 2.14: Cohorts Removed from Population for No Birth Combining Case

Table 2.5: Comparing Local and Global Combine Execution Times and Workloads

	Local Combine		Workload Ratio	Global Combine	
Processors	cohort-steps	execution (s)	Loc to Seq	cohort-steps	execution (s)
1 (seq)				6,054,622	9
2	10,073,100	8.7	1.66	6,054,525	7.3
4	16,466,202	12.0	2.72	6,054,497	12.1
6	21,834,464	18.0	3.61	6,054,483	23.8
8	26,439,529	25.4	4.37	6,054,481	36.3

option because required work load was minimized, which more than made up for the time required in extra communication. But now, given that processors are so much faster, one might be inclined to think that the best choice for modern parallel computers would be local combine.

One of the tables eliminated from the paper listed the cohort-steps required by the local combine case compared to the global combine. Such a table for our test system is listed in Table 2.5. Results are listed for a simple population consisting only of a single ecotype, so that the effects of interacting ecotypes is eliminated. (A simple population was chosen for illustration, rather than for parallel advantage.) Observe the ratio of the numbers of cohorts simulated by the local combine version with the number required by the sequential simulation. This ratio increases and approaches equality with the number of processors. How quickly it approaches depends on the birth period. Thus, for a rebalanced, local combine simulation, workload scales at the same pace as the number of processors, erasing any advantage gained by distributing the population.

Observation 2.7.1 *For a local-combined, rebalanced population simulation, where the population has a short birth period relative to the length of the simulation, then the workload measured in cohort-steps scales by n where n equals the number of processors.*

Proof Given n and a single cohort with density ρ, duplicate the cohort into n cohorts, assigning to each density ρ/n. Distribute each of these cohorts to a separate processor (rebalancing). When simulated, then each will produce progeny at the same times because births are not affected by density. These are not combined (local combine), therefore the number of cohorts is scaled by n. This process can be repeated for all cohorts in the initial population.

The action of splitting a cohort is mimicked by births. Therefore if the birth period is relatively short, then the splitting and dispersal to nodes will be effectively carried out. ∎

With this in mind, then the advantages of local combining evaporates for fast-reproducing populations like *Daphnia*, because the workload will just scale at the same rate at which processors are

added. There is a delay for larger numbers of processors as it takes time to generate the progeny which eventually are distributed to each processor which is observed in Table 2.5, but once a generating characteristic gets placed on a processor, then it will start to replicate the population at the same rate as the original. The delay in this table is illustrated by looking at the slight (20-30 simulation day) delay between the numbers for the study given. Because of this result, local combining was not studied further and global combining is turned on in all of our parallel calculations.

2.7.3 Global (Parallel) Birth Combining

Finally, this section is closed with a description of the parallel Global Combine process. Our Global Combine algorithm depends on our Local Combine algorithm which is executed by each processor on its local population. The Local Combine algorithm carries out birth combining by keeping a stack of all newborn cohorts. This stack is scanned when a new birth is generated. If there is a cohort already in the stack that has matching ecotype, then the ρ values are summed and the second cohort's data is unallocated. (Both parent cohorts point to the same offspring cohort in this case; they own a percentage of the offspring cohort. This percentage is used to scale the effect of a parent on the combined offspring cohort.) Otherwise, the new cohort is added to the stack. This stack is local to each processor. Note that the value of initial lipid for the newborn characteristics can vary, depending on which parent is processed first.

Algorithm GLOBAL_COMBINE

begin

0. Initialize the data.

$i \leftarrow$ processor-id;

$p \leftarrow$ number of processors;

$e \leftarrow$ number of ecotypes;

$locbirths \leftarrow$ local births stack;

Let $lipidmass[1...e]$ and $childdensity[1...e \cdot p]$ be arrays of floats.

1. Scan locbirths (if any) and prepare for reduction. Note that locbirths has e or fewer elements, since it is a result of the local combine algorithm.

foreach ($lb \leftarrow locbirths$)

$lipidmass[ecotype(lb)] \leftarrow$ lipid(lb);

$childdensity[i][ecotype(lb)] \leftarrow$ population density(lb);

2. Reduce across nodes. Note that SUM is really a MAX because all rows in childdensity are zero except for i-th

Let $lipidmassr[1...e]$ and $childdensityr[1...e \cdot p]$ be arrays of floats.

$lipidmassr \leftarrow parallelReduction(MAX, lipidmass)$;

$childdensityr \leftarrow parallelReduction(SUM, lipidmass)$;

3. Determine total births for each ecotype.

Let $totaldensity[1...e]$ be an array of floats.

for $(j \leftarrow 1$ to $e)$ $totaldensity[j] = \sum_{k=1}^{p} childdensityr[j][k]$;

4. Decide where combined cohort will reside in an using dest, starting always with the first processor.

$dest \leftarrow -1$;

$dest \leftarrow (dest + 1)$ mod p;

for $(j \leftarrow 1$ to $e)$

while $(childdensityr[dest][j]$ equals $0)$ $dest \leftarrow (dest + 1)$ mod p;

if (i equals $dest$)

update local newborn of ecotype j to values in $totaldensity$ and $lipidmassr$.

else delete local newborn of ecotype j.

end

After each local population has been advanced forward in time and any local births determined and combined by the local combine algorithm, then the GLOBAL_COMBINE algorithm is executed collectively (even if no local births are recorded). This algorithm must be executed every time step. This combines across each local population the births into one set of at most *number of ecotypes* elements. Thus at each time step, at most *number of ecotypes* cohorts can be generated. An upper bound on the maximum work load for a given simulation can be computed from this. As we have already seen, birth events are not equally likely, so this upper bound is not achieved. An interesting feature of the algorithm is the use of a 2D array in order to receive both node location and total population information in one reduction, so that the determination of the node retaining the newborn can be determined in parallel. Note that the maximum value at the time step for lipid per egg is the one chosen to be carried forward. This is still not completely deterministic, because the local combine algorithm does not deterministically determine this value.

2.8 Performance

In this section, we demonstrate the performance of our design by reporting the execution times for a 2500-day simulation using the same *Daphnia* population shared with the predator-prey model. By varying the cutoff value under which a cohort is removed from the population, we can control the work level without affecting the predicted outcome of the simulation. Charts are presented for cutoff values of 0.1, 0.01, 0.001, and 0.0001. The total workload increased from 127 million to 886 million across these four values. (This cutoff value was set to 0.000001 for the predator-prey populations I was given. For this value the workload would climb into the hundreds of thousands of cohorts and billions cohort-steps for a calculation. Until I found this I thought my programs had a bug that I could not locate.) We list for 1 (sequential), 2, 4, 6, and 8 processors. We list total times in seconds for the total execution time, simulation time (actual calculation of the model), communication time (parallel communication), reporting, and rebalancing. The workload values are given in number of characteristics simulated over the simulation. The speedup is the sequential execution time divided by the execution time for the parallel version. If the speedup is greater than the number of processors then it is said to be super-scalar.

We list each table twice. We used both the gcc compiler that is standard with all Macs and the Intel C++ Optimizing Compiler, v10.0 which is a commercial product. At first glance it appears that the results from the Intel compiler is not as good as from the gcc compiler, because the speedup obtained are smaller. But, the total execution times are typically about half for the Intel versions, so there is not as much room for improvement in terms of speedup. The Intel compiler adds so much performance by introducing automatic vectorization to the sequential model and adds optimized math libraries.

Each performance run was made with a moderate level of output. Each simulation produced about 700 Mb of output files. Previously, we did not include output in our published results because it evaporated much of our performance gains. For several of our runs, the reporting time exceeds the simulation time. There is often some parallel advantage during reporting, because for several types of reports the values can be partially computed on each processor in parallel and then reduced to obtain the final values recorded by the host processor. Reporting does slow down computation because the host node is busy with I/O while the compute nodes are not. I attempted to develop several parallel output options and other MPI-2 methods to try to decrease the parallel reporting time further, but found that any reductions to the execution time were overwhelmed by the time it took in post-processing to sum the data together. Sylvester in this thesis did use a post-processing

Table 2.6: Performance Chart, GCC, Cutoff = 0.1

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	183.00	130.00	0.00	53.00	0.00	127,739,252	
2	107.94	66.72	0.55	38.43	0.09	62,501,453	1.70
4	77.80	37.45	2.85	35.36	0.13	31,339,096	2.35
6	75.10	29.77	3.22	34.71	0.39	21,957,900	2.44
8	85.99	30.53	5.14	40.36	0.67	16,864,908	2.13

Table 2.7: Performance Chart, Intel, Cutoff = 0.1

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	88.00	56.00	0.00	31.00	0.00	127,739,238	
2	60.93	27.45	0.90	31.91	0.09	62,501,459	1.44
4	52.49	16.77	1.65	31.91	0.19	31,339,097	1.68
6	59.06	15.70	3.07	33.47	0.52	21,957,901	1.49
8	75.88	20.05	4.46	40.34	0.80	16,864,909	1.16

approach with reporting (Sylvester, 1995), because access to local drives was so much faster than reducing across the network of workstations.

These data presented in Tables 2.6 to 2.13 show an interesting connection between the execution times and the cutoff value for the sequential versions. As the workload doubles, the execution times of the sequential process more than double. Between cutoff levels of 0.01 and 0.001, the workload doubled, but execution time increased by more than 2.5x. The data also show that the lowest execution times are typically for 6 processors. Superscalar speedup was attained for two processors and nearly so for 4 and 6 processors. With 8 processors the times rarely improved those for 6 processors. As the workload increased, then the performance for the parallel programs continued to improve. With lower workload, then the overall performance gain was marginally above 1; the simulation times did scale with increasing numbers of processors, but the reporting I/O masked the performance gains.

2.9 Conclusions

This project occurred at an opportune time. The major issues we were dealing before with trying to get parallel performance from supercomputers are now the current major issues being dealt with for our personal computers. Gaining parallel performance from multi-core, desktop computers is a current primary computational objective (Merritt, 2007; Reinders, 2007; Marowka, 2007; Sutter, 2005). We demonstrated that the current parallel programming software libraries can be utilized to

Table 2.8: Performance Chart, GCC, Cutoff = 0.01

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	423.00	334.00	0.00	87.00	0.00	316,865,868	
2	229.43	166.54	2.04	61.17	0.06	158,835,565	1.84
4	139.83	86.45	4.59	46.54	0.16	77,585,900	3.03
6	115.31	63.33	4.04	42.22	0.38	54,553,754	3.67
8	123.77	58.58	7.77	47.63	0.79	40,380,344	3.42

Table 2.9: Performance Chart, Intel, Cutoff = 0.01

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	190.00	144.00	0.00	46.00	0.00	316,865,295	
2	109.79	68.26	1.53	40.17	0.06	158,835,305	1.73
4	77.54	36.72	2.49	35.95	0.18	77,585,901	2.45
6	77.58	30.50	3.85	36.89	0.40	54,481,805	2.45
8	103.56	36.20	7.72	49.89	0.86	40,380,344	1.83

Table 2.10: Performance Chart, GCC, Cutoff = 0.001

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	1065.00	849.00	0.00	215.00	0.00	720,010,017	
2	506.63	391.41	9.08	113.11	0.24	364,440,325	2.10
4	272.29	190.98	8.53	70.83	0.42	176,704,295	3.91
6	208.07	136.99	13.56	59.99	3.03	126,089,025	5.12
8	207.19	115.70	12.99	63.24	1.70	88,712,471	5.14

Table 2.11: Performance Chart, Intel, Cutoff = 0.001

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	521.00	384.00	0.00	135.00	0.00	720,009,986	
2	244.50	168.05	6.00	74.54	0.25	364,440,315	2.13
4	155.60	79.98	4.31	69.35	0.43	176,704,295	3.35
6	122.80	58.81	8.10	53.23	3.11	126,089,025	4.24
8	155.54	69.97	13.26	61.16	1.74	88,712,468	3.35

Table 2.12: Performance Chart, GCC, Cutoff = 0.0001

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	1425.00	1097.00	0.00	328.00	0.00	886,061,646	
2	634.08	489.96	7.40	140.93	0.23	446,954,725	2.25
4	328.22	234.59	9.44	82.16	0.44	218,018,844	4.34
6	248.89	167.44	14.95	68.30	4.02	153,695,315	5.73
8	260.88	151.68	27.78	79.10	3.46	114,387,245	5.46

Table 2.13: Performance Chart, Intel, Cutoff = 0.0001

Processors	Total	Simulation	Comm	Report	Rebalance	Workload	Speedup
1	763.00	560.00	0.00	201.00	0.00	886,061,662	
2	298.05	217.15	8.52	77.33	0.24	44,695,471	2.56
4	154.13	99.24	4.97	47.96	0.44	21,801,829	4.95
6	131.64	73.85	9.86	44.52	4.06	15,369,530	5.80
8	195.18	93.13	23.35	73.97	3.51	11,438,724	3.91

extract scalar and superscalar speedup for individual-based, population models without having to resort to lower-level programming as we had to before. So now the focus is more on the science and less on the low-level details of the computer on which it is being simulated. The standardization imposed by the near universal acceptance of MPI has also solved the portability problems we had before that lower-level solutions caused. Further as familiarity and experience is gained with the new hardware, then the underlying OS continues to be improved which produced further performance gains.

We have demonstrated that performance can be gained through parallel techniques; and, in cases of sufficient workload, super-scalar performance gains can be had. As part of the development of our techniques we simulated much larger problems. For example, one stress test simulated 729 ecotypes (=9^3) with all birth classes filled. This induced a load of 58,320 initial characteristics that grew to about 300,000 before the diversity was driven down by the dominant ecotype. This was an overwhelming simulation for sequential execution, but was completed quickly (about 20 minutes) in parallel. The no-combine cases in this chapter also exhibit the robustness of the parallel techniques to allow us to research ideas we would not have looked at before. So parallel techniques allow for increased performance and increased problem sizes and level-of-detail.

Are parallel techniques worth the effort? First, they are a useful tool and an applicable option when needed and are worth keeping in mind during development. When sufficient workload is available to benefit from parallel execution, then we demonstrated superscalar and near scalar speedup. The overall design of recognizing the correct units of work, decoupling the work units so that they can be managed and completed independently, and then executing the resulting silo of work via whatever means available is the fundamental contribution of this work. It has a direct mapping into current parallelization efforts for agent-based simulations (Reynolds, 2006; Quinn et al., 2003). Sequential versions were improved themselves by such a redesign as we developed the parallel versions. Second, we demonstrate, by producing both from the same codebase, that parallel versions do not require starting over, but can be created as extensions of a well-designed sequential version.

Third, we also demonstrate that the compiler can add performance through vectorization without additional cost. Furthermore, other benefits such as reporting are demonstrated.

In Section 2.2.5, Initial Conclusions, we found previously that a fine-grained distribution of the work load with global combining and frequent rebalancing was required to obtain speedup. We find now that load balancing is of almost no importance for its original purpose of efficiently utilizing all of the processors, but is usable for matching and maintaining the parallel resources to the dynamic size of the computational problem. In particular, this application can be used to maintain the local population distributions so that superscalar speedup can be obtained. In regards to global versus local birth combining, we find that global combining is the only valid choice for rapidly reproducing populations like *Daphnia*, because the workload for local birth combining scales directly with the number of parallel processes. Since the workload is inversely tied to the reproductive period, then simulations that are large enough to benefit from parallel computation will be primarily applicable to rapidly reproducing populations. We observe that without global combine, then workload scales directly with the number of processors for such populations if the workload is rebalanced.

We considered our original efforts to parallelize a generic structured population model to be part of the larger mission of determining efficient algorithms to simulate community models, which is the subject taken up in the next chapter.

Chapter 3

Structured Predator-Prey Feeding Algorithms and Parallelization

3.1 Introduction

In this chapter we develop two different designs for parallelizing individual-based community models. The testbed model is a predator-prey model composed of one instance each of the fish and *Daphnia* population models joined by a predation module. The two parallel designs, referred to herein as the *Pure Parallel* and the *Fish-on-All* algorithms, are new directions explored for predator-prey models. The predation module we utilize is size-based and models competition for resources by sharing proportional to body mass. The general effects on parallelization of alternative predation expressions are diagrammed and explored. The predation module and individual-based nature of both populations introduces several new interdependencies and relationships that require significant efforts to decouple. Our parallel algorithms involve decoupling the local populations and is essentially an information flow problem. Following a detailed description of the predation and competition models, we develop a complete listing of the interdependencies and stages of predation and mortality assessment. The Pure Parallel algorithm solves the information flow problem by collecting and distributing tables of information to all of the processors, so that a "pseudo-fish" can stand in the stead of a predator located elsewhere on the machine. This algorithm is generally applicable, handles large populations, features *en masse* feeding calculations, and, as we will demonstrate, exhibits near scalar performance scaling with increasing numbers of processors. But, it is complicated, requires a distinct predation module separate from the sequential version, and imposes some stringent

requirements that we had to retroactively add to our testbed models. These requirements arise from the physiology of the *Daphnia* population.

Addressing the deficiencies of the first algorithm leads to a second algorithm, Fish-on-All, that takes advantage of the inter-trophic structure of our predator-prey model by focusing on the much larger prey population. It is an extension of the sequential model and integrates directly into the source code base. It decouples the workload by duplication of the predator population on all prey nodes. All nodes begin with an identical copy of the fish population that is kept in coordination by duplicate calculations on all nodes. This reduces the parallel communications to only a few reductions similar to the population model's biomass requirements. We will show that the population model parallel performance is mimicked by this algorithm.

Body size is one of the most important components affecting physiological processes, such as metabolism and fecundity, and ecological interactions, such as predation risk and foraging. With fish consuming their prey entire, then the morphology of the gape determines the prey items that can be utilized. Incorporating such a component into model studies requires physiologically-structured population modeling techniques to connect the individual model to the population effects (De Roos and Persson, 2001). A similar model to the one presented herein has recently been used to predict alternative states and to provide guidance on restoration of top predator fish species that have been depleted by over-harvesting (Persson et al., 2007). The recovery of the top predator species is slowed because their removal allowed many stunted individuals with lower fecundity to fill up the prey-species classes. This lowers the resource available to the top-level predator, whose preferred prey are the young because they are easier to pursue, capture, and ingest whole. This indicates the importance of considering structured predation and predator-prey models. Additional motivation for the use of structured predation models is to understand the effects and flow of a lipophilic toxicant through a community via uptake in the resource, which is most directly modeled by the use of a structured model.

3.2 Structured Predation and Mortality Model

The first challenge is the formulation of expressions for predation that produce a resource level compatible with the population model for the fish and the corresponding per capita mortality rate that is compatible with the *Daphnia* models. These will be the first items taken up in this chapter. Reduction of these theoretical expressions to computational elements will follow.

At first thought, one might envision a one-to-one, this-fish-eats-that-daphnid kind of model. But emphasis must be placed on the fact that the rates are defined at the population-level, not the individual-level. Additionally, since these models do not incorporate a spatial-component, then we do not have any sense of this fish is close to that daphnid, so there is no natural way to proceed to a one-to-one expression of a predation. This lack of spatiality has the further consequence that it requires all of the predators to know about all of the potential prey items. Vice-versa, each prey cohort is potentially affected by every predator cohort. The size and complex exchange of information implicit because of this entire-to-entire correlation induces computational complexity, which is magnified when attempting parallelization.

Papers previously published related to our structured predator-prey models are: Hallam et al. (1992a), Jaworska et al. (1995), and Henson (1994). In particular, Henson (1994) presents similar derivations of shared feeding mechanisms for the predator-prey model in her sections on Communities. These published articles focus on ecotoxicology and toxicant flow through a community model, but not on the extinction and persistence relationships between the populations.

And advantage of the individual-based approach to predation is that the basis of community dynamics is contained in the individual model. The feeding mechanism of an individual predator on a prey population likewise composed of individuals can be directly observed and modeled rather than trying to apply aggregated mechanisms. Once the feeding mechanism is prescribed, then the resource uptake and growth of the individual predator are determined and the mortality caused by this predator on the prey population can be determined. The predation function expresses the resource density to which an individual fish cohort responds. Likewise, for every expression of predation, there is a calculation of the corresponding mortality. Thus each set of expressions is composed of a pair of equations.

3.2.1 New Concepts

Before we get into the specific expressions we utilize for our community models, a couple of new concepts are introduced.

Gape Size

To moderate prey choice for our size-structured models, we utilize the concept of gape size. Each fish has a certain prey window which gives a range of sizes of prey upon which it can forage. This prey window is determined from predator's overall length and has both upper and lower bounds.

The idea behind the lower bound is that prey items that are too small will either be ignored or will pass through the gills. Likewise, prey items that are too large will either not be pursued or will not fit into the fish's mouth. For our models, these bounds are taken to be linear functions of the fish's length; this is consistent with the literature (Gill, 2003). Note that the lengths of both the predators and the prey are required for this formulation. Also recall that length in our models is determined using a non-decreasing function based on the mass of protected structure.

One can think of Gape Size acting as a scaling between the populations that brings them into compatibility. Through varying gape size values, then we could easily cause the two populations to be disjoint in that no member of the prey population meets the gape size criteria for predation. A conceptual diagram of the Gape Size criteria is given in Figure 3.1. The range of prey items available to the larger fish B is larger than, but also partially overlaps with, the range of prey items available to Fish A. How to handle the competition for shared resources must be specified by the model. The relation can be inverted to give the range of possible fish lengths that can predate upon a specific daphnid. This is referred to as the *Inverse Prey Window* in this paper. Every fish that falls in this window will be presented this daphnid as part of its resource level. To simplify the mathematical expressions, through the allometric relationship between length and mass, the gape size will be converted to a prey window determined by masses of the fish and daphnids. For simulation we compute and store the lengths and endpoints required for evaluation of the prey window rather than utilize masses.

Effective Volumes

Another concept that we introduce is that of effective volumes in order to convert from numerical density to the resource volume densities (g/cm^3) for feeding. In the description of the population models there is not an explicit volume associated with the populations. While we can simulate large numbers of cohorts, ecotypes, and populations of daphnids, there is not a sense in which the daphnids take up more "room" or become more crowded. As shown earlier, density-dependent mortality does impose an optimal biomass on it and therefore implicitly has a density concept behind it, but an operating volume has never been defined. The fish model was developed with grams per cm^3 density of resource. To convert to volume density for predation, V_D is introduced and defined to be the operating volume for the prey population. Similarly, V_F is defined to be the operating volume for the predators. The ratio of V_D to V_F is used to scale the predation mortality. V_D is conceptually presumed to be smaller than or equal to V_F, but does not necessarily have to be. The idea is that the daphnid population is replicated throughout the operating volume of the fish population.

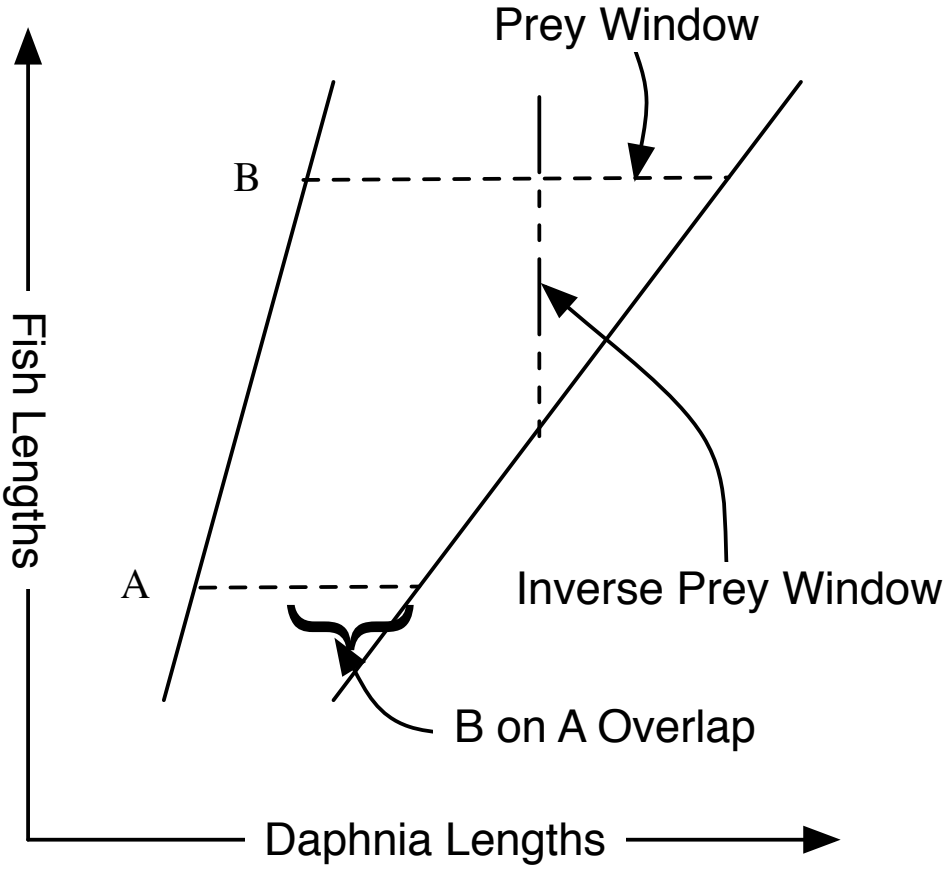


Figure 3.1: Illustration of Gape Size/Prey Window and Inverse Prey Window

Overall, there are only four new parameters introduced for the community model, further indicating that most of the structure required is already specified in the population and individual models. The parameters are the

1. Two parameters, *kmin* and *kmax*, which are used to determine minimum and maximum prey size,

2. A parameter, termed $rscale = 1/V_D$, used to scale the prey population to a density and to scale the volume up to that of the fish, and

3. A parameter, termed $fscale = V_D/V_F$, used to scale back down the prey mortality.

3.2.2 Predation Formulations

To provide a resource level to the predating population, we need to formulate an expression for the density of resource to which the predating individual responds. The functional response (equation 1.22 for fish) is used to convert from the resource density to the actual grams of resource consumed by a fish. The total grams of resource consumed, must then be converted back to mortality expressions assessed against the prey items consumed. In a broad categorization:

1. Effective density = Actual Density of Resource. E.g. filter feeders which encounter resource, but do not actively pursue it.

2. Effective density $\leq$ Actual Density. There is some level of partitioning or sharing of food among the predators. Two examples are:

   (a) Dominant Feeders — Biggest organism gets all it needs, next biggest, etc. As an illustration of the potential effects of this method of feeding, see Figure 3.2

   (b) Proportional Partitioning — Food is partitioned according to the weight of the predator in proportion to the total weight of all of the predators.

As the simplest example, if ρ is the number of prey individuals in volume V, then a predator responds to a resource density of ρ/V.

In this section we derive the predation and mortality expressions we used for our predator-prey studies. We partition resource proportionally but do not force a hierarchy based on size. In other words, all fish, when competing for a shared resource will get some portion. In our model, the resource values for the *Daphnia* are still assumed to be constant, so they continue to grow and

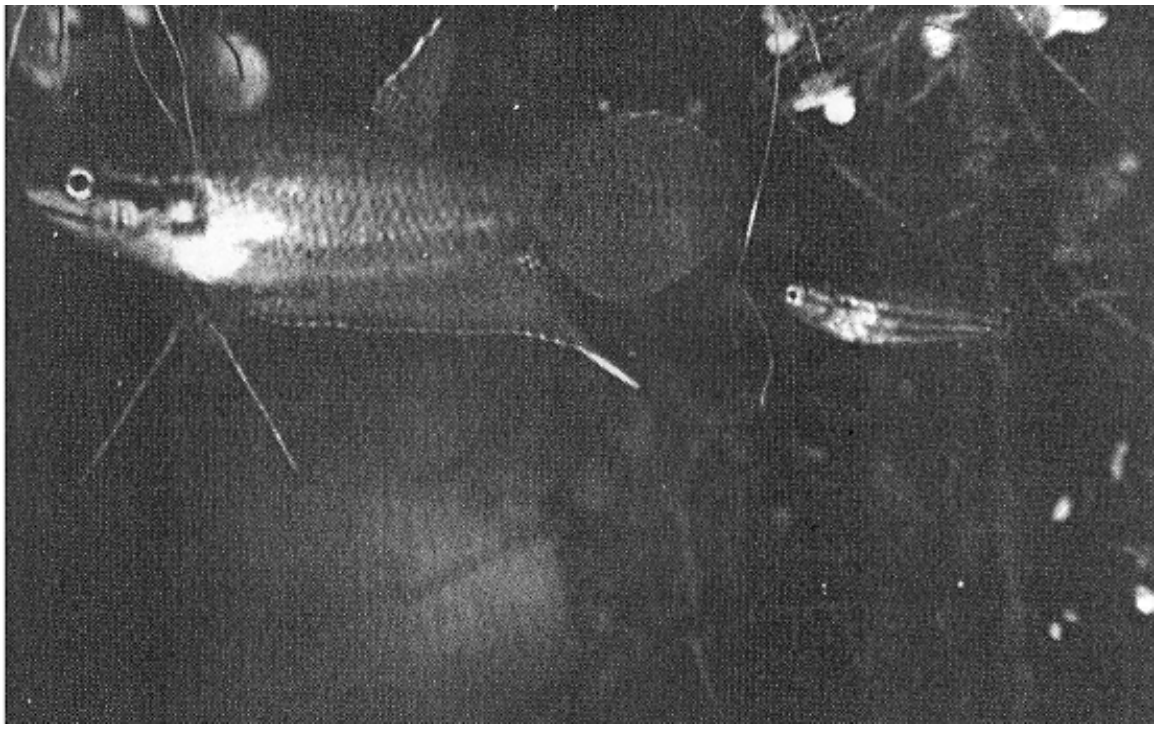


Figure 3.2: This is a picture of two fish from same brood. It is an illustration of the effects of different feeding methods. The food was given in large chunks, so tiny initial size differences were amplified by dominant feeding. (Credit: Kooijman (2000)[p. 21])

increase in size as before. Similar to what we have seen, the effect of predation mortality is only on the values of ρ along characteristics — their physical growth is not directly affected.

3.2.3 Continuous Predation Model

We model resource sharing for the individual fish to be the proportion of the mass of the fish to the total mass of the other fish which can also consume the same resource elements. Resource density level for fish is expressed in units of grams per cubic centimeter. Describing in words, the resource for a fish of mass m_F (sum of lipid and structure, g) is:

$$x_{m_F} = \frac{m_F}{\text{Total Mass of Competing Fish}} \cdot \frac{\text{Total Mass of Prey Items in Window}}{V_D}.$$

With the assumption that a fish of length L_F can only consume prey with sizes L_D in the range

$$10L_F k_{min} \leq L_D \leq 10L_F k_{max} \tag{3.1}$$

where k_{min} and k_{max} are fixed constants (recall unit conversion from cm to mm). Since length is assumed to be isometrically related to weight we can convert to terms of mass rather than calculating the lengths each time. Restating the prey window as

$$k_1 m_F \leq m_D \leq k_2 m_F$$

where m_F and m_D are the mass of the fish and a particular daphnid (sum of lipid and structure, converted to grams) , respectively, and $k_1 = k_{min}^3 \frac{a_D}{10a_F}$ and $k_2 = k_{max}^3 \frac{a_D}{10a_F}$, where a_F and a_D are the isometric constants relating length to weight for fish and *Daphnia*, respectively. Inverting this inequality gives the range of fish masses that can predate upon a daphnid of mass m_D. This range is

$$k_2^{-1} m_D \leq m_F \leq k_1^{-1} m_D$$

Using these inequalities to sum over the entire prey population yields the expression for the prey density experienced by a fish of mass m_F:

$$x_{m_F} = \frac{m_F}{V_D} \int_{k_1 m_F}^{k_2 m_F} \frac{m_D \int_0^\infty \rho_D(t, a, m_D) da}{\int_0^\infty \int_{m_D k_2^{-1}}^{m_D k_1^{-1}} m \rho_F(t, a, m) dm da} dm_D \tag{3.2}$$

where V_D is the control volume, and ρ_F and ρ_D are the density functions of fish and *Daphnia*, respectively. Note particularly the role of the inner integral in the denominator which yields the total mass of the fish that can predate upon a daphnid of mass m_D. This expression gives the resource density to which a particular fish responds. It is relatively simple and inexpensive to calculate as we will see later when this is split into discrete sums.

Now, for this given predator, a fish of mass m_F, the corresponding per capita mortality rate (grams consumed/gram of biomass) for a prey cohort of mass m_D in its prey window is computed by:

1. Calculate the functional response $f(x_{m_F})$ (Equation 1.22) which gives the total grams per day of prey eaten by a single predator of mass m_F. This converts from the resource density to the total mass consumption rate at this time step.

2. Multiply by the number of predators of mass m_F to get total grams per day eaten by all predators represented by this cohort at this time:

$$\int_0^\infty f(x_{m_F})\rho_F(t,a,m_F)da \tag{3.3}$$

3. Multiply by scaling factor (V_D/V_F) to scale mortality down to the prey population.

4. Divide by the total biomass of the prey population in order to get a per capita rate numbers per day.

The per capita mortality rate due to predation by this predator and all of its same-size cohorts is thus given by:

$$\mu_{\text{PRED}}(t,m_F) = \frac{V_D}{V_F}\frac{\int_0^\infty f(x_{m_F})\rho_F(t,a,m_F)da}{\int_0^\infty\int_0^\infty m_D\rho_D(t,a,m_D)dadm_D} \tag{3.4}$$

where $f(x_{m_F})$ is the total grams of prey eaten per unit time by a predator of mass m_F. Summing over all fish yields a total predation of

$$\mu_{\text{PRED}}(t) = \frac{V_D}{V_F}\frac{\int_0^\infty\int_0^\infty f(x_{m_F})\rho_F(t,a,m_F)dadm_F}{\int_0^\infty\int_0^\infty m_D\rho_D(t,a,m_D)dadm_D} \tag{3.5}$$

This gives the total predation mortality at time t on the total prey population. Note that the denominator is the total biomass of the prey population which is the same value used in density-dependent mortality.

The functional response converts the resource density into total grams per day consumed for single fish, so we know total grams consumed, but, in order to assess mortality on the resource

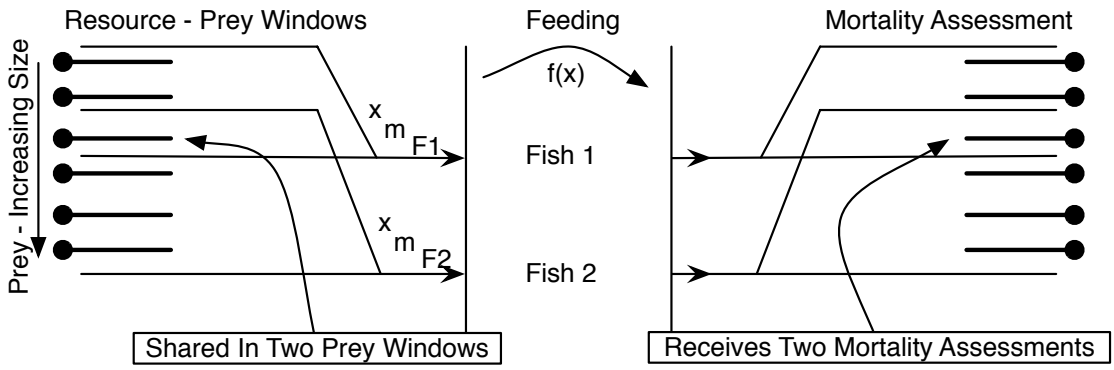


Figure 3.3: Individual-based Predation Flow Diagram

characteristics, this consumption must be converted into a mortality rate that is distributed equitably over the prey cohorts that were in its windows. We utilize the encounter rate to perform this conversion. This is developed further in a later section in this chapter, because it is very important to how the models perform and behave.

3.2.4 Discretized Predation Model

Equations 3.2 and 3.5 are the continuous forms. We now start the process of discretizing these equations. We are leading towards determining the minimal information involved in order to calculate predation and mortality for the community model, because this is the information that has to be exchanged for a parallel algorithm. It also leads us to understand potential improvements to the algorithm.

The predation flow diagram in Figure 3.3 illustrates the stages of the predation process. When prey are in overlapping prey windows, then a portion of this prey related to the relative masses of the fish is contributed to the fishes' resource levels. After the resource level, x, is determined for a fish, then the functional response $f(x)$ determines the amount of uptake. This is in units of grams per day. The rate has to be converted back to a mortality rate for assessment against the prey cohorts which provide the resource, which are obviously the ones that were in the original prey window. What is not obvious is how to apportion the mortality. The method we use is described in detail in the following subsection. For a long time I focused my efforts for parallelization on the resource level integrals, but the difficulties are in the evaluation of $f(x)$ and the assessment of mortality. The resource integrals are quite easy to compute and understand.

The discrete analogue of 3.2 for resource for a fish cohort of total mass m_F:

$$x_{m_F}(t) = \frac{m_F}{V_D} \sum_{k_1 m_F \le m_D \le k_2 m_F}^{\text{prey}} \frac{m_D \rho_D(t)}{\sum_{k_2^{-1} m_D \le m_F^* \le k_1^{-1} m_D}^{\text{pred}} m_F^* \rho_F^*(t)} \tag{3.6}$$

Notice that the sum

$$\text{tmass}(m_D) = \sum_{k_2^{-1} m_D \le m_F^* \le k_1^{-1} m_D}^{\text{pred}} m_F^* \rho_F^*(t) \tag{3.7}$$

over the predator population denotes a calculation that can be carried out independently for each prey cohort and stored onto the cohort for reuse. This requires that each prey cohort have access to the total mass and population density for each predator cohort (in its window). (Later referred to as **(Sum1)**.) The outer sum

$$\sum_{k_1 m_F \le m_D \le k_2 m_F}^{\text{prey}} \frac{m_D \rho_D(t)}{\text{tmass}(m_D)} \tag{3.8}$$

denotes a scan over the prey population for a fixed fish cohort. For this sum, the fish cohort must have access to the total mass, population density and the pre-calculated value for tmass() for each prey item (in its window). (Later referred to as **(Sum2)**.)

This completes the discrete calculations necessary to yield the resource density for each predator. Determining the resource density is not too complicated: involving only a scan over the fish population for each prey cohort to determine tmass() and a scan over the daphnid population per fish cohort. The Information Requirements to this point only have minimal information involving total mass and population densities. The complication comes with recording the mortality induced by predation. This is what we start looking at now.

Recall the derivations in Chapter 1 for the Encounter, Pursuit, and Gut Clearance Rates (Equations 1.18 – 1.21). After each of these equations, the dependencies on the daphnid population were noted. These will be summarized in a table later in this chapter when we list the information requirements, so they are not repeated here. For a given fish, to calculate $f(x_{m_F})$:

1. The entire prey population is scanned and values retrieved from these cohorts in order to form the three fundamental rates.

2. During this scan, store the encounter rate, $r_{m_F,i}$ for this fish onto the daphnid characteristic.

3. During this scan, sum the total of the encounter rates into rsum.

4. Structure and lipid component masses for calculating total percentage of lipid in resource and lipophilic toxicant uptake.

5. If the current daphnid is a parent, then retrieve the offspring characteristic and add its contribution for resource, lipid and structure components to the current fish's totals. The parent's encounter rate (and percentage of ownership if not Brood_Pouch, introduced later in this chapter) is used to scale the values for the offspring.

$f(x_{m_F})$ can now be determined for this fish by calculating the three fundamental rates and therefore the three fundamental times.

To account for this fish's predation mortality: Scan the prey population again, to update the value of predation inflicted by this fish on each prey item. On this pass, if toxicity effects are in play, then calculate the amount of toxicant consumed from each daphnid using the same rsum sharing factor and store on the fish's characteristic for later use. (The class of toxicants modelled are lipophilic, therefore the accumulated concentration varies per daphnid depending on its lipid mass.) This completes the computation of $\mu_{\mathrm{PRED}}(t, m_F)$. Repeat for each fish.

Thus, the computation of $\mu_{\mathrm{PRED}}(t, m_F)$ requires two complete passes over the prey population for each fish. Thus, if n is the number of prey cohorts, and m is the number of predator cohorts, then this involves on the order of $2m \cdot n$ operations to complete the predation mortality calculations for all fish. It is a two-pass calculation because the feeding kernel cannot be computed until individual values are retrieved from each prey item. There are several potential ways to reduce the number of operations. A couple of which are explored later in this chapter.

A physiological demand of our populations that is not apparent in the mathematical expressions arises from the fact that broods are carried internally. Thus predation of the parent affects the eggs. In the method of characteristics, the eggs are on a separate cohort and may or may not be combined with those from other parents. Contributions of the eggs to the resource and toxicant update of the consuming fish also must be calculated. This turns out to be a significant boost to the fish's diet and a significant source of mortality to eggs. This also introduces significant complications to the parallelization efforts as we shall see.

3.2.5 Encounter Rates, Competition, and Mortality

The sharing of resource invoked by overlapping prey windows introduces an element of competition to our model which turns out to be a vital control mechanism as we will see in the next chapter. In this section we describe its important features more carefully.

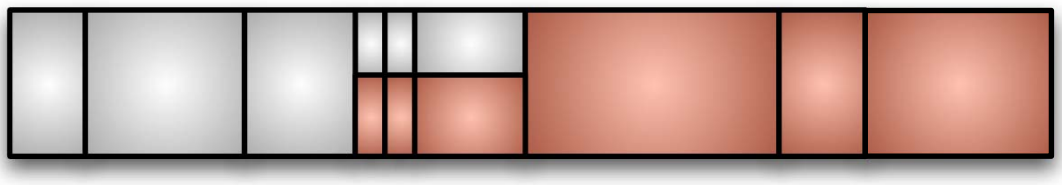


Figure 3.4: Different sized boxes represent different total masses of prey items. Colors indicate apportionment of prey items to two predators with overlapping prey windows.

With the apportionment of resource by relative masses, if a prey item is in a fish's window, then it will receive some portion of it in its resource level sum. This guarantees that the resource level is greater than zero if any prey items are in a fish's prey window. Since we do not have a spatial component, then each prey item must be evaluated by each fish.

Two factors affect the amount of resource each prey cohort represents: total mass m_D and total population density ρ_D. The first is affected only by the growth model for the *Daphnia*. Predation or lack thereof does not affect the growth of m_D. (There are species for which this is a survival mechanism: knowing the size of the predation window for the predators, they will slow growth rate just underneath the size window, then surge across the window in order to minimize the predation risk. This is called a juvenile bottleneck; for instance, see Bystrom et al. (1998).) The value for ρ_D is the value directly affected by predation mortality. The total biomass of the prey population is affected by predation since total biomass $= \sum_D \rho_D m_D$.

Figure 3.4 represents prey items as several boxes of various widths. The skinny boxes may be small because they are small in mass, density, or both. But, since they are cohorts in the population, they still must be included in the predation scans. Two fish with partially overlapping windows receive a portion of the contested resource. If $m_{F1} < m_{F2}$, then the portion is smaller for fish 1 and is precisely $m_{F1}/(m_{F1} + m_{F2})$. The number of horizontal splits depends on the number of predators competing for the same resource and the height of the split depends on the ratio of the mass of the fish to the total mass of all of the other fish for which the prey item is in their window. This is represented by **Sum1**. In the extreme, if $kmin = 0$ and $kmax = \infty$, and $n =$ number of prey cohorts and $m =$ number of predator cohorts, then every prey characteristic will be in every predator's window, so it will be split m times. The resource density for each fish is the sum of its apportionment across all n daphnids. Resulting in $m \cdot n$ operations required; I refer to this as the complexity.

To evaluate the functional response, Equation 1.16, the Encounter, Pursuit, and Gut Clearance rates must be calculated. These three rates have dependencies on the particular prey which were described in Chapter 1. The most important to recall is the Encounter Rate coefficient, a_d, which gives the volume swept out per day by the fish. As noted, its value is affected by length of the prey and the velocity of the prey. The number of prey encountered in the volume swept out depends on the prey's density. This expression provides the calculation we require to fairly assess mortality back onto the prey cohorts once total consumption is determined for a fish.

The total consumption of a predator on a prey population is called its outtake (relative to the prey population). Just because a fish is guaranteed a portion of resource, does not assure the survival and growth of the fish. The energetic/mass-equivalent demands for work required for movement and the maintenance of body mass can be higher than the resource consumed at a given time. The difference must be allocated from stores. If insufficient stores are available, then starvation occurs. This of course just describes the individual growth model for fish described in Chapter 1. What is significant is that a fish can be unable to consume sufficient resource to meet energetic requirements in two different ways:

1. If there are too many big fish competing for the same prey, then the fraction apportioned can be too small.

2. Insufficient resource is available in the fish's prey window.

If the consumption just matches the movement and maintenance requirements, then this is called the zero-growth condition. It is useful for determining the least amount of mass/energy required from a system to support a fish and can be used to determine carrying capacity for an environment. We look at this further in the following chapter. Recall also that the size/length of a fish cannot decrease, so a fish cannot "grow backwards" in order to return to a prey window where there was sufficient resource; rather, it will starve.

Finally, the assessment of mortality portion back onto the prey cohorts in the fish's prey window is a weighted sum of the encounter rates. The ratio $\frac{m_F}{tmass(m_D)}a_d$ gives the proportion of the encounter rate on this daphnid due to the current fish and is used to apportion $\mu_{\text{PRED}}(t, m_F)$. This is used to distribute the mortality. Sum over all encounter rates for all fish on a particular daphnid and call this rsum. Now, for a particular fish, determine its encounter rate relative to the particular daphnid, $r_{m_F,i}$, and assess the fraction $\frac{r_{m_F,i}}{\text{rsum}}\mu_{\text{PRED}}(t, m_F)$ as mortality against this daphnid. This method of assessing the mortality rate adds another layer of careful bookkeeping to the problem which is further complicated when we distribute the populations. Similarly, a weighted average based on the

encounter rate and total mass consumed from each cohort is used to determine aggregated values such as total lipid and structure components and total pursuit times. Through the encounter rate, there is a skewing towards the larger by mass, but still a little is apportioned from each cohort in the prey window.

A question is how closely aligned this is with optimal foraging theory which describes prey choice by a predator. The components are presumed "optimal" when the functional response was derived (Henson and Hallam, 1995; Hallam et al., 2000). But a truism of foraging theory is that a predator will choose the largest prey available because the same amount of energy is required to capture it or a smaller prey item. For our model we potentially have competition clear down from the top predators to the smallest. This actually has a controlling effect as we will see, because the growth of otherwise unmoderated newly-feeding fish can be explosive and can drive the prey population to extinction. The values for *kmin* and *kmax* can be varied to eliminate or enhance overlapping prey windows. But generally, our model does not directly give a preference to larger prey. We considered adding more than one prey population of a different *Daphnia* species than *magna* as an experiment to test the relation to optimal foraging theory, but did not carry out the experiment. Theoretical and experimental references to fish on *Daphnia* foraging in relation to foraging theory can be found in Hart and Gill (1993); Gill (2003).

3.2.6 Differences from Population Model for Fish

There are several differences between the numerics for the fish between the population model and the predator-prey model. Since the resource is a varying level composed of discrete, non-uniform organisms, then the biggest difference is that the difficulty is not in the calculation of the correct derivatives, but in the bookkeeping required to generate the discrete sums and properly apportion mortality. The resource uptake is calculated through prey window, functional response, and the varying shared resource levels, so the growth portion of the basic resource model is overridden. The basic equations of the fish growth and energetics models are the same, but since they are composed over discrete sums, then derivatives are not practical. Numerical integration is just a simple Euler's method. For our predator-prey model, the complexity for the model comes from the information flow rather the numerical method.

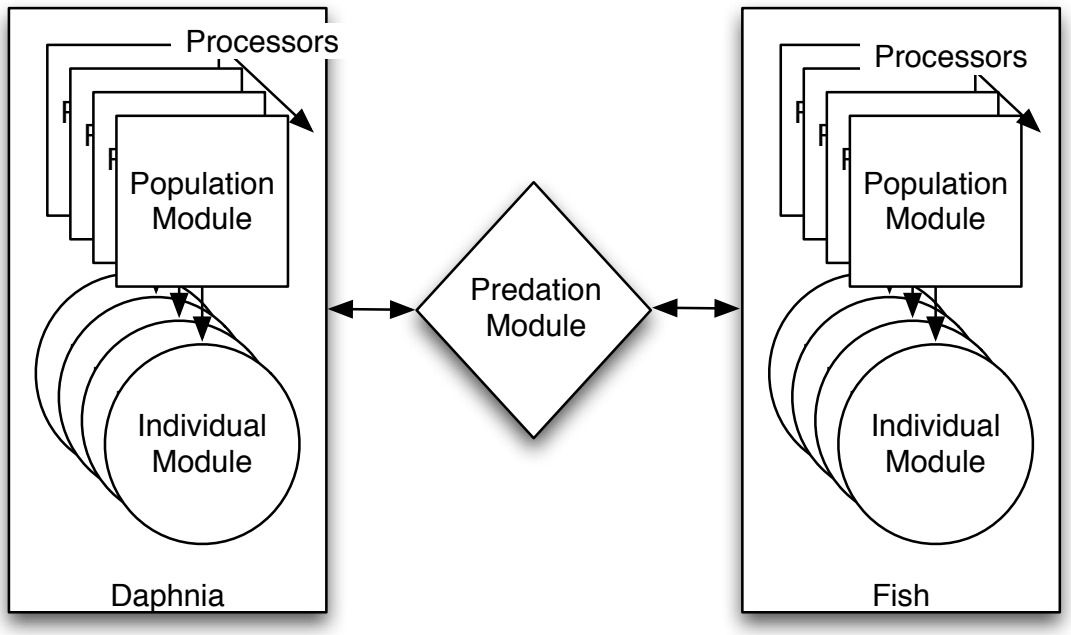


Figure 3.5: Parallel Algorithm Modular Design

3.3 Design of the Parallel Algorithm

Following what we learned from the population models, then the basic design of a predator prey would be something like that diagrammed in Figure 3.5. From experience with the population models, we learned that the parallel benefit is in two things: first, dividing the workload by cohorts, and second, distributing so that each processor has a smaller workload to process locally in order to pick up caching and memory performance benefits. We did see also from the population model that the parallel programs had increasing benefit as the total workload increased. When the workload was minimal, then one or two processors would perform just as well as eight.

As we develop different parallelizations of the predator-prey model in this section, we will morph this diagram various ways by duplicating, combining the fish and predation modules. The *Daphnia* module will remain basically the same as in the population model with a distributed population over some number of processors. Just as in the population model where birth combining caused us to have to have some overlap between the population and individual modules, the physiology of the prey will have a large impact on our parallel design. We have seen that the functional response/predation module will require several scans and updates to both populations. Another feature from the population model that we have to back away from is a single pass, unified individual

model containing feeding, growth, birth, and mortality all in one pass (see Figure 1.15). We had made these design changes to the population model in order to simplify the driver module design, eliminate synchronization points, and to more easily load balance, but they turned out to be incompatible with the back-and-forth interaction required for the predator-prey model.

(As another note related to design changes: the births calculations in all of my versions follows mortality assessment. The justification for this is that the initial density of the newborn characteristics is determined by multiplying together the population density of a birthing cohort and the number of eggs per individual; individuals that are about to be eliminated should not contribute to the newborn cohort's population density. This differs from previous implementations of this model.)

Recall Figure 1.1 and the descriptions of the brood pouch found in the introductory chapter. There is no direct predation of fish upon broods, so this removes a block of cohorts from consideration. A complication that we did not have to deal with in the population model was the mortality inflicted on a brood cohort when its parent experiences predation. This requires us to establish and maintain a connection between each parent and its offspring. The lack of this requirement in the population model allowed us extra freedom with respect to load balancing that we lose in the predator-prey model. Also, the only realistic design choices are that we redesign so that the brood cohort is internal to its parent cohort before its release, so it will be on the same node as its parent, or that we impose the requirement that the parents and their offspring to be maintained together on the same nodes. We chose the latter. Maintaining processor location information for point-to-point updates or broadcasting messages of mortality (that have to be coordinated to be received or else deadlock occurs) by each node in order to update the mortality of a brood characteristic which it may or may not have would be a poor design choice. (Although MPI-2 does introduce the concept of one-sided messages for just such a purpose; see Gropp et al. (1998).)

Another figure that will help illustrate the challenges of parallelization and the decisions we made is Figure 3.6. This figure illustrates several challenges. If the prey population is distributed as shown, then how will we distribute the prey window and resource density calculation? Likewise, once we do have the consumption by the fish, how do we distribute it back? With all of the record keeping and the intricate knowledge required by the predator calculations from each of the prey items in its prey window, then the compilation of information will be a challenge. Finally, just to add a little complexity, how can this be done if the fish are themselves also distributed? The answers to these questions is the subject of this section.

Figure 3.6 also illustrates some other predation scenarios and guides us on how they might affect parallelization efforts. After a few preliminary passes to decouple the work required, the sequential

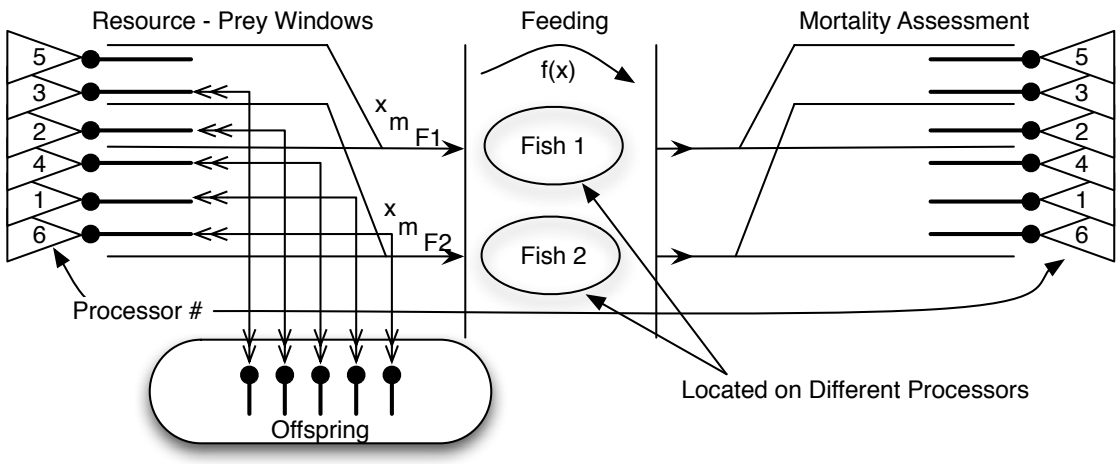


Figure 3.6: Predation on Distributed Prey Population

version of our program follows the design of working down the middle column and performing the calculations for one fish at a time. (They do not need to be in any order.) Since all of the prey items are in local memory, then there are no complications and the loop is basically take a fish and scan every prey item to see if it is in the fish's prey window. After feeding the fish, then rescan every prey item and update the mortality for the cohorts in the fish's prey window. If we were attempting to model a dominant feeding population, then there would be an ordering down the middle column, scan order would be forced to be dominant to smallest, and mortality would have to be assessed after every feeding of the fish. Finally, we will note that if there was no competition of resource and all predators were presented with the same resource density, then the left-hand side of the graphic would be just a single calculation.

For this first parallel design which we will term the *Pure Parallel* version, suppose the prey population is distributed across nodes as shown. Further suppose that the fish cohorts themselves are also distributed across the same nodes. So a particular fish will have resource that is both local to the same node and remotely located on other nodes. How can the predation module be distributed so that it can be executed in parallel? Basically, we will compose tables of information that will be compiled from the nodes and then redistributed across the nodes so that they all have access to the predator information. This table will be further annotated per node to indicate for which predators it is responsible. This is a disjoint set whose union over the processors forms the entire predator population. In a sense we are creating "ghosts" of all of the fish on all of the processors, so that the predation process can be completed in parallel. Also note that we feed all of the fish *en masse*

rather than stepping through one at a time. After the predation module is complete, then all that will remain for the fish module to perform is the Euler's method step.

Sorting out the information requirements, compiling, distributing, maintaining, and updating the distributed tables through the stages of predation, and computing the predation *en masse* are unique features of this solution. An especially significant offshoot of the algorithm was the method I used to eliminate the repeated population scans that are otherwise necessary. The first requirement is sorting out the information flow and what must be included in the distributed predation tables.

3.3.1 Parallel Algorithm - Information Flow

Building on our efforts for the population models, we extend the results to this predator-prey system. We achieved success before by distributing the populations across computational nodes. The only shared, dynamic information for the population models is biomass. With the predator-prey model, more information must be shared in order to calculate predation. In this section we first discuss the information requirements, then describe our parallel algorithm.

As before we inherit a requirement for calculating total population biomass per time step. This was required for the population models in order to access density-dependent mortality. In the predator-prey model, the total biomass for prey plays a role directly in the predation expressions. We also maintain density-dependent mortality in order to help constrain the prey population to realistic population densities for its control volume V_D. Total population biomasses are calculated for all populations per timestep just as for the population models.

In addition to biomass calculations, the calculation of predation involves the exchange of information described in Table 3.1. This table summarizes the information requirements of each stage of this calculation. The predation calculation is a separate module that computationally follows the advancement of the prey populations and precedes the advancement of the predator population. There are several predation calculations, such as for the tmass() function, Equation 3.7, that are shared among all of the predating fish, so these are completed locally before starting to advance the predator population. Since, as part of the predation process, the three fundamental rates are calculated per fish in order to determine the functional response $f(x_{M_F})$, then there remains little necessary to advance the fish population in time. Movement costs are determined from the fundamental rates. The remaining maintenance costs are simple functions. The almost complete usurping of the population model by the predation module in this design is diagrammed in Figure 3.7.

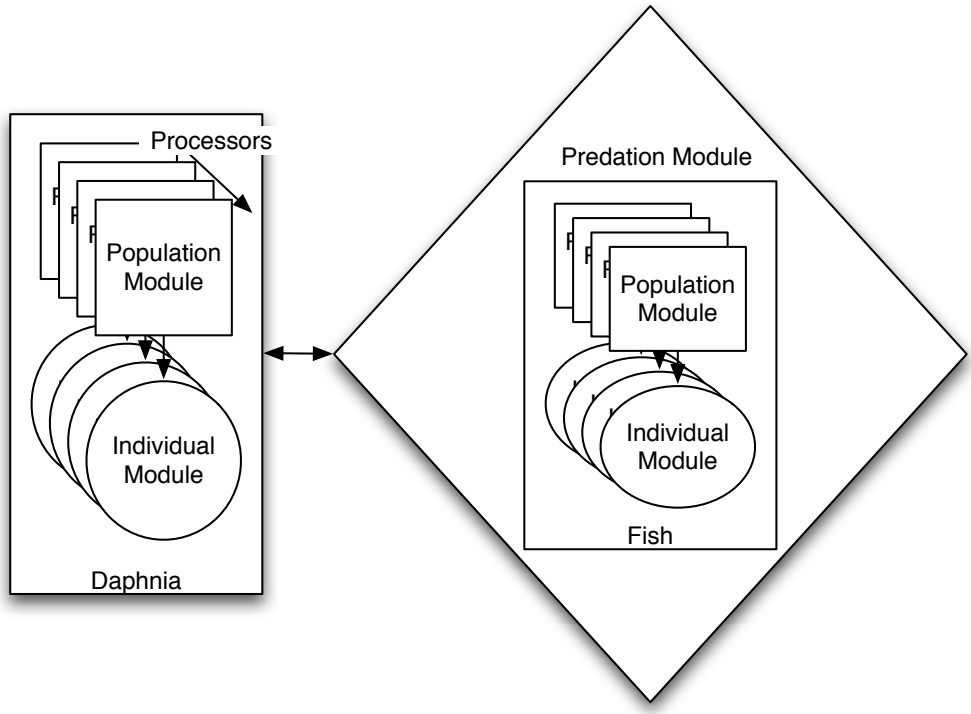


Figure 3.7: Predation Module simultaneously calculates predation and growth values for entire fish population and mortality for prey

Table 3.1: Information Requirements for Each Stage of Predation Calculation. m and n are the number of predator and prey cohorts, respectively.

Stage	Information Required	Consumers	Complexity	Notes
Sum1: tmass(m_D)	Total mass and density for each predator	Prey Cohorts	$n \cdot m$	Calculated once for each daphnid and stored
Sum2: Resource Density	Total mass, density, and tmass(m_D) for each prey cohort	Predator Cohorts	$m \cdot n$	
$f(x_{m_F})$: Feeding Kernel	Composed from R_e, R_g, R_p	Predator Cohorts	$m \cdot n$	
R_e: Encounter Rate	Total mass, numerical density, velocity, and length for each prey item consumed	-		Actually a linear expression involving length is used
R_p: Pursuit Rate	Velocity for each prey item consumed	-		
R_g: Gut Clearance Rate	Total mass for each prey item	-		We model density of each of the prey items as constant
PLX and Toxicant	Lipid and structure masses for each prey item	-		Totals used to determine percent lipid in resource and lipophilic toxicant uptake
Mortality Assessment	Individual fish on individual prey item encounter rate ($r_{m_F,i}$) and total encounter rate for individual fish (rsum).	-	-	Used to produce weighted average for equitable predation mortality assessment

To be precise, the cohorts really only need access to the information for each item inside of its forward or inverse prey windows. The easiest way to satisfy this requirement is just to make all cohorts from both populations available and scan them entirely. Thus there are several passes each costing $m \cdot n$ operations as shown in the complexity column of the Table. Memory movement can be expensive on multi-core processors as has been described, so lessening the required number of passes increases performance. One method to do this is to notice that if a fish cohort is determined to be in the inverse predation window while tmass() is being calculated for a particular daphnid, then the daphnid must be in the predation window for that fish. If this information could be recorded somehow, then the subsequent passes could be restricted to only the cohorts of interest. How I was able to implement this efficiently is described below. One could also envision taking advantage of sorting to restrict searches to only the cohorts of interest. This also will be described later in this chapter.

3.3.2 Pure Parallel Algorithm

In order to satisfy the information requirements for computation, we initially utilized MPI's combined gather/scatter operations and user-defined datatypes in order to of MPI in order to compile and exchange the minimal tables of information required to allow the computation to proceed in parallel. For population models only we only had to exchange simple vectors of numbers, but distributed tables of information about each predator cohort are required for the predator-prey model in order to decouple the local populations. We compose our own user-defined datatypes in order to exchange the rows of data needed. (I eventually replaced these advanced features with simpler ones of my own design in order to utilize alternate communication libraries, but the overall idea remains the same.) A particular feature to watch for is how the locality of cohorts is handled. In other words, a cohort described in the table may be located in local memory or it may be on another node. Similarly, each node holds a portion of both daphnia and fish populations on which it is responsible to carry out the operations required to advance them in time. As the computation progresses, then the other nodes will need to be made aware of values from these local populations; vice versa, the local populations need to be made aware of values from the cohorts on the other nodes. This is particularly driven by the sharing of resource. The procedure I designed to carry out this distributed table composition is not a basic feature of MPI. The Information Tables are described in the sequence that they appear in the computation. The interchange of tables is outlined in Figure 3.8.

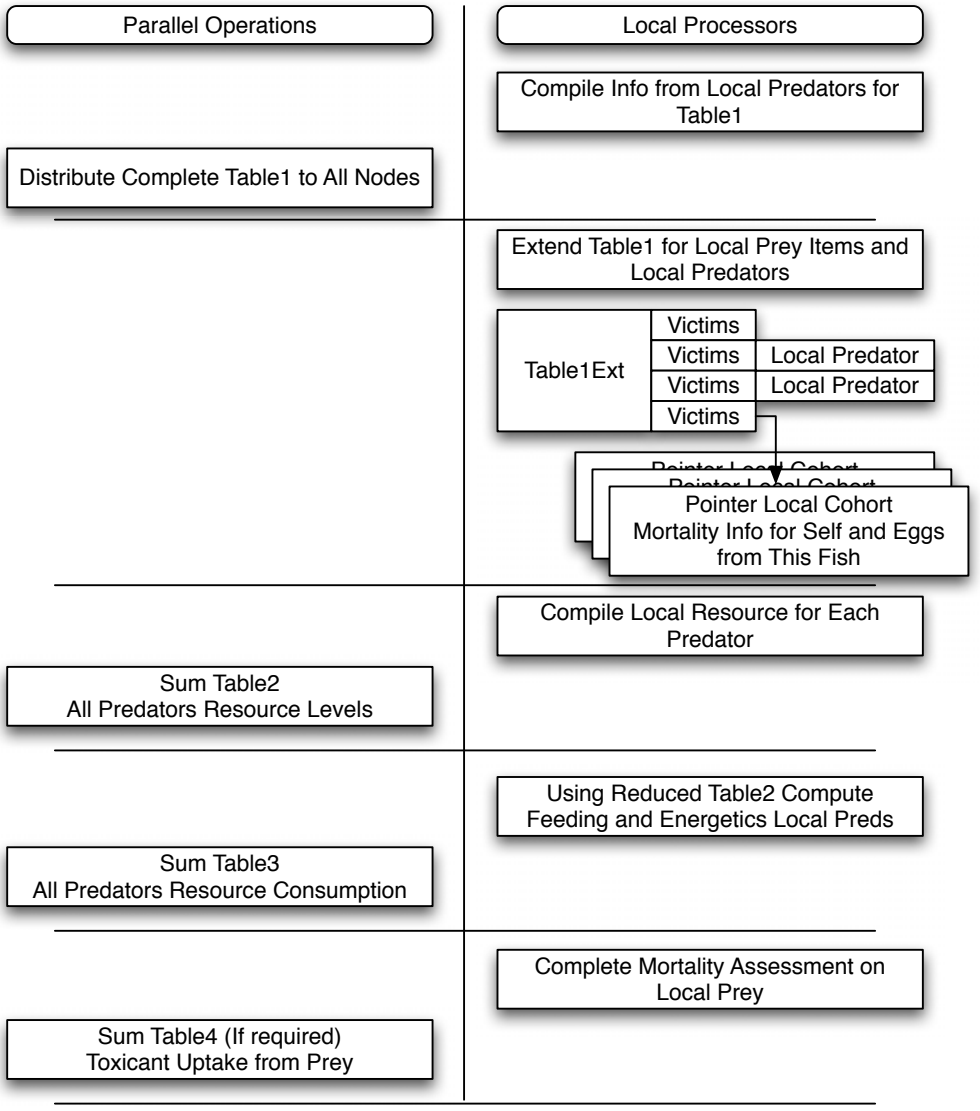


Figure 3.8: Parallel Predation Table Interchange Diagram

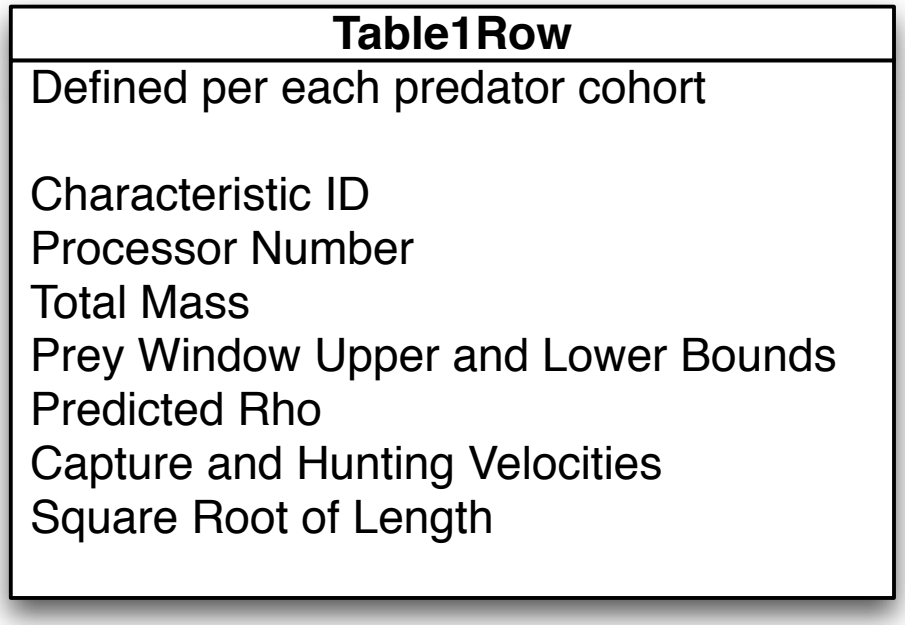


Figure 3.9: Row of Table1 - Distributed Table for All Predators

Table1 - Table of All Predators

This first table is the method by which the predatory information is distributed to the nodes. It is augmented and used to guide the computation. The table has one row for each predator cohort. Each node builds an identical copy of this table at the start of the parallel predation process. Table1 is used as an index into the local and dispersed populations and is used as a base to which the reduction tables that follow are attached. In its basic function, it provides the information necessary so that **Sum1** can be completed in a distributed way. A row for *Table1* is described in Figure 3.9. Each processor now executes Algorithm BUILDLOCTABLE1 in order to compile the entries from the local, feeding (i.e. no longer fry) predators for *Table1*. Note that the same memory structures are used to manage the cohorts as were defined for the population codes, except for the addition of *PredTableRow* which holds for each local, feeding predator its index into *Table1* once it is compiled. The *LocTable1* table created in this algorithm is combined with the corresponding tables from all of the other nodes to compose the complete *Table1*, so each row has type *Table1Row*. The *LocPreds* table created in this algorithm holds pointers back to the local predators. After Table1 is composed, this table is used to get back to the local predators. Note that if on some processor there are no predators that are feeding, then these two tables are just set locally to NULL.

Algorithm BUILDLOCTABLE1

begin

0. Initialize the local predator population.

for (scan ← local predator population)

scan.PredTableRow ← -1;

if (predator is feeding) **then**

increment numPreds;

calculate values for Table1;

end if

end for

1. Create Local Portion of Table1.

LocTable1[ ] ← allocate(Row Type: Table1Row, Count: numPreds);

LocPreds[ ] ← allocate(Row Type: pointer, Count: numPreds);

for (scan ← local predator population)

if (predator is feeding) **then**

LocPreds[k] ← scan;

LocTable1[k].proc ← processor ID;

LocTable1[k].* ← scan.*;

increment k;

end if

end for

end

The next step in the parallel process is to distribute and consolidate the complete Table1 onto each node. Each node will have an identical copy which it will link up to its local predators (if any). This consolidation involves the parallel Algorithm DISTRIBUTETABLE1. This algorithm is of my own design. It is a combined scatter/gather type parallel operation with variable length vectors on each node, which is defined by MPI, but I used a series of broadcasts instead. In particular I designed it to be compatible with the various versions of MPI which may or may not implement the user-defined types, as well as the shared memory message passing library I had created in the course of this project.

At the end of this algorithm, each node has an identical copy of *Table1*. Locally, *Table1* is extended in order to provide pointers back to the local predators and in order to provide to each row

in Table1, the base of a variable-length list of local prey which fall into the prey window. This list of local prey is used to eliminate the additional scans of the populations. Setting PredTableRow for the feeding fish enables one to step from the local predator characteristics into the corresponding entry in the tables. This is especially important for the subsequent tables; they have the same dimension and indexing as *Table1*.

Algorithm DISTRIBUTETABLE1

Given:

numPreds: Number of Feeding Predators on Local Node

LocTable1: Local portion of Table1

LocPreds: Pointers to the local predators

begin

0. Initialize the local node.

   $i \leftarrow$ processor-id;

   $p \leftarrow$ number of processors;

   Let preds[1...p] be array of p integers.

   preds[i] $\leftarrow$ numPreds;

1. Parallel: Determine the number of predators on each processor in the system via reduction.

   for ($j \leftarrow 1$ to p)

      preds[j] $\leftarrow$ the number of cohorts on processor-j;

2. Compute the total number of predators.

   totPreds $\leftarrow \sum_{j=1}^{p} preds[j]$;

3. Allocate Table1.

   Table1[ ] $\leftarrow$ allocate(Row Type: Table1Row, Count: totPreds + 1);

4. Parallel: Each node in turn broadcasts LocTable1. Likewise, each node receives broadcast and adds to Table1.

   for ($j \leftarrow 1$ to p)

      if (j equals i) **then** broadcast(LocTable1);

      else

         Table1[currentRow] $\leftarrow$ receive broadcast(LocTable1[j]);

      end if

      currentRow $\leftarrow$ currentRow + preds[j];

end for

5. Extend Table1 to Table1Ext.

for ($j \leftarrow 1$ to *totPreds*)

Table1Ext[j].predinfo ← Table1[j];

Table1Ext[j].victims ← NULL;

if (Table1[j].proc equals i) **then**

Table1Ext[j].locpred ← locPreds[k];

locPreds[k]-¿PredTableRow ← j;

increment k;

end if

end for

end

With the transmittal of the information in Table1, each node now scans its local populations of prey checking against each predator's upper and lower prey window. The list of predators is the one in Table1, not just the local predators, but the prey cohorts in this check are only the local ones. If a prey cohort is found to fit inside a prey window, then it is added to the *victims* list for that predator. A prey item can appear in multiple lists; this merely indicates that it is a potential prey item for several fish. **Sum1**, the calculation of *tmass()* for each daphnid, can now be completed. Note that the brood cohorts are not scanned, nor included in the *victims* lists. This completes the primary use of the information in Table1. The extension of Table1 is diagrammed in Figure 3.10.

Table2 - Table of Predator Resource Levels

The second distributed table, unoriginally named *Table2*, is now composed in order to complete the computation of each predator's resource levels and the values that need to be compiled to compute growth. This combines **Sum2** and provides the information from the prey population required to compute the feeding kernel for each fish. The design of a row of *Table2* is shown in Figure 3.11. The information in *Table2* is compiled by each node with values from the local prey for each predator in the total population. The details of the calculations of the rates have already been covered, so they are omitted. The important items to note at this stage are:

1. Any eggs carried by a parent are included in the resource calculations at this point.

2. The mortality partitioning weights (I term them *Relative Rates*) that could previously be stored onto the prey cohort because only a single fish at a time was being fed now must be

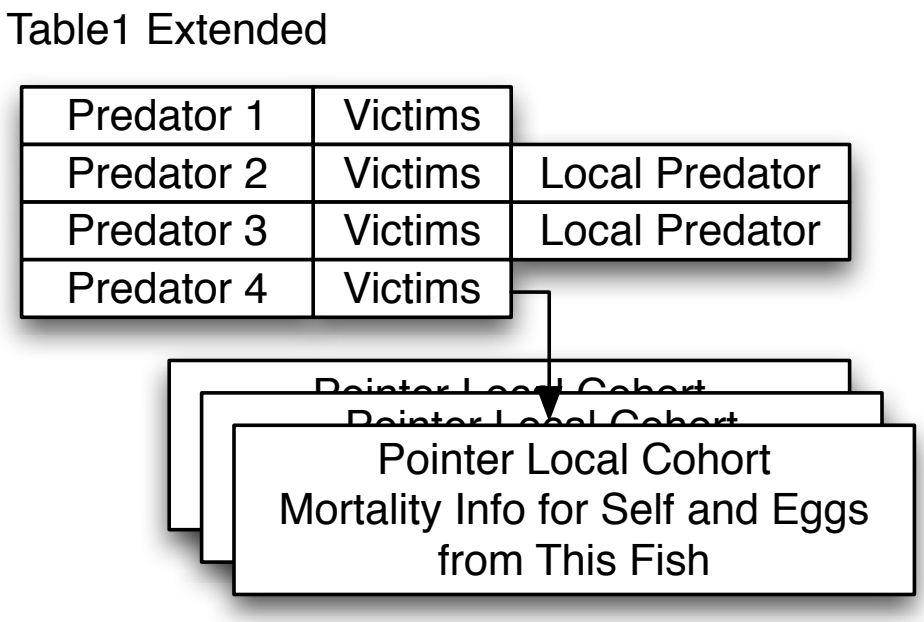


Figure 3.10: Extending Table1 to Local Predator and Prey Populations

Table2Row

Reduction of Predator Resource Levels
One Row Per Fish (all nodes)

INUM = number of local prey items for this fish
A1, Resource, Encounter Rate Sum, lipid and protein partial sums

Figure 3.11: Row of Table2 - Predator Resource Levels Compiled from Distributed Prey

duplicated. This was my original purpose behind the *victims* list. It can act as such a container because there is a one-to-one correspondence between each local prey cohort and each predator. Further note that the internal brood of eggs must also be included in the calculations. *Thus the brood characteristic must exist on the same node as its parent. The mortality partitioning weights for the brood characteristic must also be stored; this is also stored in the victims list.*

3. Values like the resource density that were previously accumulated in local variables as the prey population was scanned are now stored into a local copy of *Table2*. Before reduction, on each node *Table2* is composed of the partial sums arising from the local prey populations for each predator.

This stage is completed by reducing *Table2* across all nodes summing each corresponding entry. The reduced *Table2* now contains the complete sums for Resource Density, etc., for each predator.

Table3 - Table of Predator Resource Consumption

With the resource density information compiled across all nodes in distributed *Table2*, the computation of the feeding kernel, energetics, and growth for each predator can be completed in parallel for each local population of fish. The amount consumed by each predator is stored into *Table3*. The design of a row of *Table3* is shown in Figure 3.12. Each node only computes the feeding kernel and growth for its local predator population. All of the cohorts in the local predator population need to be advanced in time, even if they are too young to be feeding. For this reason it is not sufficient to index through the local entries in *Table1*. But, for the predators in the local population that are old enough to feed, then the *PredTableRow* index value is used to quickly extract its information from *Table2* and update its information in *Table3*. An identically sized *Table3* is allocated on each node, but, since the predator population is distributed across the nodes in a non-overlapping way, then the rows of *Table3* will also be updated in a non-overlapping way. Thus, when *Table3* is finally reduced via a parallel sum, then *Table3* will have all of the resource consumption information unified for all of the fish across all of the nodes. (Although the RSUM value for mortality partitioning was calculated in the reduction of *Table2*, it is not required until mortality is assessed by the prey population. This follows the reduction of *Table3* with the total resource consumption. I found it convenient just to copy the data from *Table2* to *Table3*, so that *Table2* can be freed at the end of this step rather than maintaining both tables.)

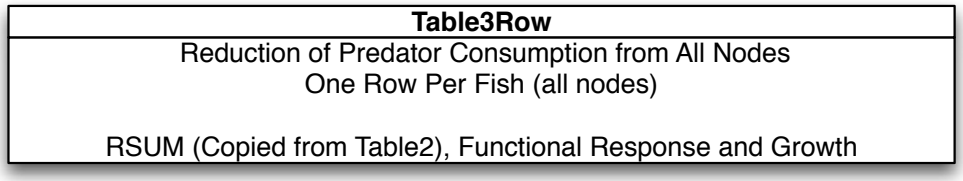


Figure 3.12: Row of Table3 - Distributing Total Predator Resource Consumption to All Nodes

Table4 - Toxicant Uptake

Finally, the reduced version of *Table3* contains the information required to complete the mortality calculations on the prey. If toxicant exposure is in effect, then the predators are exposed to the toxicant through their consumption of the contaminated prey. The amount of toxicant in the tissues of the prey items varies, so it can only be determined after the assessment of mortality on the prey. The only purpose of *Table4* is to account for the amount of toxicant uptake by each predator, so it is not pictured. After reduction, then the toxicant uptake is copied by each node onto its local predator cohorts. If toxicant exposure is not in effect, then the reduction of Table4 is skipped.

The interesting item to note in this final step is that the only local prey items that can possibly incur predator-induced mortality are the ones that appear in the lists attached to each predator in *Table1Ext*. The way that mortality is assessed is to walk the list for each predator, computing the mortality effects, while simultaneously releasing the associated memory. At the end of this procedure, then *Table1Ext* can be freed.

This completes the parallel predation algorithm. Since the growth and energetics were completed in the *Table3* phase of predation, then all that remains to complete the advancement of the predator population is to complete the Runge-Kutta step (which is just Euler's method) and to evaluate any toxicant effects. These two steps are all that remain in the predator's time advancement routine.

3.3.3 Design Analysis of the Pure Parallel Algorithm

We now take a closer look at some unique features of the Pure Parallel Algorithm.

Population Scans

The Pure Parallel algorithm required the addition of extra memory structures in order to retain an array of values for mortality assessment. These Relative Rate values have to be retained by the prey cohorts until the end of the calculation for a single fish in order to partition mortality equitably.

In the sequential design, the repeated filtering of the populations through the prey or inverse prey windows via complete scans seemed inefficient, especially for tens of thousands. I wanted a way to efficiently restrict the attention of each predator to its prey and vice versa.

By using a simple list structure attached to each fish cohort, I was able to eliminate the repeated scans, because at the end of the first scan each fish has available an exact and complete list of the prey items in its window. I needed the list structure anyway, in order to retain the Relative Rate information for each prey item in the predator's window. Since the number of prey items potentially changes with each time step, then it has to be dynamic. My first attempts were to try to take advantage of sorting by sizes in order to eliminate the extra population scans, but this algorithm produces the same benefits , is generally applicable, and does not have any additional requirements.

Brood Pouch

The second requirements imposed on the parallel simulation arise from the physiology of the *Daphnia* populations. When separating the populations across nodes the physiological feature of eggs developing in a brood pouch has to be considered. This requirement is not beneficial to the design of the algorithm like the first requirement was. Because the eggs are associated so closely to the parent, they suffer the same mortality effects. But, their mathematical dynamics are different because they are only in maintenance mode; as such, they are on a cohort distinct from its parent. The algorithmic requirements are two fold and were emphasized in the section on describing *Table2*. For the *Daphnia* population:

1. The brood characteristic must be on the same node as its parent.
2. The Relative Rate mortality information for the brood characteristic must be calculated and stored simultaneously for all predators.

The first condition requires that if a parent is moved across nodes by rebalancing, then its brood cohort must move with it. The second condition is a difficulty because the brood may be combined onto a cohort with other broods. The mortality calculation can be separated out to account for only the portion of mortality on the brood cohort due to the predation on a single parent, but if the brood has been combined with others, then it becomes difficult to maintain the correct values for ρ on the brood cohort, especially if the parent is subject to being moved to a different processor at any time step.

The solution I developed for this problem is called Brood_Pouch in the codes. The idea is to delay the "birth" combining until release time. Mathematically, this means changing the initial

condition from age zero to the age that they are released from the parent. (The linearity of the initial condition is utilized to justify this.) All of the same algorithms developed earlier apply — local and global birth combining — except they are applied at the age of release rather than the age of birth. At birth, the parents each receive a separate cohort for their brood, so its brood cohort is entirely owned by the parent. Thus if a parent is moved by load balancing, then one merely has to move its offspring cohort with it. No additional work is required. Similarly, there is a factor of one in the predation mortality assessment on the broods, so no scaling or adjustments are necessary to correctly assess mortality.

The brood pouch design does have some other benefits from a modeling perspective.

1. It respects conservation of biomass in the sense that the exact lipid per egg allocation from the parent is transmitted to the offspring cohort. (This variation is combined out of the normal versions by the birth combining process.) This variation does disappear when the cohorts are released from the brood and are combined. (Similar to the normal case, we choose the maximum lipid at the time of release from all of the cohorts of the same ecotype being released at the same time.) For processes like toxicity studies, then this can be a significant modeling feature.

2. Another benefit that is related to this is that individual birth and release timing events after the first event can be varied. In other words, the births after the first are not required to be periodic. Periodicity is forced in the normal version because the brood cohort must start feeding at some age (i.e. the age at release from the brood pouch). Since the cohort is potentially composed of several broods, then the broods of all of the parents must start feeding at the same time. The release of its current brood from the brood pouch initiates the birth process in the parent. With the Brood_Pouch design, then these timings can be varied individually. I built this potential variability of birth times in as I revised this code for Brood_Pouch.

This latter advantage could allow us to "fuzz up" some of the birth classes and eliminate some of the life cycle synchronization described by Enserink in Chapter 1. Sublethal, dissolved oxygen, and temperature effects had all required design changes centering on the size at first brood.

The Brood_Pouch design has several costs. The first is the complexity of the code required to execute it. The complexity really arises because I maintain both a normal version and a Brood_Pouch version in the same code base for the predator-prey and population models, so there are several compiler directives to maintain this dichotomy. Further, guaranteeing the conditions from the beginning of simulation requires either that the Brood_Pouch condition be imposed from the generation of

the initial populations forward (hence, the impetus behind rewriting the initial population codes described earlier), or one has to unwrap the initial population, creating additional offspring cohorts as necessary to satisfy the one-brood-per-parent condition. The load balance had to be modified to send its offspring cohort with a parent or the parent cohort with an offspring. Since our load balancing routine does not have a preference in what cohorts it sends, then either case can occur, so the offspring has to be linked to the parent too. (Originally, I only foresaw the necessity of the parent being linked to the offspring, so I signed this with single-linked lists. When I later realized that the reverse relationship was required, then I linked the otherwise unused child pointer of the offspring back to the parent, so there is some abuse of notation.) Further, since sends and receives have to match, then the receiving node has to decode the received cohort in order to detect that it needs to set up an additional receive transaction in order to receive the second cohort in a pair. Finally, but most significantly in terms of performance evaluation, the Brood_Pouch design roughly doubles the workload in terms of number of characteristics. Brood_Pouch is not absolutely required in sequential, shared memory computation, so this workload and complexity arises from a combination of parallel execution and the physiology of the simulated populations. This cuts into the potential performance gain of parallel computation.

3.3.4 And...It Didn't Work

For all of this beautiful work and unparalleled design, I could not get the models to behave correctly. I presumed that it was because of something I did. There were four problems that I observed and worked on, but could not find anything wrong.

1. Large numbers of prey cohorts made the execution intolerably slow. I presumed this was because of some overlooked consequence of the Brood_Pouch algorithm. The several days running time of the original models looked speedy by comparison.

2. Movement energetics for the fish required 10^8 J/day against an ingest rate of around 10^4 J/day. This was completely masked by the $\frac{ED}{EA}$ being capped at 1 in the model.

3. Starvation of the fish never occurred in spite of these energetic demands, perhaps because the energy factor was capped at 1.

4. Lengths for the fish were not realistic. The trout were growing to 75 centimeters long.

These problems I traced back and forth between the population and predator-prey codes. Given the complexity of the programs — especially with all of the complexity I added — and the fact that

we had published papers and runs with the initial populations I was given, I presumed there was a mistake someplace that I had made.

I did trace through the original codes and went carefully through the numerics. There were several functions that were incorrectly calculated and written in the original codes which I fixed, but then the original models exhibited the same behaviors (minus the brood pouch problem of course). These problem inspired some interesting, unique and useful ideas which are described in this short section, but overall I still could not find the source of the fundamental problems which I was working around. I am very happy to report that I finally found each problem.

The first problem is caused by the cutoff value we explored earlier when we looked at the No-Combine examples, Section 2.7.1 and which was varied for the population timings presented at the end of the last chapter. Somewhere along the line it was set to 10^-5 which for the long predator-prey runs caused sometimes hundreds of thousands of extra cohorts to be maintained although they were insignificant to the population. When combined with $kmax = \infty$ and $kmin = 0$, then they produced extraordinarily long runs. This sensitivity to workload was behind my carefully considering the complexity of the calculations, because it was slowing down noticeably with each additional fish cohort. Turning off Brood_Pouch led to the the runs being tolerable with the number of characteristics being halved.

While working with these large and long runs, we noticed that although at the beginning higher numbers of processors was slower, by the time it got to the equivalent point in the simulation, then it was faster than our normal 4 or 6 processors. The idea was to change the number of processors to be in line with the workload. The MPI2-specified way to do this is with MPI_SPAWN_PROCESS(), but this is another of the library calls that is not widely supported. I could only find it supported on SGI machines. We did it a different way which utilizes the load balance function. By starting a larger number of processes, but not assigning any population to some of them, then they will participate in all of the collective operations preventing deadlock; they just will not have any contribution. When the load reaches a currently hard-coded level of 15K on any one processor, then a new processor is added from the idle pool and the load balance routine redistributes load off of the other processors onto the new one. This allows the simulation to stay in cache and maintain its super-scalar speedup. We do not yet have a way to go backwards, but can foresee it being useful for problems that start out large and parallel execution is very beneficial and then shrink during the simulation. The individual-based bat model by Federico (2007) starts out very large and then decreases in size with execution time. By resurrecting the runtime adaptive rebalance period calculations and the heterogeneous,

runtime, workload adjustments reported by Sylvester (Sylvester, 1995) we plan to create an algorithm that will adaptively add and remove processors at runtime to yield better execution times.

The lengths and energetics turned out to be caused by the same problem: the allometric constant. The allometric constant for fish that has been utilized and published (Hallam et al., 2000) was off by a factor of 10. After correcting the value to 1.7×10^{-2}, then all of the calculations, energetics, and lengths fall into line. The appendix to Hallam et al. (2000) has a complete derivation of the energetics calculation for movement. The cost rises to the fourth power of length, so the lengths being large was causing the absurdly large values for energetics. The fish model with the corrected constant have a maximum size of about 35 cm and the energetics are much closer with the $\frac{ED}{EA}$ being less than one for smaller fish. I have located two of the three references cited for our allometric value (Staples and Nomura, 1976; Elliott, 1976), but have not located our precise source for this constant.

Finally, starvation was turned off by my predecessors by setting the starvation level to be below the smallest value of the protected structure. Since the mass of structure cannot be less than protected structure, then no starvation will ever occur. My feeling is this was done because of the energetics problem: the fish would otherwise die quickly or be so lean at birth times that they would starve with any discrete allocation of mass to eggs. I altered the value and found that a level of 90% (i.e., 10% loss of structure stores from peak) gives a 2 week starvation period for larger fish and about a month for smaller fish, but they died when the discrete allocations were made for births. A value of 80% is what I used for the simulations in order to prevent starvation at birth events, but to allow for starvation to occur if inadequate resource is available.

In the next chapter I will utilize some of the old populations, because I was trying to match some results previously published, but the dynamics studies are generated with the corrected values for the fish populations as are the timings reported below.

3.4 Performance of Pure Parallel Algorithm

The performance of the Pure Parallel algorithm is compared to the sequential version using the Intel compiler and full optimization in Tables 3.2 and 3.3. The Intel compiler actually vectorizes several of the loops too, so it is performing a partial parallelization. The initial population is the same *Daphnia* population as utilized for the Population Model performance tests mated with a fish population. The tests were run for 2600 simulation days for two different workload levels. Wall-clock timings were used, rather than processor time. This is because bus system contention is not reflected in processor times. The highest time from all of the processors and the highest workload

Table 3.2: Performance Chart, Pure Parallel version, Intel, Cutoff = 0.1, Rebalance = 1000 cycles, 2600 Simulation Days

Processors	Total	Predation	Comm	Report	Workload	Max Workload	Speedup
1	452	361	0	79	194,496,152	23492	1.00
2	672	579	45	71	64,575,079	2131	0.67
4	488	354	36	77	33,197,076	1068	0.93
6	469	283	56	92	21,755,181	752	0.96
8	498	275	147	115	16,605,057	594	0.91

Table 3.3: Performance Chart, Pure Parallel version, Intel, Cutoff = 0.001, Rebalance = 1000 cycles, 2600 Simulation Days

Processors	Total	Predation	Comm	Report	Workload	Max Workload	Speedup
1	4287	4019	0	259	1,005,322,181	106596	1.00
2	4316	4114	152	175	362,274,119	8317	0.99
4	2287	2036	235	120	185,486,582	4296	1.87
6	1708	1430	251	137	113,333,858	2733	2.51
8	1487	1053	300	130	75,396,178	2260	2.88

are recorded in these tables, so the components may not sum to the total time given. All times are in seconds. Workload is in cohorts simulated of *Daphnia*.

The runs are for 2600 days which was chosen so that the initial population would be removed by maximum age mortality before the end of the simulation. The *rscale* parameter is set to 2×10^{-6}; *fscale* is set to 2×10^{-5}; and *kmin* and *kmax* were set to 6×10^{-3} and 2.7603×10^{-1}. The values for *kmin* and *kmax* were chosen from the persistence maps values in the next chapter as a combination known to survive through the full 2600 days of simulation. They are right on a zero-growth condition boundary, so the recruited fish (born during the simulation) and the initial fish population hold each other in check through competition for resource. Once the initial population disappears, then the recruits go from under 1000 g total biomass to over 2M g total biomass in about 300 days. The simulation ends with 2.45M g of biomass in the fish population. There are 39 fish cohorts with total biomass 157,460 g, and which have age such that they almost immediately give birth once the simulation starts. There are 3069 prey cohorts in the initial population with total biomass of 1836 mg.

3.4.1 Analysis of Timings

The timings again exhibit that a minimal amount of work load must be available for parallel algorithm to perform well. In the first Table with a light workload, then the speedup is close to parity, but still negative, when compared to the sequential version. With a heavier workload, then

the performance of the parallel version exceeds the sequential version. A most interesting trend is that the algorithm gets better with more processors added. The collective feeding of the entire fish population does seem to function better with problem size as seen by the decreasing predation values. Simulation and predation timings are not separated for this version, because the predation almost entirely encompasses the simulation steps. The total predation time continues to decrease as processors are added up through the maximum of eight and the communications to work ratio improves. These are notable because it indicates that the algorithm is improving with increasing numbers of processors. Between 2 and 8 processors there is an improvement by a factor of 3.9 in predation performance, while communications only increased by a factor of 2.

Reporting was enabled and totalled about 1.4 Gb of output per execution. This explains why a significant amount of time per execution is spent in reporting. The maximum time was for the host node, because it is the one actually performing the write to disk. The other nodes spent their time waiting for the host.

The Intel compiler hides a lot of the performance because it improves the baseline performance. With gcc the sequential timings especially were more than double and super-scalar speedup was the norm. The Intel compiler automatically vectorizes when it optimizes and utilizes an advanced math library. It is also an expensive product.

It was a lot of work to get this parallelization. It was started in an era of balance between CPUs and communications and has some nice traits, but it is still unsatisfying. Is there an easier and better way to get performance?

3.5 Alternatives to Pure Parallel Algorithm

In this section a simpler alternative to the Pure Parallel algorithm are presented. The Pure Parallel version was developed in line with the direction our design experience with the population models led us: distribute the work out across the processors in roughly equal chunks. With no spatial-component to draw upon, then the downside to distributing the populations completely are the communication costs of compiling the information so that all of the interactions can be accounted for. Three (or four) synchronization points with large transfers of data are introduced to disseminate this information just for the predation calculation and then is destroyed. Comparing to the one, minor sync point for biomass in the population model, then it is apparent that communications are a much larger portion of the execution time.

3.5.1 Fish-on-All

What else could we draw from the experience with the population models? We would like the super-scalar performance gains that we were able to get from the population models, but without the complicated design exhibited by our Pure Parallel version. The alternative we built depends on the observation that we should focus on the population with the most cohorts in order to take advantage of the parallel execution benefits for the population, while doing whatever we need to do in order to lessen the communication requirements. The "Fish On All" (FOA) alternative algorithm duplicates all of the predator population on all of the nodes; whereas, the prey population is distributed as in the population model. Biologically, this makes sense in that the prey population typically vastly exceeds the predator population, but the feeding complexity is with the predator population. The nodes duplicate all of the predator population and duplicate each of the predator's calculations with the same values, so they each maintain identical copies of the predator population. This is inefficient in terms of memory and processing cycles, but eliminates most of the communication costs since the information tables exchanged were all predator-centric.

The FOA algorithm requires little redesign beyond the population model. The initial fish population is broadcast to all of the nodes rather than being distributed individually. The sequential predation code was directly modified with the addition of a few parallel reduction operations which sum up the resource levels from across all of the nodes for the current fish. The fish are fed one-at-a-time just like the original sequential design. We avoided this design-choice with the parallel algorithm, but it is much simpler to feed one and then move to the next and does not require the extra memory structures to track the mortality information. We avoided this previously because it could lead to lessening performance with increased numbers of processors.

With the duplication of the predator population, then the large Table1 from the Pure Parallel version does not need to be compiled. The nodes can immediately apply the calculations for resource density to their local populations and sum together via a parallel reduction to get the total resource level for a fish. The consumption of the fish can then be calculated by each node, which can then apply the appropriate mortality to their local populations. The only other parallel sum required is for total toxicant uptake by the fish. With this version being so similar to the population model it was anticipated that it would perform similarly.

Table 3.4: Performance Chart, FOA version, Intel, Cutoff = 0.1

Processors	Total	Simulation	Predation	Comm	Report	Workload	Max	Speedup
1	452	91	270	0	79	194,496,152	23492	1.00
2	270	54	138	30	65	97,315,655,	11736	1.67
4	328	70	138	30	70	51,279,354	7150	1.38
6	635	180	180	35	86	32,853,700	4336	0.71
8	850	185	335	53	105	28,509,208	3700	0.53

Table 3.5: Performance Chart, FOA version, Intel, Cutoff = 0.001

Processors	Total	Simulation	Predation	Comm	Report	Workload	Max	Speedup
1	4287	783	3236	0	259	1,005,322,181	106596	1.00
2	1981	400	1382	35	180	540,987,881	59199	2.16
4	1968	355	1365	46	180	351,786,915	50135	2.18
6	2000	420	1215	51	209	246,552,344	35114	2.14
8	3023	523	2025	65	220	219,141,357	31820	1.42

3.5.2 Sorting

Another algorithm that I attempted to work out for a long time was using sorting to prevent repeated scans of the populations. The ecostacks structures which are used to maintain the populations in memory have the feature that they are inherently sorted by age, because the newborn cohorts are always placed at the top. Since we have a non-decreasing expression for size once feeding begins, then one almost has sorting by size too. The variation of the initial lipid disallows the presumption of sorting by size. Reexamining Figures 3.3 and 3.1, one can see that if the prey are sorted by size, then "high-watermarks" could be recorded so that the next predator search could skip over any prey items that would be guaranteed to be outside its prey window. It is apparent now that this would fail to be of any benefit if one has many overlapping windows, which we do — oftentimes all of them overlap. The *victims* list is a generally applicable technique that makes all remaining population scans after the first one as efficient as possible. It is attached to the parallel algorithm because I had need of the memory structures for other reasons, but the algorithm could be used in other designs.

3.6 Performance Revisited

We now revisit in Tables 3.4 and 3.5 the same initial populations and setup as we used for testing the performance of the Pure Parallel algorithm and executed them with the FOA version. We also turn off load balancing and Brood Pouch in Tables 3.6 and 3.7 as examples of the performance hit induced by the Brood Pouch requirement.

Table 3.6: Performance Chart, FOA version, No Brood Pouch, Intel, Cutoff = 0.1

Processors	Total	Simulation	Predation	Comm	Report	Workload	Max	Speedup
1	228	40	128	0	49	59,970,636	3069	1.00
2	180	25	85	28	58	33,761,676	1592	1.27
4	282	57	102	29	65	21,113,033	795	0.81
6	610	168	160	33	79	11,713,772	569	0.37
8	800	180	282	50	96	19,267,265	490	0.29

Table 3.7: Performance Chart, FOA version, No Brood Pouch, Intel, Cutoff = 0.001

Processors	Total	Simulation	Predation	Comm	Report	Workload	Max	Speedup
1	648	130	452	0	67	242,919,351	5850	1.00
2	392	76	245	28	65	131,756,105	3061	1.65
4	392	84	207	30	70	78,269,120	1765	1.65
6	679	189	210	32	87	49,322,958	1108	0.95
8	948	220	506	55	111	69,893,826	1441	0.68

3.6.1 Fish-on-All Analysis

The first thing to note is that for a much simpler algorithm to implement, this version does consistently give improved performance even with lower workload levels. In some cases it exhibits the super-scalar speed up anticipated from the population models. It is interesting that we went from marginal performance gain to near superscalar based a redesign and simplification of the algorithm. This is much more satisfying.

The improvements when using the gcc compiler are much more dramatic with superscalar gains up through 6 processors. When performing calculations I would use the FOA model with 6 processors in order to complete the runs as quickly as possible. The comparison was between 15 minutes total versus an hour and a half or more otherwise.

The peak performance is with 2 to 4 processors. This is in line with the population model peaking at 4 to 6 nodes. Once the workload has scaled down below 1000 cohort per processor, then there remains little performance to gain. It is also clear that simulation/predation time scales, but our simulations are becoming I/O bound, with reporting sometimes nearly a third of the total execution time.

Rebalancing still imposes the offspring-on-same-node requirement, but rebalancing has so little benefit compared to its cost of extra workload, that if the physiological refinements afforded by the Brood_Pouch model are not required by the model, then it is best just to satisfy the constraints with the initial population distribution and then turn off rebalancing. An alternate solution is to only move juveniles which would also allow the Brood_Pouch option to be turned off. Juveniles are free

to be moved, because they are released from the brood and do not yet have an associated brood. This requires the addition of a bias towards the juveniles to the load balancing algorithm. This method will not be able to achieve the same level of load balancing, but, as we have already seen, much more imbalance can be tolerated by the modern processors. Lessening the workload is much more significant to performance.

3.7 Discussion and Conclusions

First, we developed a methodology to decouple the workload by condensing the predator population data into a set of distributed tables. This was the most direct approach to parallelizing the predator-prey model. It follows the direction established by our parallel population models of distributing the populations to gain parallel advantage, but unfortunately this design could not be implemented as a simple extension of the sequential model or population models. With the size of the information tables being directly tied to the number of predator cohorts, the advantage of this parallel design increases as the predator population's workload increases. As the ratio of work to communication improves with increasing predator workload, then, per cycle, more information is being condensed by the distributed tables and more work is being decoupled for parallel execution. This benefit was demonstrated by our Performance Analysis of this design in this chapter.

The dimensions of workload for our testbed problems — large number of prey cohorts and considerably fewer numbers of predators — does not take best advantage of the distributed table design. Our problem's dimensions are driven by going across trophic levels. For a similar but intra-trophic (fish-on-fish) model the number of cohorts never exceeds 50 for either population over 800-year simulations (Claessen et al., 2002), so such models are also not of the optimal dimension. Natural, ecological predator-prey-type situations of the correct dimension for the distributed table design, where the number of predator cohorts is large and on parity with the number of prey cohorts are rarer. Agent-based, predator-prey models are one potential source, such as BOIDS-based, bat-insect models (Raghavan, 2005; Kolli, 2007). Economics and Finance are other areas of application of individual-based techniques (Luna and Perrone, 2002; Billari et al., 2006; Holling, 2001) which could be a potential source for problems of the correct dimension for this parallel design.

The creation of pseudo-predators via the distributed tables on each node, as in the Pure Parallel design, is in the style of the parallel technique of ghosting (Reynolds, 2006). The distributed table design is generally applicable and for us has the feature of predating *en masse*, so that only one pass through the predator population is required to execute the predation module. We further

demonstrate the elimination of inefficient scans and the conveyance of required information to all nodes in an efficient manner without resorting to direct tracking of the location of particular prey or predator cohorts.

For our testbed applications, we then found that the simpler Fish-On-All design allowed us to take advantage of the dimensions of our problems. By focusing on the larger workload of the prey population and using fast CPUs to duplicate the predator calculations in order to eliminate slower communications, then we are able to take advantage of the hardware and to mimic the performance gains achieved with the population models. It does not exhibit the same parallel scaling benefits as the distributed tables method, but advantageously, it could be implemented as an extension of the sequential version.

The FOA design also solved other problems that were not apparent and enabled the efficient analysis of the predator-prey model in the next chapter. These problems include solving tracking fish predation behavior. With the Pure Parallel version, the fish could trade processors and thus change their identifier. To track this, these exchanges had to be recorded and then backtracked after the simulation was complete. Further, the predation total values were vital to finally understanding the predator-prey mechanisms. As runtime-only data that cannot be easily extracted from data files, then the data had to be recorded as it was generated. When distributed, consolidating it for I/O would have been complex. Both of these problems were solved elegantly by the FOA design.

Independent of the parallel design, we found that the requirements arising from the existence of the daphnid population's brood pouch and the corresponding assessment of predation mortality against the both parent and offspring cohorts were a source of complexity that did not arise in either the sequential or parallel population simulations. The required maintenance of one-to-one, parent-offspring relationships across all nodes was a complex requirement that had to be added to our testbed models. Although only directly required for parallel execution of *Daphnia*-fish communities, the Brood Pouch solution is a feature that affords unique benefits to our individual-based models however they are executed and would be included in any new models. Because we applied it retroactively, it required changes to our load balancing algorithms, additions of complex memory structures, and updates to the cohort-combining routines. With its application, then we were able to maintain for the parallel versions the relationships required for the correct simulation of these community models. As with the population models, we could back off these stringent requirements by not rebalancing and by distribution by ecotype for the prey population. This was unique problem for parallel simulation faced by individual-based models that arises directly from the individual's physiology in interaction with the predation model.

Although currently just hard-coded and uni-directional only, a most interesting development for changing the parallel simulation to fit the problem size is reported in this chapter. The predator-prey model could produce pathologically large workloads as the simulation progressed. Initially these problems were simulated most quickly by smaller numbers of processors, but as the workload increased, then adding processors was beneficial. Dynamically resizing the parallel resources was impossible with MPI. A method for adding processes to the parallel simulation was added to the MPI-2 specification, but it is not widely-implemented. We added this ability via another method to our simulations by not initially distributing any cohorts to a few reserve processors which were added later via the load balancing algorithm when the workload exceeded a hard-coded level on any one processor. To generalize this to a more general, dynamic resizing of the parallel machine size will make use of unused calculations that we had originally developed to solve dynamically the optimal rebalancing period problem described in Section 1.3.5. The reverse situation where problems start out large — thus potentially benefiting from parallel techniques — and then decrease in size could also be encompassed by a scaling of the parallel machine to the dynamic problem size.

Chapter 4

Persistence and Extinction Conditions of the Structured Predator-Prey Model

In this chapter, we focus on the dynamic behavior of the predator-prey model itself. Our goal is to find regions of compatibility in parameter space where a simulation of the fish and *Daphnia* community will persist indefinitely. Specifically we examine the effects, constraints, and valid value ranges for the four predation parameters. Further, the roles of the initial distributions, individual models, and dynamic structures of the two populations are explored. The population models by themselves have been explored and are well-understood, but the predator-prey model composed from these population models has not been explored. Henson, in the last part of her dissertation, made a few observations concerning prey extinctions for the predator-prey model which form the starting point for our discussion.

For the first two sections, we restrict the mortality effects imposed, so we can directly observe the effects of predation mortality without the complication of exogenous deaths.

In the first section, we focus on the prey population and indicate how to choose the predation parameters that will lead to or avoid predation-driven extinction. The basic concept introduced is called the *Extinction Threshold*, which gives the upper bound for the level of mortality that can be sustained by a given prey population arising from a given predator population. This yields a method to choose appropriate values for the two effective volumes, V_D and V_F, that determine the

predation mortality scaling factor. We duplicate and expand upon the observations of Henson which were not completely specified nor explored in her dissertation because of the constraints of how long the numerical simulations previously took to execute.

In the second section, we focus on the predator population and understand the restrictions placed upon the predators by the gape-size parameters $kmin$ and $kmax$. We will restrict our attention for most of this section to understanding predation for a single predator cohort. The prey choice effects of a predator upon a prey population are observed and validated, which leads to understanding the location and size of the prey window itself. Possible effects of the prey window that can lead to the extinction of a predator are enumerated. In particular, the effect by smaller fish of depletion of resource levels available to larger fish is explored. This effect arises because of the reduction from the smaller prey size classes that would have otherwise grown into the larger size classes utilized by larger fish. The minimum population density of a fish cohort required to totally deplete the resource in its prey window (and thus totally deplete resource for all larger fish) is the idea behind the *Quiescence Threshold* defined in this section. This concept encompasses the variety of extinction effects for a predator.

In the final section, we complete our parameter space search for the regions of compatibility. The extinction mechanisms and parameter determination techniques explored in isolation in the first two sections will be fundamental to understanding the extinction pathways observed in this final section. Density-dependent mortality is required to control the otherwise exponential growth of the *Daphnia* population, because we do not model any depletion of the resource available to them. The fish population competes for a variable resource that can be depleted. Starvation and maximum age are the only paths by which a predator cohort can be removed from the fish population. No additional types of mortality are imposed on the fish population. The effects of total predation pressure and the dynamic effects of biomass oscillations and density-dependent mortality will be observed in this section. The roles of competition, initial population distributions, and the individual models on the regions of persistence and extinction are all described in this section.

4.1 Extinction Thresholds

We start with our analysis of the structured predator-prey model, by picking up where Henson left off in her dissertation (Henson, 1994). The last chapter in her thesis is entitled *Size-Dependent Predation in Structured Predator-Prey Models*. For this first section we will repeat and expand

on this chapter. The major idea introduced in this chapter was *Extinction Thresholds*. For the remainder of this section, we operate under the assumptions that:

1. All mortality on the prey is either due to predation or old age;
2. For predators there is no mortality except due to starvation (total population density for fish is constant);
3. The *Daphnia* population has constant resource density;
4. Birth rates and prey growth are not density or volume dependent;
5. Finally, the only dependence for a predator's growth rate on volume or population densities is through the predator's resource level.

The first three assumptions lead to the consequence that extinction can only occur in one of two ways. The first scenario is if all of the daphnids are consumed, so the fish starve. This scenario is familiar from aggregated models: there exists a minimum threshold required for the initial number of prey, otherwise extinction of the prey occurs. The second extinction scenario is unique to size-dependent predation models: the fish may not find enough prey of the correct size to eat, and thus starve while the prey population survives. *Extinction Thresholds* are related to the first scenario and are focused on the survival or extinction of the prey. The second scenario is taken up later in this chapter.

Following the notation of Henson, let $\hat{\rho}(a, m)$ and $\hat{q}(a, n)$ be the fixed, initial density distributions for the prey and predator populations, respectively, in units of numbers per unit age per unit mass (m_L and m_S are combined into total mass). We now vary the initial volumetric densities by varying the volumes V_D and V_F. Recall from the previous chapter that $1/V_D$ is used to convert the *Daphnia* population $\rho(t, a, m)$ to volumetric density for fish resource density calculations, and that the predation mortality assessed on the *Daphnia* population is scaled by V_D/V_F (presumed ≤ 1). These two values correspond to *rscale* and *fscale* in the predation parameters. Note that as V_D and V_F are increased in size, thus making the initial population distributions less volumetrically dense, then V_D^{-1} and V_F^{-1} decrease. Because of this correlation of direction and the fact that V_D and V_F most often appear in the denominator in our calculations, we use V_D^{-1} and V_F^{-1} in our analysis.

Given the initial population $\hat{\rho}(a, m)/V_D$, define the *Extinction Threshold* to be the smallest V_F^{-1}, such that the initial fish density distribution $\hat{q}(a, n)/V_F$ drives the *Daphnia* population to extinction within a fixed number of days. We choose 100 days for our extinction runs to match Henson. (We

did note that on some of the runs the populations became extinct just after the 100-day period, but typically extinction occurred well before the 100 days elapsed.)

Definition 4.1.1 (Extinction Threshold) *Given the initial population distributions* $\hat{\rho}(a,m)$ *and* $\hat{q}(a,n)$ *and volume* V_D*, where it exists, define the extinction threshold to be the*

$$inf\{V_F^{-1} | \hat{q}(a,n)/V_F \text{ drives } \hat{\rho}(a,m)/V_D \text{ to extinction in 100 days}\}$$

The condition on V_F will just be termed the Extinction Condition. Note that a fixed value of V_D determines the value of the *rscale* parameter. As different values for V_F^{-1} are tested, then it is actually the calculated value for *fscale* that is varied in the parameter files. Note also that this infimum may not exist for a given set of initial conditions, because no population of fish is able to control the *Daphnia* population. Descriptions of these regions appear later in this chapter.

An alternate way I created to look at extinction level is through what I termed *Volume Replication*. Let ξ be set to the extinction threshold for a given set of initial populations and fixed volume V_D. Then

$$\begin{aligned} V_D\xi &= inf\left\{\frac{V_D}{V_F} | V_F \text{ satisfies extinction condition}\right\} \\ &= inf\left\{\text{fscale (or mortality scaling)} | V_F \text{ satisfies extinction condition}\right\} \end{aligned}$$

Definition 4.1.2 (Survival Volume Replication) *Define the Survival Volume Replication to be*

$$\begin{aligned} \nu &= \frac{1}{V_D\xi} \\ &= sup\{\frac{V_F}{V_D} | V_F \text{ satisfies extinction condition}\} \end{aligned} \tag{4.1}$$

So ν *represents the minimum number of replicants of* V_D *that will support a fish population.*

4.1.1 Extinction Thresholds as a Function of V_D^{-1}

This first experiment involves looking at how the Extinction Threshold varies as the initial volumetric density of the prey is varied. We set the gape size parameters *kmin* and *kmax* to zero and infinity, respectively, for these tests, so all predators have access to all prey items and partition between themselves accordingly. (Values for *kmin* and *kmax* for this experiment were not specified by Henson.) As a validation for our models, we retrieved from our archives the initial populations and source codes used by Henson to create her figures. We corrected several numerical calculations in the

original programs that related to the lipid and structure Jacobian calculations in the Runge-Kutta method for the *Daphnia* and some other bugs that caused the program to crash or not compile on newer computers. (Because of the corrections, some differences between the outputs are expected. The values originally obtained for the components of the Jacobian were on the order of 1.0×10^{-12}. After correction, then these values scaled to where they should be.) For direct comparison, these original populations and parameters were translated to population files compatible with our new population and predator-prey models.

By varying V_D^{-1} over six values ranging from 1.0×10^{-8} to 100 in multiples of 100 (values determined from her graph), Henson reported the figure shown in Figure 4.1. She attributed the flat section to the fish feeding below satiation, so an increase in initial density does not increase the initial density of the fish population required to drive the *Daphnia* to extinction. The figure we obtained from the updated programs is Figure 4.2. Both the original (corrected) and modern codes were tested and confirmed these values. The values obtained are recorded in Table 4.1. The figures are plotted on log-log. Further note that $ln(\text{fscale}) = ln(V_D^{-1}) - ln(V_F^{-1})$. The values for V_D, V_F, and Volume Replication are included because they represent volumes which we can understand, so the extreme values required to pick up the bottom of the curves is understood. Initially I thought only that the linear increasing portion remained and the behavior Henson observed and attributed to satiation was an artifact of the incorrect numerics. (Many more intermediate values for *rscale* were also calculated to trace out the curves, but are omitted from this final presentation.)

The threshold curve is exactly where mortality and births in the prey population balance. If the births are a little ahead, then, in the absence of other mortality, the population grows exponentially; vice versa for mortality. Thus the curve is an unstable equilibrium (zero isocline). To determine these values, I initially used both the original Henson codes and our PPrey Fish-On-All (FOA) code with 7 processors. After it was shown that the codes both produced essentially the same behaviors, then I switched completely to the FOA version. Initially this search was performed by hand, adjusting the *fscale* parameter above and below the threshold in a binary search to close in on the value. When the parameter value was below the threshold, then a typical 100-day run would take about 90 seconds, versus 5-10 minutes for the original code. When it was above then extinction usually occurred within 5 seconds. A python-script was developed to automate this binary search. Each curve represents approximately 800 executions of the model in order to refine the value, so the difference in execution time was the difference between overnight or several days.

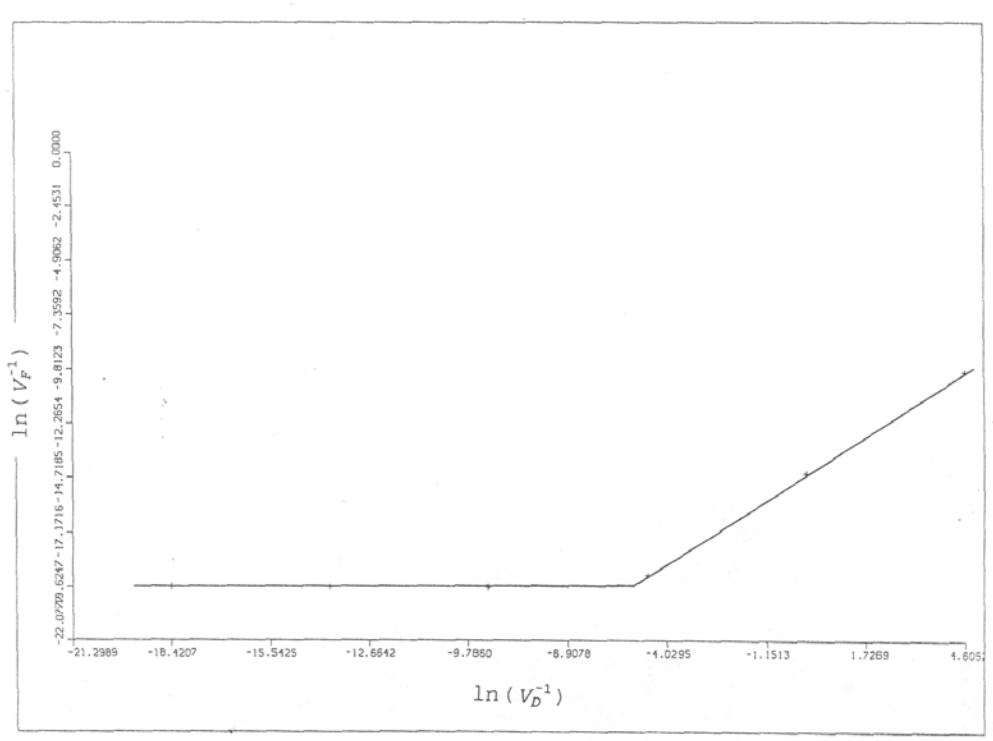


Figure 4.1: Figure 1 from Henson. Plotting the Extinction Threshold for Several Values of V_D.

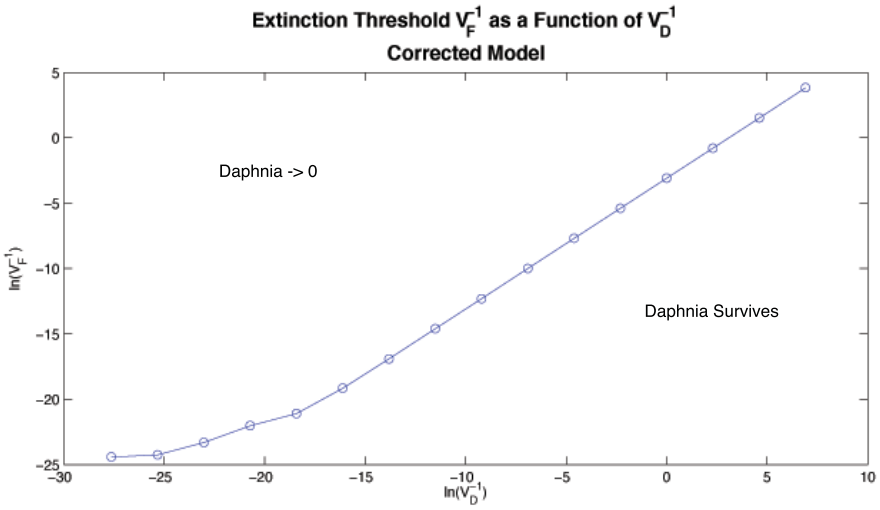


Figure 4.2: Plotting the Extinction Threshold for Several Values of V_D.

Table 4.1: Extinction Threshold Calculations

RSCALE	Extinction Threshold (V_F^{-1})	$ln(V_F^{-1})$	$ln(V_D^{-1})$	ln(FSCALE)	$V_D(cm^3)$	$V_F(cm^3)$	Volume Replication
1.00E-12	2.480E-11	-2.442E+01	-27.631	3.211	1,000,000,000,000	40,322,580,645.16	0.040
1.00E-11	2.900E-11	-2.426E+01	-25.328	1.065	100,000,000,000	34,482,758,620.69	0.345
1.00E-10	7.530E-11	-2.331E+01	-23.026	-0.284	10,000,000,000	13,280,212,483.40	1.328
1.00E-09	2.700E-10	-2.203E+01	-20.723	-1.309	1,000,000,000	3,703,703,703.70	3.704
1.00E-08	6.850E-10	-2.110E+01	-18.421	-2.681	100,000,000	1,459,854,014.60	14.599
1.00E-07	4.790E-09	-1.916E+01	-16.118	-3.039	10,000,000	208,768,267.22	20.877
1.00E-06	4.590E-08	-1.690E+01	-13.816	-3.081	1,000,000	21,786,492.37	21.786
1.00E-05	4.570E-07	-1.460E+01	-11.513	-3.086	100,000	2,188,183.81	21.882
1.00E-04	4.570E-06	-1.230E+01	-9.210	-3.086	10,000	218,818.38	21.882
1.00E-03	4.570E-05	-9.993E+00	-6.908	-3.086	1,000	21,881.84	21.882
1.00E-02	4.570E-04	-7.691E+00	-4.605	-3.086	100	2,188.18	21.882
1.00E-01	4.570E-03	-5.388E+00	-2.303	-3.086	10	218.82	21.882
1.00E+00	4.570E-02	-3.086E+00	0.000	-3.086	1	21.88	21.882
1.00E+01	4.570E-01	-7.831E-01	2.303	-3.086	0.10	2.19	21.882
1.00E+02	4.570E+00	1.520E+00	4.605	-3.086	0.01	0.22	21.882
1.00E+03	4.570E+01	3.822E+00	6.908	-3.086	0.001	0.02	21.882

Especially difficult to resolve were the smallest values of V_D^{-1}. Henson produced her graph with six points. Given the period of time this represented, this was probably several days of computation. The lowest portions being much more difficult to refine, it is not surprising that these are shown in her graph as constant. Our graph and Table 4.1, show that the graph is still decreasing in this area.

Overall, did we get the same thing as was previously reported? Did our code pass its first test? We have a much extended linear growth region starting at $rscale = 1 \times 10^{-6}$; whereas, her growth region did not start until $rscale = 1 \times 10^{-3}$. Her constant region is not quite constant in our graphs. She attributed both of these behaviors to the fish feeding below satiation for very low densities of prey, so an increase of the predator population was not necessary to control a more dense prey population. Once the fish reach saturation, then corresponding increases in initial density are required to control the prey. Is this a valid explanation? Does this explain our curves? Are our fish reaching saturation at much lower initial densities?

Investigating this question, led to looking more closely at the constant difference between $ln(V_F^{-1})$ and $ln(V_D^{-1})$ reflected in the constant value for $ln(FSCALE)$ past $rscale = 1 \times 10^{-6}$. This investigation leads to the concept of Volume Replication defined above which is trying to determine how many replicants of the prey population are required to support the predators. This describes satiation, because it gives the point at which the fish population is saturated and cannot absorb any more prey increase. Figure 4.3 plots the Volume Replication curve corresponding to our computed values. Note the distinct S-shape with the middle portion corresponding to the transition in Figure 4.2. Can these curves be attributed to satiation level? Satiation level is determined by the Functional Response, which for fish was presented in Equation 1.22. It was noted that this is in the form of a standard, hyperbolic response, whose canonical form is $\frac{x}{1+x}$. This is a curve with a horizontal asymptote and which has value zero for $x = 0$. The mystery is solved when one plots a hyperbolic functional response in log-lin as the Volume Replication curve is. The canonical hyperbolic response curve is so plotted in Figure 4.4. So we can conclude that saturation of the functional response is determining the extinction thresholds.

Transforming Extinction Thresholds for Scaled Populations

This still leaves open the question of whether our model outputs are similar? If not, then why not? If so, then why do they look so different? Can we transform our output onto hers? We did not have values for two of the parameters and we also changed the numerical calculations, but fundamentally we are still executing the same model. Further, we are producing effectively the same extinction threshold values. Are we just off by a scaling factor or something simple?

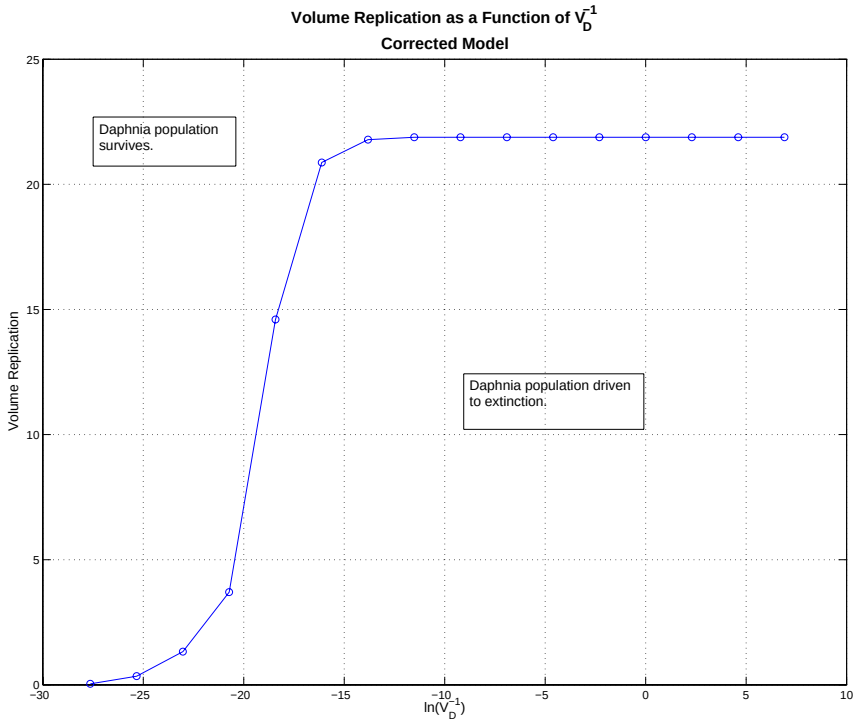


Figure 4.3: Plotting the Volume Replication for Several Values of V_D.

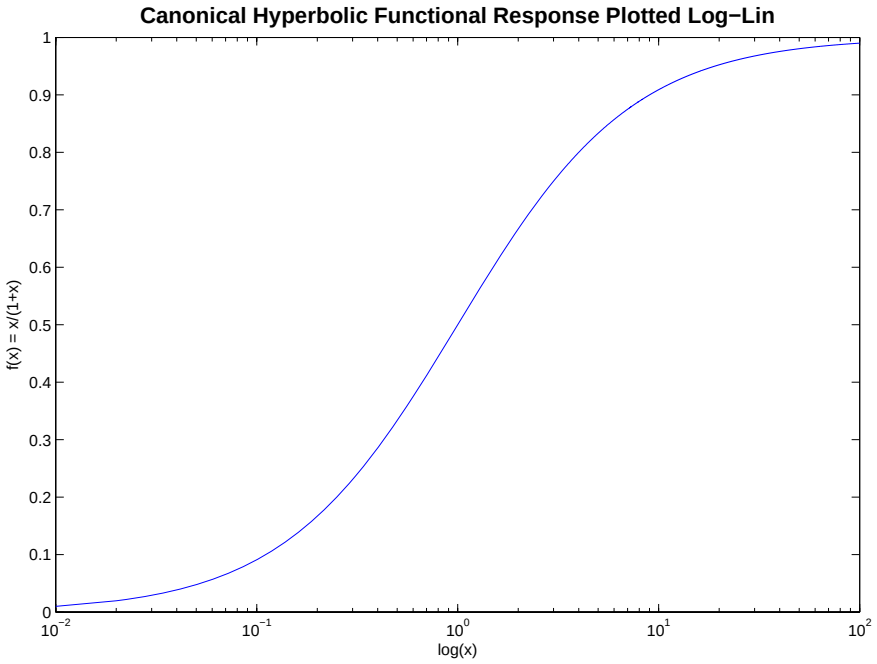


Figure 4.4: Basic Hyperbolic Functional Response Plotted Log-Lin

Up to now we have been treating V_D and V_F essentially as scaling parameters, we pause briefly to note the question how the extinction threshold transforms if the initial populations are scaled as in a mesocosm experiment. Suppose the initial populations $\hat{\rho}$ and $\hat{q}$ are scaled by α and γ, respectively, along with new volumes V_D' and V_F'. Defining "equivalent dynamics" and how the control parameters must be adjusted to produce such dynamics is the subject of the first chapter in the Communities section of Henson (1994). The question arises wanting to utilize mesocosm experiments and to isolate some dynamic of interest for study. The discussion involves careful inventories of the endogenous (fixed by population) and exogenous (adjustable) parameters and *a priori* determination of the dynamic of interest. This theorem follows from her developments for scaling for mesocosms. Its proof follows easily by substitution into conditions she develops for equivalent dynamics based on the PDE and by uniqueness of solutions, so it is omitted.

Theorem 4.1.3 (p. 135, Henson) *The predator-prey model with initial distributions* $(\alpha\hat{\rho}, \gamma\hat{q})$*, volumes* V_D' *and* V_F'*, and solution* (ρ', q') *has dynamics equivalent to those of the model with initial distributions* $(\hat{\rho}, \hat{q})$*, volumes* V_D *and* V_F*, and solution* (ρ, q) *if*

$$\frac{\rho(t,a,m)}{V_D q(t,a,n)} = \frac{\rho'(t,a,m)}{V_D' q'(t,a,n)} \tag{4.2}$$

and

$$\frac{q(t,a,n)\frac{V_D}{V_F}}{\rho(t,a,m)} = \frac{q'(t,a,n)\frac{V_D'}{V_F'}}{\rho'(t,a,m)} \tag{4.3}$$

for all t, a, m, and n.

The result important to us is that conditions 4.2 and 4.3 hold for the solution $(\rho', q') = (\alpha\rho, q)$ if and only if $\alpha V_D = \gamma V_D'$ and $V_F = V_F'$. This gives us the transformations

$$V_D^{'-1} = \frac{\gamma}{\alpha} V_D^{-1} \text{ and } V_F^{'-1} = V_F^{-1}$$

which transform the extinction threshold graph to the graph for the scaled initial distributions. Note that V_F' is not changed, while V_D' is scaled by this transformation. If we want our point of inflection at 1×10^{-7} to match her point of inflection at 1×10^{-2}, then a factor of 10^5 is required. When inventorying our archives we had found a second population set which differed from the first only in the magnitudes of ρ and q. We had noted the population as a curiosity, but it was not the one in the working directories, so we did not pursue it further. This section prompted remembrance of this oddity — why would there exist a population that differed only in the magnitude of the ρ-values?

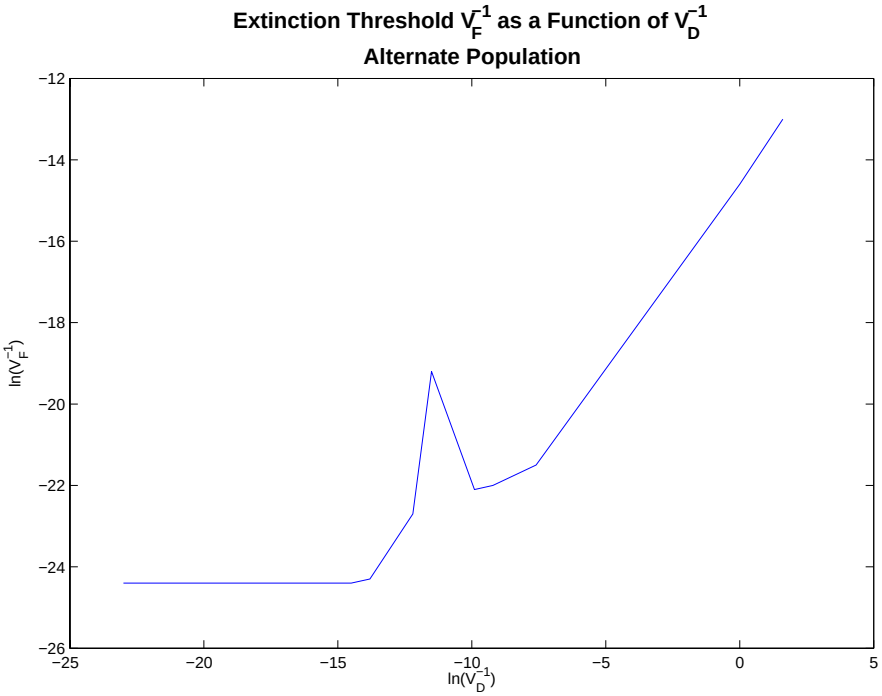


Figure 4.5: Plotting the Extinction Threshold for Several Values of V_D for Alternate Populations

Retrieving my notes about this population set, I found that the difference factors are $\alpha = 10^{-4}$ and $\gamma = 10^{1}$, which transforms precisely by a scale of 10^5.

I translated the population into our new format and tested. The graph of the extinction curve for this alternate population generated by our program is given in Figure 4.5. The shape (except for the one point, which was not calculated in Figure 4.1 anyway) reproduces the shape reported by Henson. The Volume Replication curve is shown in Figure 4.6. (I do not have an explanation for the one aberrant point other than it is in the middle of the S-curve which I had found before to be a difficult area to resolve. I calculated it several times with different levels of sensitivity and I chose different values of *rscale* around it. The values around it all calculated in line.) Overall, we can conclude that we pass our first test, but chose the wrong populations for comparison. We can further conclude that the numerical corrections did have some effect on the particular values for the extinction threshold, but not for the overall shape. The values for these alternate populations, especially for the Volume Replication graph, are conceptually out of my reach. I can understand 22 replications required to support the fish, but not billions. For this reason, I continued to use the first set of populations.

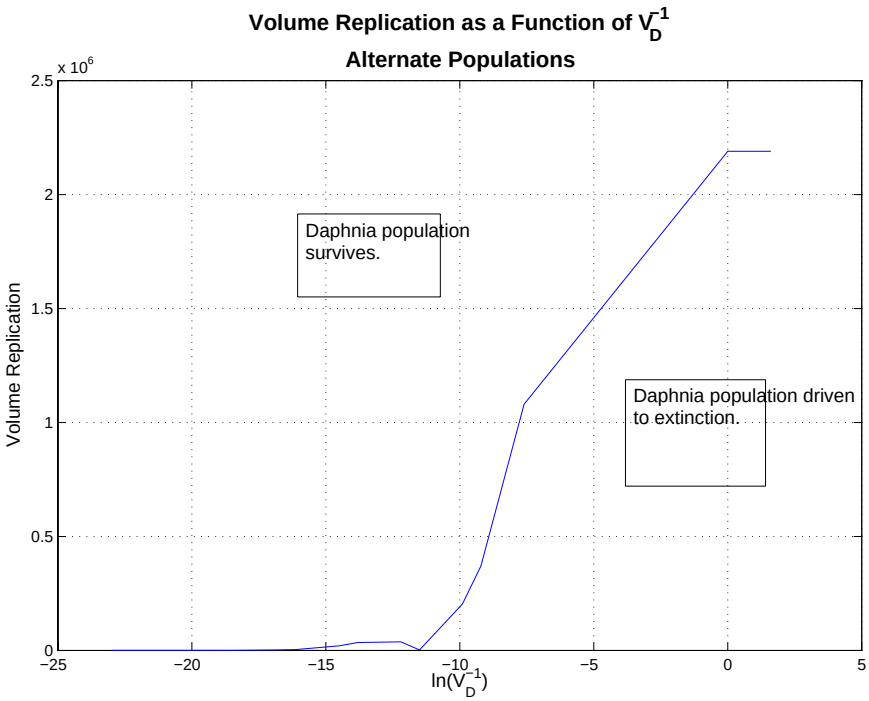


Figure 4.6: Plotting the Volume Replication for Several Values of V_D

4.1.2 Extinction Thresholds as a Function of *kmax*

For our size-structured predator-prey model the extinction thresholds also have a dependency on the prey window (gape size) of the predators. We now look at the dependency on *kmax*, which sets the upper bound of the prey window for each predator. In this section *kmin* is held at 0. In the previous work with extinction thresholds, these were set to infinity and zero, respectively, so that the gape size had no effect. By reducing the values for *kmax*, then we are removing the predation on the largest prey items.

A couple of expectations of the relationship of extinction thresholds and *kmax* can be immediately derived from our knowledge of the size-relationships and populations. We expect there to be a lower value for *kmax* below which the extinction threshold does not exist, because none of the daphnids will fit inside the prey window for any sized fish. On the other end, we expect there to be a upper value for *kmax* beyond which further increases of *kmax* will have no effect, because all of the prey items fit in the prey window for all feeding fish. With *kmax* set to a very large number in our previous analysis, then we were in this region, therefore we can use the appropriate values from Table 4.1 for the extinction threshold down to this upper value of *kmax*. Between these two values, we expect the extinction threshold curve to trace from the finite value on the right, to the vertical asymptote on the left. Further, we expect that it is non-decreasing when viewed from right-to-left.

We did plots of the extinction thresholds for a variety of fixed values of V_D^{-1} as another test of our model against the results reported in Henson (1994). Her reported curve is shown in Figure 4.7 and our curves are shown in Figure 4.8. (Henson did not note the value of V_D^{-1} that she used for her graph so we computed the curves for several. We also wanted to see if a relationship could be discerned between *kmax* and V_D^{-1}. Plotting on a surface did not reveal anything additional that cannot be seen from this presentation of all of the runs on the same graph.) The values at the left where our curves level off are an artifact of our computational method. They represent points at which a value for the extinction threshold could not be determined. Note the range of values reported by Henson and the range of values for *kmax* that she used. Her values for V_F^{-1} were on the order of 10^-10 (which is actually less dense than ours) and the values *kmax* were from below 0.003 up to about 0.0125. Her values were lower than I would have expected. The curve closest in shape and values to hers is the one for $V_D^{-1} = 1 \times 10^{-2}$. All of the curves do exhibit the expected behaviors. The rise-and-plateau structure arises from relieving segments of the prey population from predation. The larger prey members have to be eliminated by additional predation effort before they grow into

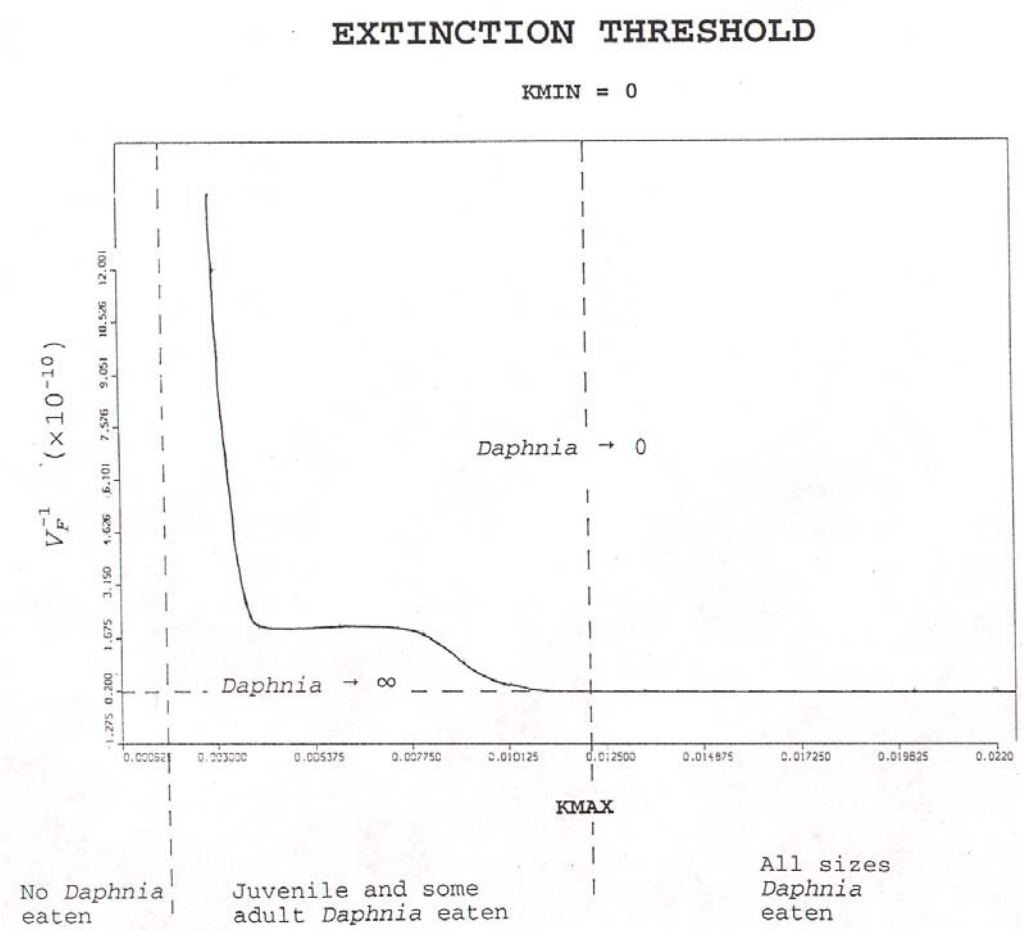


Figure 4.7: Extinction Threshold Curve from Henson for $kmin = 0$ and Varying $kmax$

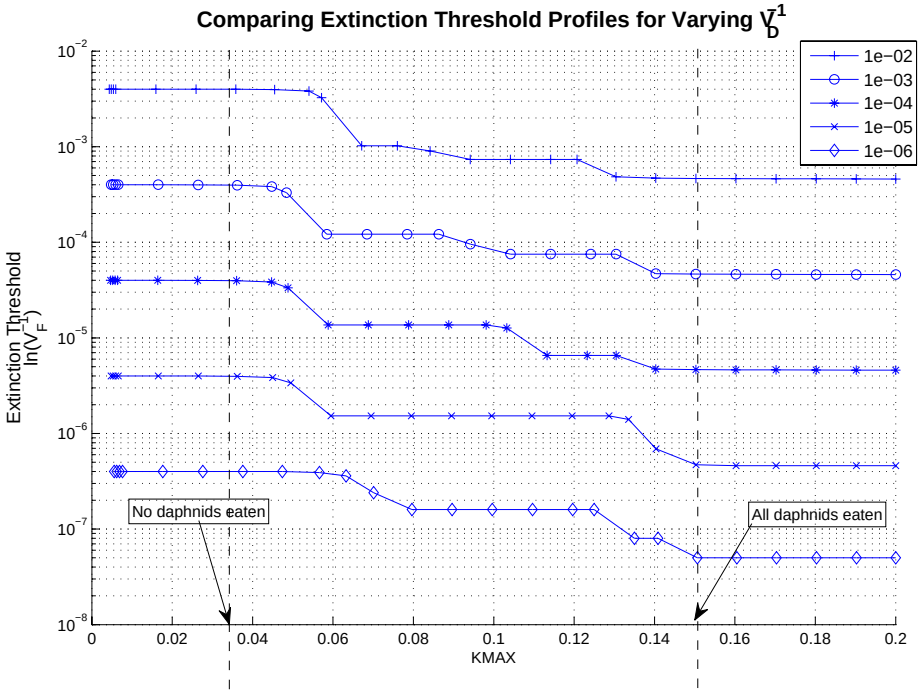


Figure 4.8: Extinction Threshold Curve for $kmin = 0$ and Varying $kmax$ for Various Values of V_D

the protected classes or else the fish will not be able to maintain control of the population. Our prey populations do not have a uniform distribution of size, hence the plateaus.

Construction Method:

For each fixed value of V_D^{-1}, we started at a large value for $kmax$ and stepped backwards across our region of interest. We adjusted our step size backwards based on how much change there was between the previous two calculated values. For each step backwards for $kmax$ we performed a binary search for the extinction threshold. We again use the technique of trapping the extinction value between "good" (prey survive) and "bad" (prey driven to extinction) values for the *fscale* and perform a binary search under automation. (The Python automation script was actually developed first for these calculations because of the thousands of runs required to determine the curves, then later applied to recompute and extend the curves that had been calculated manually in the previous subsection.) Originally the script restarted each search with the same base value for *fscale*, but it was noted that this restricted the output values to a certain fixed subset and as the curve stepped up to the left, then fewer of the runs were to extinction so the calculation was taking longer. The script was modified to take advantage it being a non-decreasing function as $kmax$ is stepped backwards, so the last known good value from the previous step can be used as the starting value for the good value

on the current step. This balanced the extinction and complete runs so that the overall computation time was decreased.

Given various parameters and knowledge we have of the individual models, can we calculate the upper and lower limits for *kmax*? Recall Equation 3.1 the expression of the prey window:

$$10 \cdot L_F k_{min} \leq L_D \leq 10 \cdot L_F k_{max}$$

Let L_D^{min} and L_D^{max} be the smallest and largest attainable daphnid lengths. Likewise, let L_F^{min} and L_F^{max} be the smallest and largest attainable fish lengths. For what value of *kmax* would the fish be unable to predate on any daphnids? If the upper bound of the prey window were smaller than the smallest daphnid, then no fish could possibly feed on any daphnids. Thus if

$$k_{max} < \frac{L_D^{min}}{10 \cdot L_F^{max}}$$

then the condition will be satisfied. Similarly, if

$$k_{max} > \frac{L_D^{max}}{10 \cdot L_F^{min}}$$

then all daphnids will fit into the prey window for all of the fish. Recalling that length is determined from protected structure levels through the allometric relationship

$$length = \Big(\frac{m_{PS}}{\text{Allometric Coefficient}}\Big)^{1/3}$$

then the max and min lengths can be determined for these populations if the maximum protected structure levels are known. Table 4.2 lists the values for our initial populations which results in lower and upper effective limits for *kmax* of 0.0036 and 0.57, respectively. The lower limit agrees with our graphs, but the upper limit seems to be reached by our graphs much earlier than 0.57. Recall that we observed that our population of daphnids tended to saturate growth at around 50% of the maximum. Taking this into account yields a effective upper limit on *kmax* of about 0.3, which is in agreement with our graphs. (This is much smaller than the values we had been given to us for testing in all of our predation parameter sets, so our simulations have traditionally been with wide-open (upper) gape.)

Table 4.2: Parameters for Max and Min Length Calculations

Parameter	*Daphnia*	Units	Fish	Units
Max Structure	0.75	mg	1080	g
Min Structure	0.0082	mg	0.0238	g
% Protected Structure	50%		71%	
Allometric Coefficient	0.002		0.017	
Max Mass PS	0.375	mg	766.8	g
Min Mass PS	0.0041	mg	0.017	g
Max Length	5.72	mm	35.6	cm
Max Length	1.27	mm	1.0	cm

4.1.3 Extinction Thresholds as a Function of *kmin*

By reversing and varying *kmin* while setting *kmax* to infinity, then one would expect similar behavior. The change relative to the predation though is to start releasing the small and young from predation. Once the juveniles are allowed to reproduce at least once, with a clutch size greater than one, then no finite fish population will be able to control the population. We might therefore expect the extinction threshold curves to be more sensitive to *kmin*. This is what we report in Figure 4.9. Compared to Figure 4.10 from Henson, none of our curves reproduce the long shoulder she reported. Our step size was larger than hers, and our initial populations saturate and grow faster than hers as was seen earlier. Thus the juvenile period, the end of which is only dependent on reaching a fixed length, is shorter. This explains why we do not see quite same length of shoulder before approaching the right-hand asymptote. As before, the level portion to the right in our figure is a result of maxing out mortality range which we allowed to be searched. This region indicates that the extinction threshold cannot be determined and probably does not exist.

Similar to our work with *kmax*, can we calculate the upper and lower limits for *kmin*? Below what value of *kmin* would all of the fish be presented with all sizes of prey? If the lower bound of the prey window of the largest fish were smaller than the smallest daphnid, then daphnids of all sizes would fit in the prey window of all fish. Thus if

$$k_{min} < \frac{L_D^{min}}{10 \cdot L_F^{max}}$$

then the condition will be satisfied. The right hand side is the same value calculated for the left-hand asymptote for *kmax*. The size required for onset of reproduction is 2.5 mm. Similarly, if

$$k_{min} > \frac{2.5}{10 \cdot L_F^{min}}$$

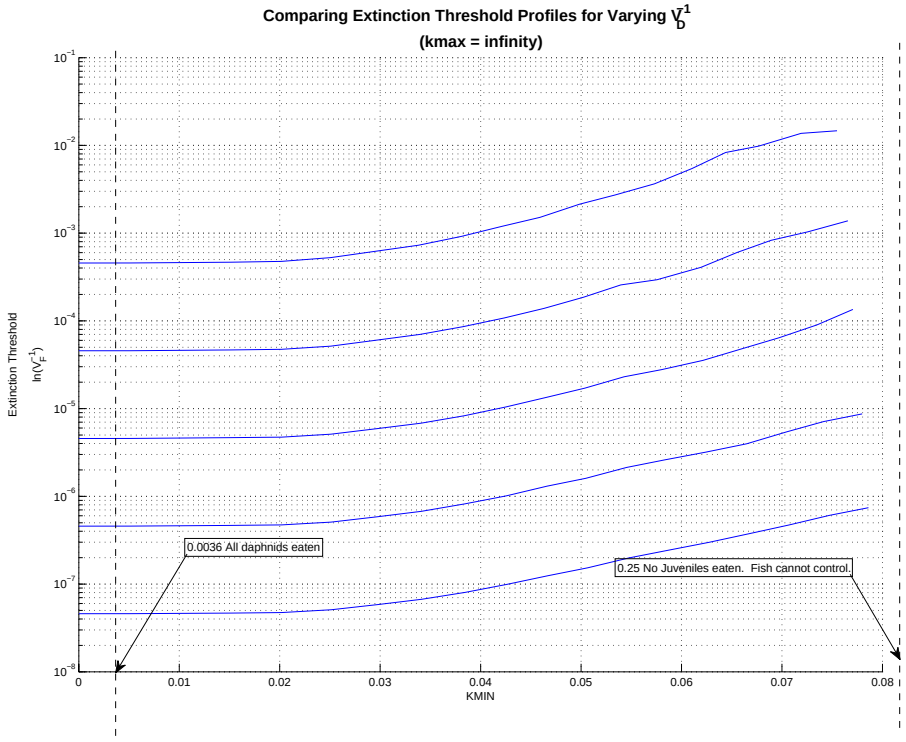


Figure 4.9: Extinction Threshold Curve for $kmax$ = infinity and Varying $kmin$ for Various Values of V_D

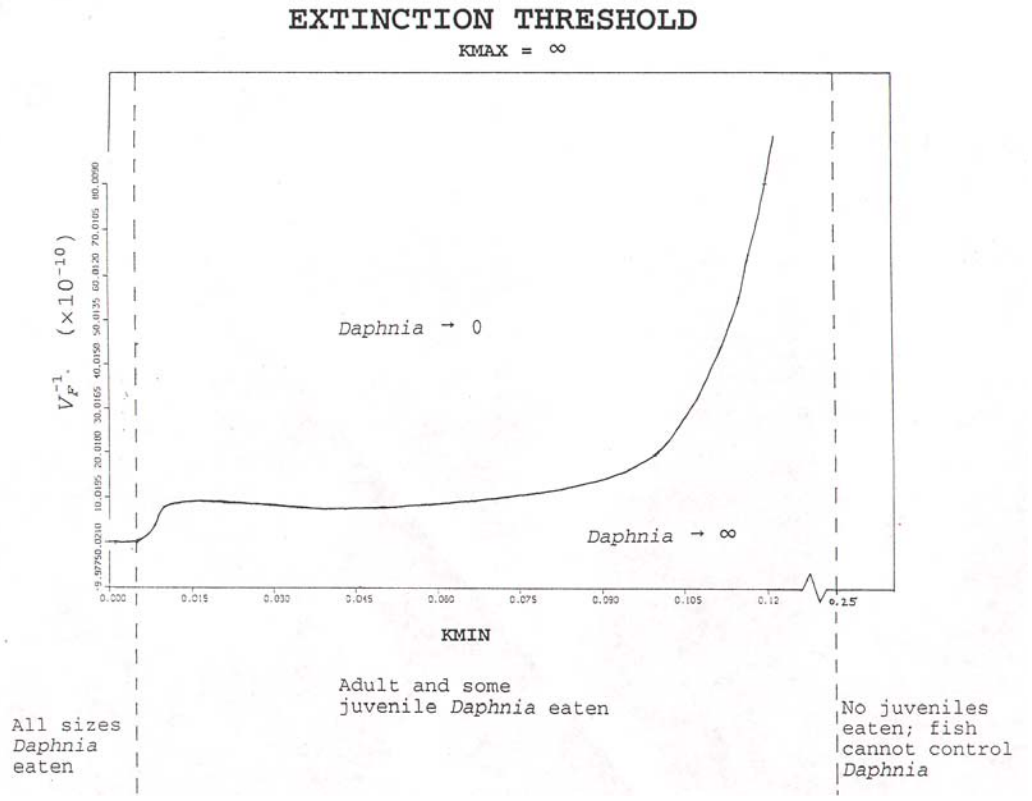


Figure 4.10: Extinction Threshold Curve from Henson for *kmax*= ∞ and Varying *kmin*

then all juvenile daphnids will escape predation and reproduce at least once, thus no finite fish population could control them. This results in lower and upper effective limits for *kmin* of 0.0036 and 0.25, respectively.

4.1.4 Comments

Although the extinction threshold is defined for a relatively short period of time and under conditions of unrestricted growth for the prey population, the results and techniques can be extended and used for populations in which density-dependent mortality is acting. It gives a rough idea of where the edge of support is, even for longer runs. Further, the concept of replication volumes is useful to keep in mind as it indicates how insulated a prey population is from extinction from predation. If it is close to the edge of supportability, then its dynamics will be much wilder than one further from the edge.

4.2 Size-Dependent Predation

In this section, we are focused on the persistence and extinction of the *predators* due to size-related effects. The previous section addressed the extinction of the entire prey population and thus the predators. In our models the predators have a selective window in which they draw their resource. Such prey windows are observed in nature, with some prey even responding by adjusting their growth rates in order to avoid the predation risk (Bystrom et al., 1998).

With a restricted window from which the fish are allowed to draw their resource, there are several potential results that lead to starvation.

1. The most basic problem could be that there is no resource available in the window either because the prey are all too big or too small.

2. Relatedly, because the window is a function of the fish's length and because length is a non-decreasing function in our model, the fish could grow off of their resource support. They cannot shrink to pick up additional resource, nor can they advance in size without additional resource. In the absence of resource entering the window, then only starvation can result.

3. Lack of sufficient resource in the prey window could be caused by the size classes being under populated or the prey moving so quickly through the predation window via rapid changes in size, that the predator again is left with an insufficient or a wildly fluctuation resource level.

4. Further, perhaps the predators deplete the resource in their window, thus invoking starvation on themselves.

These scenarios arise from the dynamics of the size structure in the prey population. Finally, because of the competition of resource based on weight between the other fish, a smaller fish's growth may be stunted or prevented altogether through competition with the larger fish because their portion of the resource is insufficient for growth.

With our models being across trophic levels, we have additional complications that arise from very different abilities for the populations to respond by increasing or decreasing population numbers. This is termed the *numerical response*, and is the complement of functional response. Both terms were introduced by Holling (Holling, 1959). The fish have an annual window in which they can give birth; whereas, the generation time for the *Daphnia* is four to six days. In order to explore the depletion limits of the fish population we will artificially adjust the numerical response.

4.2.1 Goals

In order to better understand size-based predation, first, I wanted to visualize the effects of predation on the prey both immediate and long-term. Secondly, I wanted to see the resource density dynamics caused by varying resource. Next I wanted to understand and put specific meaning to the term *Predation Pressure* and to see what I kept envisioning as *Predation Waves.* I had a picture in my head of these concepts, but I wanted to figure out a way to visualize them. (Perhaps more significantly, I additionally had the motivation to figure out why the models were recalcitrant. So I focused on visualizing the actual predation in order to confirm correct behavior. What amount of prey was one predator receiving? Could the number of prey be blowing the top out of the energetics because something was not being summed correctly? These questions were also behind some of the ideas in this section.)

Predation Pressure is used often in the literature. From the perspective of the prey it is used as a synonym for predation risk. Or it is used to indicate a change in predation state, for example if a species is found in two different lakes, but is only exposed to predation in one and thus takes some action like hiding in a refuge or moving deeper in the lake, then this action may be described as a result of predation pressure. From the perspective of the predator it may be used to describe the amount of outtake or the amount of energy required of the prey system. The term is used often, but I could not find a mathematical description. (It even appears in one of our own predator-prey model

papers Jaworska et al. (1995).) I was trying to understand predation pressure from an individual perspective.

A feature of our models that masks these concepts is that of sharing resource. With two different predators which share resource in overlapping prey windows, the effects become muddled. For this reason I decided to restrict my attention to the effects of one fish and to try to understand its effects. This is analogous to the analysis technique behind functional response: understand the effect of one predator on a prey population and then multiply by the number of predators to get the effect for all.

4.2.2 Effects of Structured Predation

We maintain the initial populations and assumptions utilized in the previous section, in particular that no mortalities are imposed except those due to predation or reaching a fixed maximum age. We maintain this in this section so that the effects of predation are not obscured by other mortality effects.

Total Predation Effect

The overall idea in this section is to use the fact that we have both a population and a predator-prey model available that are identical in their effects on the prey population except for those imposed by predation. (As part of the development process I repeatedly confirmed that the output of the predator-prey version when restricted to the prey population matched exactly the output of the population version.) Thus we can run the same initial population through both models and examine the difference. Comparing the two models, then we can see the differences caused by predation since we have not imposed any other mortalities. (This idea seems obvious now, but I was searching for so long for some mathematical expression and did not think of this idea until thinking about differencing integral expressions for predation.) Since we are trying to understand the effects of one fish, then we limit the predator population to one cohort.

The experimental design was to take the initial populations utilized in the previous section and isolate out one fish, testing it against the prey population. There was little diversity of size and numbers left in the fish population as tested in the previous section, but I wanted to stay tied to the first section. We will remedy this in the next section. I chose a single fish cohort with CHARID 135, age 409 days, initial population density of 0.2, lipid mass of 89 grams, structure of 277 grams, and protected structure of 210 grams, which is equivalent to a length of about 50 cm. (These populations

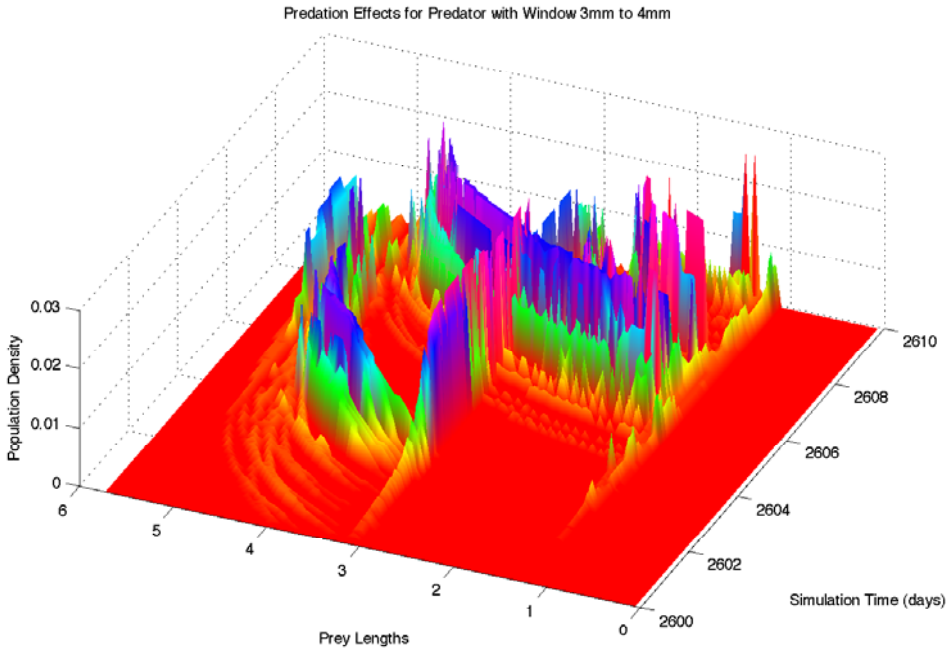


Figure 4.11: Predation Effects for Chosen Fish over 10 Days

used the old allometric parameter value, so the lengths are inflated.) The populations were run for 10 days of simulation. One run was performed with the predator-prey model in "Daphnia-only" mode. The second run was done normally. *kmin* and *kmax* were set such that the one fish could prey on *Daphnia* with lengths in the range [3 mm, 4 mm]. The reason this range was chosen was that previous experience indicated this to be a range of lengths through which the daphnids quickly grow, so we should observe the effects of fluctuating resource level.

The resulting differences between the prey in the unstressed state and the prey stressed by the predator are shown in Figure 4.11. This was not quite what I expected and it revealed some things I had not previous observed about the prey population. The predation clearly starts its effects in the correct size range. In terms of overall numbers of prey removed was small, and of no consequence to the prey population. As these daphnids move out of the window, their trails are clear. At about 2602 days the first effects on reduced birth count is seen in the size classes near 1 mm. The movement out of the prey window and the delayed effect on the egg classes was expected. A larger predation effect is observed at about 2603 days. Why the sudden predation effect there? Further, why does the wall build up along lengths around 3 mm? It seems that the predator was not uniformly predating, but perhaps was favoring smaller size classes.

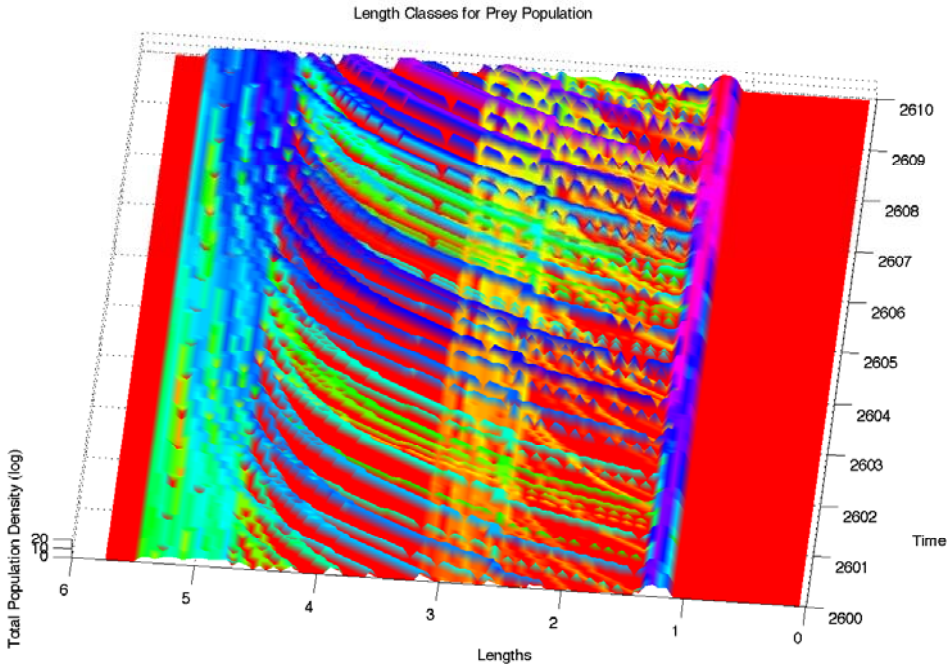


Figure 4.12: Prey Population Length Class Structure over Same 10 Days

The mysteries are cleared up by looking at the length classes for the prey population in Figure 4.12. I chose the oblique angle in order to emphasize that there is a dichotomy of sizes built-up for this prey population. There is a segment of the prey population that is undersized with maximum length just over 3 mm. And there are the normal sized ones which grow to about 5.5 mm. The characteristic curves emphasize the variable numbers of prey in the size classes through time. There is a concentration of cohorts moving through the prey window at about 2602.5 days followed by a relative sparsity. The predator is taking from the undersized daphnid classes in between spurts from the normal sized classes. This is what it should do. It illustrates the dynamic relationship to the prey population. If the prey window were just a little higher and missed the resource building up at 3 mm, then the fish would have been in a feast or famine situation. This is illustrated in Figure 4.13 where several resource density curves are pictured for different locations of the prey window on this same prey population run over the same 10 days. (Note that the high-densities of resource ($> 1g/cm^3$) are because density-dependent mortality is not being applied.)

This version of predation effect illustrates the total effect of a predator over a period of time. Total effects include subsequent changes in births and the ripple effect through time caused by predation. The direct effects are the result of the outtake integral and direct feeding. The indirect

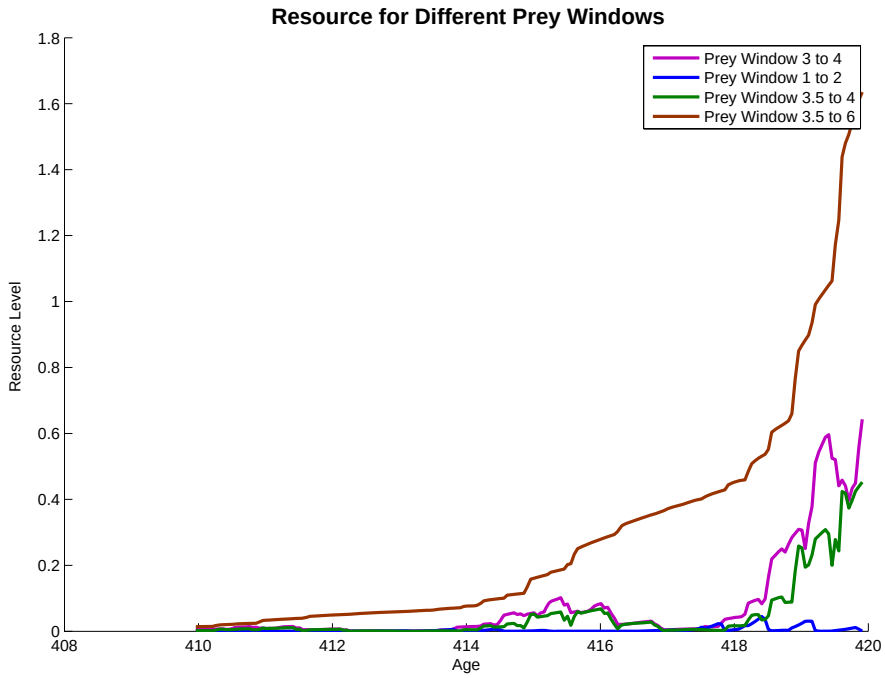


Figure 4.13: Resource Levels for the Same Fish and Population, but Different Prey Windows. Note that Prey Window 3.5-4 Resource is consistently under that for Prey Window 3-4. Further note that the lowest prey window has comparatively little resource, because egg classes cannot be directly predated.

effects are the subsequent changes in the population structure that are caused by the removal of some number of prey. These indirect effects quickly clutter the graph and confuse the effect.

4.2.3 Effects of Structured Predation

We now go through each of the potential reasons for lack of sufficient resource which were given in the introduction to this section.

All Prey Too Big or Small

For sufficiently small values of *kmax* and sufficiently large values of *kmin*, the limits of the prey window will positioned below or above the prey size classes. Similarly, if *kmin* equals *kmax*, then no prey items can be consumed.

These extreme cases could be viewed as arising from incorrect setup of the model. In the case that *kmin* equals *kmax* obviously it is. But there can arise "alternative states" as they are termed, where the predator finds itself positioned above or below the available prey sizes. Such a case is described in Persson et al. (2007), where the removal of a top predator species of fish has allowed the previously predated species of fish to grow too large to be easily predated, but insufficiently large to have higher fecundity, so the preferred YOY of the prey are suppressed. This slows or prevents recovery of the top predator species. This observation led to the counterintuitive culling of the prey species in order to promote the recovery of the top predator species by returning the lakes back to their previous predator-dominated states.

Growth Off of Support

Related to the first case is where the predator receives sufficient resource for a period of time to grow, but the growth then moves the lower limit of the prey window to be above the size classes that were supporting the fish. I term this as *growing off of its support.* This is not a result of incorrect setup of the model. Note that this does not mean that the resource level for the fish is zero. It just means that it is insufficient for additional growth. Starvation may or may not occur depending if the resource available is sufficient over time to meet energetic requirements. This situation can also be caused by smaller fish consuming the resource before it can grow into the size classes required by the larger predator. (This is a common natural method by which the smaller fish can outcompete larger fish; see Bystrom et al. (1998); De Roos et al. (2003); van Kooten et al. (2007); Claessen et al. (2002).) Further note that new resource will not immediately result in increased length of the

predator because the mass of structure must increase to a point above the previous high-water mark at which the level of protected structure was set previously.

Fluctuating Availability

The resource level can be fluctuating as we saw in the visualization and Figure 4.13. This may cause the growth of the predator to be slowed or prevented when its prey window is passing over a region of high fluctuation. Fluctuations in resource can be exacerbated by smaller cohorts consuming from smaller prey size classes.

Depletion

Depletion of the resource in the prey window by the fish cohort itself is typically not sufficient to drive the prey population to extinction. This is similar to an extinction threshold for a single cohort. If the juvenile classes of prey can reproduce once, then no finite predator population can control the prey population. So if the prey window of the fish does not include the juvenile classes, then it cannot drive the prey population to extinction. It can consume all of the resource available to it, so new resource is only available from the growth of the smaller size classes of prey. This input resource may or may not be sufficient to support the predators and is a tenuous survival at best. With density-dependent mortality or some other effect, then indirectly the prey population could be driven to zero by sufficient removal of segments of the population.

One can define what I term a *Quiescence Threshold* similarly to the *Extinction Threshold*. Define the function $np(kmin, kmax, L_F)$ to be the number of prey cohorts available as resource in its prey window for a fish of length L_F.

Definition 4.2.1 (Quiescence Threshold) *Given a fixed number of days, N, values for kmin and kmax, the initial population distribution $\hat{\rho}(a, m)$, volumes V_D and V_F, and a fixed fish cohort, F, define the quiescence threshold for F to be the*

$$inf\{q \in \Re^+ | F \textit{ with density } q \textit{ drives } np(kmin, kmax, L_F) \textit{ to zero in } N \textit{ days}\}$$

What we are doing is by arbitrarily increasing the population size on the fish cohort, when does it deplete its resource? It could also grow off of its support, but the presumption is N is small, so that growth is not a significant concern.

Conceptually I picture a gape-size prey window as a type of band-pass filter: it reduces or eliminates any frequencies in a precise range. Prey items above the effects of the filter have a limited

lifetime. If no prey items survive through the filter, then all larger size classes will be eliminated. The Quiescence Threshold for a fish gives an indication of how strong of a filter it is. Obviously, this is not an experimental value that is directly obtainable, and it depends on the prey population. It is an exploration of the concept of predation pressure from an individual perspective. It also gives a value at which juvenile classes could block all resources from progressing upward. This latter is a natural occurrence and is behind the Dwarf-Giants dynamics reported in Claessen et al. (2002) for cannibalistic fish populations. The populations are dominated by small fish that consume all of the zooplankton resource. A few fish grow to an extraordinary size because they grow large enough by cannibalize. The other size classes die out by starvation.

One can calculate the Quiescence Threshold exactly if N is one time step, dt. Multiplying Equation 3.2 by V_D gives the total resource mass for a single fish with a certain prey window. Equation 3.3, the total outtake integral, gives the total consumption by all fish of the same size. Since there is only one cohort, then the outtake integral reduces to a single value with units grams/day. Multiplying by the time step, equating, and solving for ρ_F, yields the exact value for the Quiescence Threshold for fish cohort F.

$$\text{Quiescence Threshold}(F) = \frac{m_F}{f(x_{m_F})dt} \int_{k_1 m_F}^{k_2 m_F} \frac{m_D \int_0^\infty \rho_D(t,a,m_D)da}{\int_0^\infty \int_{m_D k_2^{-1}}^{m_D k_1^{-1}} m\rho_F(t,a,m)dmda} dm_D \tag{4.4}$$

If one gives a longer time for quiescence to occur, then integration over time does give an equation for Quiescence Threshold, but a program that varies the population density and then runs the simulation over the prescribed time period is required to determine a value. The first point of zero resource could be earlier than N days and new resource characteristics will encroach for a period of time as they grow into the window. I performed this search with another python script and a simple modification to the predator-prey model that outputs a notice if there are no items in the prey window.

Table 4.3 gives the Quiescence Threshold for the fish characteristic chosen for the predation visualization. As the size of the window is increased, then it takes more fish to deplete the prey which is a behavior one would expect from pressure. As we increase the segment of the population over which the predation is applied, then the population density must also be increased in order to exert the same amount of predation pressure.

Table 4.3: Quiescence Threshold Calculations

Lower Bound	Upper Bound	Window Size	Quiescence Threshold
3	3.25	0.25	0.0162125
3	3.5	0.5	3.3063889
3	3.75	0.75	7.8954697
3	4	1	28.6922455
3	4.25	1.25	43.7097549
3	4.5	1.5	92.2079086
3	4.75	1.75	142.5619130

Competition

The last effect that can constrain uptake in structured predation is competition and sharing of resource. I had developed the other ideas for one fish cohort, because sharing complicates the concepts and mathematics, but this turns out to be an important limit on the otherwise explosive growth of small fish as we shall see in the next section.

We have already been through the design and reasons for competition, encounter rates, and sharing of resource, so they are not repeated here. But do note that if *kmin* is zero, then competition for resource will always be present and the larger fish will dominate resource uptake. This dominance can be so heavy that the smallest fish subsist or starve. If *kmin* is greater than zero, then there is a corresponding decrease in competition from larger fish on the resource available to small fish.

4.2.4 Predation Pressure and Zero Growth Condition

Looking at the definition, calculation, and behavior of the Quiescence Threshold, then a process leading towards one possible definition of predation pressure emerges. We use Equation 3.3, the total outtake integral, to tie the individual predator's functional response/outtake to the cohort of which it is a representative. Since the population is a composition of cohorts, then integrating over all fish sizes yields a total outtake for the population in units of grams (of prey biomass) per day. The prey population must be able to sustain this rate of predation. What had previously gone into increasing the biomass of the prey population by growth is now partially siphoned off by predation, thus acting to decrease the prey population's biomass. We will observe this effect in the next section, where its interplay with density-dependent mortality will be significant. The Quiescence Threshold gives the extreme value of this depression at which all growth is consumed by predation.

Representing total consumption rate and its effects on the prey population are both features I was looking for in a definition of predation pressure. There are some features that I wanted too that total consumption does not feature. It does not give clear direction on sufficiency of the resource, long-term survival/growth of the individuals, and carrying capacity of the environment. Further, it is dependent on the prey population as part of its definition. As we just saw, underlying several of the consequences of structured predation is a condition where no further growth is possible without additional resource. The level of resource may be sufficient to sustain, but is insufficient for additional growth. At what point is this?

A related concept used by de Roos and Persson repeatedly in their papers as an analysis tool they term both *zero-growth* and *critical resource level*. See Claessen et al. (2002); De Roos and Persson (2001); Claessen et al. (2000); Persson et al. (1998) for development and application of these functions to an aquatic, size-dependent predation, physiologically-structured, predator-prey model which is very similar to ours. It features a predation window, cohorts, birth combining, periodic and discrete births, energetics, etc., similar to ours. Their model has a zooplankton resource, but the fish can also meet energetic requirements through cannibalism. Further, see Persson and De Roos (2006) where it is reported that the location of the minimum of the zero-growth function determines one of three ultimate population dynamics. The Zero-Growth equation arises from going back to the individual model and setting the equations expressing growth to zero. This expression gives the ingestion required at each step in order to exactly balance the work and maintenance losses. This is used to calculate the lowest resource density that an individual of a specific size needs for maintenance which is termed the *critical resource level*. The critical attack, metabolic, and gut clearance rate functions are all directly in terms of the length and size of the fish. The minimum and shape of the critical resource density curve determines the ultimate population dynamics. They had very carefully examined the time series outputs before and knew there were three types of outcomes which are tied to this curve (Claessen et al., 2002). The important feature to us of this equation is that it indicates the level of resource at which an individual can persist without growth or starvation. This is calculated from the individual without dependence on the prey population.

Theoretically we can apply the same technique to our models by zeroing the two growth model ODEs in Equations 1.14 and 1.15. The loss terms in this equations express the amount labile lipid/structure required for energetic requirements. Direct calculation is prevented by our current formulation because we do not directly tie length to some of the components and have functions like percent lipid in resource that vary with the prey. Calculation is further complicated by the energy integrator feature. Although, until I fixed the ED/EA fraction was always one, so the loss term was

simplified considerably, so that one could by estimating a few values calculate the critical resource level. (See the appendix to Hallam et al. (2000) for derivation of the energetics equations.) Another problem is that the same level may not apply to both equations. Theoretical issues aside, we will detect zero-growth and near-zero growth conditions in our extinction/persistence maps in the next section.

The zero growth condition does have several features that such as not depending on the prey population directly, can give estimates on carrying capacity, can be directly calculated from the model (at least approximately), and precisely defines the conditions of growth or subsistence underlying structured feeding. Summing the actual outtake leads to the rate at which the prey population must replenish to survive. Whereas the zero-growth condition gives the resource level at which the predators must feed to survive. Conceptually, I settled on these as practical and theoretical expressions of predation pressure. We will see both effects in the dynamics maps in the next section.

A note for completeness: When predation pressure is interpreted as predation risk, then the question becomes how long is a prey item exposed in certain prey windows? Since we have constant resource and the growth equations, then we could calculate predation risk in terms of exposure times. But, since we do not model any direct mechanism by which the prey item can change its exposure to predation — it cannot grow faster or slower, for instance — there is no benefit to pursue this direction with the models as they are currently.

4.3 Extinction, Persistence, and Compatibility

In the sections preceding we have analyzed extinction (or quiescence) for the prey and predators separately. In this final section we examine conditions for long-term persistence: Given two populations which are themselves persistent as population models, for the four parameters which define our structured predation, is there a region of parameter space such that the predator-prey model persistent (i.e., the populations coexist with the prey population providing sufficient resource for the predating population to thrive)? I term this region the *Compatibility Region*. This section puts together all of the analysis tools introduced in this chapter thus far and was inspired by the dynamics studies in Claessen et al. (2002). Note that for the definitions of Extinction and Quiescence Thresholds in the previous sections that additional types of mortalities can be imposed on the populations.

4.3.1 Mortalities Imposed

The first problem to solve is to be able to run the model longer than 100 days. The initial populations chosen are the same ones described and used previously in Section 3.4 for the Performance Runs. Here we make the additional notes that the ages span 364.47-2915.43 days and the sizes span 21.6-31.9cm. The initial total biomasses for each population are *Daphnia* = 1836 mg and Fish = 157,460 g. A total run time of 2600 was chosen so that all of the fish in the initial population would be removed before the end of the simulation; so if the population is to remain viable long-term, then it must be through replenishment by the fish born during the course of the simulation (recruitment).

In addition to maximum age, we now impose density-dependent mortality effects on the *Daphnia*. Without this control on the *Daphnia* population, given unlimited resource, it will exhibit exponential growth. Recall the shape of density-dependent mortality curve in Figure 1.13, with a well around an optimal biomass. For populations whose total biomass exceeds the upper lip of the well, a very high mortality is imposed which drives their biomass down. Dropping below the optimal biomass begins an increase in density-dependent mortality (models undercrowding) which causes further decreases in biomass. The effect for a population whose biomass declines below this value is usually a rapid decline to extinction. It was proven in Henson (1994) that density-dependent mortality drives out the ecotypic diversity of the population in a population model. This is another reason I delayed the invocation of this until now, but it is required now for us to study the long-term effects on the predator population.

Only maximum age mortality is imposed on the fish population as a whole. Starvation will occur for an individual through lack of sufficient resource once all its stores are depleted. Starvation is considered to occur when the structure stores drop to a certain percentage above the level of protected structure. As described previously I settled on a value of 80% of peak mass structure for the starvation threshold compared to 71% for the protected structure threshold. This yielded starvation times of about 80 days for large fish and about 240 days for smaller fish. These are excessively long, but starvation can also occur at reproduction times, when bulk allocations from lipid and structure stores are shunted to eggs. Starvation during the mass allocations for births did not occur with it set to this value as it did for higher values. (A value of 90% yielded starvation times of 15 days for large fish.) Young-of-year mortality is imposed on the newborn fish aged 18 to 60 days pushing the towards a fixed population density (not directly biomass related). This was usually set to 8000 with a resulting newborn biomass of 250 g, but I did vary it some to see if I could induce persistence by allowing additional recruit population size. The role of density-dependent

mortality is not required as a control for the fish population, because they are competing through shared resource.

It is built into the model that reproduction for both the fish and *Daphnia* can be delayed or prevented altogether if insufficient stores are available when the reproduction window arrives. Thus starvation or subsistence can prevent reproduction. We have not noted this before this point because we were always operating under conditions of sufficient resource for growth and reproduction.

4.3.2 Experimental Design and First Results

With knowledge from the Section 4.1, we fixed the value for *rscale* to be 0.2×10^{-5} which corresponds to V_D equal to 500 liters or approximately 130 gallons. I determined through a binary search that an extinction threshold for 1000 days for a wide-open gape was above the value *fscale*$= 0.510 \times 10^{-3}$. This corresponds to a value for V_F of 1 million liters or 265,000 gallons. This represents a volume of replication value of 2000 — certainly conceivable. Further, I did not observe the biomass of the *Daphnia* population to drop below 1000 mg for this value (bottom threshold for density-dependent mortality is 860 mg), so I thought it a reasonable value to begin with. The only other value I had from my archives for *fscale* was 0.2×10^{-4} which corresponds to 25 million liters or 6.6 million gallons which seemed to be way too conservative of a value for anything interesting to occur and a volume of replication of 51,000 which seemed way out of range. For values above *fscale*$= 0.530 \times 10^{-3}$ the *Daphnia* population went extinct before the end of my test simulations.

With these two values chosen, all that remains is to choose are values for *kmin* and *kmax*. Choices for these two parameters form the parameter space over which I searched for regions of compatibility with *kmax* on the horizontal axis and *kmin* on the vertical. What constraints are there on these besides they are non-negative, real-valued parameters? We know $kmin < kmax$, because otherwise the gape size is completely closed. This restricts our search region to the lower triangle in the first quadrant. An additional constraint on *kmax* comes from the smallest of fish at least must be able to have at least the smallest of daphnids in its prey window. Thus $10 kmax L_F^{min} \geq L_D^{min}$. Plugging in values from our populations (see Table 4.4) yields an effective lower bound for *kmax* to be 0.1272. Finally, similar to when we searched for Extinction Thresholds for *kmax*, when *kmax* is chosen such that the entire prey population is in the smallest of fish's prey window, then there will be no difference between using this value for *kmax* and setting $kmax = \infty$. Using our populations' values, this yields an effect upper limit of 0.5735 on *kmax*. Thus the region to be searched for compatible values of *kmin* and *kmax* and the expected outcomes at the edges is shown in Figure 4.14.

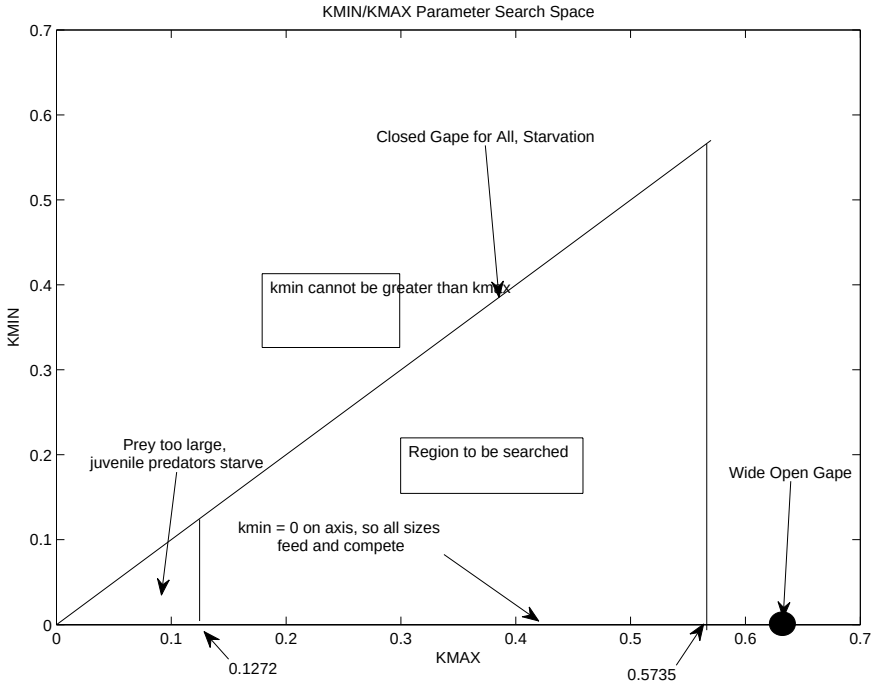


Figure 4.14: The KMAX-KMIN Parameter Space

Table 4.4: Maximum and Minimum Individual Sizes

Parameter	Value	Units
Daphnia		
Max Structure	0.75	mg
Min Structure	0.0082	mg
% Protected Structure	50 %	
Allometric Coefficient	2×10^{-3}	
Max Length	5.72	mm
Min Length	1.27	mm
Fish		
Max Structure	1080	g
Min Structure	0.0238	g
% Protected Structure	71 %	
Allometric Coefficient	1.7×10^{-2}	
Max Length	35.6	cm
Min Length	0.998	cm

The initial population has lengths of 21 to 32 cm. For any resource to be available to the initial population from the prey population, then *kmin* must be less than 0.0159. This is the value at which the largest of the initial fish (31.9 cm) can still feed on the largest of the daphnid population (5.1 mm). We search the entire parameter space, but this condition puts a strong condition on the viable values for *kmin* to be near the $kmin = 0$ axis.

Generally, for a given fish of length L_F, in order for any prey items to appear in its prey window, then $kmin < \frac{L_P^{max}}{10L_F}$. This gives the restriction for the largest of prey items to appear at the bottom edge of the fish's prey window. Thus, for all except the smallest of fish (1 cm), this means that most values of *kmin* will eliminate their ability to forage on this prey population.

I added displays of *Daphnia* and fish population total biomasses as the programs ran which were recorded in addition to all predation and population structure data. The fish population biomass output was further split into that for the initial population and the recruits. With these I could monitor what was happening while the program ran. A python script was written that parsed the space into about 100 runs, created the appropriate predation parameters files and ran the predator-prey model repeatedly. After it completed, then I analyzed the dynamics of each output file and compiled a table of the results based on the outcomes. For all combinations of the parameters extinction occurred, but there were a variety of pathways through which extinction occurred. The Extinction Map for this set of runs is shown in Figure 4.15. The size of the dots indicate how close to a complete run the simulation reached. None completed, but the ones along the kmin=0 axis ended at about 2550 days.

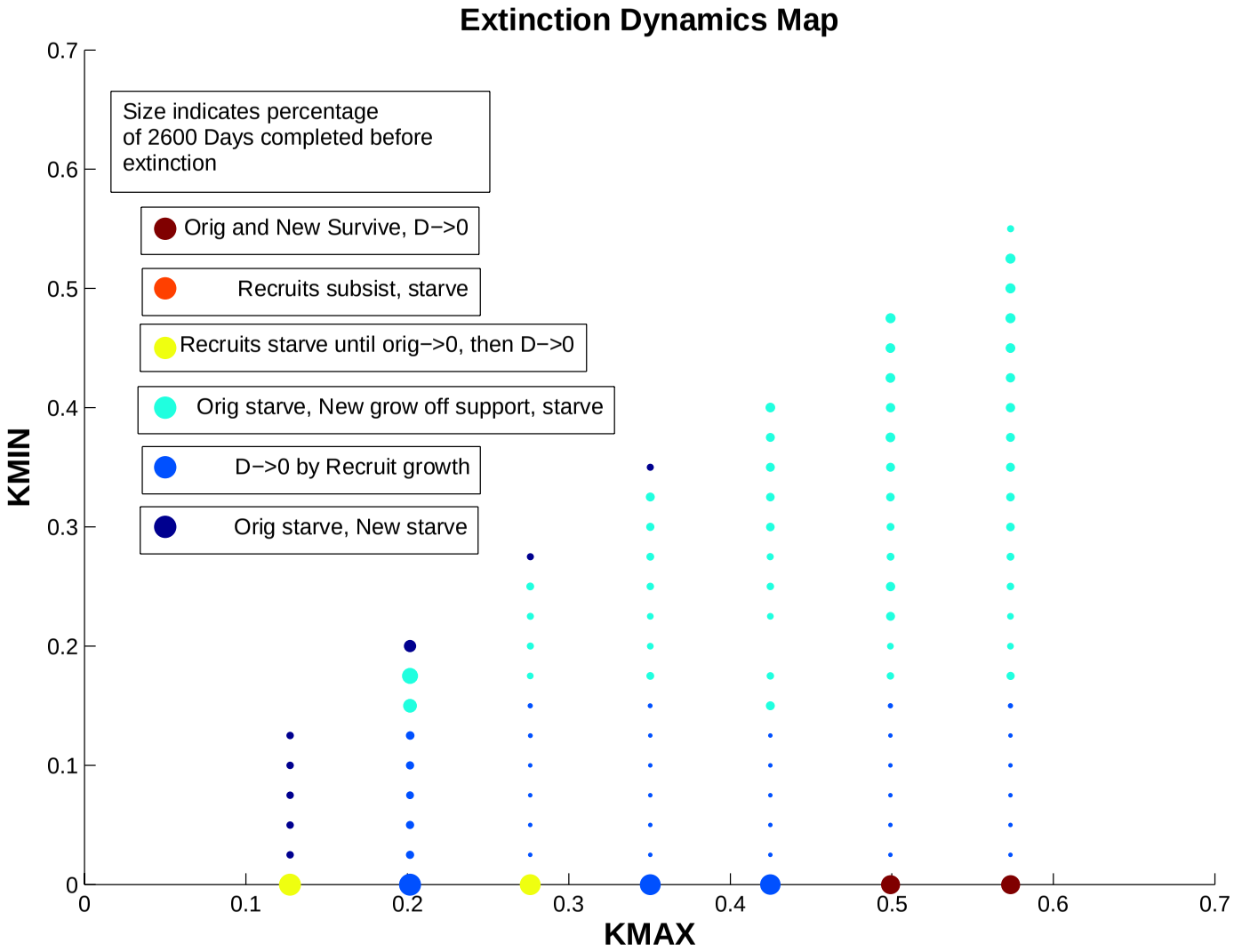


Figure 4.15: Extinction Map for First Set of Runs

4.3.3 Extinction Pathways

I was surprised by two things once I plotted this map. One is that I did not find any values leading to persistence. The second was by the variety of methods through which extinction occurred. So many of the runs looked like they would make it, the populations seemed healthy and growing, but then would go extinct for one reason or another. These different pathways to extinction are revealing and grouped into regions in the parameter space. Each pathway is now described from my original list and examples given. I later pared this list down.

Type 1: Initial and Recruit Populations Starve

The first extinction pathway occurs at the upper and left edge of the parameter space. On the upper edge the gape size is nearly closed so minimal feeding occurs. On the left edge, for the non-zero values of *kmin* tested the initial population could not see any resource. For the recruits there was too little resource available in their window, because *kmax* was chosen such that the recruits just had access to the smallest (non-brood) items of the prey population.

The total biomass values for this extinction pathway are shown in Figure 4.16 for the prey and predator population. Figure 4.17 shows the recruits and initial fish population biomasses. Note the number of days to starvation. There is some feeding occurring for the recruits, but at unsustainable levels. Also note the precipitous drop for the recruits caused by YOY mortality; the biomass levels off at about 250 g. The decreasing modulation of the biomass levels for the *Daphnia* population is typical when density-dependent mortality is imposed.

Type 2a: Prey Driven to Extinction by Recruits

In this scenario, the initial population had been removed from the fish population by starvation through growth off of its support. This left the recruit population without competition. Often, in this situation, with relatively few numbers and biomass, it was able quickly drive the prey population to extinction.

I give two different examples for this extinction pathway. In the first example, Figures 4.18 and 4.19, shows the typical example where the recruits, as soon as they start feeding, rapidly drive the prey population to extinction. In the second example, Figures 4.20 and 4.21, shows an example where the recruit population grows slowly, but once it increases in size sufficiently to start growing rapidly, it again drives the *Daphnia* to extinction.

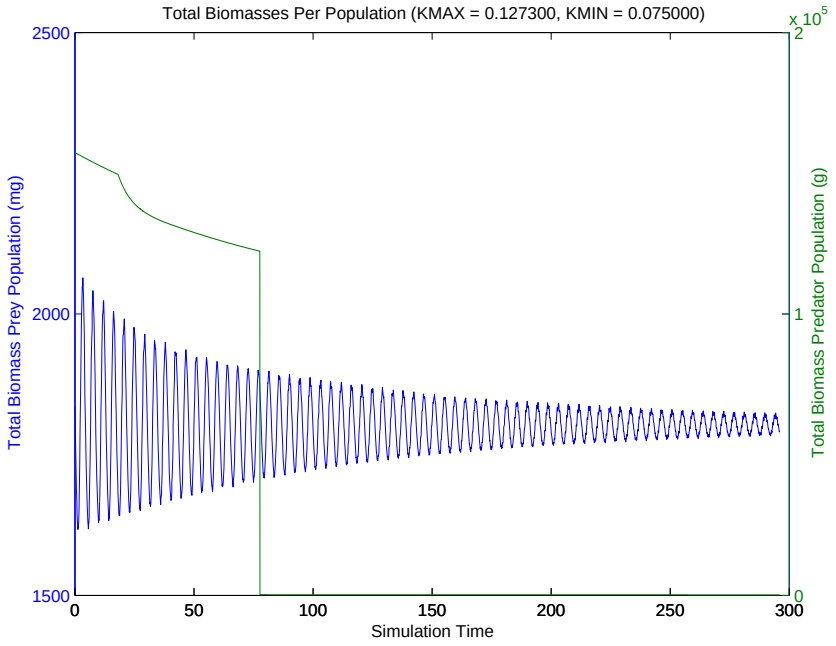


Figure 4.16: Total Biomasses for *Daphnia* and Fish Populations for Type 1 Extinction

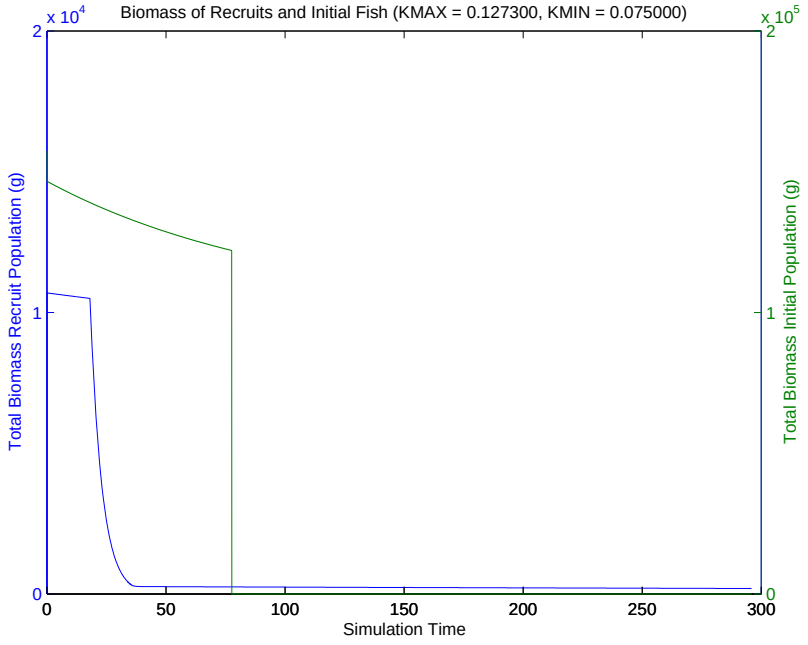


Figure 4.17: Total Biomasses for Recruits and Initial Fish Populations for Type 1 Extinction

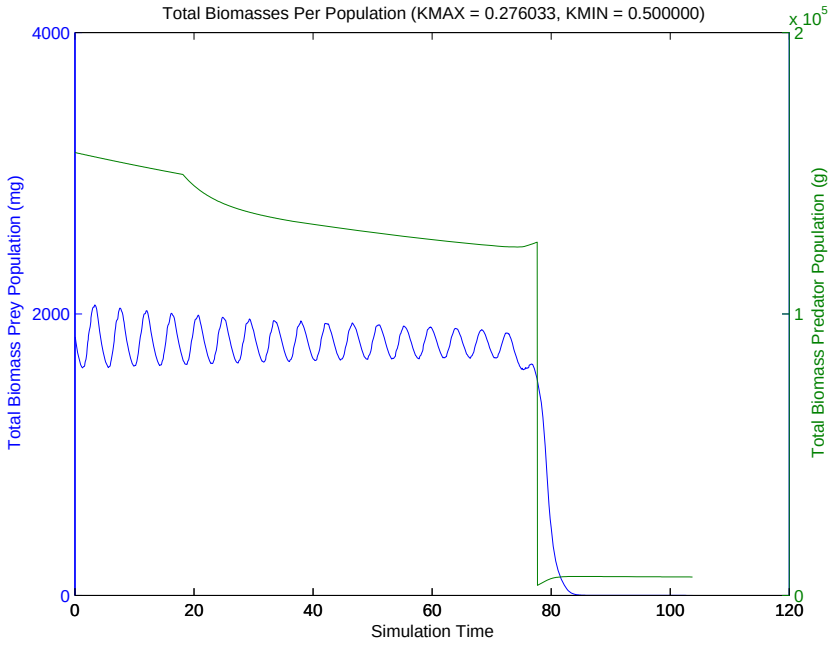


Figure 4.18: Total Biomasses for *Daphnia* and Fish Populations for Type 2a Extinction

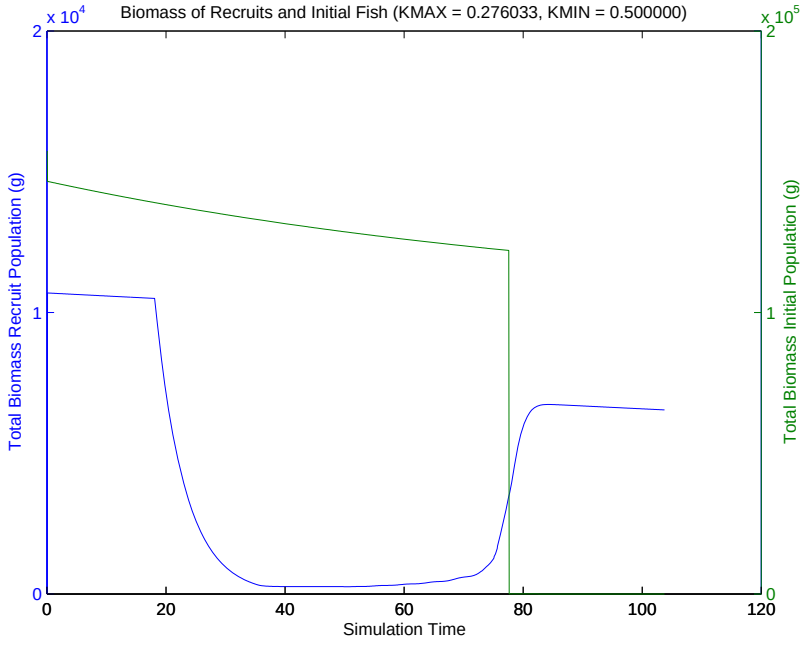


Figure 4.19: Total Biomasses for Recruits and Initial Fish Populations for Type 2a Extinction

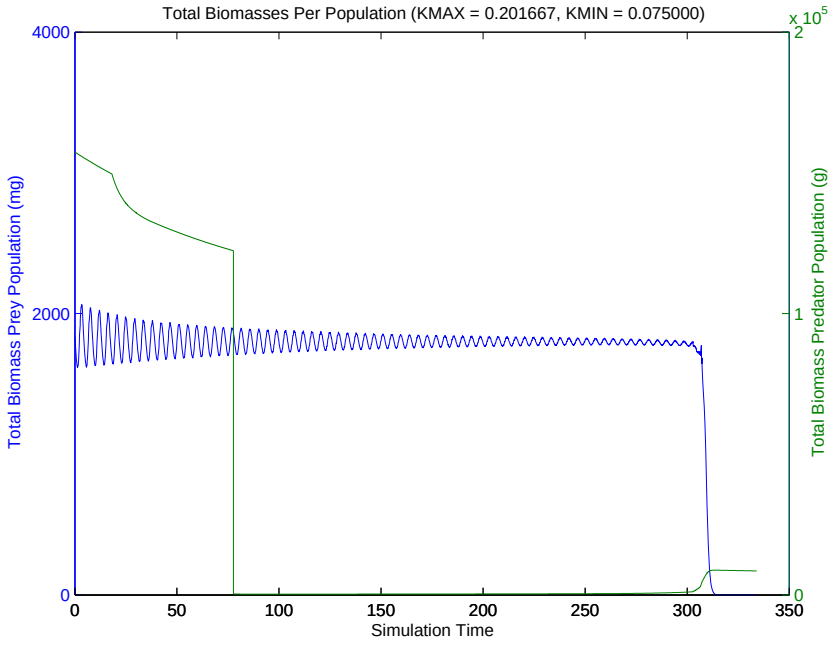


Figure 4.20: Alternate Example Total Biomasses for *Daphnia* and Fish Populations for Type 2a Extinction

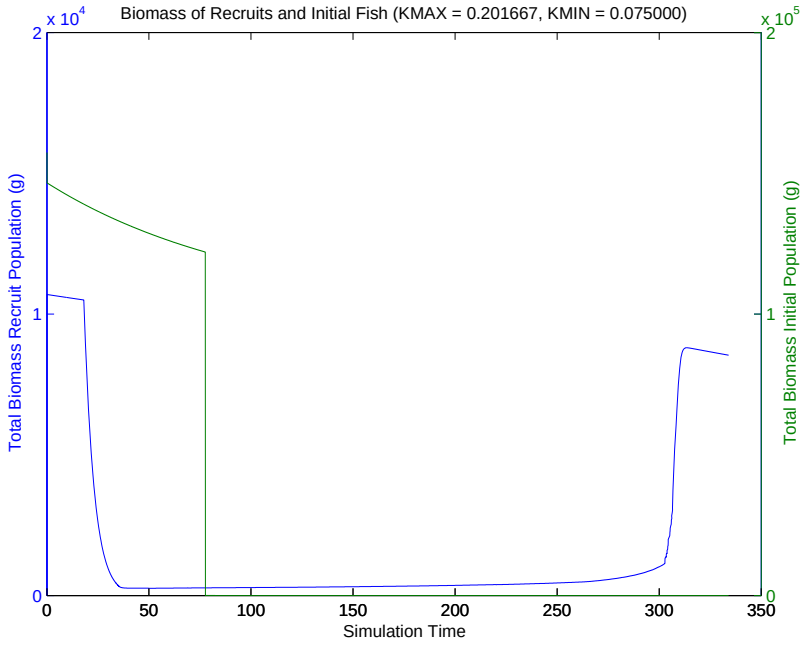


Figure 4.21: Alternate Example Total Biomasses for Recruits and Initial Fish Populations for Type 2a Extinction

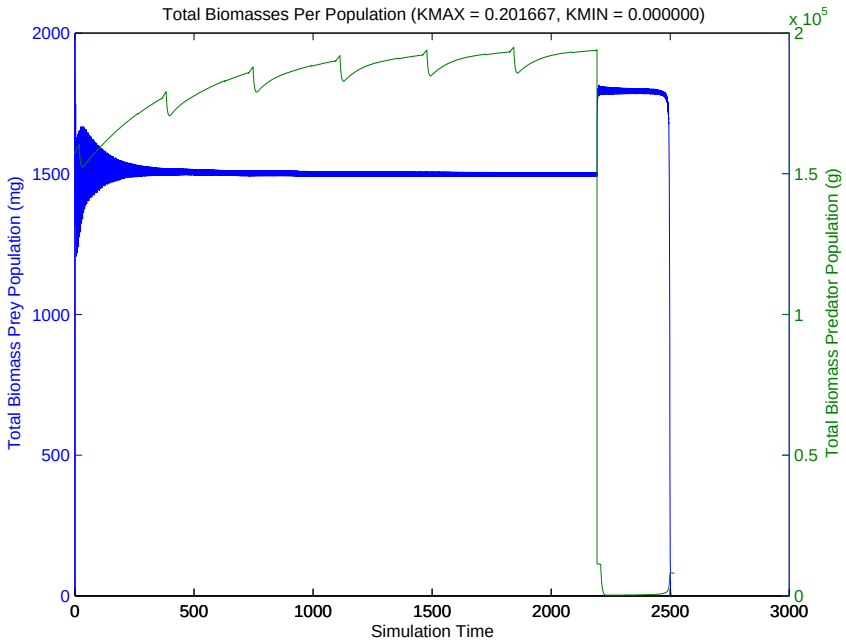


Figure 4.22: Total Biomasses for *Daphnia* and Fish Populations for Type 2b Extinction

Type 2b: Recruits Starve Until Initial Population Reaches Max Age, Then Drives Prey to Extinction

This pathway is a interesting variation on the one above. I later stopped making a distinction between this pathway and "Prey Driven to Extinction." This type only occurred on the $kmin = 0$ axis. Figures 4.22 and 4.23 show that the recruit population for the first seven generations was outcompeted by the larger fish to the point of elimination from the population. Each recruit class would starve off due to lack of sufficient resource because they could not compete against the initial population. But the last generation which was hatched just before the last of the initial population was removed, grew so rapidly that they drove the prey to extinction. This demonstrates both the effects of competition and rapid growth. This simulation does almost reach 2600 days. Note the jump in biomass when the *Daphnia* population is released from the predation pressure exerted by the initial fish population.

Type 2c: Initial and Recruit Populations Survive, Drive Prey to Extinction

This is also a variation on the "Prey Driven to Extinction" pathway. Figures 4.24 and 4.25, shows that both Initial and Recruit populations remain at the end of the simulation, but together had

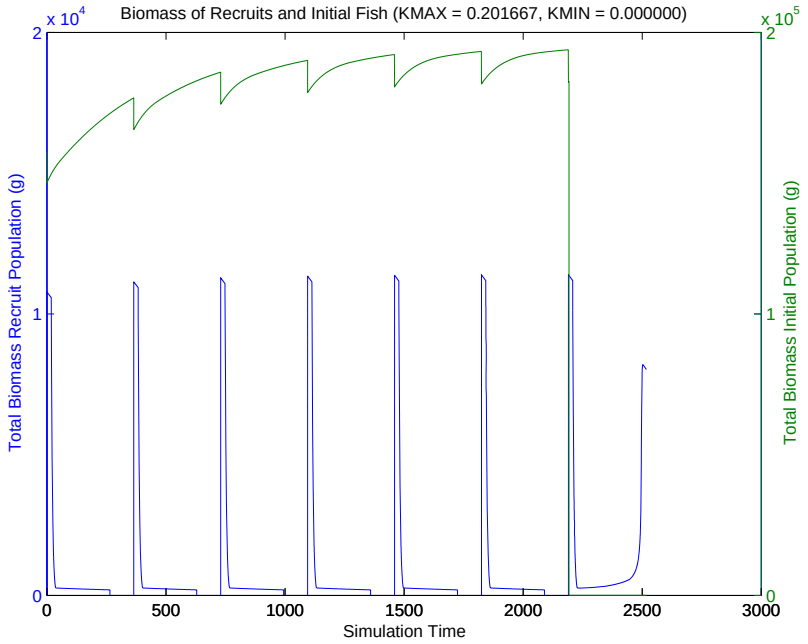


Figure 4.23: Total Biomasses for Recruits and Initial Fish Populations for Type 2b Extinction

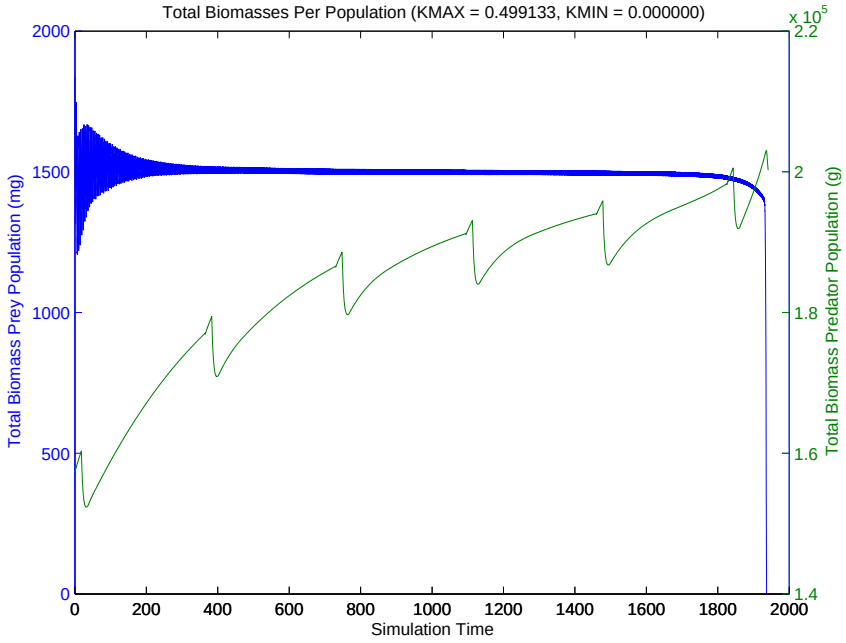


Figure 4.24: Total Biomasses for *Daphnia* and Fish Populations for Type 2c Extinction

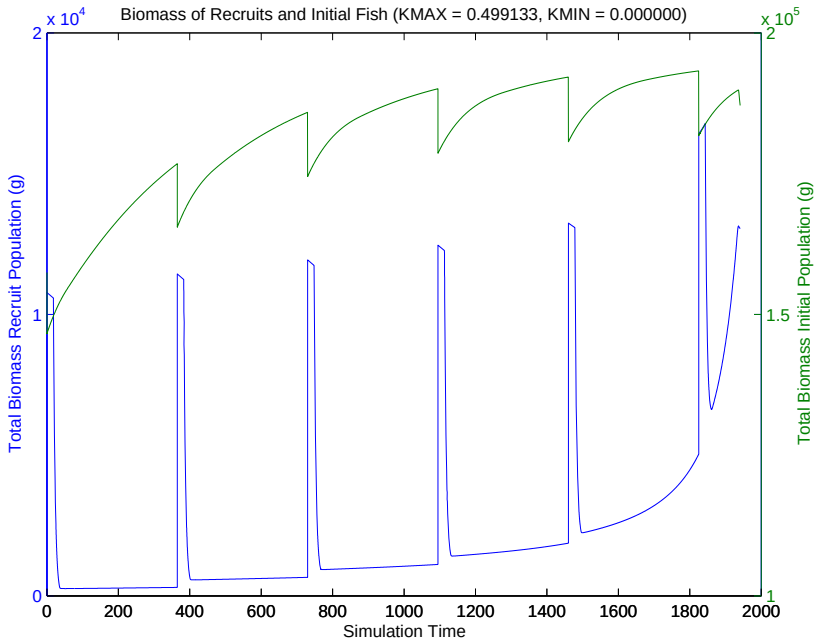


Figure 4.25: Total Biomasses for Recruits and Initial Fish Populations for Type 2c Extinction

driven the prey population to extinction. The simulation ended before the initial population had been removed by old age. This type only occurred on the $kmin = 0$ axis at the far right. It is exhibiting a Quiescence Threshold-type extinction, because the recruit class is growing and surviving between generations, eventually accumulating enough numbers to drive the prey population to extinction.

Type 3: Initial and Recruit Populations Grow Off of Support

The most common pathway of extinction is caused by both segments growing off of their support and starving. This was sometimes difficult to distinguish from starvation caused by a closed gape. Looking at the .pred output file in which I record all predation values and activity, I was able to distinguish by confirming that the lower value of the prey window had exceeded the maximum length of the prey population. This turned out to be a fast filter for my later analysis. I could look at the .pred file and eliminate all of the parameter values which indicated growth off of the prey population.

The total biomass values for this extinction pathway are shown in Figures 4.26 for the prey and predator population. Figure 4.27 shows the recruits and initial fish population biomasses. The recruits begin feeding at 51 days after they have consumed their yolk sac. The feeding causes a drop in the biomass of the *Daphnia* population, but with the rapid increase in size, the recruits grow off of their support and the *Daphnia* population recovers.

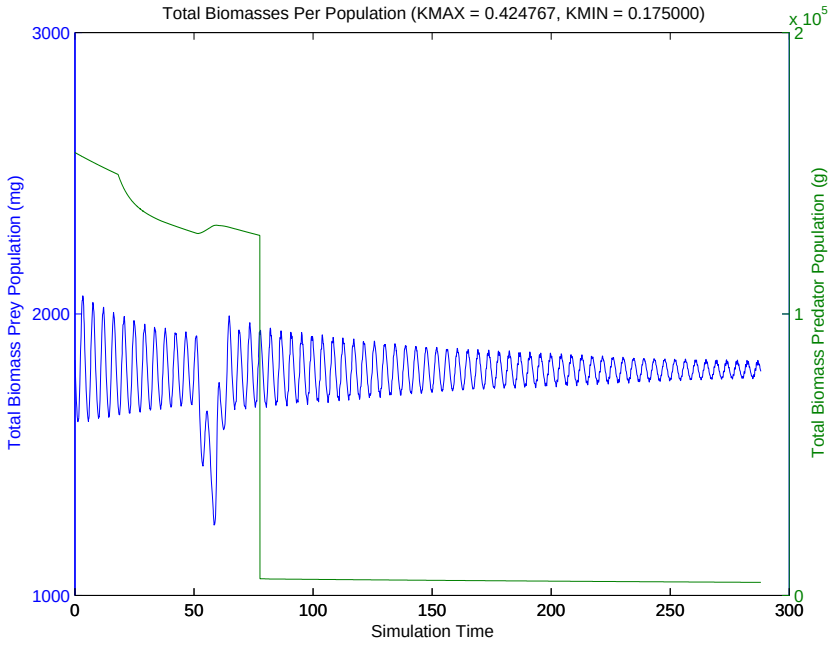


Figure 4.26: Total Biomasses for *Daphnia* and Fish Populations for Type 3 Extinction

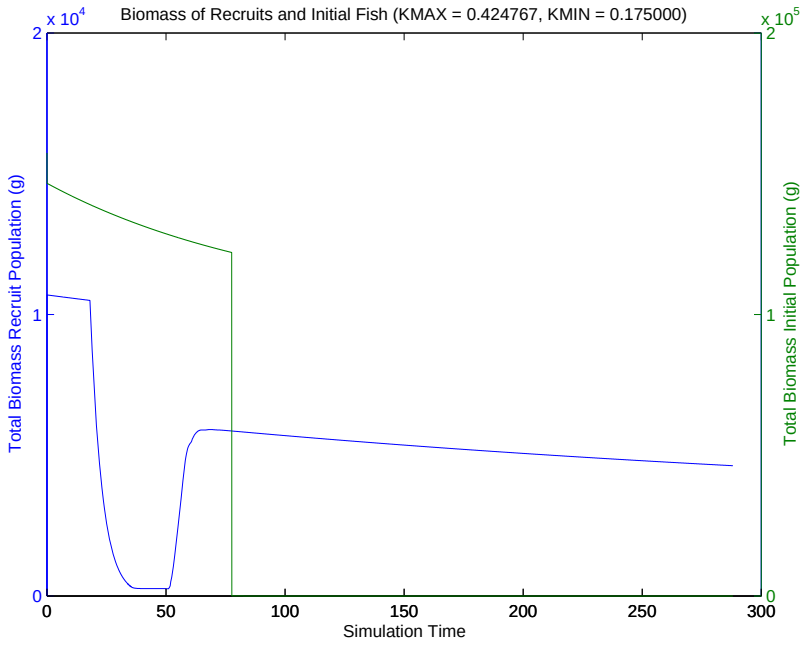


Figure 4.27: Total Biomasses for Recruits and Initial Fish Populations for Type 3 Extinction

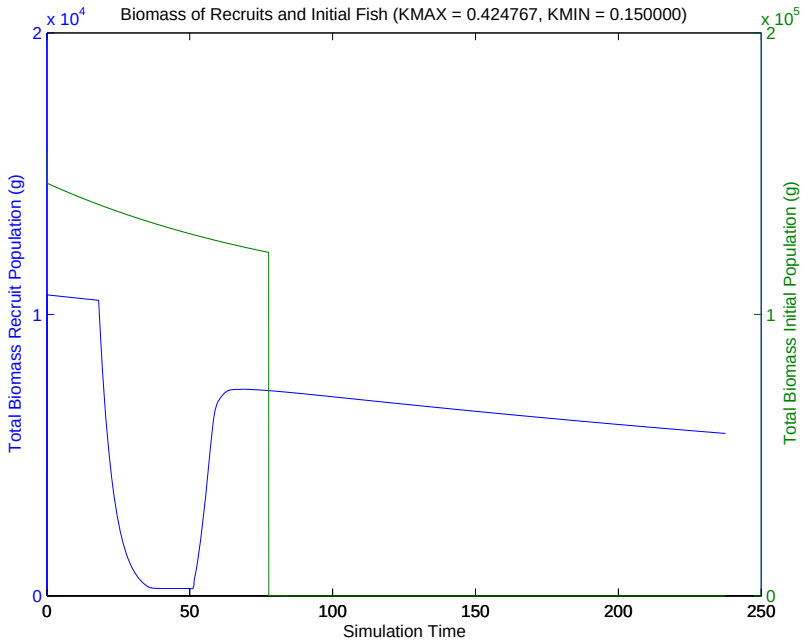


Figure 4.28: Total Biomasses for *Daphnia* and Fish Populations for Boundary Case

An Interesting Boundary Case

In this final set of Figures 4.28 and 4.29, $kmax = 0.424767$, and $kmin = 0.15$ which is on the boundary between extinction of the fish by growth off of support, and extinction by driving the daphnids to zero. The lone fish cohort almost drives the prey population to zero, but its resulting growth restricts it to only the top portion of the prey population ($> 4.7mm$). The *Daphnia* population is able to continue to reproduce, although with very small numbers; whereas the lone fish starves. This is also a longer-term, Quiescence Threshold-type of extinction.

4.3.4 Analysis of Extinction Map

The most surprising aspect of this Extinction Map are the Type 2 Extinctions. Repeatedly, the *Daphnia* was driven to extinction by a relatively number of recruits. This is partially explained by the explosive nature of the early growth of the recruits, which when they are moderated by competition is suppressed, but it still did not explain why a relatively huge population of fish exerting much more pressure on the daphnids (enough to suppress the normal biomass level by several hundred mg) were able to thrive and survive on the same prey population, but just a few kilograms of small fish could not survive on the same population. I initially thought it was their explosive rate plus

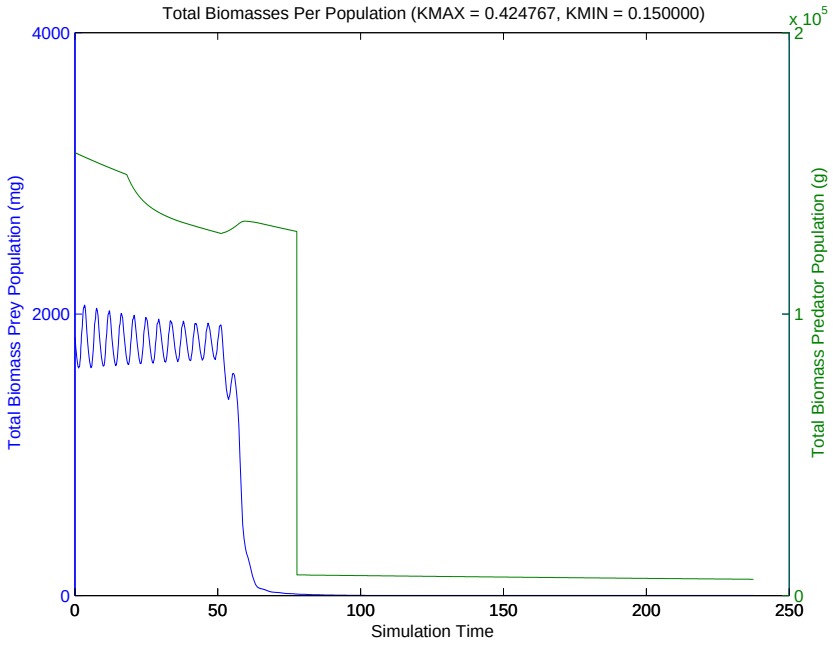


Figure 4.29: Total Biomasses for Recruits and Initial Fish Populations for Boundary Case

normal oscillations of biomass, which was knocking the daphnid population under the bottom edge of its density-dependent mortality bowl; thus leading to the rapid extinction of the daphnids. This is partially true and leads to the rapid elimination of the large daphnids, but it does not explain completely how such a small population was able to effect such a large change in biomass.

It turns out to be another Quiescence Threshold-type of effect where the juvenile fish are out-competing the larger fish for food and eliminating any from growing into the size classes above. Figure 4.30 is the lengths plotted over the simulation time for the first example shown as Type 2a. It clearly demonstrates the depletion of the lower-size classes (brood classes are protected). The depletion of the juvenile classes quickly prevents recruitment into the larger size classes.

I did think that perhaps lessening the YOY mortality imposed would allow the biomass of the fish born in the course of the simulation to rise sufficiently to suppress the explosive feeding of newborn cohorts that follow it. As it is, each new population of YOY fish born in the course of the simulation are collectively limited to about 8000 which represents a total biomass of about 250 grams. This is decrease from a population of hundreds of thousands and thousands of grams as the graphs have shown. I ran such a set of tests allow four times as many to remain. There were no regions of compatibility that appeared.

The conclusion seemed obvious. I had apparently chosen the value for *fscale* too close to the extinction threshold. Choosing a new value well away from the extinction threshold would certainly insulate the *Daphnia* population from the shocks of newborn fish. This would at least eliminate this pathway to extinction and lead to regions of compatibility where the fish growth would be balanced moderated by older fish and would not grow off of its support until at least after first reproduction. This lead to my second set of experiments.

But before we get into the second set of experiments, there are a couple of other synchronicities with the analysis results that I want to point out. In Claessen et al. (2002) they focus heavily on time series analysis of the dynamics. Their dynamics map separating their parameter into regions inspired the extinction/persistence maps of this study. They also mention that oscillation frequencies in the populations portend a change in dynamic. Our two previous papers published on this predator-prey model also describe the oscillations (Hallam et al., 1992a; Jaworska et al., 1995). Figures 4.31 and 4.32 demonstrate these oscillations. I did not especially study the time series data, so I do not have any other comment on the fluctuations.

The other comment is that one can explore different regions and different parameter values through the inequality relations in order to try to induce overlap or buffer the foraging regions of the initial and recruit populations. I worked out many different scenarios and ratios trying to detect

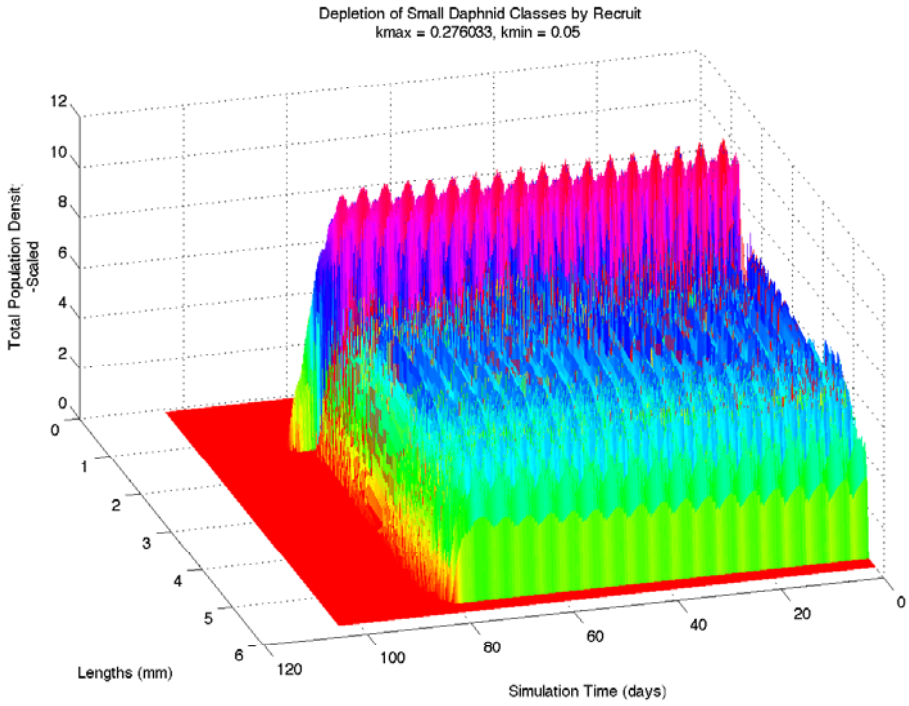


Figure 4.30: Depletion of the Lower Size Classes by Small Fish

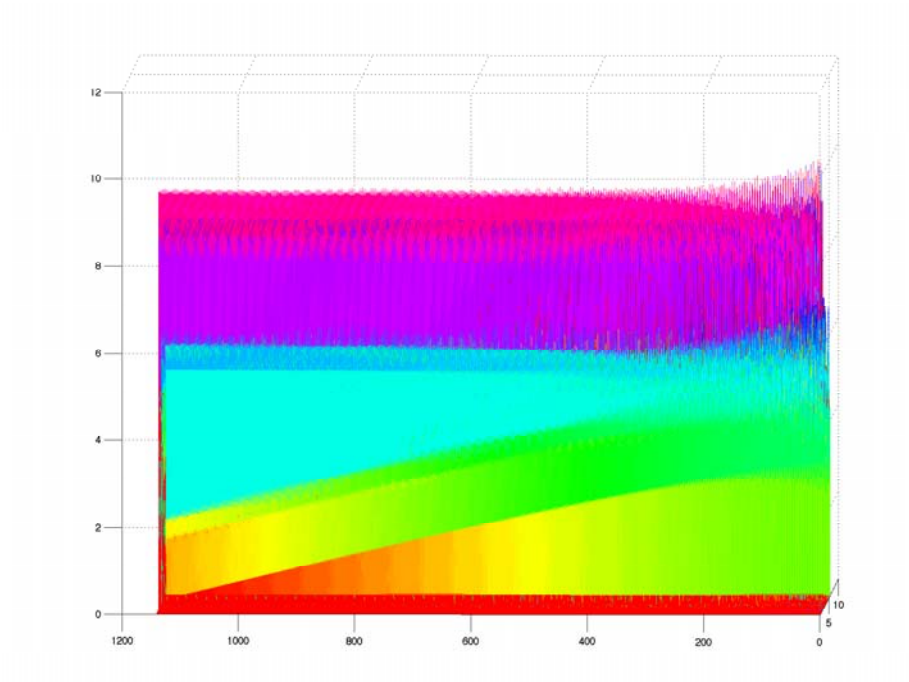


Figure 4.31: Short-Term Fluctuations in Size Classes

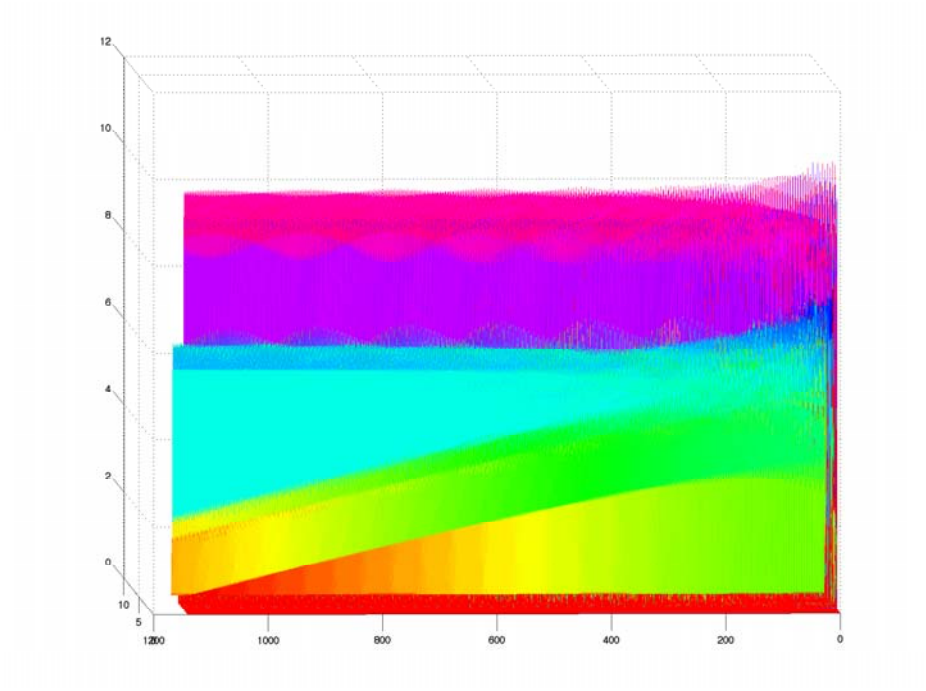


Figure 4.32: Long-Term Fluctuations in Size Classes

critical regions and boundaries. There is some analogous work in Claessen et al. (2002). But, with the general parameter space search, these values are explored, so I do not report any results from these scenarios, because they are usurped by the general maps.

4.3.5 Second Set of Experiments

For the second set of experiments, I took the other value I had for *fscale* in order to be on the safe side and to be certain to generate regions of compatibility presuming there were such regions. So *fscale* was set to 0.2×10^{-4} and the runs were repeated. During the course of the runs I was able to see that several parameter sets did lead to persistence as expected. The biomass level for *Daphnia* would often hardly budge from a value of about 1800 mg. This was also expected since we were scaling down mortality by a factor of 200,000. With such insulation, I theorized that the Extinction/Persistence Maps would be boring. In particular, there should be no regions where the insulated daphnid population is driven to zero. Other than the new value for *fscale*, all other values and populations were fixed between the first and second set of experiments.

The first Extinction/Persistence Map I generated is given in Figure 4.33. Most of the parameter space is covered by growth off of support as expected. There were several parameter sets along the bottom axis that survived to the required 2600 days, and an off-axis combination at 0.2 also survived. Starvation along the left and top edge was not surprising. What was surprising was a single green dot indicating that the prey population had once again been driven to extinction. When I was analyzing the data set, when I came to this run I was puzzled as to what happened; the fish seemed fine. I was surprised to see the extinction of the prey — I had not been looking at the prey biomass. Upon later review I found a whole row of such extinctions for $kmin = 0.025$. It seemed like a replay each time: the recruit biomass would reach 1.85M g and the daphnid population would go to zero. What was going on? Was the model breaking? The same prey population was supporting much larger biomass levels when $kmin = 0$. How could the *Daphnia* population possibly be driven to zero?

An additional set of runs was made that was made in step sizes of 0.001 vertically and was bounded above by 0.02. I was looking right along the axis to try to find values off of the axis that were in the compatibility region. It did not make sense that the interior of the compatibility region should be empty, since the smallest sizes of *Daphnia* are around 1.0 mm. It should be an open set of some sort. Analyzing these new runs yielded Figure 4.34. It indicated that there was indeed an extended off-axis area of persistence for $kmin < 0.01$ with a bulbous region forming near the left.

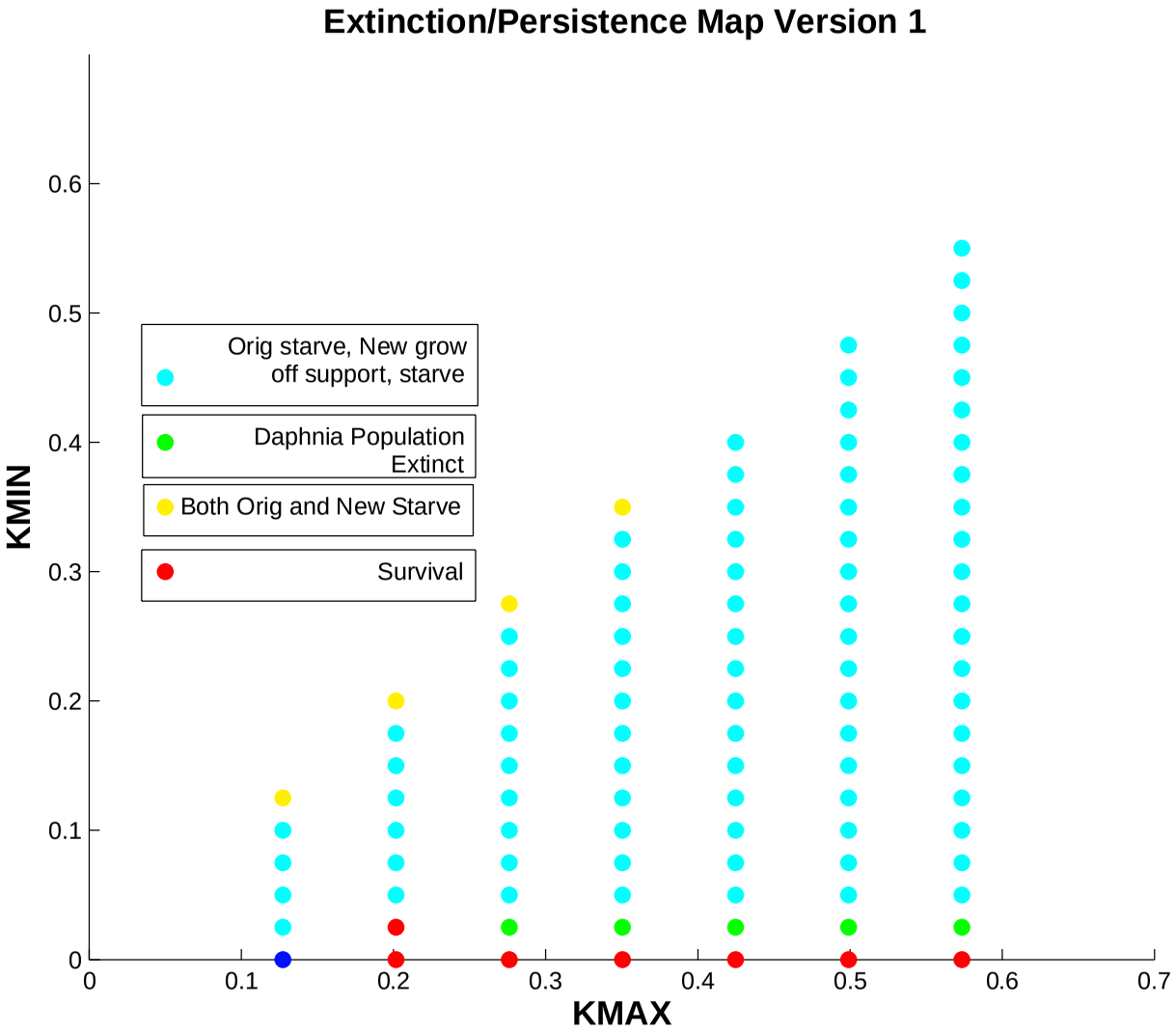


Figure 4.33: First Version of Extinction/Persistence Map for Second Set of Runs

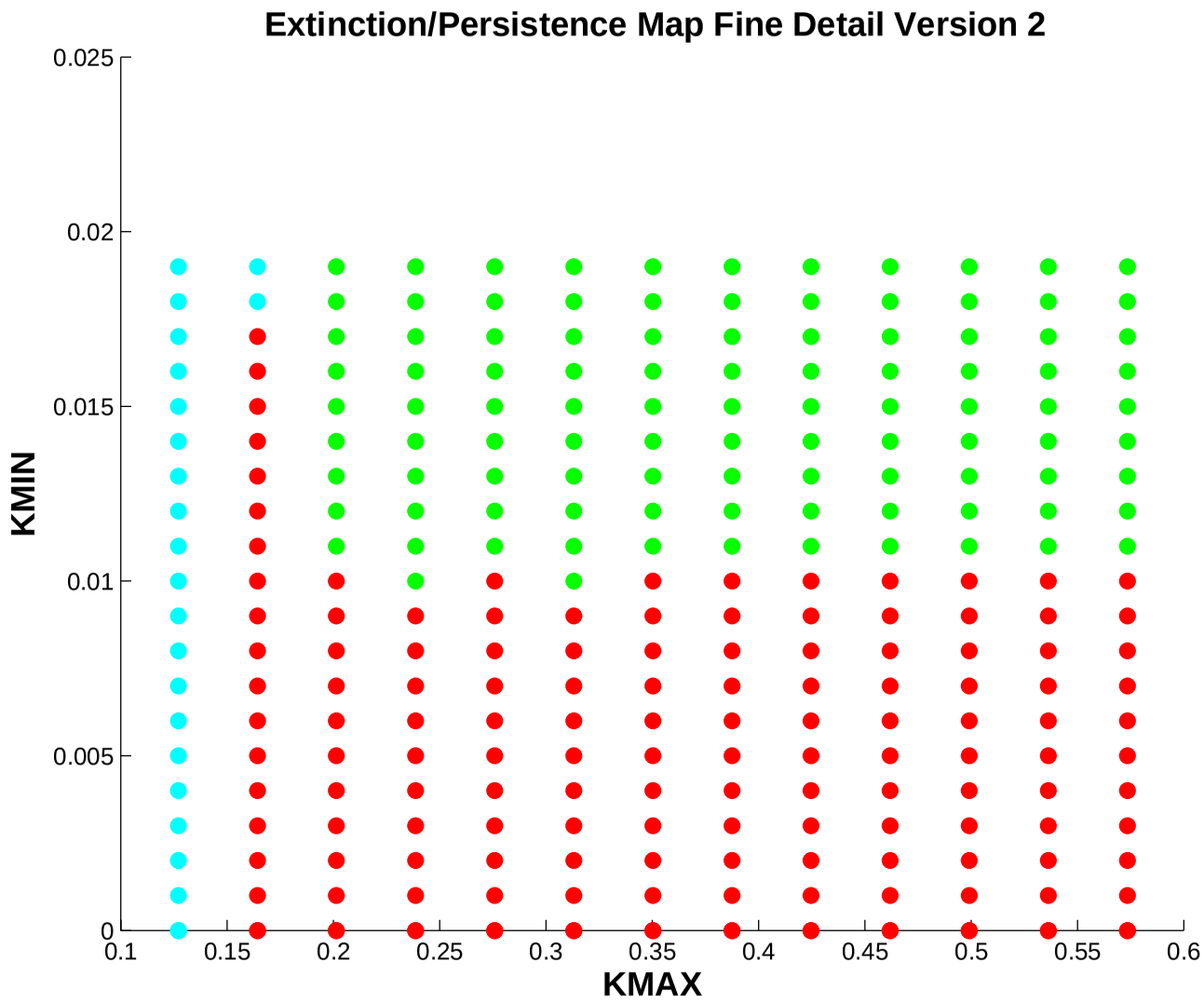


Figure 4.34: Second Version of Extinction/Persistence Map for Second Set of Runs

But it also revealed an extended region where the *Daphnia* were driven to zero. This region really surprised me both for its existence, location, and size. Did it make sense and why was it persistently there?

In the first set of experiments we did see the effect that juveniles could cause extinction by eliminating the smallest size classes. With the longer runs and the fish population allowed to grow larger, another mechanism emerges that duplicates the same effect. The fish population size can grow to such a point that its outtake suppresses the biomass of the *Daphnia* population below optimal biomass. This can occur by just a random oscillation starting the feedback loop towards increasing mortality with decreasing population size. Figure 4.35 gives such an example where the predating cohorts all had prey windows above 3 mm, so the juvenile classes were not being directly predated upon. Note the flaring of the prey biomasses near the end. I did not try to distinguish between this type and extinction caused by the juvenile classes being eliminated from the *Daphnia* population; they both had the same outcome.

To further analyze these emerging regions and to try to determine boundaries between the regions I realized that I needed more runs, but my ability to accurately analyze the outputs and assign extinction types was flagging. I had already filled half of a new 500 Gb hard drive with output files. So I wrote a python script to perform a binary search for the boundaries of the persistence, prey-extinct, and predator-extinct regions. The script was consistent where I was not and could extract terminal execution times and biomasses which I had been doing initially but quit doing because of the tedium. The first output of this binary search combined with all of the data I had collected to this point is summarized in the Existence/Persistence Map in Figure 4.36.

A few more surprising features emerge from this map. The continued existence of the Type 2 Extinction region notable, capping the persistence region and separating it from the region of Zero-Support. Further, many of the runs in the Type 2 region ran to a substantial portion of the 2600 day limit. Time before extinction decreased as the parameters neared the Zero-Support region. The band of survival across the top of the Type 2 Extinction region at the boundary between the predator-to-zero and prey-to-zero region is surprising to detect. Such an interface is rare to find, although mathematically it exists. Finally, the size of the bulbous region at the left is ten times higher than the rest of the survival band.

I focused on the left-hand side to try to understand what was happening there. See Figure 4.37. Surprisingly, more areas of persistence showed up detached from the lower axis. These were near the upper boundary. Had I been overlooking regions of compatibility, presuming there to be nothing interesting above the Zero Support region? I had finally discovered the strong restriction on *kmin*

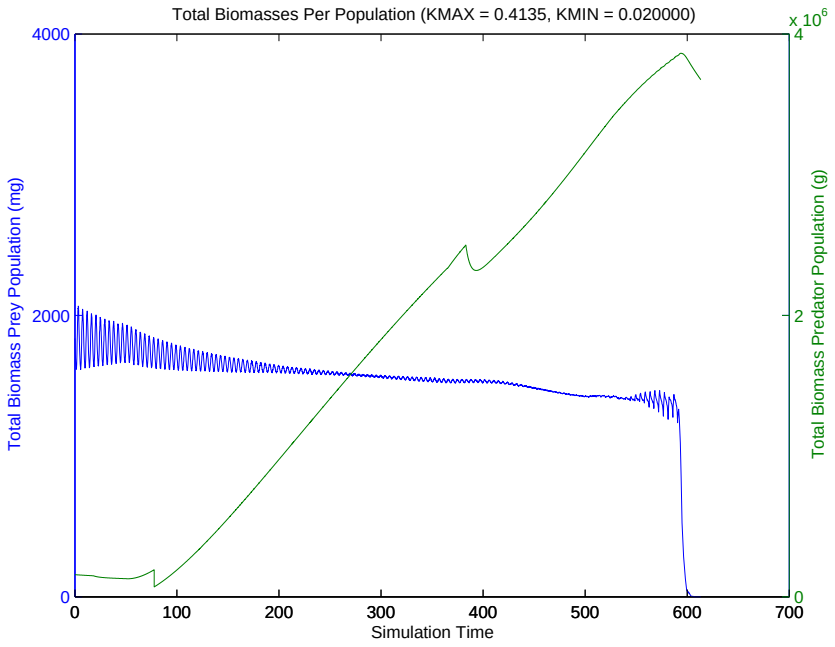


Figure 4.35: Total Biomasses for *Daphnia* and Fish Populations Demonstrating Type 2 Extinction Caused by Predation Pressure

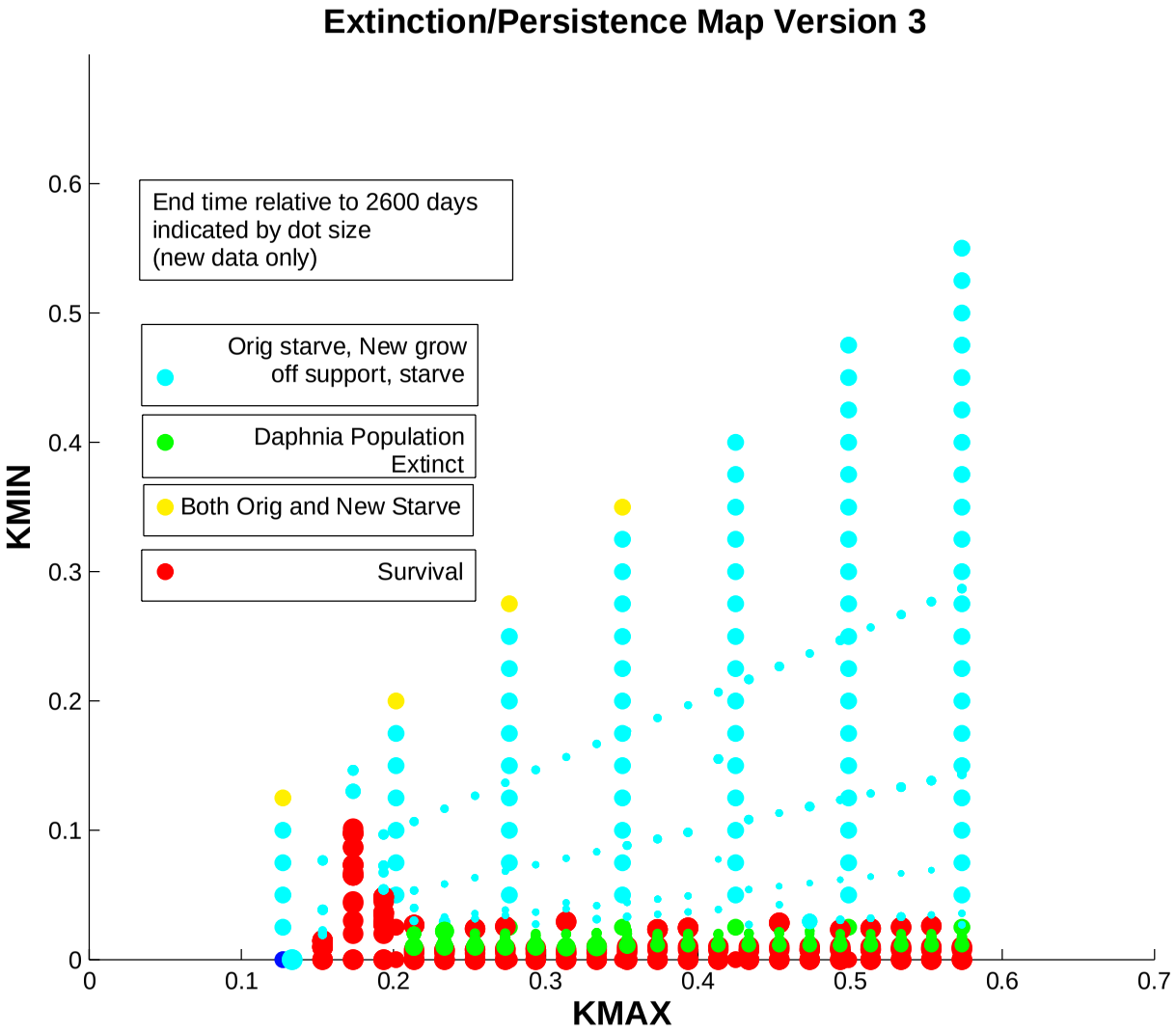


Figure 4.36: Third Version of Extinction/Persistence Map for Second Set of Runs

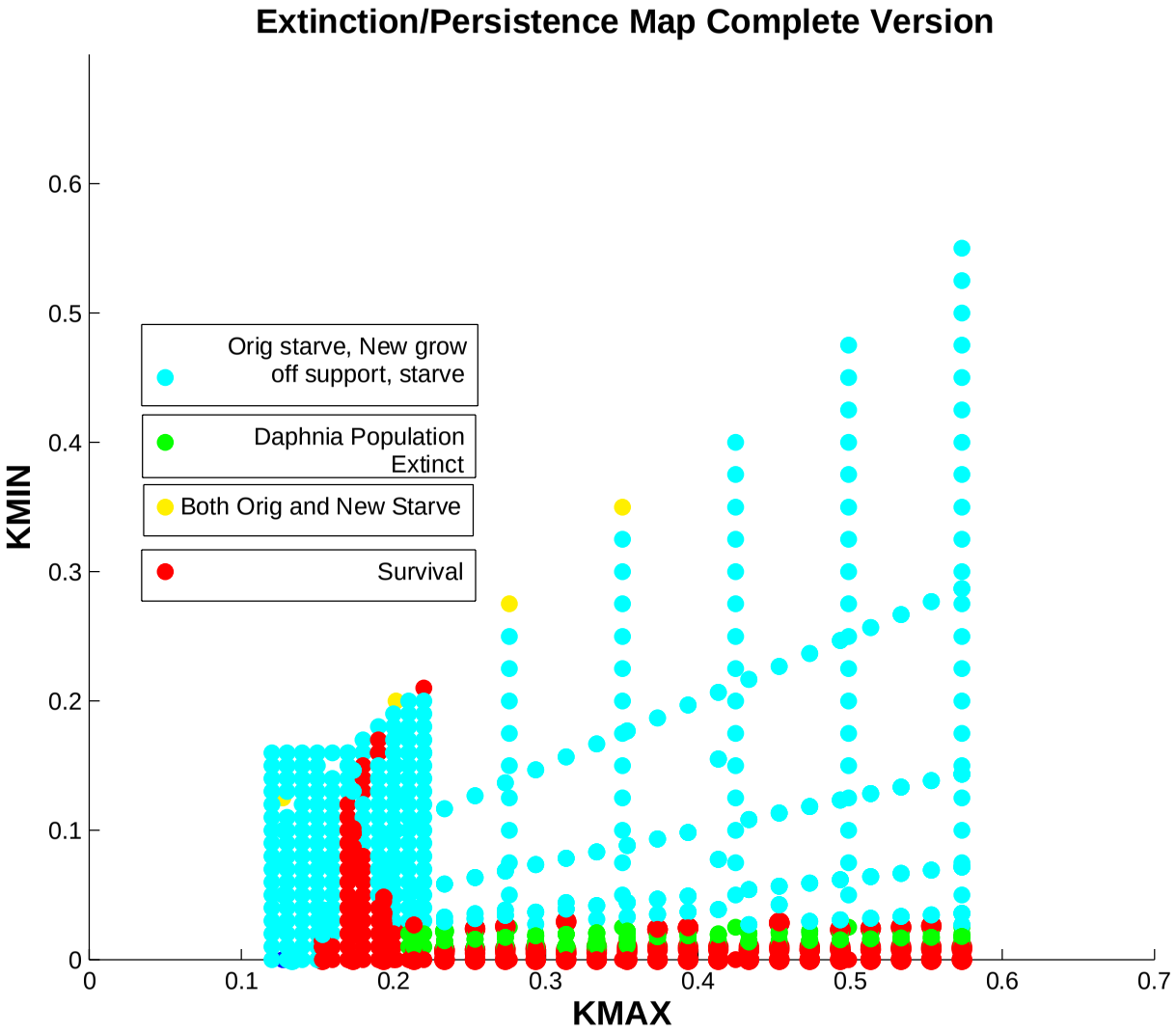


Figure 4.37: Complete Version of Extinction/Persistence Map for Second Set of Runs

resulting from the large sizes of the initial population and I had observed the required competition to suppress the rapid growth of the juvenile class. These together explained the compatibility region along the bottom axis, but what could I make of the off-axis regions? The mystery is cleared up when the plots are made with relative to the terminal total biomass for the fish populations as shown in Figure 4.38. What is happening is that the recruits were subsisting, but not increasing in biomass above their original levels. By my definition, the populations survive to 2600 days and should therefore be included in the persistence region, but they fail to meet the qualification of truly representing compatible populations which would persist indefinitely. The recruits cannot generate enough stores to reproduce, therefore they will not persist. This observation indicates the role that the choice of 2600 days plays. It plays a similar role to our choice of time period in the definition of Quiescence and Extinction Thresholds.

This last observation led to wondering if any of the regions truly show persistence. Was it only a matter of time before the fish population built up to an unsupportable level? I had repeatedly observed the biomass level of the *Daphnia* being drawn down lower and lower with each successive generation of fish. Could it be drawn under the optimal biomass, thus leading to a Type 2 Extinction? Is some other mortality required to induce compatibility? To this end I ran a final set along the bottom axis for 10,000 days. I had thought that the compatibility regions might disappear, but had indication that the middle region ran just a little bit closer to pushing the prey under optimal biomass. All of the populations persisted except for the ones with low *kmax*. Figure 4.39 reports the terminal biomass for both populations. As observed, the middle values of *kmax* did have slightly higher terminal biomasses for fish which pushed the prey population slightly closer to the optimal biomass of 1400 mg. This did surprise me that none of them built up quite enough to push the *Daphnia* to zero. Figure 4.40 demonstrates why none of them built up too high. When the fish population had built up almost enough biomass to push the *Daphnia* under optimal, then the oldest characteristic would be removed because of maximum age. The *Daphnia* population would then recover.

4.4 Analysis and Conclusions

At first appearance *Extinction Thresholds* seem to indicate that persistence of the community model is determined solely from the value of the predation scaling factor $fscale = V_D/V_F$. If the predation scaling factor is chosen below the threshold, then the community model survives. In fact, the

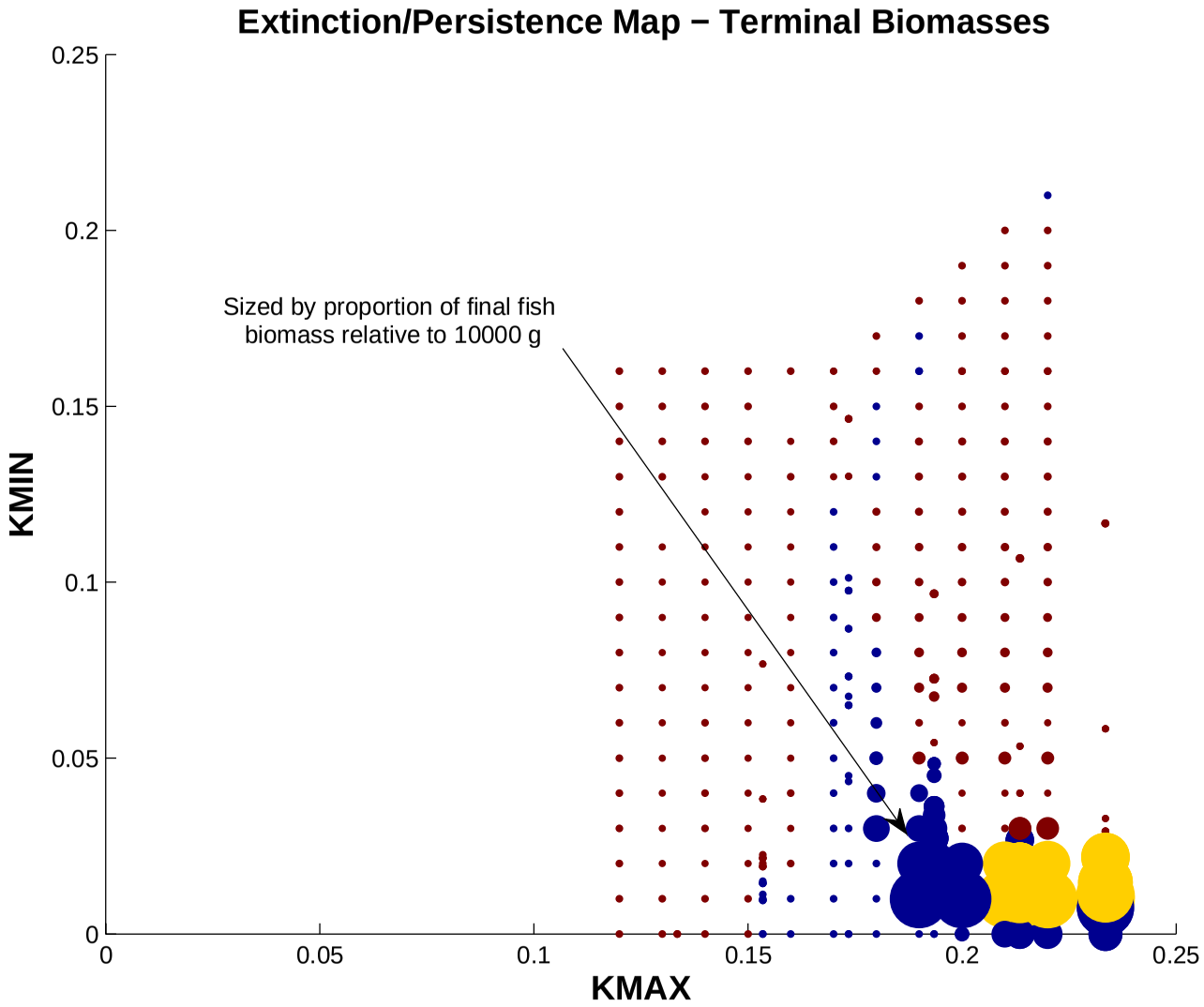


Figure 4.38: Extinction/Persistence Map for Second Set of Runs Plotted Relative to Terminal Fish Biomasses

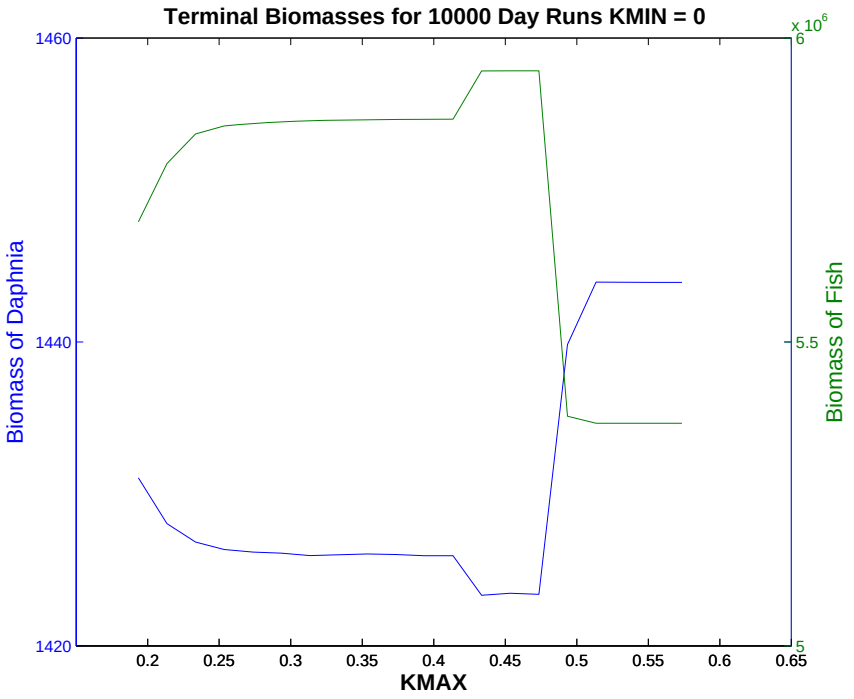


Figure 4.39: Terminal Biomasses for *Daphnia* and Fish Populations for 10000 day Simulation

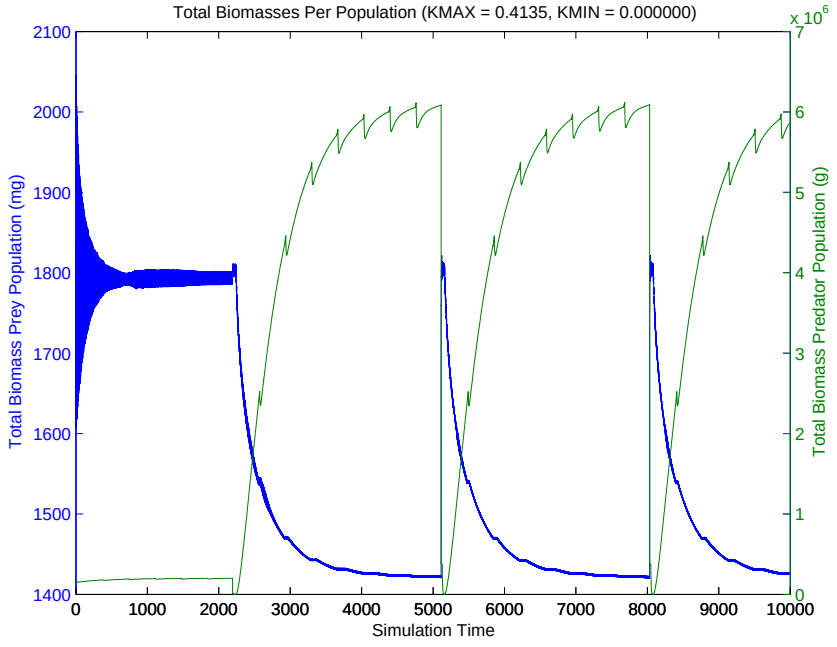


Figure 4.40: Total Biomasses for *Daphnia* and Fish Populations for 10000 day Simulation

Daphnia population either goes to zero or infinity, depending on if the predation is sufficient to control the prey. The absence of density-dependent mortality gives this rudimentary structure and sharp threshold. The overall method of determining the extinction threshold can still be applied in the presence of more complicated mortalities.

Similarly, the *Quiescence Threshold* is defined in the absence of density-dependent mortality so that predation mortality is isolated. Further, isolating a predator cohort and then defining the *Quiescence Threshold* in terms of the minimal population required to deplete the resource in its prey window seems to be without direct application. But, we see in Figures 4.24 and 4.25 that addition to small classes of the fish population over successive generations can built up to a point at which the smaller classes of the prey population are depleted and the prey population collapses. Again, the overall idea behind the quiescence threshold is applicable in the presence of more complicated mortalities.

For the complete community model with density-dependent mortality imposed on the prey population it is demonstrated that decreases of *fscale* are insufficient to ensure persistence of the community model for all valid values of the gape-size parameters *kmin* and *kmax*. The Persistence/Extinction Maps demonstrate that in the *kmin*-*kmax* parameter space three types of regions persist under changes to the *fscale* parameter. The predator-extinction region was the largest. For parameter values in this region, then the predators grew off of their supporting resource. In spite of increased insulation of the prey population from predation mortality afforded by decreases in *fscale*, a region of prey-extinction persists. We demonstrated two causal mechanisms for this behavior in that both the rapid initial growth of the juvenile fish and their predation on the juvenile prey classes, assisted by density-dependent mortality and natural oscillations in biomass can lead to this conclusion. Finally, a region of persistence emerges along the $kmin = 0$ axis. The persistence region is constrained to this region because of the requirements for competition from larger fish to control the juvenile classes. For the fish from the initial population to survive and compete with the juvenile classes, then the size-structure of the initial fish population and the overall size-structure of the prey population determined from the individual model delineate the maximum value of *kmin* that can produce persistence.

Overall, we found that the stability of the community model is threatened when the predator population is dominated by juveniles. Conversely, stability is maintained by the presence of larger fish. This indicates that determination of the health and stability of fish populations requires inclusion of size-distribution in addition to assessment of population or biomass.

The model is very sensitive to the choice of *kmin*, while *kmax* is less significant. This was first indicated in Section 4.1 where we had difficulty determining the extinction threshold for *kmin*. Because *kmin* is fundamental in prescribing the prey window overlap and the level of competition between the size classes, the reasons for the sensitivity of *kmin* is clear. Although for a population model-only and through different mathematical analyses, this conclusion is in line with the results of Persson and de Roos who found that the lower window parameter determined the dynamical behavior and outcome while the upper window parameter only had effect on the life history and population structuring. Another similarity was the role of the juvenile classes on restricting food from advancing up to the larger fish.

4.5 Future Research

With these results established, now one could proceed to study long-term dynamics and bifurcation diagrams as was done in the papers by de Roos and Persson for a fish population. Their primary analysis tool was time series analysis with the ability to discern predator-prey dynamics from cohort-driven dynamics. As a future research project, we could do a similar analysis. Questions of interest include the following. Do our regions of persistence further segment into dynamical regions as they found? Do the shapes of the critical rate functions in the functional response for the fish also determine these dynamics?

Another potential project would be examine how the diagrams change with perturbation in starvation sensitivity. The size structure of the prey population is not evenly distributed. If the fish were particularly sensitive to feast-or-famine cycles, then different outcomes could be expected. The starvation levels and preventing starvation during birth cycles are model features that need to be reconsidered.

Finally, as we saw in this chapter, our maximum age condition is unrealistically abrupt. We do have a formulation for an age-based mortality, but it was never applied. Smoother dynamics would result if the older cohorts were slowly reduced over time. Similarly, starvation as modelled by Persson and de Roos is smoother, reducing the cohort numbers over time rather than an abrupt removal.

Bibliography

Bibliography

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1985). *Data Structures and Algorithms*. Series in Computer Science and Information Processing. Addison-Wesley.

Almasi, G. and Gottlieb, A. (1989). *Highly Parallel Computing*. Benjamin-Cummings, New York.

Bartell, S., Gardner, R. H., and O'Neill, R. V. (1992). *Ecological Risk Assessment*. Lewis, Chelsea.

Billari, F. G., Fent, T., Prskawetz, A., and Scheffran, J., editors (2006). *Agent-Based Computational Modelling: Applications in Demography, Social, Economic and Environmental Sciences*. Physica-Verlag: Heidelberg.

Breck, J. and Gitter, M. (1983). Effect of fish size on the reactive distance of bluegill (*lepomis macrochirus*) sunfish. *Canadian Journal of Fisheries and Aquatic Sciences*, 40:162–167.

Butenhof, D. (1997). *Programming with Posix Threads*. Addison-Wesley Professional.

Bystrom, P., Persson, L., and Wahlstrom, E. (1998). Competing predators and prey: Juvenile bottlenecks in whole-lake experiments. *Ecology*, 79(6):2153–2167.

Claessen, D., de Roos, A. M., and Persson, L. (2000). Dwarfs and giants: Cannibalism and competition in size-structured populations. *American Naturalist*, 155(2):219–237.

Claessen, D., Van Oss, C., de Roos, A. M., and Persson, L. (2002). The impact of size-dependent predation on population dynamics and individual life history. *Ecology*, 83(6):1660–1675.

De Roos, A. M. and Persson, L. (2001). Physiologically structured models - from versatile technique to ecological theory. *Oikos*, 94(1):51–71.

De Roos, A. M., Persson, L., and McCauley, E. (2003). The influence of size-dependent life-history traits on the structure and dynamics of populations and communities. *Ecology Letters*, 6(5):473–487.

DeAngelis, D. L. and Gross, L. J., editors (1992). *Individual-based Models and Approaches in Ecology: Populations, Communities, and Ecosystems*. Chapman and Hall, New York, NY.

DeAngelis, D. L., Rose, K. A., and Huston, M. A. (1995). Individual-oriented approaches to modeling ecological populations and communities. In Levin, S. A., editor, *Frontiers in Mathematical Biology*, volume 100 of *Frontiers in Mathematical Biology*, pages 390–410. Springer Verlag.

DeRoos, A. (1988). Numerical methods for structured population models: The escalator boxcar train. *Numerical Methods for Partial Differential Equations*, 4:173–195.

Dudycha, J. L. and Tessier, A. J. (1999). Natural genetic variation of life span, reproduction, and juvenile growth in daphnia. *Evolution*, 53(6):1744–1756.

Elliott, J. M. (1976). Body composition of brown trout (salmo trutta l.) in relation to temperature and ration size. *The Journal of Animal Ecology*, 45(1):273–289.

Enserink, E. L. (1995). *Food mediated life history strategies in Daphnia magna : their relevance to ecotoxicological evaluations*. PhD thesis, Free University of Amsterdam.

Federico, P. (2007). *Bat Population Dynamics: An Individual-Based Model Approach*. PhD thesis, University of Tennessee.

Funasaki, E. T. (1997). *Examination of Dynamical Behavior and Estimation of Toxicant Levels in Chemically Stressed Population Models*. PhD thesis, University of Tennessee.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing*. The MIT Press.

Gerritsen, J. (1984). Size efficiency reconsidered - a general foraging model for free-swimming aquatic animals. *American Naturalist*, 123(4):450–467.

Gerritsen, J. and Strickler, J. R. (1977). Encounter probabilities and community structure in zooplankton - mathematical-model. *Journal Of The Fisheries Research Board Of Canada*, 34(1):73–82.

Gill, A. B. (2003). The dynamics of prey choice in fish: the importance of prey size and satiation. *Journal Of Fish Biology*, 63:105–116.

Grimm, V. and Railsback, S. (2005). *Individual-Based Modeling and Ecology*. Princeton Series in Theoretical and Computational Biology. Princeton University Press.

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, W., Saphir, W., and Snir, M. (1998). *MPI—The Complete Reference, Volume 2, The MPI-2 Extensions*. The MIT Press.

Grove, D., Loizides, L., and Nott, J. (1978). Satiation amount, frequency of feeding, and gastric emptying rate in *salmo gairdneri*. *Journal of Fish Biology*, 12:507–516.

Haefner, J. W. (1992). Parallel computers and individual-based models: An overview. In DeAngelis, D. L. and Gross, L. J., editors, *Individual-based Models and Approaches in Ecology*, pages 126–164. Chapman and Hall, New York, NY.

Hallam, T. G., Canziani, G. A., and Lassiter, R. R. (1993). Sublethal narcosis and population persistence - a modeling study on growth effects. *Environmental Toxicology And Chemistry*, 12(5):947–954.

Hallam, T. G. and Lassiter, R. R. (1994). Individual-based mathematical modeling approaches in ecotoxicology: A promising direction for aquatic population and community ecological risk assessment. In Kendall, R. J. and Lacher, Jr., T. E., editors, *Wildlife Toxicology and Population Modeling: Integrated Studies of Agroecosystems, Proceedings of Ninth Pellston Workshop, July 22-27, 1990*, pages 531–542, Lewis Publishers, Boca Raton, USA.

Hallam, T. G., Lassiter, R. R., and Henson, S. M. (2000). Modeling fish population dynamics. *Nonlinear Analysis-Theory Methods & Applications*, 40(1-8):227–250.

Hallam, T. G., Lassiter, R. R., Jaworska, J., and McKinney, W. (1992a). Physiologically based models in predator-prey ecology: An introduction. In Agarwal, R., editor, *Recent Trends in Differential Equations*, volume 1 of *World Scientific Series in Applicable Analysis*, pages 285–299. World Scientific Publishing Company.

Hallam, T. G., Lassiter, R. R., Li, J., and McKinney, W. (1990a). Toxicant-induced mortality in models of daphnia populations. *Environmental Toxicology And Chemistry*, 9(5):597–621.

Hallam, T. G., Lassiter, R. R., Li, J., and McKinney, W. (1992b). An approach for modelling populations with continuous structured models. In DeAngelis, D. L. and Gross, L. J., editors, *Individual-based Models and Approaches in Ecology*, pages 312–337. Chapman and Hall, New York, NY.

Hallam, T. G., Lassiter, R. R., Li, J., and Suarez, L. A. (1990b). Modeling individuals employing an integrated energy response - application to daphnia. *Ecology*, 71(3):938–954.

Hart, P. J. B. and Gill, A. B. (1993). Choosing prey size - a comparison of static and dynamic foraging models for predicting prey choice by fish. *Marine Behaviour And Physiology*, 23(1-4):91–104.

Henson, S. M. (1994). *Individual-based physiologically structured population and community models*. PhD thesis, University of Tennessee.

Henson, S. M. and Hallam, T. G. (1994). Survival of the fittest: Asymptotic competitive exclusion in structured population and community models. *Nonlinear World*, 1:385–402.

Henson, S. M. and Hallam, T. G. (1995). Optimal feeding via constrained processes. *Journal of Theoretical Biology*, 176(1):33–37.

Holling, C. S. (1959). The components of predation as revealed by a study of small mammal predation of the european pine saw fly. *The Canadian Entomologist*, 91:293–320.

Holling, C. S. (2001). Understanding the complexity of economic, ecological, and social systems. *Ecosystems*, 4(5):390–405.

Hwang, K. (1993). *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, New York.

Jaworska, J., Hallam, T. G., Henson, S. M., and McKinney, W. (1995). Ecotoxicology of predator-prey communities: An individual-based modeling perspective. In Hughes, J., editor, *Environmental Toxicology and Risk Assessment - Third Volume*, volume ASTM STP 1218. American Society for Testing and Materials, Philadelphia.

Koh, H. L., Hallam, T. G., and Lee, H. L. (1997). Combined effects of environmental and chemical stressors on a model daphnia population. *Ecological Modelling*, 103(1):19–32.

Kolli, H. (2007). Study of interaction between mexican free tailed bats (tadarida brasiliensis) and moths and counting moths in a real time video. Master's thesis, University of Tennessee.

Kooijman, S. (1986). Population dynamics on basis of budgets. In Metz, J. and Diekmann, O., editors, *The Dynamics of Physiologically Structured Populations*, volume 68 of *Lecture Notes in Biomathematics*. Springer Verlag.

Kooijman, S. (2000). *Dynamic Energy and Mass Budgets in Biological Systems*. Cambridge University Press, Great Britain, 2nd edition.

Kot, M. (2001). *Elements of Mathematical Ecology*. Cambridge University Press, Great Britain.

Krohn, C. M. (2001). *An Individual-based approach to population dynamics with applications to sockeye salmon and iteroparous organisms*. PhD thesis, Univeristy of Tennessee.

Lassiter, R. R. (1986). A theoretical basis for modelling element cycling. In Hallam, T. G. and Levin, S. A., editors, *In Mathematical Ecology: An Introduction*, volume 17 of *Biomathematics*, pages 341–380. Springer Verlag.

Lassiter, R. R. and Hallam, T. G. (1990). Survival of the fattest - implications for acute effects of lipophilic chemicals on aquatic populations. *Environmental Toxicology And Chemistry*, 9(5):585–595.

Li, J. and Hallam, T. G. (1988). Survival in continuous structured populations models. *Journal Of Mathematical Biology*, 26(4):421–433.

Lika, K. (1996). *Interactions of predator-prey ecological processes and advective movement in a spatially heterogeneous environment*. PhD thesis, University of Tennessee.

Lovelock, C. M. (1996). The effects of temperature and dissolved oxygen on a model fish population. Master's thesis, University of Tennessee.

Luna, F. and Perrone, A., editors (2002). *Agent-based Methods in Economics and Finance: Simuations in Swarm*. Kluwer Academic Publishers, Boston.

MAPLE (1996). *MAPLE*. Waterloo Maple, Inc., 57 Erb Street West, Waterloo, ON, Canada, http://www.maplesoft.com.

Marowka, A. (September 2007). Parallel computing on any desktop. In *Communications of the ACM*, volume 50(9).

Merritt, R. (July 23, 2007). M'soft: Parallel programming model 10 years off. *EETimes*. http://www.eetimes.com/showArticle.jhtml?articleID=201200019. Retrieved August 7, 2007.

Metcalf, M. and Reid, J. (1993). *Fortran 90 Explained*. Oxford University Press.

Metz, J. and Diekmann, O., editors (1986). *The Dynamics of Physiologically Structured Populations*, volume 68 of *Lecture Notes in Biomathematics*. Springer Verlag.

MPICH2 (2007). *MPICH2*. Argonne National Laboratory, Mathematics and Computer Science Division. http://www-unix.mcs.anl.gov/mpi/mpich2.

Parnot, C. (2007). XGrid@Stanford. http://http://cmgm.stanford.edu/ cparnot/xgrid-stanford. Retrieved August 2007.

Persson, L., Amundsen, P. A., De Roos, A. M., Klemetsen, A., Knudsen, R., and Primicerio, R. (2007). Culling prey promotes predator recovery - alternative states in a whole-lake experiment. *Science*, 316(5832):1743–1746.

Persson, L. and De Roos, A. M. (2006). Size-structured interactions and the dynamics of aquatic systems. *Polish Journal Of Ecology*, 54(4):621–632.

Persson, L., Leonardsson, K., de Roos, A. M., Gyllenberg, M., and Christensen, B. (1998). Ontogenetic scaling of foraging rates and the dynamics of a size-structured consumer-resource model. *Theoretical Population Biology*, 54(3):270–293.

Peters, R. H. and De Bernardi, R. (1987). *Daphnia*, volume 45 of *Memorie dell'Istituto Italiano di Idrobiologia Dott. Marco de Marchi*. Verbania Pallanza.

Quinn, M., Metoyer, R., and Hunter-Zaworski, K. (2003). Parallel implementation of the social forces model. In *Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics (August 2003)*, pages 63–74.

Raghavan, A. (2005). Modeling study of individual and group behavior of brazilian free-tailed bats (tadarida brasiliensis) and dynamic bat counting using real-time infrared thermal video. Master's thesis, University of Tennessee.

Ramachandramurthi, S., Hallam, T. G., and Nichols, J. A. (1997). Parallel simulation of individual-based, physiologically structured population models. *Mathematical and Computer Modelling*, 25(12):55–70.

Reinders, J. (2007). *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly.

Reynolds, C. W. (2006). Big fast crowds on PS3. In *ACM Sandbox Symposium 2006, Boston, Massachusetts, July 29-30, 2006*.

Rost, R. J. (January 25, 2006). *OpenGL(R) Shading Language*. Addison-Wesley Professional, 2nd edition.

Sinko, J. W. and Streifer, W. (1967). A new model for age-size structure of a population. *Ecology*, 48(6):910–918.

Sinko, J. W. and Streifer, W. (1969). Applying models incorporating age-size structure of a population to daphnia. *Ecology*, 50(4):608–615.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. (1998). *MPI—The Complete Reference, Volume 1, The MPI Core*. The MIT Press.

Staples, D. J. and Nomura, M. (1976). Influence of body size and food ration on energy budget of rainbow-trout salmo-gairdneri richardson. *Journal Of Fish Biology*, 9(1):29–43.

Stevens, W. R. (1993). *Advanced Programming in the UNIX Environment*. Addison-Wesley. 2nd Edition, 2005. S. Rago is co-author.

Suter, I. G. W. (1993). *Ecological Risk Assessment*. Lewis, Boca Raton.

Sutter, H. (March 2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3).

Sylvester, S. (1995). PVM parallelization of a mathematical ecology *daphnia* models. Master's thesis, University of Tennessee.

Tessier, A. J., Henry, L. L., Goulden, C. E., and Durand, M. W. (1983). Starvation in daphnia - energy reserves and reproductive allocation. *Limnology And Oceanography*, 28(4):667–676.

van Kooten, T., Persson, L., and de Roos, A. M. (2007). Size-dependent mortality induces life-history changes mediated through population dynamical feedbacks. *American Naturalist*, 170(2):258–270.

Wilson, K. (2007). Performance and Architecture Group, Apple, Inc., WWDC2007. Personal Communication, Apple World Wide Developer's Conference, 2007.

Appendix

Appendix A

Numerical Simulation of the Individual-based *Daphnia* and Fish Population Models

Contained in this Appendix are the derivations of the equations used in the numerical simulation of both the fish and *Daphnia* population models. The Individual-based aquatic ecosystem models developed at the University of Tennessee share a common numerical simulation scheme which has previously not been well-documented, so it has either been ignored or passed down by folklore. This paper is a collection of all of the information necessary to understand the numerics in the Daphnia, fish, and predator-prey models. The general numerical formulae are first presented, followed by sections on each separate model. Each significant equation involved in the numerical simulation was confirmed using the Waterloo Maple symbolic mathematics system (MAPLE, 1996) in order to provide assurance that the programs on which we base our results are executing the correct mathematics.

A.1 General Numerical Formulae

The base for each of these models was originally coded by Bill McKinney. He originally included the option to use two different Runga-Kutta-type numerical schemes for their simulation. A third scheme has since been added for some of these models. The common names of the first two are Backward Euler and Crank-Nicolson which are both implicit schemes. The other scheme is an one explicit one

suggested by McKinney and implemented by Dina Lika for the fish model. It is termed Corrected Euler, because it is basically a double application of Euler's Method. Backward Euler is a common implicit scheme used for ODEs. Crank-Nicolson is most often applied to the simulation of PDEs, so it is in books on the numerical simulation of PDEs where one will most often find it described. Corrected Euler is an explicit scheme based on Euler's Method. It has several advantages over the two implicit schemes: 1) partial derivatives are not needed; 2) it is much easier to implement; and 3) it is a two-step simulation which allows function updates in between the steps if desired. Each of these schemes is described herein.

A.1.1 McKendrick-von Foerster Population Equations

The population and predator-prey models all share the same underlying mathematical description. Individuals are incorporated into a population by a system of extended McKendrick-von Foerster partial differential equations. A derivation from basic principles of the McKendrick-von Foerster equation for age and size structured models, as well as its relationship with other standard models of populations, can be found in Sinko and Streifer (1967, 1969). For a fixed individual model, its McKendrick-von Foerster equation is:

$$\rho_t + \rho_a + (g_L\rho)_{m_L} + (g_S\rho)_{m_S} = -\mu(t, a, m_L, m_S, \rho)\rho, \tag{A.1}$$

where $\rho(t, a, m_L, m_S)$ is a density function which gives the number (or density) per unit age per unit mass of lipid per unit mass of structure of individuals which are age a, and have masses m_L and m_S at time t. The functions g_L and g_S are functions expressing the growth of the variables m_L and m_S with respect to time; these are given by the individual model equations. The mortality function μ expresses the per capita rate of mortality for individuals of age a, and masses m_L and m_S; several expressions for this function are included in each code; it can depend on the population density. In general, we assume that $\mu = \mu_A + \mu_S + \mu_D$ where μ_A, μ_S, and μ_D are the age, size, and density dependent mortality rates. Equation (A.1) requires initial and boundary conditions in order to be well-posed. The initial population distribution is specified by

$$\rho(0, a, m_L, m_S) = \phi(a, m_L, m_S). \tag{A.2}$$

The boundary condition is an expression for the newborn individuals:

$$\rho(t,0,m_{L0},m_{S0}) = \int_0^\infty \int_0^\infty \int_0^\infty \beta(t,a,m_{L0},m_{S0},m_L,m_S,\rho)\rho(t,a,m_L,m_S)dadm_Ldm_S, \tag{A.3}$$

where m_{L0} and m_{S0} are the initial sizes of the newborn, and β is the birth rate at time t by individuals of masses m_L and m_S for newborns of masses m_{L0} and m_{S0}. Note that β can depend also depend on ρ. Together equations (A.1), (A.2), and (A.3) form a well-posed model of a population of individuals whose growth is described by equations of the individual model.

To solve this PDE, the method of characteristics is employed to reduce this PDE to a system of ODE's. The characteristic equations for equation (A.1) are

$$\frac{dt}{ds} = 1 \tag{A.4}$$
$$\frac{da}{ds} = 1 \tag{A.5}$$
$$\frac{dm_L}{ds} = g_L(m_L,m_S) \tag{A.6}$$
$$\frac{dm_S}{ds} = g_S(m_L,m_S) \tag{A.7}$$
$$\frac{d\rho}{ds} = -(\mu + (g_L)_{m_L} + (g_S)_{m_S})\rho. \tag{A.8}$$

A small transformation of ρ is performed in order to get a density function $n(t,a,m_L,m_S)$ such that only mortality is acting on the characteristics. The equation which replaces equation (A.8) is

$$\frac{dn}{dt} = -\mu(t,a,m_L,m_S,n)n. \tag{A.9}$$

See Hallam et al. (1992b) for an explanation of this transformation. The basic steps are to let $\hat{\rho} \equiv \rho(t,a,m_L(t,a),m_S(t,a))$ and $n(t,a) \equiv \hat{\rho}(t,a)h(t,a)$ where $h(t,a)$ solves the PDE $h_t + h_a = [(g_L)_{m_L} + (g_S)_{m_S}]h$. Applying this transformation to (A.8) will yield (A.9). Note in particular that the same expression for μ is obtained. Throughout the models, wherever ρ is referred to, it is really n.

Equations (A.4) through (A.7) and (A.9) are simulated numerically by the models. Equations (A.4) and (A.5) are trivial to solve. Equations (A.6) and (A.7) are coupled through m_L and m_S and are solved simultaneously using one of the Runge-Kutta methods. These updated values of m_L and m_S are then used to evaluate μ, and the final equation is solved.

In order to model diversity in the population, various values of the parameters in the individual model's equations are used to model the different growth characteristics of ecotypes. Each set of these parameters determines a different McKendrick-von Foerster equation. These PDE's are simulated simultaneously by the population model and are coupled through density dependent mortality.

The numerical schemes used to simulate these equations are now described.

A.1.2 Backward Euler

Each of the following numerical schemes are applied to the general formulation of an ODE initial-value problem:

$$\begin{aligned} y' &= f(t,y) \\ y(a) &= b \end{aligned} \tag{A.10}$$

where y can be a vector or scalar.

For this problem, at simulation time t_n, the Backward Euler method is described by

$$\begin{aligned} \frac{y^{n+1}-y^n}{\tau} &= f(t_{n+1},y^{n+1}) \\ \text{so } y^{n+1} &= y^n + \tau f(t_{n+1},y^{n+1}) \end{aligned} \tag{A.11}$$

where y^n is the approximation to $y(t_n)$, y^{n+1} is the approximation to $y(t_{n+1})$, $t_{n+1} = t_n + \tau$ and τ is the time step.

Applying this to equations (A.6) and (A.7) one gets the equations

$$\begin{pmatrix} m_L^{n+1} \\ m_S^{n+1} \end{pmatrix} = \begin{pmatrix} m_L^n \\ m_S^n \end{pmatrix} + \tau \begin{pmatrix} g_L(t_{n+1}, m_L^{n+1}, m_S^{n+1}) \\ g_S(t_{n+1}, m_L^{n+1}, m_S^{n+1}) \end{pmatrix}$$

Letting $x = (x_1, x_2) = (m_L^{n+1}, m_S^{n+1})$ these transform to

$$\begin{pmatrix} x_1 - m_L^n - \tau g_L(t_{n+1}, x_1^k, x_2^k)) \\ x_2 - m_S^n - \tau g_S(t_{n+1}, x_1^k, x_2^k)) \end{pmatrix} = 0$$

Applying Newton's Method (iterated in k)

$$\begin{pmatrix} x_1^{k+1} \\ x_2^{k+1} \end{pmatrix} = \begin{pmatrix} x_1^{k} \\ x_2^{k} \end{pmatrix} + (\nabla h)^{-1} h$$

to solve these implicit equations, one ends up with the system

$$\begin{pmatrix} x_1^{k+1} \\ x_2^{k+1} \end{pmatrix} = \begin{pmatrix} x_1^{k} \\ x_2^{k} \end{pmatrix} + \tag{A.12}$$
$$\begin{pmatrix} 1-\tau\frac{\partial g_L}{\partial x_1}(t_{n+1},x_1^k,x_2^k) & -\tau\frac{\partial g_L}{\partial x_2} \\ -\tau\frac{\partial g_S}{\partial x_1} & 1-\tau\frac{\partial g_S}{\partial x_2} \end{pmatrix}^{-1} \begin{pmatrix} x_1 - m_L^n - \tau g_L(t_{n+1},x_1^k,x_2^k) \\ x_2 - m_S^n - \tau g_S(t_{n+1},x_1^k,x_2^k) \end{pmatrix}$$

After iterating through Newton's enough times, one has values the new approximations m_L^{n+1} and m_S^{n+1}. Note that the seed values (x_1^0, x_2^0) used to start Newton's Method are the linear approximations for m_L^{n+1} and m_S^{n+1} based on the previous two values. Finally, these new values are used to calculate the mortality function μ at time t_{n+1}, so that the equation for ρ can be solved.

$$\rho^{n+1} = \frac{\rho^n}{1+\tau\mu}$$

A.1.3 Crank-Nicolson

Crank-Nicolson can be considered an average of Euler's and Backward Euler's methods. Its mathematical statement is

$$\begin{aligned} \frac{y^{n+1}-y^n}{\tau} &= f(t_{n+1/2}, y^{n+1/2}) \\ \text{so } y^{n+1} &= y^n + \tau f(t_{n+1/2}, y^{n+1/2}) \\ \text{where} & \\ y^{n+1/2} &= \frac{y^{n+1}+y^n}{2}. \end{aligned}$$

Applying this to equations (A.6) and (A.7) as before one gets the system

$$\begin{pmatrix} x_1^{k+1} \\ x_2^{k+1} \end{pmatrix} = \begin{pmatrix} x_1^k \\ x_2^k \end{pmatrix} + \tag{A.13}$$
$$\begin{pmatrix} 1 - \tau \frac{\partial g_L}{\partial x_1}(t_{n+1/2}, \frac{x_1^k + m_L^n}{2}, \frac{x_2^k + m_S^n}{2})\frac{1}{2} & -\tau \frac{\partial g_L}{\partial x_2}\frac{1}{2} \\ -\tau \frac{\partial g_S}{\partial x_1}\frac{1}{2} & 1 - \tau \frac{\partial g_S}{\partial x_2}\frac{1}{2} \end{pmatrix}^{-1} \begin{pmatrix} x_1 - m_L^n - \tau g_L \\ x_2 - m_S^n - \tau g_S \end{pmatrix}$$

The interesting twist with Crank-Nicolson is that it requires values at the "midpoint." Thus in order to solve the equation for ρ using Crank-Nicolson, the values for m_L and m_S at $t_{n+1/2}$ must be determined. The steps followed in the codes are

1. Compute m_L and m_S at $t_{n+1/2}$ using C-N.
2. Store values of m_L and m_S for ρ-equation.
3. Linearly extrapolate m_L and m_S forward from $t_{n+1/2}$ to t_1.
4. Calculate μ at $t_{n+1/2}$.
5. Calculate ρ using C-N.
$$\rho^{n+1} = \frac{\rho^n(1 - (1/2)\tau\mu)}{(1 + (1/2)\tau\mu)}$$

Comparing the resulting approximations generated by these two schemes, one will find out that Backward Euler will generally undershoot the value determined by Crank-Nicolson (increasing function).

A.1.4 Corrected Euler's Method

Corrected Euler's method is a direct method obtained by applying Euler's Method to get estimates for the future values, which are then used to calculate an average which is used to calculate the values. The mathematical statement is:

$$\begin{aligned} \tilde{y}^{n+1} &= y^n + \tau f(t_n, y^n) \\ y^{n+1} &= y^n + \frac{\tau}{2}(f(t_n, y^n + f(t_{n+1}, \tilde{y}^{n+1}). \end{aligned}$$

Notice that no derivatives are required. Also, with it being two-stage, one can perform an update in between. For instance, Dina Lika used this to perform an update of resource values for her fish movement models.

A.2 Daphnia Population Model

Since the Daphnia population model shares the same McKendrick-von Foerster equations as the other models, it suffices to describe the equations for the growth of the lipid and structure compartments. These expressions for g_L and g_S specify the right-hand side of equations (A.6) and (A.7). These are components specified in the individual model.

The original paper describing the Daphnia model is Hallam et al. (1990b). It has since undergone several changes and additions, including the addition of sublethal toxicant effects Hallam et al. (1993), and environmental effects Koh et al. (1997). The original code was written by McKinney when he was a graduate student. His program actually combined both the fish and Daphnia models in one code using a global variable to indicate if the model being run was for fish or Daphnia. The only function that had to be written special for the two models was the one were the numerical equations and derivatives are calculated. The "driver" portion of the code was shared by both models. It was written in Fortran 77. This code base was copied and duplicated to form the first predator-prey model. Later additions to the Daphnia model have splintered off from the original code. Graciella Canziani developed a whole separate line for the studies using WASP and sublethal effects. She did so by pulling out only what she needed from the original code, basically scrapping the fish code. This version is what the later additions were made to. Several important abilities were removed in the Canziani translation, so several different codes had to be run in order to create and tune a new population. To unite these codes and provide a GUI to the model was the purpose of Smart-Alec written by Scott Sylvester and Mike Peek. I have since recombined the Daphnia and fish models with all of the additions into one code written in C called fishdaph. There was a precursor to all of these models which was written by Jia Li, using the family-tree concept directly. It was written in Pascal for PCs.

A.2.1 Individual Model and Numerical Simulation

The basic expressions for g_L and g_S were originally developed in Hallam et al. (1990b). The reader is referred to it for details of the derivations. A summary of these expressions is presented here:

Equations for *Daphnia*

Juvenile Ages: $0 \leq a \leq P$

$$\begin{aligned} g_L &= -A_{14} m_L \\ g_S &= -A_{15} m_S \\ m_L(0) &= m_{L0} \text{ (determined from parent)} \\ m_S(0) &= m_{S0} \text{ (fixed amount)} \end{aligned}$$

Adult Ages: $P \leq a \leq$ maximum age. (Assuming $D \leq E$.)

$$\begin{aligned} g_L &= \frac{A_{0L} x_L m_S}{A_1 m_S^{1/3} + A_2 x} - A_3 (m_L - \epsilon m_{PS}) \frac{D}{E} \\ g_S &= \frac{A_{0S} x_S m_S}{A_1 m_S^{1/3} + A_2 x} - A_4 (m_S - m_{PS}) \frac{D}{E} \\ m_L(P) \text{ and } m_S(P) &= \text{terminal values of juvenile stage} \end{aligned}$$

where

$$\begin{aligned} D &= A_5 (m_L + m_S)^{1/3} + A_6 (m_L + m_S)^{2/3} + A_7 m_L + A_8 m_S \\ E &= 37.68 A_3 (m_L - \epsilon m_{PS}) + 16.75 A_4 (m_S - m_{PS}) \end{aligned}$$

An undocumented change in the expression for $A2$ was made by Shandelle Henson. The new expression for it is

$$A2(m_S) = \frac{m_S}{k M_g}$$

where k is determined allometrically by $k = k_1 (m_{PS})^{-k_3}$ and M_g is the mass capacity of the gut which is calculated by $c_g(m_{PS}) BDP$ where c_g gives the fraction of body volume devoted to the gut and BDP is the (average) body density of the prey (Grove et al. (1978)). This expression is derived in the same way as the gut clearance rate component is derived for fish (see below). The inclusion of m_S is merely a result of simplifying the uptake terms in g_L and g_S by multiplying top and bottom by m_S. The reason given for this change was "this modification keeps Daphnia from growing arbitrarily large with increasing resource levels." The inclusion of m_S in the expression for $A2$ both explicitly and implicitly through m_{PS} makes the numerical expressions much more complicated. Other additions such as sublethal effects and environmental effects are done as simple

scaling multipliers on relevant parameters, so their effect on the numerics is minimal. The specific necessary numerical equations are now derived.

In both Backward Euler and Crank-Nicolson the difficult expressions are those for the partial derivatives involving g_L and g_S. In order to numerically simplify their computation, they are broken down into a series of steps. These computations are now documented using Maple.

A.3 Maple Derivation of Daphnia Model

The main functions for the Daphnia model are calculated in the function dfterm_c(). The formulas are derived and correlated to the expressions in the *Ecology* paper (Hallam et al., 1990b). Table 1 in the *Ecology* paper describes the parameter values of the model. Table 2 lists the values and units for the parameters that were used for the simulations. Table 2 also lists the sources for the values.

The order in which the formulas are presented here is the order of the simulation program, not the order of the presentation in the paper.

A.3.1 Work Coefficients

The energy expended by Daphnia on locomotion and similar functions — termed generically as work — is minimal. The expressions used to describe the energy requirements for locomotion are derived from Gerritsen (1984, ref). As stated in the Ecology paper, the expression of work in units of power is:

$$locomotionPower := (\phi, \mu, rS, \nu, M, g) \mapsto \left(6\,\pi\,\phi\,\mu\,rS\,\nu^2 + \tfrac{3}{32}\,M\nu^3\right) g$$

Here ϕ is a non-dimensional coefficient of form resistance; μ is viscosity; rS is the radius of an equivalent spherical volume; ν is the velocity; M is the wetted surface area; and g is related to the muscular swimming efficiency of the individual.

Assuming average density of 1.0 mg/mm^3, then the equivalent radius, rS, is:

$$equivRadius := m \mapsto 1/4\,\sqrt[3]{3}4^{2/3}\,\sqrt[3]{\tfrac{m}{\pi}}$$

$$rS := equivRadius\,(m)$$

$$locomotionPower\,(\phi, \mu, r_S, \nu, M, g)$$

$$0.000001432500000\,\pi\,\sqrt[3]{3}4^{2/3}\,\sqrt[3]{\tfrac{m}{\pi}}\nu^2 + 0.00009375000000\,M\nu^3$$

$$surfArea := (m, \delta) \mapsto \delta\,\sqrt[3]{4}3^{2/3}\left(\pi^{-1}\right)^{2/3} m^{2/3}$$

$$lP := locomotionPower\,(\phi, \mu, r_S, \nu, surfArea\,(m, \delta), g)$$

$$1.270696287 \times 10^{-14}\,\nu^2\,\sqrt[3]{m}\,(878819863.0 + 11357003710.0\,\delta\,\sqrt[3]{m}\nu)$$

$paperA5 := coeff\ (\sqrt[3]{m})$

$$0.00001116713137\,\nu^2$$

$paperA6 := coeff\ \left(lP, m^{2/3}\right)$

$$0.00009375000000\,\delta\,\sqrt[3]{4}3^{2/3}\left(\pi^{-1}\right)^{2/3}\,\nu^3$$

The two coefficients $paperA5$ and $paperA6$ are the two that appeared in the Ecology paper. We now begin to correlate these to the c1d and c2d expressions used in the simulation codes.

$c1d := (\nu, \phi, \mu, g) \mapsto 3\,\sqrt[3]{2}\,\sqrt[3]{3}\pi^{2/3}\phi\,\mu\,\nu^2 g$

ϕ is a coefficient representing form resistance and has value 0.001. μ represents the viscosity of the medium and is taken to be 0.955. g represents the animal's muscular efficiency with value 0.001. The value for δ, which represents proportion between $length^2$ to surface area. It is calculated below for $c2d$ to be set to $\delta = 0.002$. Substituting, then $c1d$ simplifies to:

$c1d(v, \phi, \mu, g)$

$$0.000002865\,\sqrt[3]{3}\,\sqrt[3]{2}\pi^{2/3}\nu^2$$

And now correlating $c2d$ to $paperA6$. $c2d := (\nu, g, \delta) \mapsto \frac{3}{32}\,\frac{3^{2/3}2^{2/3}\nu^3 g\delta}{\pi^{2/3}}$

$$0.001539338926\,v^3\delta$$

Which results in a value for δ of

$$\delta = 0.001948888545$$

so the value of δ is confirmed.

Note: The units are ergs/sec which must be converted to our normal units of J/d. There are 10^7 ergs/J and 86400 sec/day. The line of the code marked as changing from ergs/sec to J/d is to divide by 0.2 and multiply by 86400 which seems incorrect. See the Appendix to Hallam et al. (2000) to see how this is derived.

A.3.2 Growth Functions for Lipid and Structure

The expressions for the growth of lipid and structure, g_L and g_S, are broken down into several components in order to be able to simplify the calculation of the derivatives. One thing to notice is that the first term only involves ms, so it drops out of the derivatives involving ml.

The expression for protected structure is modeled as non-decreasing function. It represents the structure that is unavailable to the organism for energy demands. It is calculated as a portion of the total structure, but prevented from decreasing by comparing to its previous values. Because of its non-decreasing value is used as an expression for length through an allometric relationship. It

has the form $mps(ms, t_n) = max(A6ms, mps(t_{n-1}))$ for adults. If the adult is in a growth phase, then the derivative of mps is A6 (w.r.t. ms); otherwise its derivative is zero.

The following equations are in terms of ml and ms, mass of lipid and mass of structure, respectively.

Juvenile Phase

If the daphnid is still in the brood pouch, then only consumption of the stores in the egg are modeled. Therefore, $gl = 0.0$, $gpd = 0.0$, $work = 0.0$, $a3val = A3ZERO$, $a4val = A4ZERO$. This models simple consumption of egg stores.

Adult Phase

The function mps (zmpp1 is the variable) is the non-decreasing expression of protected structure. It is left here as a function in order to make manifest its role in the numerics. Its derivative has to be handled carefully, as mentioned above. There is a similar fraction taken of structure (zmps1 is the variable) which represents the starvation level. It is a fraction just a little larger than the proportion taken for protected structure. It represents the structure component level at which it is considered that all available stores have been converted to energy and the organism dies from starvation.

The function A2 is the new expression for the ingestion rate. The coefficients k3 and k1 are the allometric constants for the gut clearance expression, cg is the proportion of the body that is "gut" and bdp is the body density of the prey. The derivation of the original ingestion rate described in the Ecology paper is a lengthy section. The presumptions made in the paper were that the ingestion rate I_m should be proportional to volume and that the filtering rate, F_m, should be proportional to surface area. This was updated later by Henson in the models in order to prevent the daphnids from growing arbitrarily large.

$A2 := (ms) \mapsto \frac{ms}{k3\,(mps(ms))^{-k1}\,cg\,mps(ms)\,bdp}$

The function gd is the denominator of the first term of both g_L and g_S.

$gd := (ms) \mapsto A1\ \sqrt[3]{ms} + A2\,(ms)\,X$

which expands to

$$A1\ \sqrt[3]{ms} + \frac{ms\,X}{k3\,(massps(ms))^{-k1}\,cg\,massps(ms)\,bdp}$$

The function gg is the basic component of the first term of both g_L and g_S, noting that they differ only by a factor applied to X. Handling it this way allows one to compute the partials for both equations by just applying different factors to the partial of gg. The coefficient A0 is the assimilation

percentage for lipid. The A9 parameter which represents structure assimilation efficiency is relative to A0. charinfo[PLX] is the individual's percentage of lipid in the resource. The remaining peportions considered structure. PLX is one of the structuring parameters.

$gg := (ms) \mapsto \frac{A0\ X ms}{gd(ms)}$

The expression for work performed by the Daphnid forms the first two terms of the energy demand. Note that the derivatives of work with respect to ml and ms are equal, so it can be handled a little easier. The values for c1d and c2d calculated earlier are used for A5 and A6.

$work := (ml, ms) \mapsto A5\ \sqrt[3]{ml + ms} + A6\ (ml + ms)^{2/3}$

The function ed is the energy demand which is the work plus maintenance. Note that the conversion factors for going from mg of lipid or structure to energy are used, but are not in the published papers. These are 37.68 and 16.75, respectively, which are the same factors used in ea. Note that these factors do not appear in the corresponding function for fish, because they are included in the values used for A7 and A8.

$ed := (ml, ms) \mapsto work\,(ml, ms) + A7\ zjmgl\ ml + A8\ zjmgp\ ms$

The function ea is the energy available.

$ed := (ml, ms) \mapsto 37.68\ A3\ (ml - \epsilon\ mps\,(ms)) + 16.75\ A4\ (ms - mps\,(ms))$

The function fctr is the ratio of energy demand versus energy available. According to the model, this function is minimized against 1 at all points. ??? Work out to watch for derivatives?

$fctr := (ml, ms) \mapsto \frac{ed(ml,ms)}{ea(ml,ms)}$

The function zl is the remainder of the energy loss term in g_L.

$zl := (ml, ms) \mapsto A3\ (ml - \epsilon\ mps\,(ms))$

The function zs is the remainder of the energy loss term in g_S.

$zs := (ml, ms) \mapsto A4\ (ms - mps\,(ms))$

If plx is the percent of lipid in the resource and A9 is the assimilation rate for structure divided by A0, then the functions g_L and g_S are expressed in terms of the functions above as

$gl := (ml, ms) \mapsto plx\ gg\,(ms) - zl\,(ml, ms)\ fctr\,(ml, ms)$

$gs := (ml, ms) \mapsto (1 - plx)\ A9\ gg\,(ms) - zs\,(ml, ms)\ fctr\,(ml, ms)$

A.3.3 Derivatives

The derivatives for g_L and g_S are now calculated using the above functions. The derivatives may be skipped if iterating Newton's method more than once. We normally only iterate once, because of the rapid convergence of Newton's Method.

Because several of the functions involve mps(ms), the first derivative that should be calculated is its partial with respect to ms. Generally, this is simply A6, but in cases where it is being held constant, it should have derivative zero. This has not been consistently performed, which is part of the reason for the careful derivations in this paper. In the calculations below, this differentiation is left open, so that its role is more obvious.

The first derivative to be calculated is the derivative of A2(ms) with respect to ms. As noted above, the implicit inclusion of ms through mps greatly increases the complexity of the functions used in the numerics.

$$da2dms := \frac{d}{dms} A2\,(ms)$$
$$= \frac{(massps(ms))^{k1-1}}{k3\ cg\ bdp} + \frac{ms\,(massps(ms))^{k1-2}\left(\frac{d}{dms} massps(ms)\right)(k1-1)}{k3\ cg\ bdp}$$

Verifying against what was in the code originally. What was originally written is wrong at least because it doesn't take into account mps having zero derivative sometimes. Making the assumption that $mps := A6ms$ always, then the expression in the code was correct (in some versions) although it was also much more complicated than it really had to be as the second line shows.

The derivative of gd is next performed, because it involves the derivative da2dms.

$$diff(gd(ms),\ ms) := 1/3\,\frac{A1}{ms^{2/3}} + \left(\frac{d}{dms} A2\,(ms)\right) X$$

The derivative of gg with respect to ms can now be calculated in terms of the derivative of gd. The derivatives of the two uptake terms with respect to ms are easily determined from this, because they are simple scaling of this function as noted before.

$$diff\,(gg(ms), ms) := \frac{A0\ X}{gd(ms)} - \frac{A0\ X ms\ \frac{d}{dms} gd(ms)}{(gd(ms))^2}$$

Moving on to the loss terms, the derivatives are calculated in two parts: the energy fraction and the mobilization expression. The energy demand equation involves work(ml,ms). Note that the derivatives of work with respect to ml and ms are the same, so it is calculated only once.

$$diff(work(ml,\ ms),\ ml) := 1/3\,\frac{A5}{(ml+ms)^{2/3}} + 2/3\,\frac{A6}{\sqrt[3]{ml+ms}}$$

The derivatives of the energy fraction with respect to ml and ms are now calculated. The derivatives of ea with respect to ms includes mps, so it must be handled carefully. Two temporary variables, z1 and z2, are used to hold these.

$$diff\,(ea(ml, ms), ml) := 37.68\,A3$$
$$diff\,(ea(ml, ms), ms) := -37.68\,A3\,\epsilon\,\frac{d}{dms} massps\,(ms) + 16.75\,A4\,\left(1 - \frac{d}{dms} massps\,(ms)\right)$$
$$dfdml := diff\,(fctr(ml, ms), ml)$$
$$= \frac{\frac{\partial}{\partial ml} work(ml,ms) + A7\ zjmgl}{ea(ml,ms)} - \frac{(work(ml,ms) + A7\ zjmgl\ ml + A8\ zjmgp\ ms)\frac{\partial}{\partial ml} ea(ml,ms)}{(ea(ml,ms))^2}$$
$$dfdms := diff\,(fctr(ml, ms), ms)$$

$$= \frac{\frac{\partial}{\partial ms} work(ml,ms) + A8\ zjmgp}{ea(ml,ms)} - \frac{(work(ml,ms) + A7\ zjmgl\ ml + A8\ zjmgp\ ms)\frac{\partial}{\partial ms} ea(ml,ms)}{(ea(ml,ms))^2}$$

Putting these derivatives together with the expressions for mobilization, zl and zs, one ends up with the four partials

$$deldml := diff\left(zl(ml,ms) * fctr(ml,ms), ml\right)$$
$$= A3\ fctr\left(ml,ms\right) + A3\ \left(ml - \epsilon\ massps\left(ms\right)\right) \frac{\partial}{\partial ml} fctr\left(ml,ms\right)$$

The first term is represented by z1 in the code

$$deldms := diff\left(zl(ml,ms) * fctr(ml,ms), ms\right)$$
$$= -A3\ \epsilon\ \left(\frac{d}{dms} massps\left(ms\right)\right) fctr\left(ml,ms\right)\ + A3\ \left(ml - \epsilon\ massps\left(ms\right)\right) \frac{\partial}{\partial ms} fctr\left(ml,ms\right)$$
$$desdml := diff\left(zs(ml,ms) * fctr(ml,ms), ml\right)$$
$$= A4\ \left(ms - massps\left(ms\right)\right) \frac{\partial}{\partial ml} fctr\left(ml,ms\right)$$

The first term of desdms is represented by z2 in the code

$$desdms := diff\left(zs(ml,ms) * fctr(ml,ms), ms\right)$$
$$= A4\ \left(1 - \frac{d}{dms} massps\left(ms\right)\right) fctr\left(ml,ms\right)\ + A4\ \left(ms - massps\left(ms\right)\right) \frac{\partial}{\partial ms} fctr\left(ml,ms\right)$$

This completes the subordinate derivatives. The four partials of g_L and g_S can now be easily calculated from these components.

A.4 Fish Population Model

The population model for fish shares the same heritage as the *Daphnia* population model. In fact most of the source code was identical originally when McKinney wrote the first versions. Because they have undergone separate development paths since then under different people, the source codes had diverged. They have been reunited (for the most part) in the program FishDaph. This program is described in a following section.

Again, for this model it suffices to describe the equations for the growth of the lipid and structure compartments. The original paper describing the fish model is Hallam et al. (2000). The discussion of the fish model in this paper and its appendix is very complete. The reader is referred to it for a description of the derivation of the fish model and the parameter values used.

Not as many changes have been made to the fish model. Dina Lika worked extensively with the fish model developing a model for fish movement. Because she was making so many changes to the fundamental equations, she eventually rewrote the numerical portion of the code to use the Corrected Euler scheme, because it became too difficult to make sure that implicit schemes requiring derivatives were implemented correctly (Lika, 1996). Cyndi Lovelock has developed for fish an analogue of the

environmental effects models earlier done for *Daphnia* (Lovelock, 1996). She continued with a similar individual-based fish model in her dissertation Krohn (2001).

A.4.1 Individual Model and Numerical Simulation

The basic expressions for g_L and g_S are presented for completeness. The reader is referred to the original papers for details of the derivations. The notation has been changed slightly in order to match that presented above for *Daphnia*.

Equations for Fish

Juvenile Ages: $0 \leq a \leq P$

$$\begin{aligned} g_L &= -A_{14} m_L \\ g_S &= -A_{15} m_S \\ m_L(0) &= m_{L0} \text{ (determined from parent)} \\ m_S(0) &= m_{S0} \text{ (fixed amount)} \end{aligned}$$

Adult Ages: $P \leq a \leq$ maximum age. (Assuming $D \leq E$.)

$$\begin{aligned} F &= \frac{x}{[a_d]^{-1} + [\frac{s_d}{M_p \delta v} + [kM_g]^{-1}]x} \\ g_L &= A_{0L}(PLX)F - A_3(m_L - \epsilon m_{PS})\frac{D}{E} \\ g_S &= A_{0S}(1 - PLX)F - A_4(m_S - m_{PS})\frac{D}{E} \\ m_L(P) \text{ and } m_S(P) &= \text{terminal values of juvenile stage} \end{aligned}$$

where

$$\begin{aligned} D &= W(m_L, m_S) + A_7 m_L + A_8 m_S \\ W &= 1.188 \cdot 10^{-2} L_f^4 \beta_2 ((blsh)^{5/2} T_e F + (blsc)^{5/2} T_c F) \\ E &= 3.768 \cdot 10^4 A_3(m_L - \epsilon m_{PS}) + 1.675 \cdot 10^4 A_4(m_S - m_{PS}) \end{aligned}$$

The Juvenile stage is described as before as decreasing exponentials and initial conditions set from the parent. The fish model has a larger number of parameters, but for the most part their inclusion does not complicate the numerical equations. The fact that F appears in the expression of work due

to swimming does add some complexity. The numerical equations are now derived as they appear in the programs.

As described in the published fish model (Hallam et al. (2000)), the first simplification made for the numerics is to rewrite F in terms of characteristic times for encounter, capture, and digestion, T_e, T_c, and T_d, respectively. In terms of these,

$$F = \frac{1}{T_e + T_c + T_d}$$

The computations are done separately for each of these functions, and the resulting expressions are put together like what was done for the *Daphnia* model.

A.5 Maple Derivation of Fish Model

This worksheet describes the equations used in the numerical simulation of the fish population models.The main functions for the fish model are calculated in the function ffterm_c(). Note that ffterm_c() is not used for predator-prey, but only in the fish population models.

A.5.1 Functions

The expressions for g_L and g_S are broken down into several components in order to be able to simplify the calculation of the derivatives.

The function mps is the non-decreasing expression of protected structure. It is left here as a function in order to make manifest its role in the numerics. Its derivative has to be handled carefully, as mentioned above. Note that I added the growth flag just as in dfterm_c(), so that growth detection is correct.

The structural mass of the prey is first determined, which is used to calculate the length of a prey item, which is used in the calculation of the reactive distance of the fish and the velocity of the prey. The average mass of prey (mp) is given as a parameter in this model. The fact that PLX is a ecotypic parameter is the only variation that occurs in these expressions. These become true functions in the predator-prey model.

$lp := \sqrt[3]{\frac{(1-PLX)mp}{daphbeta}}$

$vp := blsp\ lp$

$zlp := a\ lp + b$

The total mass (zmass), length (lc), and various velocities (vc, vh) of the fish are now calculated. Note that the allometric beta is acoef in the code. I recorded it here as fishbeta to be consistent with the Daphnia allometric constant naming scheme.

$lf := ms \mapsto \sqrt[3]{\frac{mps(ms)}{fishbeta}}$

$vc := ms \mapsto blsc\ lf\ (ms)$

$vh := ms \mapsto blsh\ lf\ (ms)$

The reactive distance can now be calculated. This is a product of the linear function involving length of prey and the square root of the length of the fish.

$sd := (ms) \mapsto zlp\ \sqrt{lf\ (ms)}$

The difference in the prey and pursuit velocities. This is converted from cm per second to cm per day to be consistent (spd = seconds per day which is 8.64×10^4).

$dv := (ms) \mapsto spd\ |vc\ (ms) - vp|$

The variable ad is the encounter rate coefficient.

$ad := (ms) \mapsto 1/3\ \frac{\pi\ (sd(ms))^2 spd\ \left(vp^2 + 3\ (vh(ms))^2\right)}{vh(ms)}$

Gut volume (volgut) is assumed to be a fraction of the protected structure. There used to be an additional added constant called dg. It was always set to 0.0, so I deleted it. If dg is ever put back in, then it needs to be added to volgut.

$volgut := (ms) \mapsto cg\ mps\ (ms)$

Gut clearance rate coefficient which is another allometric expression.

$k := (ms) \mapsto \frac{k3}{(mps(ms))^{k1}}$

In terms of these functions, the three characteristic times for fish can be calculated. The three times are for Capture, Encounter, and Digestion.

Capture Time:

$tc := (ms) \mapsto \frac{sd(ms)}{dv(ms)mp}$

Encounter Time:

$te := (ms) \mapsto \frac{1}{ad(ms)X}$

Digestion Time: Note that volgut * density of the prey gives a mass capacity of the gut.

$td := (ms) \mapsto \frac{1}{k(ms)volgut(ms)bdp}$

Filtering rate for fish, A1, is not constant, because K3 is used to define ecotypes. It is calculated for reporting purposes, but it is not used in the calculations directly.

Now the core of the uptake rate can be calculated in terms of these times.

$F := (ms) \mapsto (te\ (ms) + tc\ (ms) + td\ (ms))^{-1}$

Finally, the growth of lipid and structure can be written as (splitting out of total components). The total growth of both compartments is stored on the characteristic as GROWTH.

$growthl := (ms) \mapsto F(ms)\, A0L\, PLX$

$growthp := (ms) \mapsto F(ms)\, A0P\, (1 - PLX)$

Now that we have calculated the rates, we can compute the work exerted by the individual fish in the course of searching and feeding.

$tca := (ms) \mapsto tc(ms)\, F(ms)$

$tea := (ms) \mapsto te(ms)\, F(ms)$

$c2 := (ms) \mapsto blsh^{5/2} tca(ms) + blsc^{5/2} tca(ms)$

$work := (ms) \mapsto 0.01188000000\, (lf(ms))^4\, beta2\, c2(ms)$

Now, in the same design as the Daphnia, the energy demands and energy available at this moment are calculated. The function zl is the remainder of the energy loss term in g_L. It is the energy mobilizable from lipid. Note, repfat in the code equals epsilon.

$zl := (ml, ms) \mapsto A3\, (ml - \epsilon\, mps(ms))$

The function zs is the remainder of the energy loss term in g_S. It is the energy mobilizable from structure.

$zs := (ml, ms) \mapsto A4\, (ms - mps(ms))$

The function ed is the energy demand which is the work plus maintenance. This function does not need energy conversion factors, because they are built into the values of the parameters.

$ed := (ml, ms) \mapsto work(ms) + A7\, ml + A8\, ms$

The function ea is the energy available.

$ea := (ml, ms) \mapsto 37.68\, zl(ml, ms) + 16.75\, zs(ml, ms)$

In the normal case that energy demand does not exceed energy available, this is the fraction utilized. The function fctr is the ratio of energy demand versus energy available. According to the model, this function is minimized against 1 at all points.

$fctr := (ml, ms) \mapsto \frac{ed(ml,ms)}{ea(ml,ms)}$

Finally, we can put the growth and losses together for the total change in the lipid and structure components. If plx is the percent of lipid in the resource and A9 is the assimilation rate for structure divided by A0, then the functions g_L and g_S are expressed in terms of the functions above as

$gl := (ml, ms) \mapsto growthl(ms) - zl(ml, ms)\, fctr(ml, ms)$

$gs := (ml, ms) \mapsto growthp(ms) - zs(ml, ms)\, fctr(ml, ms)$

A.5.2 Derivatives

The derivatives for g_L and g_S are now calculated using the above functions.

Because several of the functions involve mps(ms), the first derivative that should be calculated is its partial with respect to ms. Generally, this is simply A6, but in cases where it is being held constant, it should have derivative zero. This was consistently done for the fish model, mainly because the numerics have not been tinkered with as much as those for Daphnia.

$dlfdms := \frac{d}{dms} lf\,(ms)$

$$= 1/3\,\frac{d}{dms} mps\,(ms)\left(\frac{mps(ms)}{fishbeta}\right)^{-2/3} fishbeta^{-1}$$

$dsddms := diff(sd(ms), ms)$

$$= 1/2\,\frac{(alp+b)\frac{d}{dms} lf(ms)}{sqrt(lf(ms))}$$

$dvcdms := diff(vc(ms), ms)$

$$= blsc\,\frac{d}{dms} lf\,(ms)$$

$dvhdms := diff(vh(ms), ms)$

$$= blsh\,\frac{d}{dms} lf\,(ms)$$

Note that the derivative of absolute value is the signum function or sign in F77. It is denoted by Maple as abs(1,f(x)).

$ddvdms := diff(dv(ms), ms)$

$$= -spd\;abs\,(1, -blsc\;lf(ms) + blsp\;lp)\;blsc\,\frac{d}{dms} lf\,(ms)$$

Notice that Maple caught a lot of simplification that occurs because sd is squared in the top. This produces an lf in the top that cancels against the one in vh in the bottom. This leaves only one term with ms in it, so the derivative is simple.

$daddms := diff(ad(ms), ms)$

$$= 1/3\,\frac{\pi\,(alp+b)^2\left(\frac{d}{dms} lf(ms)\right) spd\left(blsp^2 lp^2 + 3\,(vh(ms))^2\right)}{vh(ms)}$$

$$+2\,\pi\;(alp+b)^2\;lf\,(ms)\;spd\,\frac{d}{dms} vh\,(ms)$$

$$-1/3\,\frac{\pi\,(alp+b)^2 lf(ms) spd\left(blsp^2 lp^2 + 3\,(vh(ms))^2\right)\frac{d}{dms} vh(ms)}{(vh(ms))^2}$$

Testing daddms against what is in the code. First, we have to prevent Maple from simplifying ad too much. After differentiating the unsimplified ad(ms), then the coefficients qq1 and qq2 in the code are from grouping on the derivative of sd and the derivative of vh. Dividing the bottom into the top for both qq1 and qq2 yields exactly what is in the code.

$udaddms := diff(ad(ms), ms)$

$$2/3\,\frac{\pi\;sd(ms) spd\left(vp^2 + 3\,(vh(ms))^2\right)\frac{d}{dms} sd(ms)}{vh(ms)}$$

$$+2\,\pi\ (sd\,(ms))^2\ spd\ \tfrac{d}{dms}\,vh\,(ms)$$

$$-1/3\ \tfrac{\pi\,(sd(ms))^2 spd\left(vp^2+3\,(vh(ms))^2\right)\frac{d}{dms}vh(ms)}{(vh(ms))^2}$$

$qq1\ :=\ 1/3\ \tfrac{vp^2+3\,(vh(ms))^2}{vh(ms)}$

$qq2\ :=\ -1/3\ \tfrac{-3\,(vh(ms))^2+vp^2}{(vh(ms))^2}$

Computing the derivative of tc with respect to ms. To get what is in the code, divide out the expression for tc.

$dtcdms := diff\,(tc(ms), ms)$

$$= \tfrac{\frac{d}{dms}sd(ms)}{dv(ms)mp} - \tfrac{sd(ms)\frac{d}{dms}dv(ms)}{(dv(ms))^2 mp}$$

$dtedms := diff\,(te(ms), ms)$

$$= -\tfrac{\frac{d}{dms}ad(ms)}{(ad(ms))^2 X}$$

$dtddms := diff\,(td(ms), ms)$

$$= \tfrac{(massps(ms))^{k1-2}\left(\frac{d}{dms}massps(ms)\right)(k1-1)}{k3\ cg\ bdp}$$

This expression for dtddms can be verified against what is in the code. Note that I use diff(mps(ms),ms) in place of A6 in the code.

$dFdms := diff\,(F(ms), ms)$

$$= -\tfrac{\frac{d}{dms}te(ms)+\frac{d}{dms}tc(ms)+\frac{d}{dms}td(ms)}{(te(ms)+tc(ms)+td(ms))^2}$$

Moving on to the loss terms, the derivatives are calculated in two parts: the energy fraction and the mobilization expression. The energy demand equation involves work(ms). Since work(ms) is an involved expression, it is calculated in several steps.

$dtcadms := diff\,(tca(ms), ms)$

$$= \left(\tfrac{d}{dms}tc\,(ms)\right) F\,(ms) + tc\,(ms)\,\tfrac{d}{dms}F\,(ms)$$

$dteadms := diff\,(tea(ms), ms)$

$$= \left(\tfrac{d}{dms}te\,(ms)\right) F\,(ms) + te\,(ms)\,\tfrac{d}{dms}F\,(ms)$$

$dc2dms := diff\,(c2(ms), ms)$

$$= blsh^{5/2}\,\tfrac{d}{dms}tca\,(ms) + blsc^{5/2}\,\tfrac{d}{dms}tca\,(ms)$$

$dwork := diff\,(work(ms), ms)$

$= 0.04752000000\ (lf\ (ms))^3\ beta2\ c2\ (ms)\ \frac{d}{dms}lf\ (ms) + 0.01188000000\ (lf\ (ms))^4\ beta2\ \frac{d}{dms}c2\ (ms)$

This expression for dwork can be verified against what is in the code.

Derivatives of growth functions.

$dgrowthldms := diff\,(growthl(ms), ms)$

$$= \left(\tfrac{d}{dms}F\,(ms)\right) A0L\,PLX$$

$dgrowthpdms := diff\,(growthp(ms), ms)$

$$= \left(\frac{d}{dms} F\left(ms\right)\right) A0P\ \left(1 - PLX\right)$$

The derivatives with respect to structure of the mobilization expressions are done separately because they involve mps(ms), which has derivative zero in non-growth conditions.

$dzldms := diff\left(zl(ml, ms), ms\right)$

$$= -A3\ \epsilon\ \frac{d}{dms} massps\left(ms\right)$$

$dzsdms := diff\left(zs(ml, ms), ms\right)$

$$= A4\ \left(1 - \frac{d}{dms} massps\left(ms\right)\right)$$

Derivatives of energy demand equation with respect to ms and ml are now simple.

$deddml := diff\left(ed(ml, ms), ml\right) = A7$

$deddms := diff\left(ed(ml, ms), ms\right)$

$$= \frac{d}{dms} work\left(ms\right) + A8$$

The derivatives of the energy fraction with respect to ml and ms are now calculated. (Note: Derivative of work with respect to ml is zero for fish.)

$dfdml := diff\left(fctr(ml, ms), ml\right)$

$$= \frac{A7}{37.68\ A3\ (ml - \epsilon\ massps(ms)) + 16.75\ A4\ (ms - massps(ms))}$$
$$-37.68\ \frac{(work(ms) + A7\ ml + A8\ ms)A3}{(37.68\ A3\ (ml - \epsilon\ massps(ms)) + 16.75\ A4\ (ms - massps(ms)))^2}$$

$dfdms := diff\left(fctr(ml, ms), ms\right)$

$$= \frac{\frac{d}{dms} work(ms) + A8}{37.68\ A3\ (ml - \epsilon\ massps(ms)) + 16.75\ A4\ (ms - massps(ms))}$$
$$-\frac{(work(ms) + A7\ ml + A8\ ms)\left(-37.68\ A3\ \epsilon\ \frac{d}{dms} massps(ms) + 16.75\ A4\ \left(1 - \frac{d}{dms} massps(ms)\right)\right)}{(37.68\ A3\ (ml - \epsilon\ massps(ms)) + 16.75\ A4\ (ms - massps(ms)))^2}$$

Putting these derivatives together with the expressions for mobilization, zl and zs, one ends up with the four partials.

$deldml := diff\left(zl(ml, ms) * fctr(ml, ms), ml\right)$

$$A3\ fctr\left(ml, ms\right) + A3\ \left(ml - \epsilon\ massps\left(ms\right)\right) \frac{\partial}{\partial ml} fctr\left(ml, ms\right)$$

$deldms := diff\left(zl(ml, ms) * fctr(ml, ms), ms\right)$

$$= -A3\ \epsilon\ \left(\frac{d}{dms} massps\left(ms\right)\right) fctr\left(ml, ms\right)\ + A3\ \left(ml - \epsilon\ massps\left(ms\right)\right) \frac{\partial}{\partial ms} fctr\left(ml, ms\right)$$

$desdml := diff\left(zs(ml, ms) * fctr(ml, ms), ml\right)$

$$= A4\ \left(ms - massps\left(ms\right)\right) \frac{\partial}{\partial ml} fctr\left(ml, ms\right)$$

$desdms := diff\left(zs(ml, ms) * fctr(ml, ms), ms\right)$

$$= A4\ \left(1 - \frac{d}{dms} massps\left(ms\right)\right) fctr\left(ml, ms\right)\ + A4\ \left(ms - massps\left(ms\right)\right) \frac{\partial}{\partial ms} fctr\left(ml, ms\right)$$

This completes the subordinate derivatives. The four partials of g_L and g_S can now be easily calculated from these components.

Vita

Jeffrey Alan Nichols was born in Tulsa, Oklahoma, on October 17,1968, the first son of Jerry and Joyce Nichols.

He became enthused about mathematics after moving to Alabama and participating on competitive math teams for three years; several times placing at the state and national levels. After graduating in 1986 from Huffman High School, Birmingham, Alabama, he attended the University of Alabama at Birmingham on a 4-year Honors Scholarship where he received a Bachelor of Science degree in Mathematics, *summa cum laude*, in 1990. He was also recognized as the top mathematics student his senior year.

During his undergraduate studies, he worked for three years at the Southern Research Institute, where some of the first anti-viral AIDS tests were performed in which he played an information management, computer automation role. He started a computer consulting business while in Birmingham called NiComp Consulting (nicomp.com).

He participated in an NSF Research Experience for Undergraduates summer session at the University of Tennessee studying under Dr. Don Hinton in 1989.

Jeff entered graduate school in August 1990 at the University of Tennessee in Knoxville along with several other former REU students. He received a Science Alliance Fellowship for his first year in graduate school, and NSF Honorable Mention, and recognition as the top first-year graduate student.

While leading a hike for the new REU students in the Smokies to Andrew's Bald on June 15, 1991, Jeff met his future wife, Sherry, at Newfound Gap. They were engaged on the same spot where they met on December 13, 1991, and were married in Butler, PA, on June 13, 1992. Several children were born to Jeff and Sherry including several who died *in utero*. Rebecca was born in August 1995 as a high-risk pregnancy managed by Dr. Bill Holls. Philip, their only son, was born August, 1997. Karla was born October, 1999. Finally, Rachel was born in November 2002.

Jeff worked at UT Medical Center with Dr. Holls. He left graduate school and began Consulting in 1999 to America's Collectibles Network as a Linux consultant. This eventually led to positions as IT Director and Solutions Architect with ACN.

But the research itch would not go away. He returned to the academic world as a doctoral student at The University of Tennessee in Mathematics in Spring Semester 2006. With support from ACN, he left their employment in March 2006 and became a consultant again.

He completed his Doctor of Philosophy degree in December 2007.

In the future, Jeff hopes to continue to scratch the research itch.