

**DYNAMIC WATER QUALITY MODELING
USING CELLULAR AUTOMATA**

by

Antonio Paulo Castro

A dissertation submitted to
the faculty of Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY
in
Civil Engineering

APPROVED:



Daniel L. Gallagher, Chairman



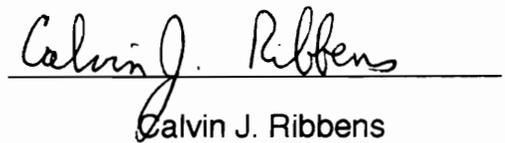
Antonio S. Câmara



Theo A. Dillaha



G. V. Loganathan



Calvin J. Ribbens

August, 1996
Blacksburg, Virginia

Keywords: cellular automata, water quality modeling, parallel processors

c.2

LD
5655
V856
1996
C382
c.2

DYNAMIC WATER QUALITY MODELING USING CELLULAR AUTOMATA

by

Antonio Paulo Castro

Daniel L. Gallagher, Chairman

Civil Engineering Department

(ABSTRACT)

Parallel computing has recently appeared as an alternative approach to increase computing performance. In the world of engineering and scientific computing the efficient use of parallel computers is dependent on the availability of methodologies capable of exploiting the new computing environment. The research presented here focused on a modeling approach, known as cellular automata (CA), which is characterized by a high degree of parallelism and is thus well suited to implementation on parallel processors. The inherent degree of parallelism also exhibited by the random-walk particle method provided a suitable basis for the development of a CA water quality model. The random-walk particle method was successfully represented using an approach based on CA. The CA approach requires the definition of transition rules, with each rule representing a water quality process. The basic water quality processes of interest in this research were advection, dispersion, and first-order decay. Due to the discrete nature of CA, the rule for advection introduces considerable numerical dispersion. However, the magnitude of this numerical dispersion can be minimized by proper selection of model parameters, namely the size of the cells and the time step. Similarly, the rule for

dispersion is also affected by numerical dispersion. But, contrary to advection, a procedure was developed that eliminates significant numerical dispersion associated with the dispersion rule. For first-order decay a rule was derived which describes the decay process without the limitations of a similar approach previously reported in the literature. The rules developed for advection, dispersion, and decay, due to their independence, are well suited to implementation using a time-splitting approach. Through validation of the CA methodology as an integrated water quality model, the methodology was shown to adequately simulate one and two-dimensional, single and multiple constituent, steady-state and transient, and spatially invariant and variant systems. The CA results show a good agreement with corresponding results for differential equation based models. The CA model was found to be simpler to understand and implement than the traditional numerical models. The CA model was easily implemented on a MIMD distributed memory parallel computer (Intel Paragon). However, poor performance was obtained.

ACKNOWLEDGMENTS

This research has been supported by the National Science Foundation, the Virginia Tech Core Research Program, and grants from the Junta Nacional de Investigação Científica e Tecnológica, Portugal (grants BD/1003/90-RN and BD/3033/94). I would like to thank the Virginia Tech Computer Science Department and the Oak Ridge National Laboratory for access to the Intel Paragon.

I am grateful to Dr. Daniel Gallagher, my major advisor, for his research guidance, understanding, patience, kindness, support, and friendship during the many years of this project.

I would like to thank Dr. Antonio Câmara for his crucial contribution to the launching of this project and for serving in the research committee.

I thank Dr. Calvin Ribbens of the Virginia Tech Computer Science Department for his constant help and support regarding the use of Intel parallel computers and for serving in the research committee.

I also thank Dr. Theo Dillaha and Dr. G. V. Loganathan for serving in the research committee.

A special thanks to Madalena and Álvaro, my parents, for the eternal support and friendship. And to Karin for being such a good friend.

And finally, thanks to the Wilderness of North America for being a constant source of inspiration and well-being.

TABLE OF CONTENTS

ABSTRACT	ii
ACKNOWLEDGMENTS	iv
TABLE OF CONTENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
1 INTRODUCTION	1
REFERENCES	3
2 DYNAMIC WATER QUALITY MODELING USING CELLULAR AUTOMATA:	
MODEL DEVELOPMENT	5
2.1 INTRODUCTION	5
2.1.1 Random-Walk Particle Method	5
2.1.2 Cellular Automata	7
2.2 ADVECTION.....	11
2.2.1 Methodology	11
2.2.1.1 Rule Definition	11
2.2.1.2 Advection Induced Numerical Dispersion.....	12
2.2.2 Results and Discussion	14
2.3 DISPERSION	20
2.3.1 Methodology	20
2.3.1.1 Rule Definition	20
2.3.1.2 Effect of P_{disamp} on Dispersion	24
2.3.2 Results and Discussion	26
2.4 DECAY	35
2.4.1 Methodology	35
2.4.1.1 Rule Definition	35
2.4.1.2 Effect of P_{dec} on Decay.....	37
2.4.2 Results and Discussion	38
2.5 WATER QUALITY MODEL DEVELOPMENT	46
2.6 CONCLUSIONS	47
APPENDIX A: Characterization of the mass distribution resulting from the advection induced numerical dispersion	48

APPENDIX B: Derivation of the expression for the dispersion probability	51
REFERENCES	53
3 DYNAMIC WATER QUALITY MODELING USING CELLULAR AUTOMATA:	
MODEL APPLICATION USING PARALLEL PROCESSORS	59
3.1 INTRODUCTION	59
3.1.1 Parallel Computing	59
3.2 METHODOLOGY	62
3.2.1 General Model	62
3.2.2 BOD/DO Model	68
3.2.3 Simulation Scenarios	69
3.2.3.1 One-Dimensional Line Pulse Input	69
3.2.3.2 Two-Dimensional Pulse Input	70
3.2.3.3 One-Dimensional Steady-State BOD/DO	72
3.2.3.4 One-Dimensional Tidal BOD/DO	77
3.2.4 Parallel Computer Implementation	80
3.3 RESULTS AND DISCUSSION	82
3.3.1 One-Dimensional Line Pulse Input	82
3.3.2 Two-Dimensional Pulse Input	85
3.3.3 One-Dimensional Steady-State BOD/DO	87
3.3.4 One-Dimensional Tidal BOD/DO	97
3.4 CONCLUSIONS	100
REFERENCES	101
4 EUTROPHICATION MODELING WITH CELLULAR AUTOMATA	104
4.1 WATER QUALITY CONSTITUENTS	104
4.2 WATER QUALITY PROCESSES	109
4.2.1 Advection and Dispersion	109
4.2.2 Aerobic Biodegradation	109
4.2.3 Reaeration/Deaeration	110
4.2.4 Denitrification	111
4.2.5 Nitrification	113
4.2.6 Hydrolysis	114
4.2.7 Photosynthesis	115
4.2.8 Respiration	118
4.2.9 Death	119

4.2.10 Grazing	119
4.2.11 Settling	121
4.3 EXTERNAL SOURCES AND SINKS	123
4.4 IMPLEMENTATION ON PARALLEL PROCESSORS	123
REFERENCES	124
5 CONCLUSIONS	125
APPENDIX: LISTING OF THE C SOURCE CODE FOR THE MAIN COMPONENTS OF THE CA WATER QUALITY MODEL	128
VITA	170

LIST OF TABLES

Table 3.1	Parameter values used in the one-dimensional line pulse input simulation.	71
Table 3.2	Parameter values used in the two-dimensional pulse input simulation.	73
Table 3.3	Parameter values used in the one-dimensional steady-state BOD/DO simulation for a river with uniform cross-section and a single continuous discharge.	74
Table 3.4	Parameter values used in the one-dimensional steady-state BOD/DO simulation for a river with variable cross-section and multiple continuous discharges.	75
Table 3.5	Parameter values used in the one-dimensional tidal BOD/DO simulation.	79

LIST OF FIGURES

Figure 2.1	Cellular automata evolving in a discrete time interval (Δt). (a) One-dimensional cellular automaton. (b) Two-dimensional cellular automaton.	9
Figure 2.2	Time evolution of the mean (a) and standard deviation (b) of the mass distribution as a function of the advection probability using the CA surrogate method. ($\Delta x = 10, u = 5.$)	15
Figure 2.3	Relation between the advection induced numerical dispersion (E_{num}) and the advection probability.....	17
Figure 2.4	Relation between $E_{num}/(u\Delta x)$ and the advection probability.	18
Figure 2.5	Guideline for the selection of Δx_{max} for typical values of u_{max} and $E_{num,max}$ for rivers and estuaries.	21
Figure 2.6	Guideline for the selection of $\Delta t_{adv,max}$ for typical values of u_{max} and $E_{num,max}$ for rivers and estuaries.	22
Figure 2.7	Time evolution of the standard deviation of the distribution of number of particles as a function of P_{disp} using the CA dispersion rule, and comparison with the analytical solution for the dispersion differential equation for an instantaneous input. ($N_p^0 = 1000, \Delta x = 10, E = 1.$)	27
Figure 2.8	Relation between $(\alpha_{ca}/\alpha_a)^2$ and P_{disp}	28
Figure 2.9	Time evolution of the standard deviation of the distribution of number of particles as a function of P_{disp} using the CA dispersion rule corrected for numerical dispersion and comparison with the analytical solution. ($N_p^0 = 1000, \Delta x = 10, E = 5.$)	32
Figure 2.10	Comparison of mass distributions for the CA with dispersion rule corrected and non-corrected for numerical dispersion and the analytical solution as a function of P_{disp} . ($N_p^0 = 1000, \Delta x = 10, E = 5, t \approx 50.$)	34

Figure 2.11	Time evolution of the peak of mass distributions for the CA and the analytical solution as a function of P_{disp} : (a) dispersion rule without correction for numerical dispersion; (b) with correction. ($N_p^0=1000, \Delta x=10, E=5.$)	36
Figure 2.12	Time evolution of the fraction of mass remaining as a function of the decay probability using the CA surrogate method and comparison with the analytical solution for first-order decay. ($k_{dec}=0.5.$)	39
Figure 2.13	Relation between the ratio $\beta = \frac{k_{dec}^{ca}}{k_{dec}}$ and the decay probability for first-order decay.	41
Figure 2.14	Convergence values for β and P_{dec} as a function of $k_{dec}\Delta t_{dec}$ for first-order decay.	42
Figure 2.15	Relation between $\Delta t_{dec,max}$ and $k_{dec,max}$ for first-order decay.	44
Figure 2.16	Time evolution of the fraction of mass remaining as a function of the decay probability for the CA decay rule using equation (2.30) and comparison with the analytical solution for first-order decay. ($N_p^0=10000, k_{dec}=0.05.$)	45
Figure A1	Comparison between the mass distributions obtained for the CA advection surrogate method (solid line) and the analytical solution for the advection-dispersion differential equation for an instantaneous input (dotted line). The distributions are shown for different advection probability values and at the end of various simulation steps.	50
Figure 3.1	Layout of the variable cross-section river system corresponding to the model input parameters of Table 3.4.	76
Figure 3.2	Comparison of concentration profiles, noise to signal ratio, and computation time from CA model simulations for evaluation of the effects of the packet fraction approach. (Model input parameters from Table 3.1.)	84
Figure 3.3	Concentration plume of a conservative constituent at successive times after a pulse discharge at a distance of 0.25 km and zero depth. Comparison between the CA model and the two-dimensional advection-dispersion differential equation. (Model input parameters from Table 3.2.)	86

Figure 3.4	Concentration profiles corresponding to longitudinal and vertical transects passing through the maximum concentration point of the plume in Figure 3.3 at a time of 25 minutes after the discharge. Comparison between the CA model and the two-dimensional advection-dispersion differential equation.	88
Figure 3.5	Concentration profiles for BOD and DO at steady-state for a river with uniform cross-section and a single continuous discharge. The BOD is being continuously discharged at a distance of 20 km. Comparison between the CA model and the Streeter and Phelps model. The CA results are from a single model simulation. (Model input parameters from Table 3.3.)	89
Figure 3.6	Concentration profiles for BOD and DO at steady-state for a river with variable cross-section and multiple continuous discharges. Comparison between the CA model and the Streeter and Phelps model. The CA results are from a single model simulation. (Model input parameters from Table 3.4.)	90
Figure 3.7	Distribution of the number of BOD and DO particles and the computation time among the worker nodes, corresponding to the CA simulation results of Figure 3.5. The worker nodes are numbered based on an upstream to downstream ordering of their subdomains.	92
Figure 3.8	Distribution of the number of BOD and DO particles and the computation time among the worker nodes, corresponding to the CA simulation results of Figure 3.6. The worker nodes are numbered based on an upstream to downstream ordering of their subdomains.	93
Figure 3.9	Execution times for the uniform cross-section steady-state BOD/DO simulation as a function of the number of worker nodes used in the simulation. (Model input parameters from Table 3.3.)	95
Figure 3.10	Graphical representation of the time evolution of the advective velocity for the simulation scenario corresponding to Table 3.5. The circles indicate the time at which model results are shown in Figure 3.11.	98
Figure 3.11	Concentration profiles for BOD and DO at successive slack water times over an entire tidal cycle. The BOD is being continuously discharged at a distance of 15 km. Comparison between the CA model and a differential equation model as described in the text. The CA results are from a single model simulation. (Model input parameters from Table 3.5.)	99
Figure 4.1	Water quality constituents and processes typically included in a eutrophication model.	105

Figure 4.2 External Sources and sinks typically included in a eutrophication model. 106

1 INTRODUCTION

Since the advent of the electronic computer in the 1950's a typical ten-fold improvement in speed performance has occurred every five years, mainly as a result of considerable advances in electronic integrate circuitry. However, such technological progress has not been sufficient to satisfy the increasing computational demand from engineering and scientific applications. Thus, parallel computation appeared as an alternative approach to increase computer performance. This involves incorporating multiple computational units in a single computer and operating them concurrently, thereby substantially increasing system performance (Green, 1991; Messina, 1991).

Parallel computers have evolved substantially during the last decade and that trend is expected to continue (Messina, 1991; Fox *et al.*, 1994). The possibility of successfully scaling to large number of processors is shown by the testimony of high performance machines now operational (Messina, 1991; Fox *et al.*, 1994). Many parallel computer architectures have proved reliable, and successful in engineering and scientific applications involving large-scale computations (Fox *et al.*, 1988; Fox, 1991; Messina, 1991; Camp *et al.*, 1994; Dabdub and Seinfeld, 1994; Fox *et al.*, 1994).

In the world of engineering and scientific computing, the efficient use of parallel computers is dependent on the availability of methodologies capable of exploiting the new computing environment. Modeling methodologies successfully implemented on sequential (single processor) machines are efficient in exploiting the computational power of a single processor. However, the most efficient use of parallel processors comes from methodologies that are inherently parallel (Camp *et al.*, 1994).

Some modeling approaches characterized by a high degree of parallelism may already have been developed in the past and possibly neglected due to their poor performance when implemented on sequential machines. However, the emerging parallel

computing field opens a door of opportunity to new and old modeling techniques able to exploit the newer computing environment.

The research presented here focuses on a modeling approach, known as cellular automata (CA), which is characterized by a high degree of parallelism, and is therefore well suited to implementation on parallel processors (Toffoli and Margolus, 1987; Amato, 1991; Fox *et al.*, 1994). Cellular automata were first introduced in the late 1940's by John von Neumann (von Neumann, 1966). Cellular automata gained popularity three decades later through John Conway's work in the game of Life (Fogelman *et al.*, 1987; Toffoli and Margolus, 1987). However, the potential of CA as a modeling tool only recently has been realized with the advent of parallel computing (Toffoli and Margolus, 1987; Fox *et al.*, 1994).

Other potential benefits of CA as a water quality modeling tool include: (1) a better representation of the real physical system by bridging the differences between macroscopic and microscopic representations; (2) no power series truncation and, in certain implementations, no round-off error; (3) easy extension from a one-dimensional to a higher dimensional representation; and (4) since CA models can be based on simple, microscopic behavior, the focus of water quality modeling can be placed on the water quality mechanisms and not on the numerical solution technique.

The overall goal of this research is to evaluate the potential of the cellular automata methodology as a simulation tool for water quality modeling. The criteria for this evaluation include characterization of numerical dispersion, and model validation through comparison of simulation results with established water quality models.

The specific objectives of this research are to:

- develop CA representations of the more common water quality modeling processes;

- evaluate the numerical accuracy of CA representations, in particular the evaluation of numerical dispersion introduced through the discrete nature of the model;
- integrate individual process rules into typical water quality models;
- compare cellular automata model results with existing analytical and numerical solutions for typical modeling scenarios;
- evaluate the feasibility and performance gains associated with the implementation of the cellular automata water quality model on parallel processors.

REFERENCES

- Amato, I., "Speculating in Precious Computronium." *Science*, 253, 856-857 (1991).
- Camp, W. J., S. J. Plimpton, B. A. Hendrickson, and R. W. Leland, "Massively Parallel Methods for Engineering and Science Problems." *Communications of the ACM*, 37, 31-41 (1994).
- Crowl, L. A., "How to Measure, Present, and Compare Parallel Performance." *IEEE Parallel & Distributed Technology, Spring Issue*, 9-25 (1994).
- Dabdub, D., and J. H. Seinfeld, "Air Quality Modeling on Massively Parallel Computers." *Atmospheric Environment*, 28, 1679-1687 (1994).
- Fogelman, F., Y. Robert, and M. Tchente, *Automata Networks in Computer Science: Theory and Applications*. Princeton University Press, Princeton, NJ, 264 p. (1987).
- Fox, G. C., M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors: General Techniques and Regular Problems, Volume I*. Prentice Hall, Englewood Cliffs, NJ, 592 p. (1988).

- Fox, G. C., "Achievements and Prospects for Parallel Computing." *Concurrency: Practice and Experience*, 3 (6), 725-739 (1991).
- Fox, G. C., D. R. Williams, and P. C. Messina, *Parallel Computing Works!*. Morgan Kaufmann, San Francisco, CA, 977 p. (1994).
- Green, S., *Parallel Processing for Computer Graphics*. MIT Press, Cambridge, MA, 233 p. (1991).
- Messina, P. C., "Parallel Computing in the 1980s - One Person's View." *Concurrency: Practice and Experience*, 3 (6), 501-524 (1991).
- Toffoli, T., and N. Margolus, *Cellular Automata Machines: a New Environment for Modeling*. MIT Press, Cambridge, MA, 259 p. (1987).
- von Neumann, J., *Theory of Self-Reproducing Automata*. A. W. Burks, ed., University of Illinois Press, Urbana, IL, 388 p. (1966).

2 DYNAMIC WATER QUALITY MODELING USING CELLULAR AUTOMATA: MODEL DEVELOPMENT

2.1 INTRODUCTION

Modeling the fate and transport of environmental contaminants in general, and water quality modeling in particular, frequently requires mathematical formulations involving differential equations for which analytical solutions do not exist unless simplifying assumptions are made. Thus numerical methods are used to provide solutions for those complex mathematical representations.

One important category of numerical methods is the Lagrangian particle models. Various types of particle models have been considered in the past, and applied to the simulation of a range of physical problems (Hockney and Eastwood, 1988). In some cases the computational particles represent actual physical particles such as molecules (Hockney and Eastwood, 1988). More often, the computational (or fictitious) particles are used to represent a discrete parcel of the parameter to be simulated: fluid elements of a fluid flow application, or mass parcels in a simulation of an environmental contaminant (Hockney and Eastwood, 1988; Zannetti, 1990). The dynamics of computational particles, such as their motion, can be considered to be either deterministic or stochastic (through Monte-Carlo techniques) (Zannetti, 1990).

2.1.1 Random-Walk Particle Method

The random-walk particle method (RWPM), which is a stochastic Lagrangian particle model, has been applied to the fate and transport modeling of environmental constituents in both groundwater (Ahlstrom *et al.*, 1977; Prickett *et al.*, 1981; Bear and Verruijt, 1987; Ackerer, 1988; Kinzelbach, 1988; Uffink, 1988; Valocchi and Quinodoz,

1989; Tompson and Gelhar, 1990; Dougherty, 1991; Tompson and Dougherty, 1992; Mahinthakumar and Valocchi, 1993) and surface water (Williams and Hinwood, 1976; Allen, 1982; Shen *et al.*, 1987; Shen and Yapa, 1988; Józsa, 1989; Kleinschmidt and Pearce, 1992; Bogle *et al.*, 1993; Dimou and Adams, 1993). When nonlinearities (such as source/sink terms) are absent, computational particles are completely independent of each other, thus allowing particle behavior to be computed in parallel (Ahlstrom *et al.*, 1977). In fact, several implementations of the RWPM using parallel computers have been reported (Dougherty and Tompson, 1990; Dougherty, 1991; Mahinthakumar and Valocchi, 1993).

The original motivation for this research was to develop a water quality model based on the cellular automata (CA) approach to be implemented using parallel processors. Cellular automata are characterized by a high degree of parallelism (Toffoli and Margolus, 1987). The inherent degree of parallelism also exhibited by the RWPM was assumed to provide a suitable basis for the development of a CA water quality model. In fact, Brieger and Bonomi (1991) while using a different approach than in the current work have shown that a random-walk methodology can be adapted to CA.

The derivation of the RWPM is based upon analogies established between the transport equations and probability distributions (Józsa, 1989; Tompson and Gelhar, 1990). The RWPM in its basic form comprises a deterministic component representing the advective particle transport due to an average velocity, and a stochastic component which represents the randomness associated with particle movement due to dispersion (Ahlstrom *et al.*, 1977; Bear and Verruijt, 1987; Ackerer, 1988; Uffink, 1988; Valocchi and Quinodoz, 1989; Dougherty, 1991; Tompson and Dougherty, 1992; Dimou and Adams, 1993; Mahinthakumar and Valocchi, 1993). In addition, the method often incorporates reactive terms that have been considered deterministic (e.g., Ahlstrom *et al.*, 1977; Dougherty, 1991) where the mass associated with the particles is allowed to change

with time, and stochastic (e.g., Kinzelbach, 1988; Valocchi and Quinodoz, 1989) where the number of particles, not their mass, changes with time.

Reported advantages of the RWPM include (Ahlstrom *et al.*, 1977; Ackerer, 1988; Kinzelbach, 1988; Józsa, 1989; Tompson and Gelhar, 1990; Dimou and Adams, 1993): (1) inherent stability; (2) absence of cumulative numerical dispersion; (3) easy extension to higher dimensional problems; (4) easy handling of complex geometry; (5) high degree of parallelism; and (6) more realistic (natural) representation of the occurring processes. Disadvantages of the RWPM are (Ahlstrom *et al.*, 1977; Allen, 1982; Kinzelbach, 1988): (1) greater computational resources compared to traditional numerical methods with the possible exception of some three-dimensional problems; (2) random noise associated with model results; and (3) model accuracy dependency on the number of particles while a general criterion defining their optimum number is still to be developed.

2.1.2 Cellular Automata

Computational resources have suggested new approaches to the modeling of systems. Digital computing devices are finite and discrete in nature, and their potential can be best realized when applied to discrete dynamic systems (Fogelman *et al.*, 1987). Moreover, many physical phenomena can be better viewed as discrete dynamic systems, in contrast to their traditional continuous representation using differential equations. Various discrete dynamic modeling approaches have been used with some success (Fogelman *et al.*, 1987). Cellular automata appear to be one of those methodologies with an increasingly important role for conceptual and practical modeling of discrete dynamic systems (Toffoli and Margolus, 1987).

The theory of CA was first introduced by John von Neumann in the late 1940's (von Neumann, 1966). This concept gained popularity three decades later through John Conway's work in the game of Life (Fogelman *et al.*, 1987; Toffoli and Margolus, 1987).

Cellular automata can be defined as dynamic systems in which space, time, and the dependent variable are all discrete quantities. In addition, the dependent variable is typically represented as a finite discretization, i.e., through a small set of possible values or cell states (Boghosian, 1990). Cellular automata are therefore based on a discrete lattice of cells. Each cell state evolves in discrete time steps according to deterministic or stochastic transition rules that depend only on the cell states of a local neighborhood of cells (Wolfram, 1984; Zeigler, 1984; Toffoli and Margolus, 1987; Boghosian, 1990). The CA configuration at the next time step ($t + \Delta t$) is the result of simultaneously applying the transition rule to all cell neighborhoods, using the cell states corresponding to the present time step (t). Thus the update of the cell states uses an explicit scheme. Since the transition rules are based on local or microscopic behavior, they are generally quite simple. However, the resulting overall behavior of the system can appear to be quite complex.

Examples of one and two-dimensional CA are shown in Figure 2.1. Each cell has two possible states (black and white), and the local neighborhood of a cell is defined by two adjacent neighbor cells. The transition rule simply specifies that the state of a cell at time $t + \Delta t$ is equal to the state of its two neighbors at time t if these have the same state; otherwise the state of a cell will remain unchanged.

Cellular automata as a modeling tool have been viewed as both (1) an alternative to floating-point based numerical methods for the solution of partial differential equations, and (2) as a complete modeling tool and an alternative to partial differential equations (Boghosian, 1990).

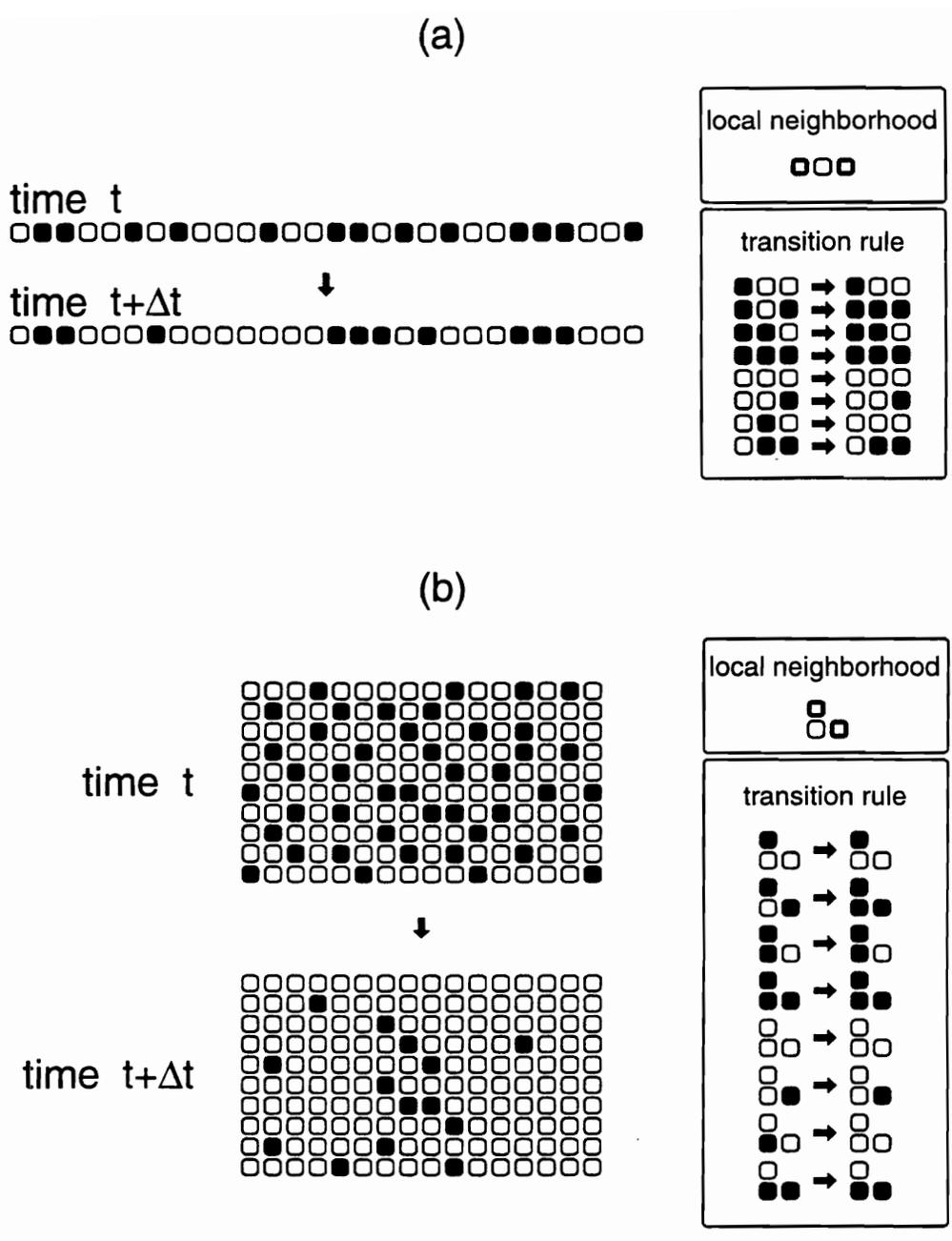


Figure 2.1 Cellular automata evolving in a discrete time interval (Δt). (a) One-dimensional cellular automaton. (b) Two-dimensional cellular automaton.

A special type of CA known as lattice gases are formed by a lattice on which particles are allowed to move, with resulting collisions occurring between particles (Boghosian, 1990). Lattice gases are known for their ability to simulate dynamic systems characterized by conserved quantities (Boghosian, 1990). They have been used for the solution of several problems including the Navier-Stokes and the diffusion equations (Eli, 1987; Doolen *et al.*, 1990; Rothman, 1990; Bernardin *et al.*, 1991; Boon, 1991; Chen *et al.*, 1991; Cliffe *et al.*, 1991; Kong and Cohen, 1991; Kougias *et al.*, 1991; Fox *et al.*, 1994).

The CA modeling approach appears to have the following potential benefits: (1) it can provide a better representation of the real physical system by bridging the differences between macroscopic and microscopic representations; (2) it does not involve any power series truncation; (3) in certain implementations it is not subject to round-off; (4) it is easily extended from a one-dimensional to a higher dimensional representation; (5) it is easily implemented using parallel processors to decrease execution time; and (6) since CA models can be based on simple, microscopic behavior, the focus of water quality modeling can be placed on the water quality mechanisms and not on the numerical solution technique. In addition, while comparing finite elements and CA methods for the computation of drag coefficients, Duarte and Brosa (1990) point out the superiority of the CA approach in terms of numerical stability and easy incorporation of boundary conditions.

In this research the development of the CA methodology was driven by a problem-specific approach. The rules were constructed based on a macroscopic view of the system, to solve specific water quality modeling issues. This allowed the development of CA rules based on the RWPM and, ultimately, on a differential equation representation of the phenomena. Therefore, the resulting CA model incorporates the same model coefficients typical of water quality models derived from differential equations. This

approach brings the CA model presented here close to traditional modeling techniques. This approach, however, does not constitute the only way CA can be used as a modeling tool. A more fundamental approach can be pursued based on a microscopic representation of the system. The CA rules can be defined at a microscopic level, possibly as very simple rules leading to complex macroscopic behavior. These simple rules should translate the fundamental physico-chemical laws governing the system. The microscopic approach may be more useful for improving fundamental understanding of water quality processes, but would likely be difficult to apply to site specific problems.

The remainder of this chapter deals with the methodology used to develop and test the CA representation for the RWPM, in particular for the advection, dispersion, and decay processes. Some other aspects of the development of a water quality model based on the CA approach are also discussed.

2.2 ADVECTION

In this section an approach is developed which represents the advection component of the RWPM using CA. The approach was tested for its ability to accurately represent the advection process.

2.2.1 Methodology

2.2.1.1 Rule Definition

The following discussion assumes advection along the longitudinal direction of flow. The CA is then defined as a line of cells in the longitudinal direction. In this section, as well as in the dispersion and decay sections, each cell is considered to have a finite (zero or more) number of particles, with each particle representing a fixed mass of a water quality constituent. The advection process is represented through a probability of a

particle to move to the next cell in the direction of flow during a simulation time step. The advection probability, P_{adv} , is defined as:

$$P_{adv} = \frac{u\Delta t_{adv}}{\Delta x} \quad 0 \leq P_{adv} \leq 1 \quad (2.1)$$

where u is the advective velocity ($L T^{-1}$), Δt_{adv} is the time step for the advection process (T), and Δx is the cell size in the longitudinal direction (L). A careful choice of values for Δt_{adv} and Δx can assure that $P_{adv} \leq 1$.

For each particle in each cell, a uniformly distributed random number, r , between 0 and 1 is generated and compared with the value of advection probability. If r does not exceed P_{adv} then the particle moves to the adjacent cell along the flow direction; otherwise the particle stays in the original cell.

2.2.1.2 Advection Induced Numerical Dispersion

As indicated above the choice of values for Δt_{adv} and Δx can be used to guarantee that the advection probability does not exceed one. When the advective velocity is invariant in time and space, a single set of values for Δt_{adv} and Δx can be used which guarantee that P_{adv} is exactly one. When the advective velocity is invariant in time, although varying in space along different reaches (assuming each reach has a uniform velocity), P_{adv} can still be made equal to one. This implies selecting a single global value for Δt_{adv} , and several values for Δx , each being a function of the velocity in a particular reach.

However, in many situations the advective velocity is expected to vary in time. In this case it is no longer possible to guarantee that P_{adv} stays equal to one without varying the value(s) of Δx over time as well. Updating the value(s) of Δx as the simulation progresses leads to a constant redefinition of the simulation grid. Although a simple task

for the one-dimensional case, it could be an untractable and computational intensive task for a large three-dimensional grid.

The motivation for restricting the value of P_{adv} to be equal to one is due to the relation between P_{adv} and numerical dispersion. When P_{adv} is exactly one, all the particles in a cell are advected to a neighbor cell, thus moving by a distance $\Delta x = u \Delta t_{adv}$. Therefore, every particle moves exactly the distance it is supposed to move. As soon as P_{adv} is smaller than one, although on average the particles in a cell move by a distance $u \Delta t_{adv}$, only a fraction of them (given by P_{adv}) is actually displaced. This fraction of the particles moves a distance $\Delta x > u \Delta t_{adv}$, while the remaining fraction (represented by $1 - P_{adv}$) remains in the original cell. The overall result is that some particles move faster while others move slower than the real velocity, thus leading to numerical dispersion.

To evaluate the magnitude of this numerical dispersion a surrogate method based on an exact probabilistic approach was used to simulate the behavior of the CA advection rule. The reason for using this surrogate method was that it represents the basic CA behavior without the random variability associated with the use of random numbers which makes it more difficult to determine trends in model behavior.

In this approach, an initial amount of mass (M^0) of a conservative constituent is introduced into the first cell of a one-dimensional system at the beginning of the simulation to represent an instantaneous discharge. Then a fraction (given by P_{adv}) of the mass present in each cell is moved to the adjacent downstream cell while the remaining fraction ($1 - P_{adv}$) stays in the original cell. This process is repeated for each simulation time step. The evolution of the mass distribution among the different cells as the simulation progresses was used to evaluate the magnitude of numerical dispersion.

The simulation was performed for several values of P_{adv} . At each simulation step, the mean (m') and standard deviation (s') of the mass distribution were calculated as:

$$m' = \sum_{i=1}^N i f(i) \quad (2.2a)$$

$$s' = \sqrt{\sum_{i=1}^N ((i - m')^2 f(i))} \quad (2.2b)$$

where $f(i)$ is the mass present in cell i as a fraction of the total mass (M^0), and N is the total number of cells in the system. Note that i is inside the summation terms in these expressions since i represents the x -coordinate in terms of cells.

The values of m' and s' are thus expressed in terms of cells and were then converted to a mean (m) and standard deviation (s) in units of distance (L), using the expressions:

$$m = m' \Delta x \quad (2.3a)$$

$$s = s' \Delta x. \quad (2.3b)$$

Several values for Δx and u were selected. From equation (2.1), and using a set of values for Δx and u , a value for Δt_{adv} was obtained for each value of advection probability. Knowing the value of Δt_{adv} , the evolution of the mean m and standard deviation s as a function of the number of simulation steps n can then be expressed as a function of time. The objective was to use those relations between s and time to quantify numerical dispersion and then evaluate any dependency of numerical dispersion on P_{adv} .

2.2.2 Results and Discussion

Figure 2.2 shows these relations for several values of P_{adv} , and for the case of $\Delta x = 10$ and $u = 5$. The results of Figure 2.2(a) indicate that the mean is a function of ut , with the term $\Delta x/2$ being the result of the spatial discretization of the model. The results of Figure 2.2(b) clearly show a power-law relationship between the standard deviation

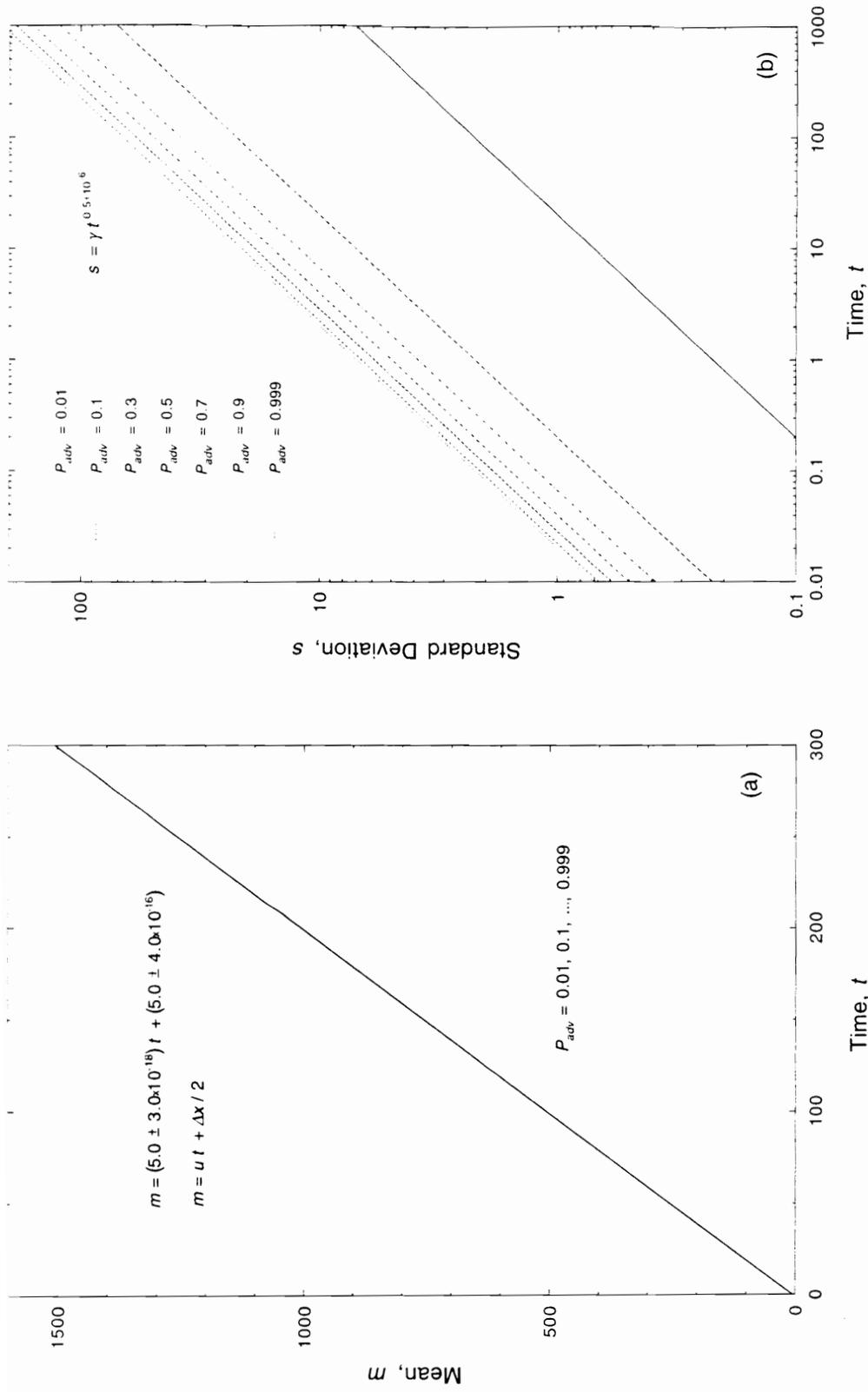


Figure 2.2 Time evolution of the mean (a) and standard deviation (b) of the mass distribution as a function of the advection probability using the CA surrogate method. ($\Delta x = 10$, $\mu = 5$.)

and time of the form $s = \gamma t^{0.5}$ for any of the values of P_{adv} . These results suggest the assumption (see also Appendix A) that the mass distribution follows a normal probability density function similar to the solution of the advection-dispersion differential equation for an instantaneous input (Thomann and Mueller, 1987). This implies $s = \sqrt{2E_{num} t}$, where E_{num} is a coefficient representing the advection induced numerical dispersion. This equation is a power-law identical to $s = \gamma t^{0.5}$ with $\gamma = \sqrt{2E_{num}}$. The values for E_{num} corresponding to the results in Figure 2.2(b) can then be obtained from $E_{num} = \gamma^2 / 2$.

Figure 2.3 shows an inverse linear relation between E_{num} and P_{adv} for different values of $u \Delta x$. The data points for $u \Delta x = 50$ were derived from Figure 2.2(b), and a similar approach was used for the other values of $u \Delta x$. Thus the relation between E_{num} and P_{adv} is dependent not on the single values for u and Δx , but on the product $u \Delta x$. Furthermore, as shown in Figure 2.4, a unique linear relation exists between the dimensionless quantity $E_{num}/(u\Delta x)$ and P_{adv} given as:

$$\frac{E_{num}}{u\Delta x} = 0.5 (1 - P_{adv}) \quad (2.4)$$

Substituting equation (2.1) for P_{adv} in equation (2.4) and rearranging leads to:

$$E_{num} = 0.5 (u\Delta x - u^2 \Delta t_{adv}) \quad (2.5)$$

Given that $P_{adv} \leq 1$, and assuming that Δt_{adv} and Δx are constants, equation (2.1) implies that a maximum value for the advective velocity (u_{max}) exists for which $P_{adv} = 1$ and equation (2.1) reduces to:

$$\frac{u_{max} \Delta t_{adv}}{\Delta x} = 1 \quad (2.6)$$

Equation (2.6) can be rewritten as:

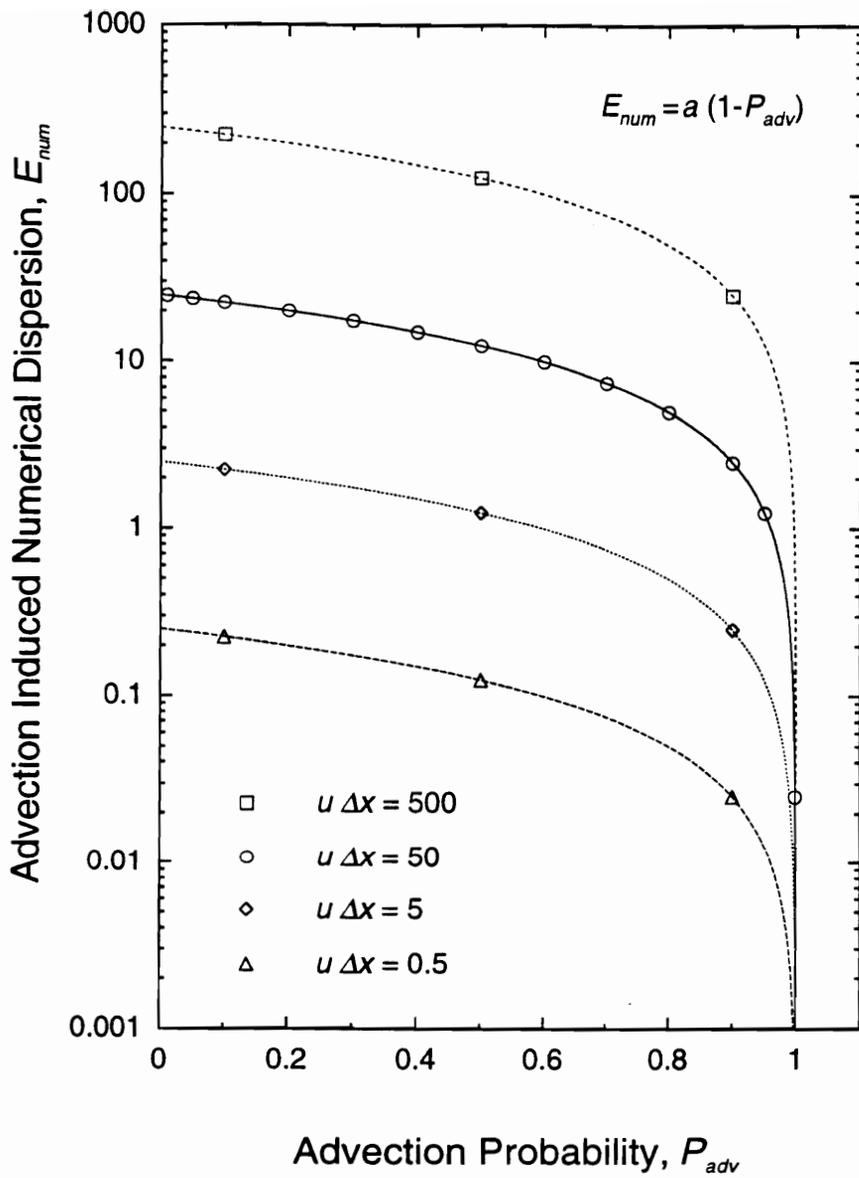


Figure 2.3 Relation between the advection induced numerical dispersion (E_{num}) and the advection probability.

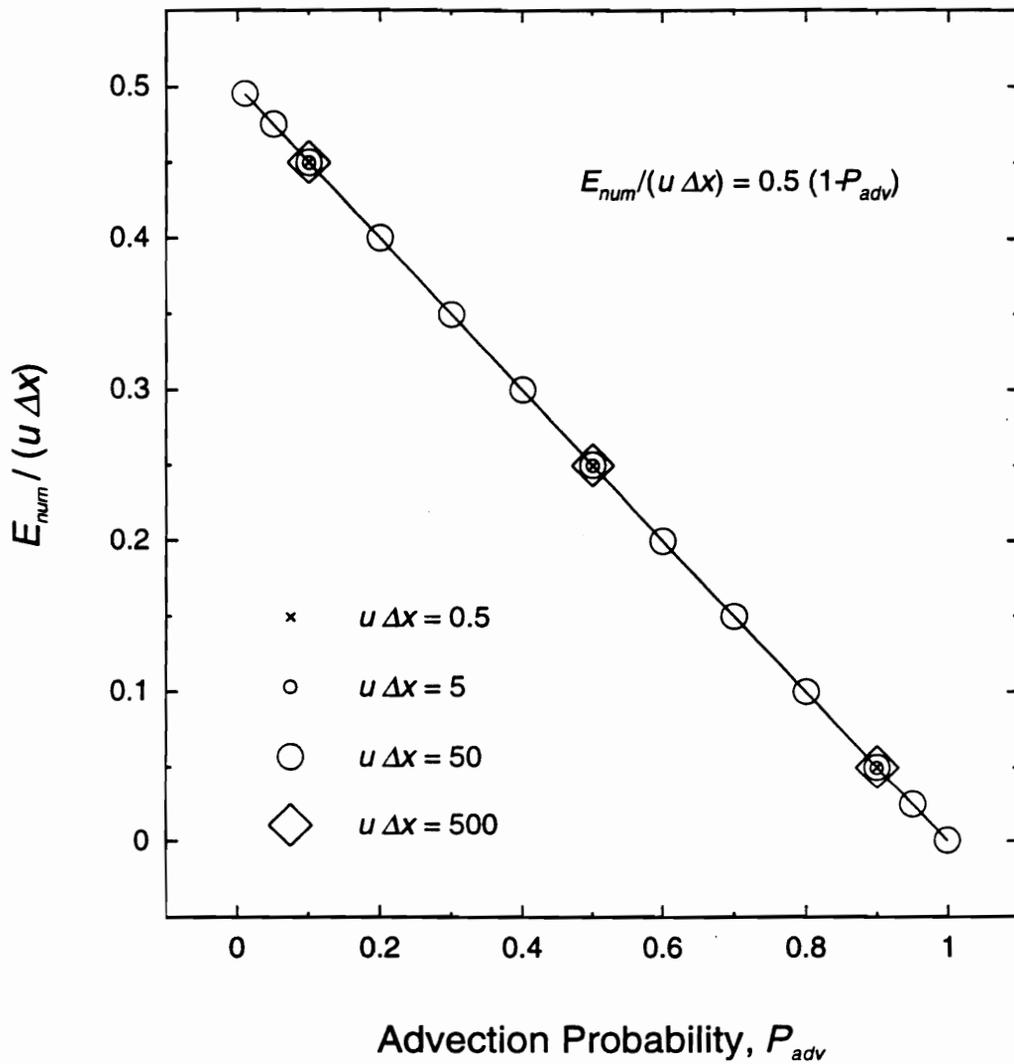


Figure 2.4 Relation between $E_{num}/(u\Delta x)$ and the advection probability.

$$\Delta t_{adv} = \frac{\Delta x}{u_{max}}. \quad (2.7)$$

Substituting equation (2.7) for Δt_{adv} in equation (2.5) and rearranging one obtains:

$$E_{num} = 0.5u\Delta x \left(1 - \frac{u}{u_{max}} \right). \quad (2.8)$$

As u varies from zero to u_{max} , E_{num} increases from zero to a maximum value and then decreases back to zero. In the present discussion of advection it has been assumed from the beginning that u is a non-negative quantity since the flow follows the positive direction. In a more general case in which u can be negative or positive, equation (2.8) still holds although u must then be replaced by $|u|$. The value of u for which E_{num} reaches its maximum is obtained from the solution to the equation $\frac{dE_{num}}{du} = 0$. That value is $u = 0.5u_{max}$. The expression for the maximum value of E_{num} is then obtained by substituting $u = 0.5u_{max}$ in equation (2.8), and is given by:

$$E_{num,max} = \frac{u_{max}\Delta x}{8}. \quad (2.9)$$

Equation (2.9) shows that the maximum value for the numerical dispersion is a direct function of both u_{max} and Δx . This equation can then be used to define a guideline for the selection of a maximum value for Δx given a particular maximum value to eventually be reached by the advective velocity (u_{max}), and a numerical dispersion value not to be exceeded ($E_{num,max}$). This value of $E_{num,max}$ can be defined in relative terms as a certain percentage of the minimum expected value for the longitudinal dispersion coefficient. The same considerations can be extended to the Δt_{adv} by combining equations (2.7) and (2.9).

Figure 2.5 illustrates the suggested guideline showing values of Δx_{max} (maximum value to be selected for Δx) for typical values of u_{max} and $E_{num,max}$ expected for rivers and estuaries. Figure 2.6 illustrates the same approach for the selection of the maximum advection time step. Recall that the derivations leading to equation (2.9) and thus to the relationships shown in Figures 2.5 and 2.6 required the assumption that equation (2.6) is always true. This means that the final values for Δx and Δt_{adv} obtained from equation (2.9) or Figures 2.5 and 2.6 have to obey the relationship imposed by equation (2.6).

The previous discussion indicates that although numerical dispersion cannot be neglected, it can be made insignificant in relation to the physical dispersion of the system. However, such an approach may require excessive spatial and temporal model resolutions in some cases, and thus substantially increase the required model computations. The situation is most severe in systems with a large maximum value for the advective velocity, and a small longitudinal dispersion.

2.3 DISPERSION

This section presents an approach based on CA for the representation of the dispersion component of the RWPM. The approach was tested for its ability to accurately represent the dispersion phenomena.

2.3.1 Methodology

2.3.1.1 Rule Definition

The following discussion assumes dispersion in the longitudinal direction of flow, although the same approach can be used to describe dispersion in other directions. In the RWPM a particle moves by a random amount of a maximum magnitude as a result of dispersion (Bear and Verruijt, 1987). Assuming a uniform distribution of the random

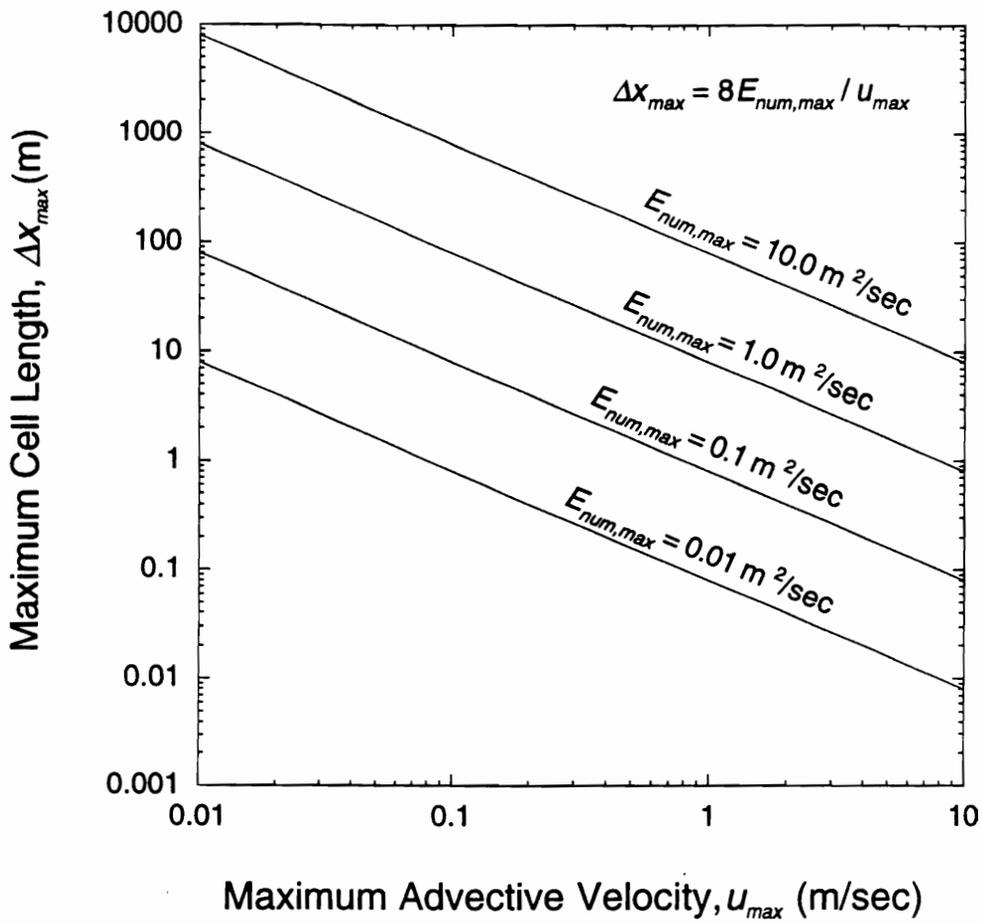


Figure 2.5 Guideline for the selection of Δx_{max} for typical values of u_{max} and $E_{num,max}$ for rivers and estuaries.

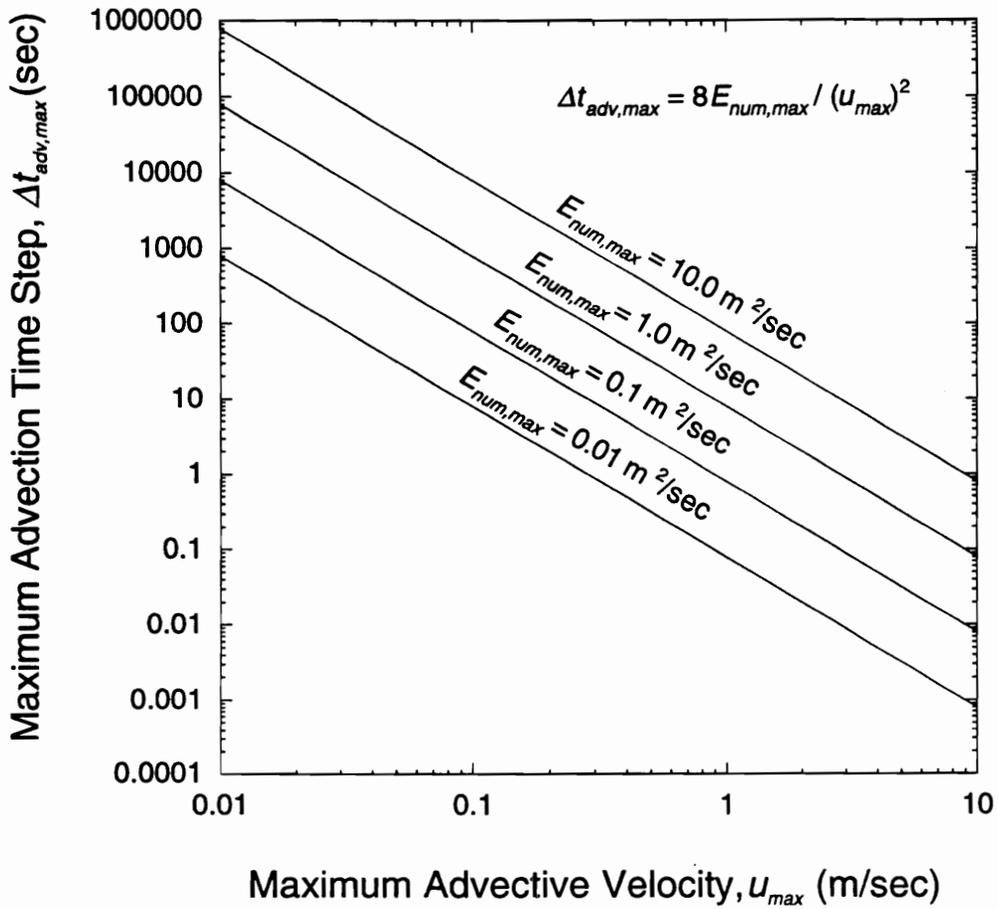


Figure 2.6 Guideline for the selection of $\Delta t_{adv,max}$ for typical values of u_{max} and $E_{num,max}$ for rivers and estuaries.

movement of a particle, the probability distribution of the particle movement for a large number of independent steps should follow a normal distribution. This probability distribution can be shown to be identical to the analytical solution of the one-dimensional dispersion differential equation for an instantaneous contaminant spill (Bear and Verruijt, 1987). As a result, the maximum magnitude (amplitude) of the random movement of a particle is a function of both the dispersion coefficient, E , and the simulation time step for the dispersion process, Δt_{dis} , and is given by $\sqrt{6E\Delta t_{dis}}$. Based on this conclusion, an expression for the dispersion probability needed for CA can be derived (see Appendix B for details).

The dispersion process is based on the probability of a particle to move from its present cell to the adjacent upstream or downstream cell during a simulation time step. The dispersion probability, P_{dis} , is defined as:

$$P_{dis} = (2q - 1)P_{dis.amp} \quad -P_{dis.amp} \leq P_{dis} \leq P_{dis.amp} \quad (2.10a)$$

$$P_{dis.amp} = \frac{\sqrt{6E\Delta t_{dis}}}{\Delta x} \quad 0 \leq P_{dis.amp} \leq 1 \quad (2.10b)$$

where $P_{dis.amp}$ is the amplitude of the dispersion probability, E is the longitudinal dispersion coefficient ($L^2 T^{-1}$), and q is a uniformly distributed random number between 0 and 1. A careful choice of values for Δt_{dis} and Δx assures that $P_{dis.amp} \leq 1$.

For each particle in a cell, a uniformly distributed random number, q , between 0 and 1 is generated and used to compute the dispersion probability. Then, a uniformly distributed random number, r , between 0 and 1 is generated and compared with the value of dispersion probability. If r does not exceed the absolute value of P_{dis} then the particle moves to the adjacent upstream cell (when P_{dis} is negative) or to the adjacent downstream cell (when P_{dis} is positive); otherwise the particle stays in the original cell.

2.3.1.2 Effect of P_{disamp} on Dispersion

As previously mentioned P_{disamp} varies from 0 to 1. When P_{disamp} is zero there is no dispersion. When dispersion occurs then a value is selected for P_{disamp} in the range $0 < P_{disamp} \leq 1$. However, in order to evaluate in what extent such selection affects simulation results an in depth analysis of the problem was pursued.

Due to the relative higher complexity of the dispersion rule, in comparison to advection and decay, it was not practical to develop a surrogate method using a non-random probabilistic approach capable of simulating the basic behavior of the CA rule. However, to reduce the effect of randomness associated with the CA dispersion rule on the simulation results, each simulation was performed several times and corresponding results averaged.

The simulated system consists of a line of cells. An amount of particles (N_p^0) was introduced into one of the cells at the beginning of the simulation to represent an instantaneous discharge. Then the particles were allowed to disperse upstream and downstream accordingly to the rule for dispersion, and using a particular value for P_{disamp} in the range $0 < P_{disamp} \leq 1$. This process was repeated for each simulation time step. The overall simulation was repeated one-hundred times to counteract the randomness associated with the dispersion rule.

The evolution of the distribution of the number of particles among the cells during the course of a simulation was used to characterize the respective dispersion. At each simulation step, the mean (m') and standard deviation (s') of the particle distribution was calculated as:

$$m' = \sum_{i=1}^N i f(i) \quad (2.11a)$$

$$s' = \sqrt{\sum_{i=1}^N ((i - m')^2 f(i))} \quad (2.11b)$$

where $f(i)$ is the number of particles present in cell i as a fraction of the total number of particles (N_p^0), and N is the total number of cells in the system.

For each simulation step, a total of 100 values for s' was obtained due to repetition of the simulation. The corresponding mean (s'_m) and standard deviation (s'_s) of s' were calculated as:

$$s'_m = \frac{1}{100} \sum_{j=1}^{100} s'_j \quad (2.12a)$$

$$s'_s = \sqrt{\frac{1}{99} \sum_{j=1}^{100} (s'_j - s'_m)^2}. \quad (2.12b)$$

The values of m' and s' , and therefore of s'_m and s'_s , are thus expressed in terms of cells. A value for s'_m was converted to a value, s , in units of distance (L), using the expression:

$$s = s'_m \Delta x. \quad (2.13)$$

A similar conversion was used for s'_s .

For a particular set of values for Δx and E , equation (2.10b) was used to evaluate different Δt_{dis} corresponding to different values for P_{disamp} . Based on a value for Δt_{dis} , the evolution of the standard deviation s as a function of the number of simulation steps n can then be expressed as a function of time. The idea was to use these relations between s and time as a measure of the impact of the values for P_{disamp} on the dispersion produced by the CA rule.

2.3.2 Results and Discussion

Figure 2.7 shows these relations for several values of P_{disamp} , and for the particular case of $N_p^0=1000$, $\Delta x=10$, and $E=1$. The vertical bars represent the standard deviation associated with s , i.e., $s \pm s' \Delta x$. The results clearly show a power-law relationship between the standard deviation and time of the form $s_{ca} = \alpha_{ca} t^{0.5}$ for any value of P_{disamp} . In fact, the slope of any P_{disamp} line varies from 0.5 by just 0.002 at the most. These results strongly suggest the validity of the assumption that the particle distribution follows a normal probability density function similar to the solution of the dispersion differential equation for an instantaneous input (Thomann and Mueller, 1987). In this analytical solution, the standard deviation of the distribution has the form $s_a = \sqrt{2Et}$. This expression is a power-law of the form $s_a = \alpha_a t^{0.5}$ with $\alpha_a = \sqrt{2E}$. This power-law is also plotted in Figure 2.7. The relation between the various P_{disamp} lines and the analytical solution suggests that the CA rule for dispersion overestimates the dispersion process and the overestimation increases as P_{disamp} decreases.

These results indicate that some numerical dispersion is associated with the CA rule for dispersion. In contrast to the RWPM in which particles move in a continuous path by exact increments thus precluding numerical dispersion, in CA particles move by finite increments therefore introducing numerical dispersion. Such numerical dispersion occurs since some particles tend to spread faster than it would be expected from the dispersion coefficient alone. Since P_{dis} allows values in the full range $0 < |P_{dis}| \leq P_{disamp}$, faster spreading of particles will occur, even when $P_{disamp}=1$ (as seen in Figure 2.7).

To quantify the numerical dispersion, a ratio $(\alpha_{ca}/\alpha_a)^2$ was evaluated as a function of P_{disamp} using several plots similar to Figure 2.7. In those plots several combinations of values for E , Δx , and N_p^0 were considered. A summary of these results is depicted in Figure 2.8. This figure shows the existence of a power-law relationship

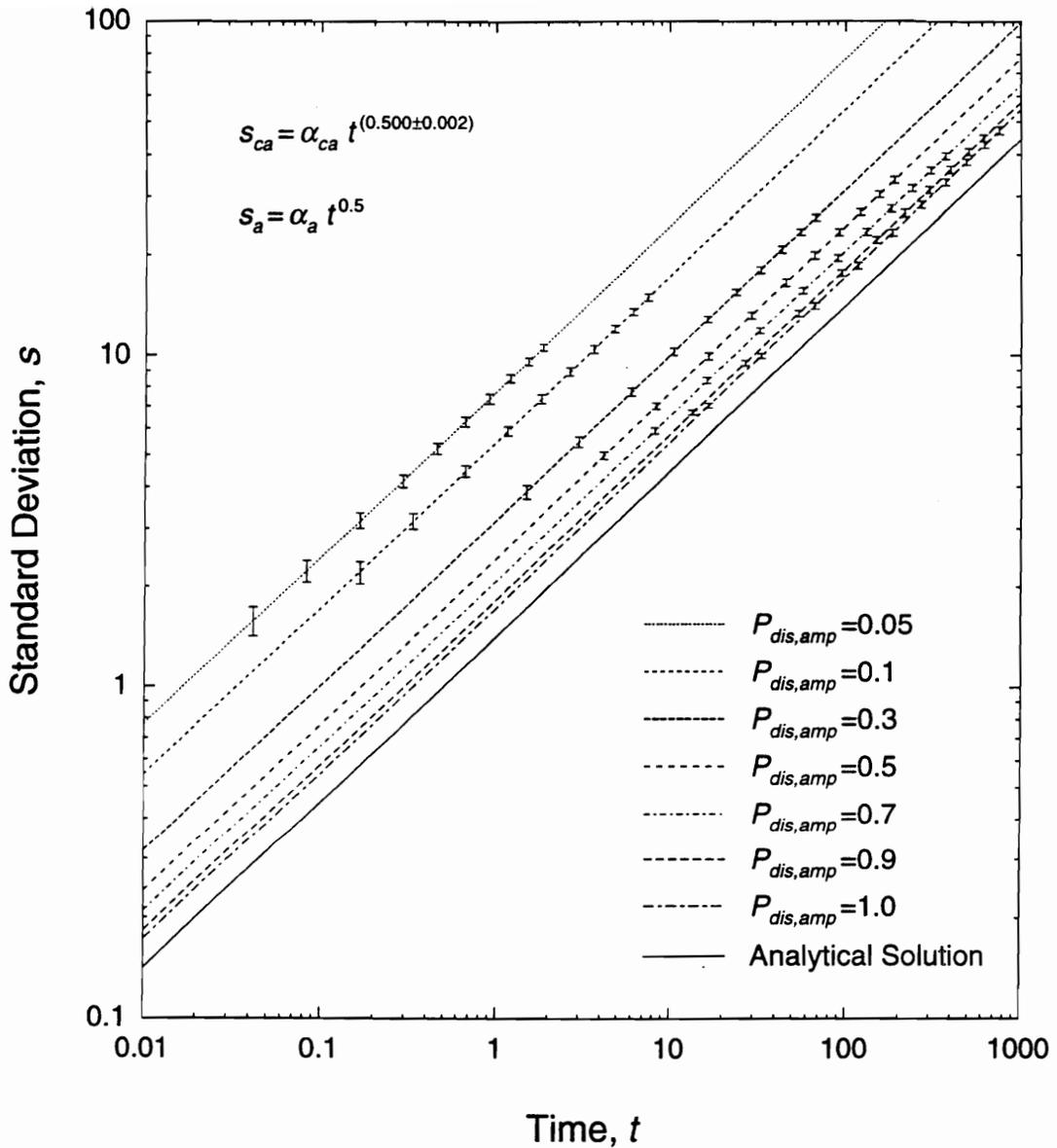


Figure 2.7 Time evolution of the standard deviation of the distribution of number of particles as a function of $P_{dis,amp}$ using the CA dispersion rule, and comparison with the analytical solution for the dispersion differential equation for an instantaneous input. ($N_p^0 = 1000$, $\Delta x = 10$, $E = 1$.)

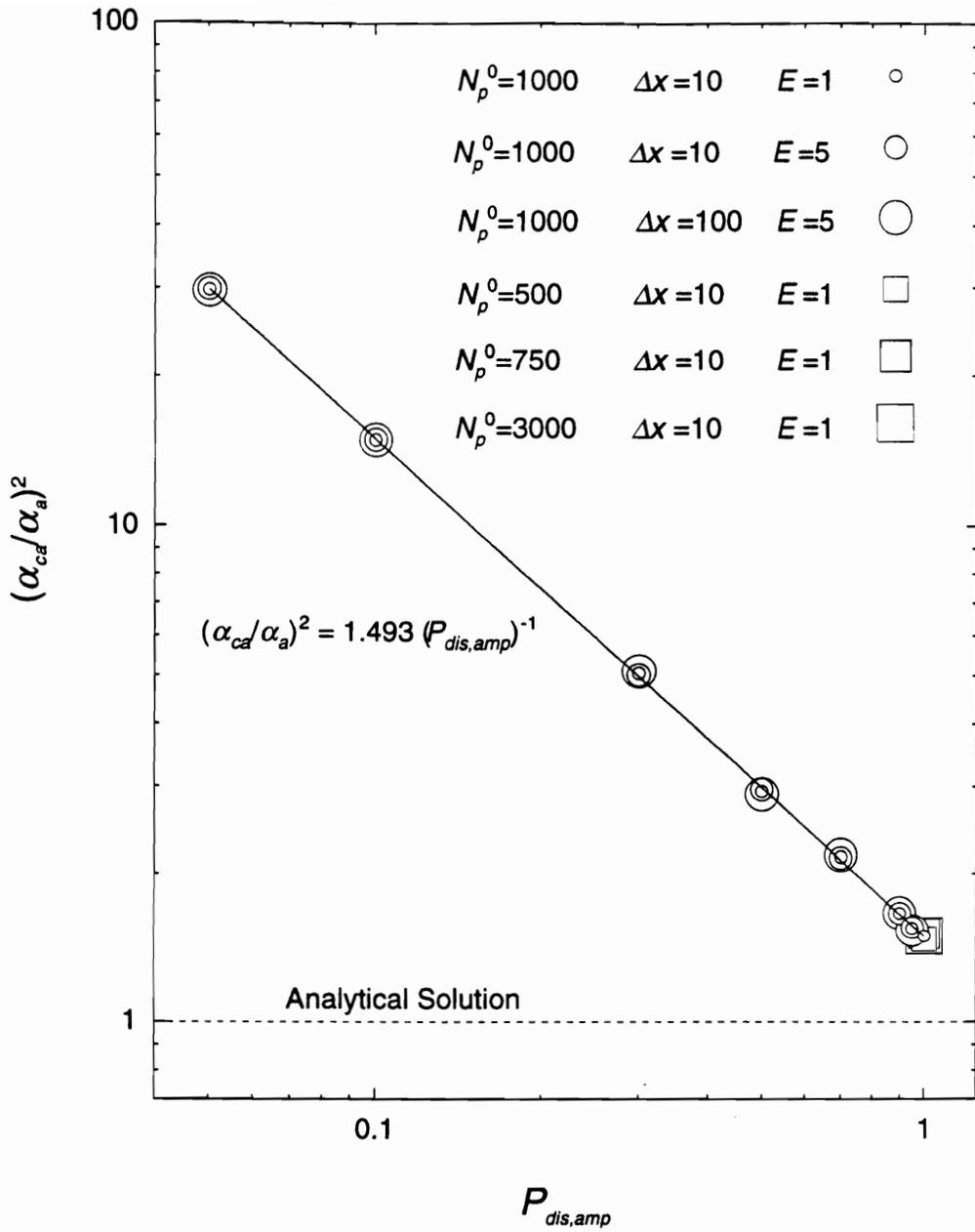


Figure 2.8 Relation between $(\alpha_{ca}/\alpha_a)^2$ and $P_{dis,amp}$.

between the ratio $(\alpha_{ca}/\alpha_a)^2$ and P_{disamp} . Furthermore, this relation appears to be completely independent of E , Δx , and N_p^0 , and is given as:

$$(\alpha_{ca}/\alpha_a)^2 = \frac{1.493}{P_{disamp}}. \quad (2.14)$$

In relation to Figure 2.7 it was previously indicated that $\alpha_a = \sqrt{2E}$. This expression can be rewritten as $\alpha_a = \sqrt{2E_a}$ to emphasize that the analytical solution represents the dispersion associated with a coefficient E_a . Since the CA rule also represents a dispersion process but of larger magnitude, it can be assumed that $\alpha_{ca} = \sqrt{2E_{ca}}$, where $E_{ca} > E_a$. From this it follows that

$$\left(\frac{\alpha_{ca}}{\alpha_a}\right)^2 = \frac{E_{ca}}{E_a}. \quad (2.15)$$

The numerical dispersion associated with the dispersion rule is by definition $E_{ca} - E_a$, and is always larger than zero. Also, equation (2.15) implies that

$$E_{ca} - E_a = E_a \left(\left(\frac{\alpha_{ca}}{\alpha_a} \right)^2 - 1 \right). \quad (2.16)$$

Substitution of equation (2.14) in equation (2.16) leads to:

$$E_{ca} - E_a = E_a \left(\frac{1.493}{P_{disamp}} - 1 \right). \quad (2.17)$$

This expression shows that for the best case ($P_{disamp}=1$) numerical dispersion still leads to an overestimation of dispersion by 49.3%. When P_{disamp} is 0.5 and 0.1, the numerical dispersion is about 2 and 14 times the actual dispersion, respectively.

An important finding from the previous result is that the CA rule for dispersion simulates the dispersion process corresponding to a dispersion coefficient somewhat

larger than the one intended to be represented. Based on this conclusion a procedure can be developed to counteract this problem.

Consider that the dispersion coefficient E represents the actual dispersion to be simulated by the CA rule. Then, and based on equation (2.15), the dispersion coefficient to be used in the dispersion rule should be a corrected value, E^c , given as:

$$E^c = \left(\frac{\alpha_a}{\alpha_{ca}} \right)^2 E. \quad (2.18)$$

This relation implies that $E^c < E$. Substituting equation (2.14) into equation (2.18), one obtains:

$$E^c = \left(\frac{P_{dis.amp}}{1.493} \right) E. \quad (2.19)$$

Substitution of $P_{dis.amp}$ from equation (2.10b) into equation (2.19) leads to:

$$E^c = \frac{E\sqrt{6E^c\Delta t_{dis}}}{1.493\Delta x}. \quad (2.20)$$

Solving equation (2.20) for E^c yields:

$$E^c = \frac{6E^2\Delta t_{dis}}{(1.493\Delta x)^2}. \quad (2.21)$$

Then, substituting E^c in equation (2.21) by equation (2.19) and canceling terms one obtains:

$$P_{dis.amp} = \frac{6E\Delta t_{dis}}{1.493(\Delta x)^2}. \quad (2.22)$$

This equation can then be used in place of equation (2.10b) to calculate $P_{dis.amp}$.

Assuming that in a typical situation both a maximum value for E , represented as E_{max} , and Δx are known quantities, the largest possible value for Δt_{dis} , denoted $\Delta t_{dis,max}$, is obtained from equation (2.22) with $P_{dis,amp}=1$ (the upper limit for $P_{dis,amp}$). The value for $\Delta t_{dis,max}$ is therefore given as:

$$\Delta t_{dis,max} = \frac{1.493(\Delta x)^2}{6E_{max}}. \quad (2.23)$$

The derivation followed in this section seems to point out that any value $\Delta t_{dis} \leq \Delta t_{dis,max}$ can be chosen without affecting the accuracy of the simulation. In other words, the dispersion rule is now expected to be identically accurate regardless of the value for $P_{dis,amp}$ in the range $0 < P_{dis,amp} \leq 1$.

To verify this claim, the results of some of the previous simulations were re-analyzed to reflect the new definition for $P_{dis,amp}$ as given by equation (2.22), which corrects for numerical dispersion. The results are presented in Figure 2.9, for $N_p^0=1000$, $\Delta x=10$, and $E=5$. Comparison of these results with previous ones (see Figure 2.7) strongly suggests that numerical dispersion is no longer associated with the dispersion rule. It also suggests that an equally good representation of the analytical solution is obtained regardless of the value for $P_{dis,amp}$. (The standard deviation associated with the standard deviation of the distributions is at approximately the same magnitude as the plot symbols.)

So far, the analysis of the results has been based solely on the standard deviation of the particle distributions. Thus, to provide a more complete discussion the particle distributions (which can be easily viewed as mass distributions, since particles of uniform mass are assumed) were analyzed and compared with the corresponding distributions associated with the analytical solution of the dispersion differential equation for an

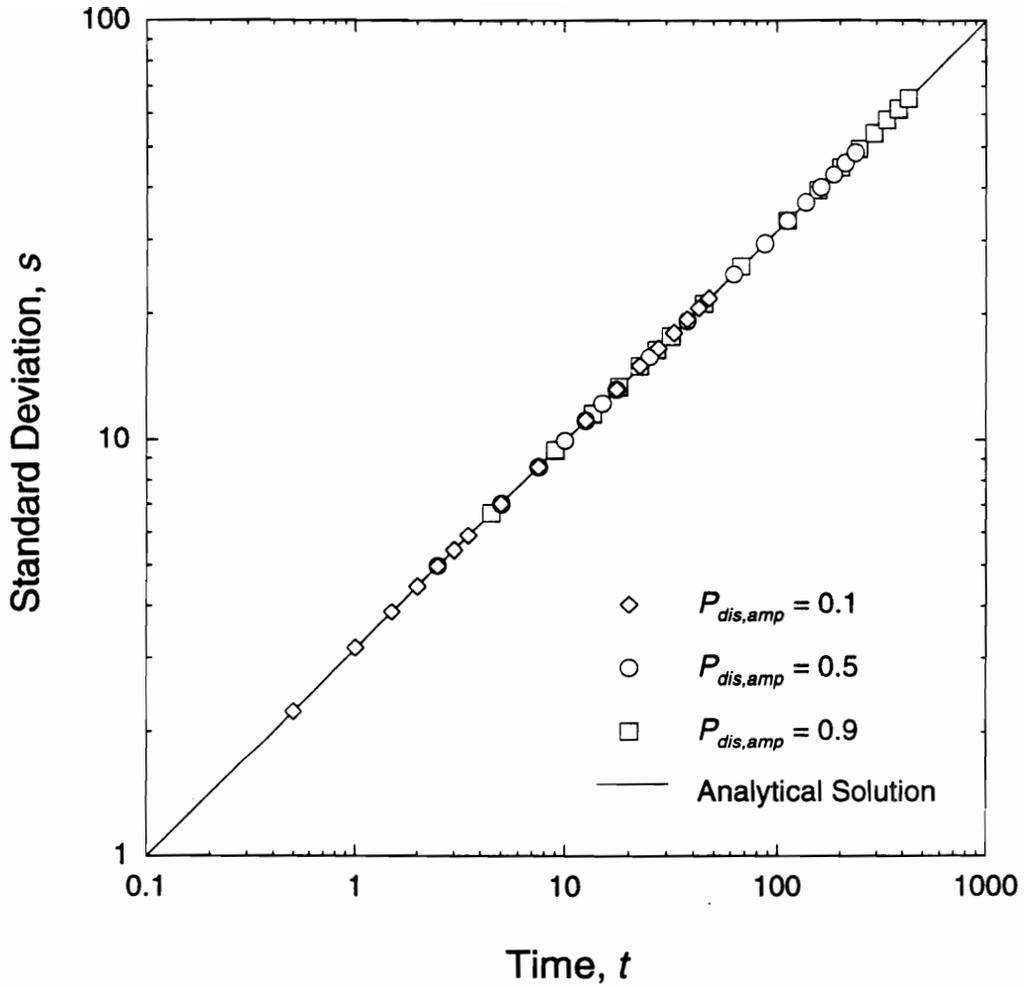


Figure 2.9 Time evolution of the standard deviation of the distribution of number of particles as a function of $P_{dis,amp}$ using the CA dispersion rule corrected for numerical dispersion and comparison with the analytical solution. ($N_p^0=1000$, $\Delta x=10$, $E=5$.)

instantaneous input (Thomann and Mueller, 1987). The mass present in each cell, as a fraction of the total mass, is:

$$\frac{M_i}{M^0} = \frac{1}{2\sqrt{\pi Et}} \int_{x_i - \frac{\Delta x}{2}}^{x_i + \frac{\Delta x}{2}} e^{-\frac{x^2}{4Et}} dx \quad (2.24a)$$

$$= \frac{1}{2} \left[\operatorname{erf} \left(\frac{2x_i + \Delta x}{4\sqrt{Et}} \right) - \operatorname{erf} \left(\frac{2x_i - \Delta x}{4\sqrt{Et}} \right) \right] \quad (2.24b)$$

where M_i/M^0 represents the fraction of the total mass present in cell i , with i being any integer ($\dots, -2, -1, 0, 1, 2, \dots$). The total mass M^0 is considered to be instantaneously introduced in cell $i = 0$, at time $t = 0$. The x -coordinate of each cell i is defined as the mid-point of the cell and is given by $x_i = i \Delta x$. The above equation is then typically evaluated for particular values of t corresponding to the product $n \Delta t_{dis}$.

Figure 2.10 compares CA simulation results using both definitions for P_{disamp} , given by equations (2.22) and (2.10b), with the analytical solution (equation (2.24b)). Natural cubic spline interpolation is used to draw the line for the analytical solution based on the discrete values of the cells. This results refer to the particular case of $N_p^0 = 1000$, $\Delta x = 10$, $E = 5$, and $t \approx 50$. Each CA distribution is the mean distribution for the 100 simulation repetitions. The vertical bars show the magnitude of the corresponding standard deviation.

Figure 2.10 supports previous results suggesting a very good agreement between the behavior of the dispersion rule, when corrected for numerical dispersion, and the analytical solution for different values of P_{disamp} . This figure also confirms previous findings of an overestimation of the dispersion process (which becomes more pronounced as P_{disamp} decreases) by the CA dispersion rule without a correction for numerical

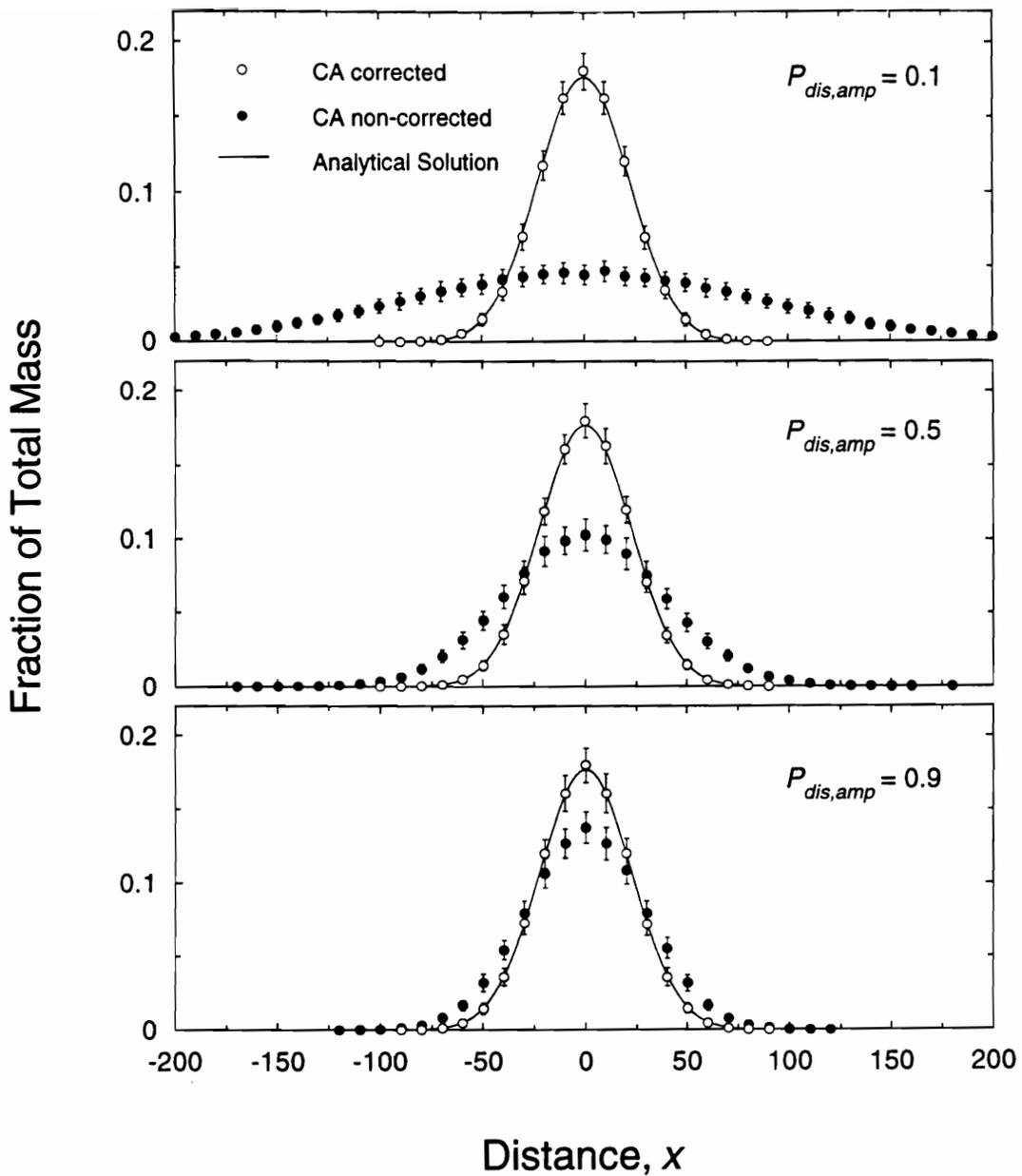


Figure 2.10 Comparison of mass distributions for the CA with dispersion rule corrected and non-corrected for numerical dispersion and the analytical solution as a function of $P_{dis,amp}$. ($N_p^0 = 1000$, $\Delta x = 10$, $E = 5$, $t \approx 50$.)

dispersion. Furthermore, it supports an earlier assumption that the particle distributions resulting from the CA dispersion rule follow a normal distribution.

Another way to compare the CA results with the analytical solution is provided by the time evolution of the distribution peaks. From equation (2.24a), the peak predicted by the analytical solution is simply the fraction of total mass for the cell $i=0$, thus $x_i=x_0=0$, which is given as:

$$\frac{M_0}{M^0} = \text{erf} \left(\frac{\Delta x}{4\sqrt{Et}} \right). \quad (2.25)$$

Figure 2.11 shows the comparative time evolution of the peak fraction of total mass for the CA and analytical solution, for $N_p^0=1000$, $\Delta x=10$, and $E=5$. These results indicate again a significant improvement on the performance of the CA dispersion rule when correction for numerical dispersion is included. After an initial short time (Figure 2.11(b)) the CA results agree very well with the analytical solution.

2.4 DECAY

In this section an approach is introduced to represent the decay component of the RWPM under the CA framework. The approach was tested for its ability to accurately represent the decay process.

2.4.1 Methodology

2.4.1.1 Rule Definition

Decay processes usually follow first-order kinetics or negative exponential functions. Toffoli and Margolus (1987) present an illustrative example on how to model exponential decay phenomena using CA. The decay process is considered through a

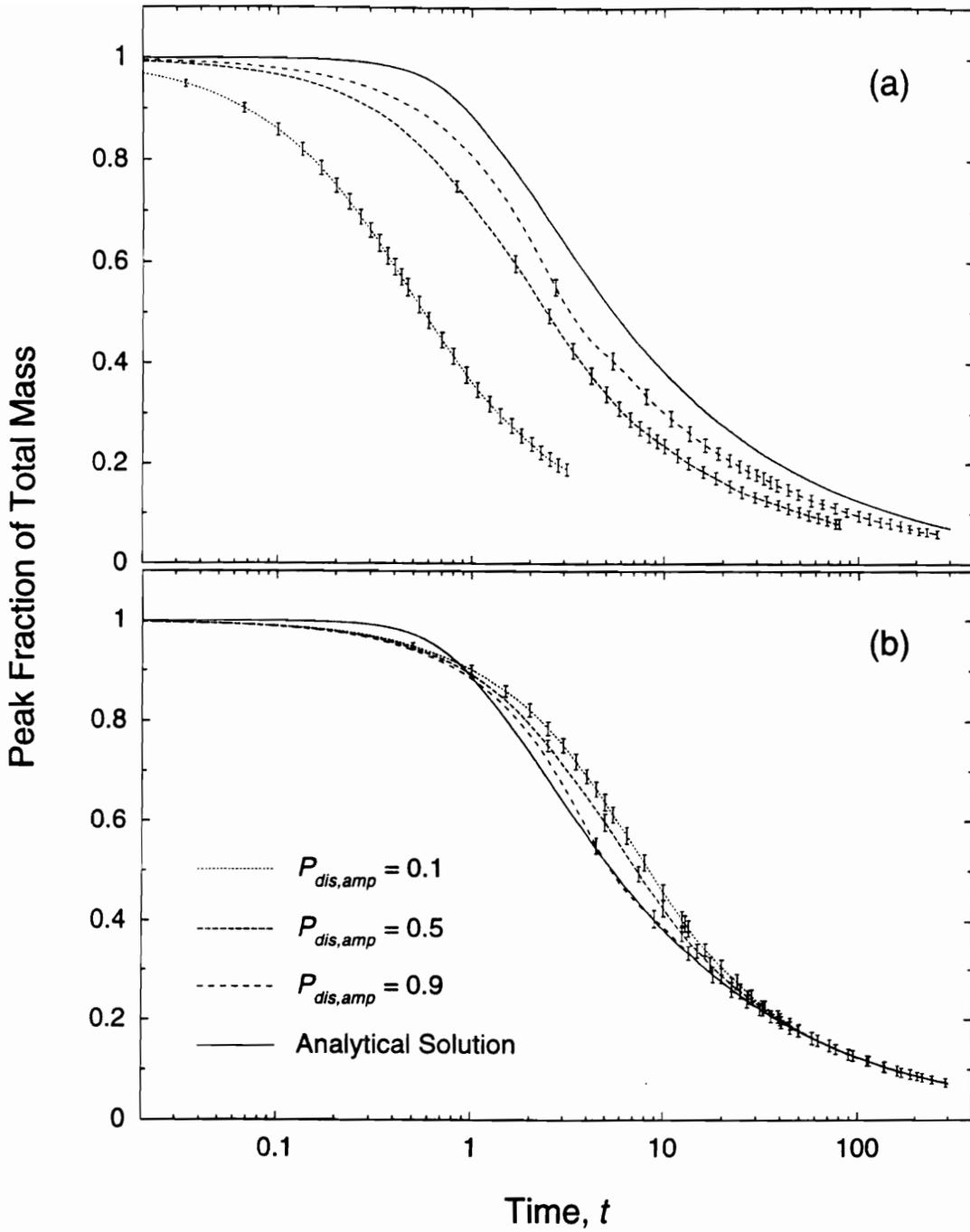


Figure 2.11 Time evolution of the peak of mass distributions for the CA and the analytical solution as a function of $P_{dis,amp}$: (a) dispersion rule without correction for numerical dispersion; (b) with correction. ($N_p^0=1000$, $\Delta x=10$, $E=5$.)

probability of a particle to be removed from a cell during a simulation time step. The first-order decay probability, P_{dec} , is given as:

$$P_{dec} = k_{dec} \Delta t_{dec} \quad 0 \leq P_{dec} \leq 1 \quad (2.26)$$

where k_{dec} is the first-order decay rate constant (T^{-1}), and Δt_{dec} is the time step for the decay process. The value of $P_{dec} \leq 1$.

For each particle in a cell, a uniformly distributed random number, r , between 0 and 1 is generated and compared with the decay probability. If r does not exceed P_{dec} then the particle is removed from its cell (and from the system); otherwise the particle stays in its cell.

In the context of random-walk, Kinzelbach (1988) presents an identical decay probability in which the time step should be made small enough to always keep the probability value less than one. Valocchi and Quinodoz (1989) using a similarly defined probability but in the context of first-order adsorption within random-walk, further suggest that the probability should be substantially less than one. The question then arises as to how much to constrain the value of Δt_{dec} , and therefore the resulting value of P_{dec} .

2.4.1.2 Effect of P_{dec} on Decay

Since the CA decay rule uses random numbers to simulate the behavior of the particles, such an approach introduces some random variability in the results making it more difficult to determine trends in model behavior. To overcome this difficulty, a CA surrogate method based on an exact probabilistic approach was used.

Given an initial mass of a nonconservative constituent, this approach simply keeps track of the mass remaining at successive simulation time steps. At each simulation step, a fraction of the mass remaining from the previous step is removed. The remaining

mass is then available for loss at the next simulation step. The fraction of mass removed at each step is represented by the decay probability.

The mass remaining after a certain number of simulation steps is given as:

$$M^n = M^0 \left[1 - P_{dec} \left(1 + \sum_{i=1}^{n-1} (1 - P_{dec})^i \right) \right] \quad (2.27)$$

where M^n is the remaining mass after n simulation time steps and M^0 is the initial mass.

Expression (2.27) was evaluated as a function of the number of simulation steps for different values of P_{dec} . The mass remaining was expressed as a fraction of the initial mass, i. e., as M^n/M^0 . The fraction of mass remaining as a function of the number of simulation steps was then expressed as a function of time given a particular value of first-order k_{dec} . Due to the relation imposed by equation (2.26) this implies using different Δt_{dec} values, each associated with a particular probability P_{dec} . These relations between fraction of mass remaining and time were then used to quantify the decay and its dependency on P_{dec} .

2.4.2 Results and Discussion

Figure 2.12 shows remaining mass versus time for $k_{dec} = 0.5$. The results for different decay probability values are compared with the analytical solution for first-order decay given by:

$$M^t = M^0 e^{(-k_{dec}t)} \quad (2.28)$$

where M^t is the remaining mass at time t .

The relation between the slopes of the lines plotted in Figure 2.12 seems to suggest that the CA surrogate method consistently overestimates the true (analytical) decay rate, k_{dec} . This means that a decay rate $k_{dec}^{ca} > k_{dec}$ is being represented by the CA

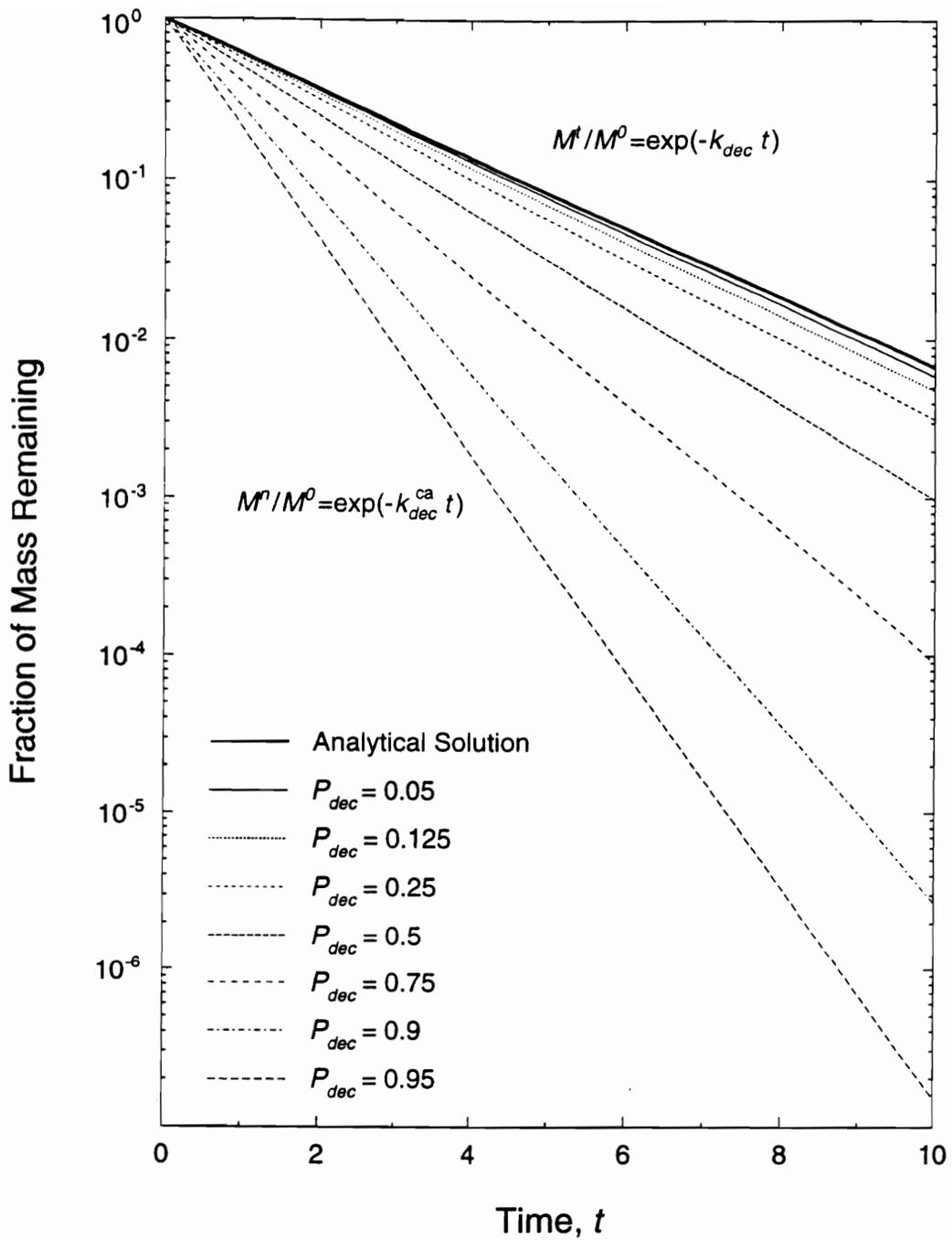


Figure 2.12 Time evolution of the fraction of mass remaining as a function of the decay probability using the CA surrogate method and comparison with the analytical solution for first-order decay. ($k_{dec}=0.5$.)

surrogate method, with k_{dec}^{ca} being a function of P_{dec} . Larger values of P_{dec} lead to larger values of k_{dec}^{ca} . To emphasize this behavior Figure 2.13 is presented in which the ratio k_{dec}^{ca}/k_{dec} , named β , is plotted as a function of P_{dec} . It was found that this relationship $\beta = f(P_{dec})$ holds for any value of k_{dec} . For the CA decay rule to be accurate, very small values of the decay probability (≤ 0.01) for which $\beta \approx 1$, and thus $k_{dec}^{ca} \approx k_{dec}$, are required.

However, the previous results also suggest a procedure to relax this constraint. From the definition of β , then $k_{dec}^{ca} = \beta k_{dec}$. Since the CA approach will result in a decay rate k_{dec}^{ca} larger than the original k_{dec} , this suggests that for k_{dec}^{ca} to be equal to k_{dec} , a smaller k_{dec} (with notation k_{dec}^c) should be substituted for the original value of k_{dec} . This conversion has the form $k_{dec}^c = k_{dec}/\beta$. Given that $\beta = f(P_{dec})$ and also $P_{dec} = f(k_{dec}^c)$ an iterative procedure must be used to calculate k_{dec}^c .

For a particular value of Δt_{dec} , an initial guess for P_{dec} is obtained using equation (2.26). This value of P_{dec} is used to estimate β using the relationship from Figure 2.13. An estimate for k_{dec}^c is then obtained from k_{dec} and β . A new value for P_{dec} is obtained from equation (2.26) using the previously calculated value for k_{dec}^c . This allows for a new estimate of β and therefore k_{dec}^c . This process is repeated until P_{dec} , β , and k_{dec}^c converge within a specified convergence error.

This procedure was implemented and tested for several values of Δt_{dec} and k_{dec} . Figure 2.14 summarizes convergence values obtained for β and P_{dec} , with a convergence error of 10^{-5} . A polynomial equation was fitted to β of the form:

$$\beta = -0.003282(k_{dec} \Delta t_{dec})^3 + 0.065914(k_{dec} \Delta t_{dec})^2 + 0.563833(k_{dec} \Delta t_{dec}) + 0.973541 \quad (2.29)$$

Since $P_{dec} = k_{dec}^c \Delta t_{dec}$, and $k_{dec}^c = k_{dec}/\beta$, it follows that

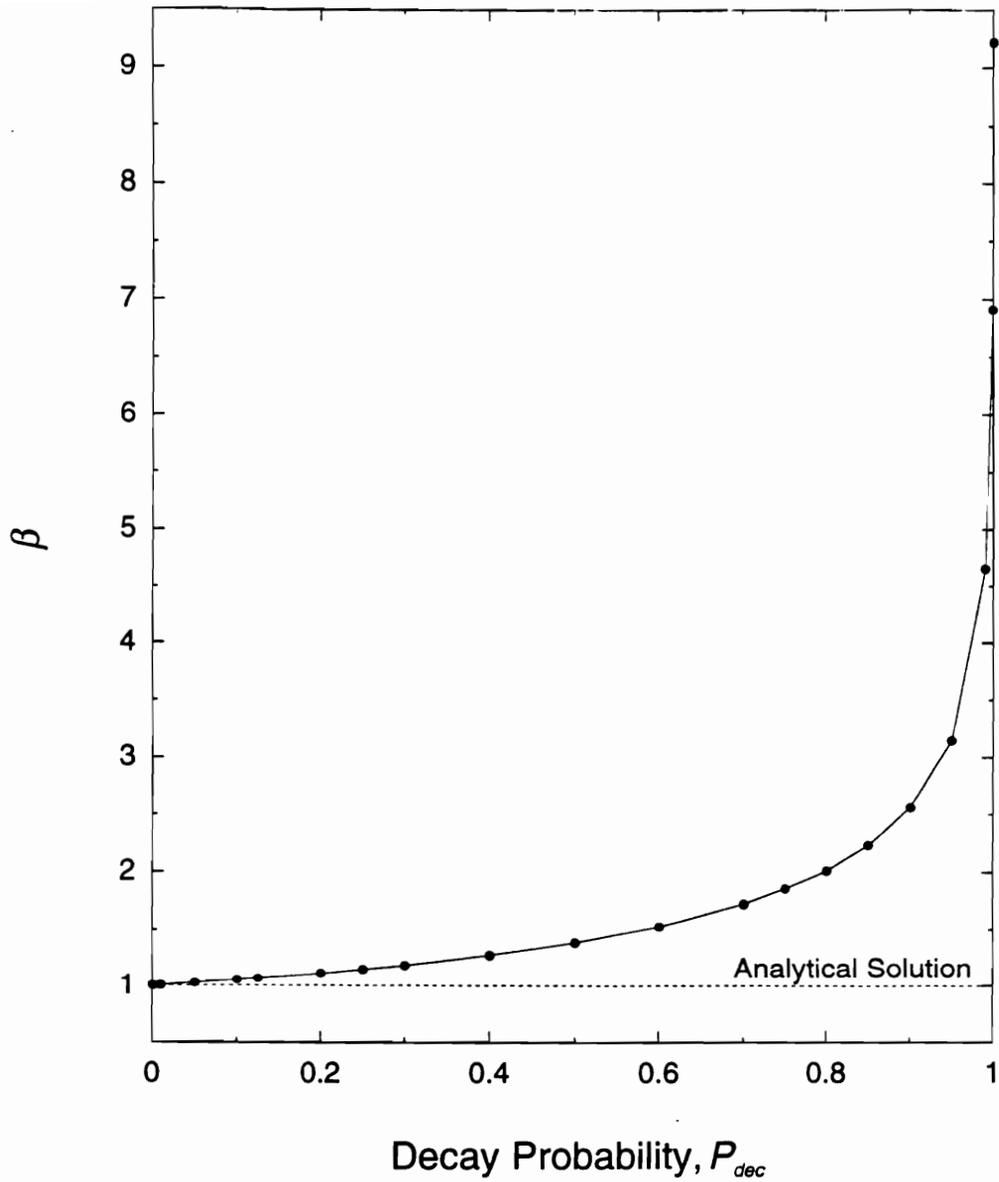


Figure 2.13 Relation between the ratio $\beta = \frac{k_{dec}^{ca}}{k_{dec}}$ and the decay probability for first-order decay.

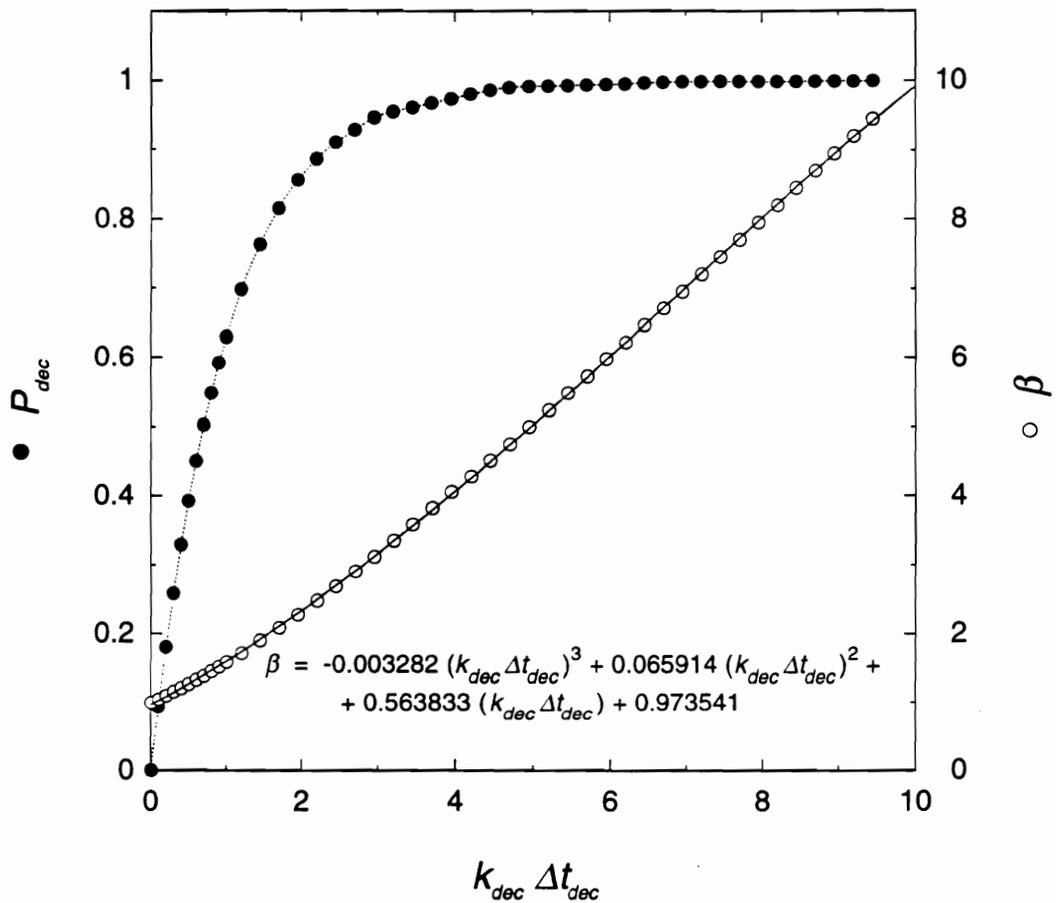


Figure 2.14 Convergence values for β and P_{dec} as a function of $k_{dec} \Delta t_{dec}$ for first-order decay.

$$P_{dec} = \frac{k_{dec}\Delta t_{dec}}{-0.003282(k_{dec}\Delta t_{dec})^3 + 0.065914(k_{dec}\Delta t_{dec})^2 + 0.563833(k_{dec}\Delta t_{dec}) + 0.973541}. \quad (2.30)$$

This equation should be used instead of equation (2.26) to define P_{dec} .

Assuming that in a typical situation the maximum value for k_{dec} , denoted $k_{dec,max}$, is known, the largest possible value for Δt_{dec} , represented by $\Delta t_{dec,max}$, is obtained from equation (2.30) with $P_{dec}=1$ (the upper limit for P_{dec}). Figure 2.15 shows $\Delta t_{dec,max}$ as a function of $k_{dec,max}$. The relation obtained is given as:

$$\Delta t_{dec,max} = \frac{8.876}{k_{dec,max}}. \quad (2.31)$$

The validity of equation (2.30) was tested with simulations of the CA decay rule for different values of P_{dec} . Each simulation was repeated one-hundred times, and the mean and standard deviation were calculated. The total number of particles, with an arbitrary uniform mass, at the beginning of each simulation was $N_p^0=10000$ and the value of $k_{dec}=0.05$.

The results are presented in Figure 2.16, where the mean of the fraction of mass remaining is plotted as a function of time. The magnitude of the standard deviations is less than the size of the plot symbols. The analytical solution is given by equation (2.28). It is clear that equation (2.30) provides a much better representation of the decay process than equation (2.26). Equally satisfactory results are obtained despite wide variation in the values of P_{dec} . This implies that specific constraints on the value of P_{dec} are not necessary to obtain satisfactory results.

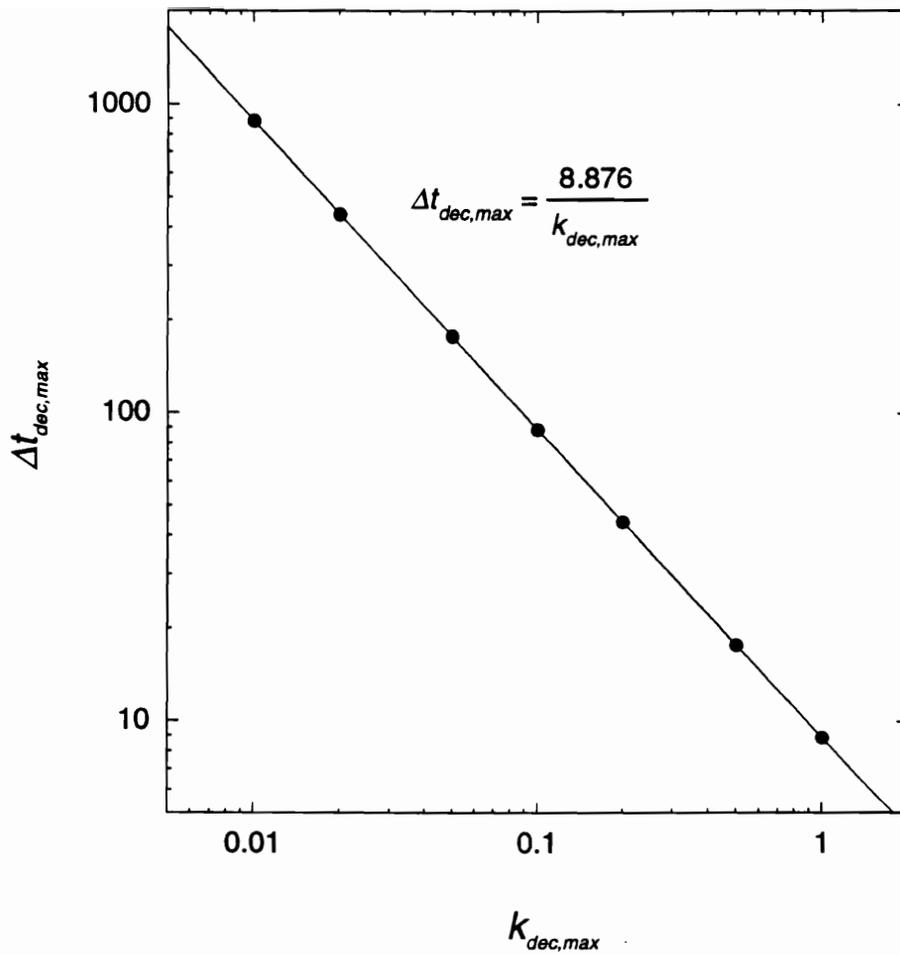


Figure 2.15 Relation between $\Delta t_{dec,max}$ and $k_{dec,max}$ for first-order decay.

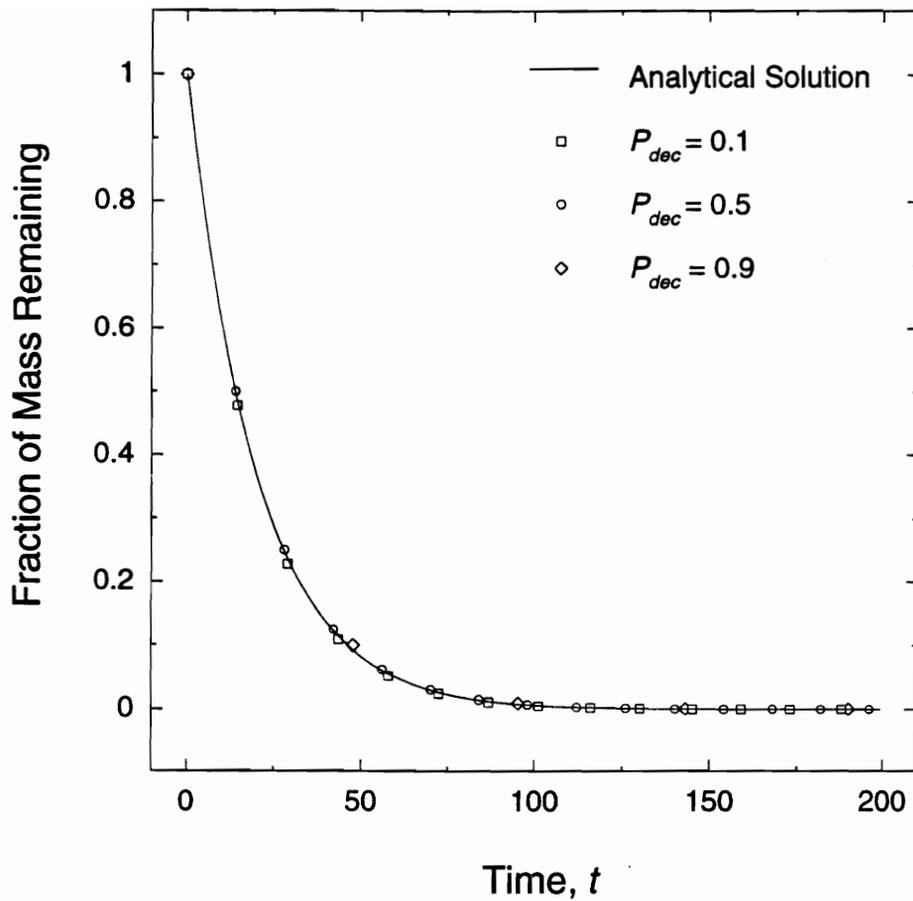


Figure 2.16 Time evolution of the fraction of mass remaining as a function of the decay probability for the CA decay rule using equation (2.30) and comparison with the analytical solution for first-order decay. ($N_p^0=10000$, $k_{dec}=0.05$.)

2.5 WATER QUALITY MODEL DEVELOPMENT

Given the CA representation of several water quality processes as described above, those different representations can be combined to form a full water quality model. In such a CA model, each rule (representing a process) is considered independent of other rules (representing other processes). Cellular automata evolve as a result of applying a sequence of independent rules. However, a particular rule can be applied more than once in the sequence before the rule for a different process is used. In this way, each rule can be designed to best represent the respective process by removing unnecessary constraints imposed by other rules. This is particularly true since all the rules depend on the time step.

A time-splitting approach, similar to the one used in other modeling methodologies (Wheeler and Dawson, 1988), allows one to consider different values of time step for different rules. One advantage of this approach is that it allows the selection of time step values that optimize the performance of the different rules. Synchronization of the various rules in the sequence is obtained by applying proportionally more often those rules using smaller time steps.

To simplify synchronization of the different rules, a main time step, Δt , is defined by the user corresponding to the time interval at which the values of input variables are to be updated. Therefore, at every main time step, a sequence of different rules is applied with each rule being repeated a number of times, φ . The value of φ typically will vary among different processes, although is obviously always a function of the main time step. Additionally, and depending on the processes, it is also a function of specific coefficients (u_{max} , E_{max} , or $k_{dec.max}$), Δx , and $E_{num.max}$.

The advection rule is characterized by numerical dispersion. As shown previously the control of numerical dispersion is accomplished by constraining both Δx and Δt_{adv} .

Since other rules are potentially dependent on Δx , the procedure to calculate φ is first done for advection. From previous results (Figures 2.5 and 2.6) Δx_{max} and $\Delta t_{adv,max}$ can be obtained. A value Δt_{adv} is then derived as the highest submultiple of Δt not exceeding $\Delta t_{adv,max}$. Finally the value of φ_{adv} is given by the integer defined by the ratio $\Delta t / \Delta t_{adv}$. In addition Δx is calculated as $u_{max} \Delta t_{adv}$, therefore satisfying $\Delta x \leq \Delta x_{max}$. When these calculations lead to $\Delta t_{adv} < \Delta t_{adv,max}$ (therefore $\Delta x < \Delta x_{max}$) this obviously implies (see Figures 2.5 and 2.6) a corresponding decrease in the value for $E_{num,max}$. The new value is obtained from equation (2.9).

Given the value of Δx above, the procedure can be applied to rules depending on Δx as is the case for dispersion. From equation (2.23) the value of $\Delta t_{dis,max}$ can be obtained. A value Δt_{dis} is next derived as the highest submultiple of Δt not exceeding $\Delta t_{dis,max}$. The value of φ_{dis} is given by the integer defined by the ratio $\Delta t / \Delta t_{dis}$.

For decay, equation (2.31) gives the value for $\Delta t_{dec,max}$. As before, a value Δt_{dec} is then derived as the highest submultiple of Δt not exceeding $\Delta t_{dec,max}$. The value of φ_{dec} is similarly given by the integer defined by the ratio $\Delta t / \Delta t_{dec}$.

As mentioned previously the rules are applied in a sequence. For simplicity that sequence is fixed, which means that at each main time step advection is applied first (repeated φ_{adv} times), followed by dispersion (repeated φ_{dis} times), and finally decay (repeated φ_{dec} times). Although effects arising from the order in which the rules are applied have not been investigated, if such effects are important then the order of the rules can be manipulated to improve results.

2.6 CONCLUSIONS

The RWPM can be successfully represented using a CA approach. Due to the discrete nature of CA, the rule for advection introduces considerable numerical dispersion. However, the magnitude of this numerical dispersion can be minimized by

proper selection of the cell size and the time step. Similarly, the rule for dispersion is also affected by some numerical dispersion. But, in contrast to advection, a procedure was developed that eliminates numerical dispersion associated with the dispersion rule. For first-order decay a rule was derived that describes the decay process without the limitations of a similar approach reported in the literature. The rules developed for advection, dispersion, and decay, due to their independence, are well suited to implementation using a time-splitting approach.

APPENDIX A: CHARACTERIZATION OF THE MASS DISTRIBUTION RESULTING FROM THE ADVECTION INDUCED NUMERICAL DISPERSION

The assumption suggested in section 2.2.2, that the mass distribution resulting from the numerical dispersion associated with the advection process follows a normal distribution, is investigated in more detail in this appendix.

This involves comparing the mass distribution from the CA advection surrogate method (as described in section 2.2.1.2) with the analytical solution for the advection-dispersion differential equation for an instantaneous input (Thomann and Mueller, 1987). The mass present in each cell as a fraction of the total mass is:

$$\frac{M_i}{M^0} = \frac{1}{2\sqrt{\pi E_{num} t}} \int_{x_i - \frac{\Delta x}{2}}^{x_i + \frac{\Delta x}{2}} e^{-\frac{((x-0.5\Delta x)-ut)^2}{4E_{num}t}} dx \quad (A1a)$$

$$= \frac{1}{2} \left[\operatorname{erf} \left(\frac{x_i - ut}{2\sqrt{E_{num} t}} \right) - \operatorname{erf} \left(\frac{x_i - \Delta x - ut}{2\sqrt{E_{num} t}} \right) \right] \quad (A1b)$$

where M_i is the mass present in cell i , and M^0 is the total mass (which was instantaneously introduced in cell $x_j = 0.5\Delta x$, at time $t = 0$). The x-coordinate of a cell i is defined at the mid-point of the cell, and is given as $x_i = (i - 0.5)\Delta x$.

The value of E_{num} in equation (A1b) is provided by equation (2.4). In addition, t is given by the product $n\Delta t_{adv}$, with Δt_{adv} being provided by equation (2.1). After substitutions are made in equation (A1b) one obtains:

$$\frac{M_i}{M^0} = \frac{1}{2} \left[\operatorname{erf} \left(\frac{(i - 0.5) - nP_{adv}}{\sqrt{2n(1 - P_{adv})P_{adv}}} \right) - \operatorname{erf} \left(\frac{(i - 1.5) - nP_{adv}}{\sqrt{2n(1 - P_{adv})P_{adv}}} \right) \right]. \quad (\text{A2})$$

The value of i is any integer ($\dots, -2, -1, 0, 1, 2, \dots$), with $i=1$ representing the cell into which a mass M^0 is instantaneously introduced at the beginning of the simulation, i.e., when $n=0$.

The mass distribution from equation (A2) is then compared with the mass distribution obtained for the CA surrogate method. The results are summarized in Figure A1, in which the distribution curves result from interpolation, using the natural cubic spline method, of the respective discrete values associated with the cells.

Figure A1 shows that the mass distribution from the CA approach follows a normal probability density function similar to the analytical solution for the advection-dispersion differential equation. However, a few simulation steps are required for the mass distribution to develop a shape identical to the analytical solution. A minimal number of steps is required when $P_{adv}=0.5$, while a larger number is required for other values of P_{adv} . This is related to the initial skewness developed by the mass distribution, which obviously tends to be more pronounced as P_{adv} diverges from 0.5.

This good level of agreement between the two distributions strongly supports the previous assumption that the mass distribution from the CA approach follows a normal

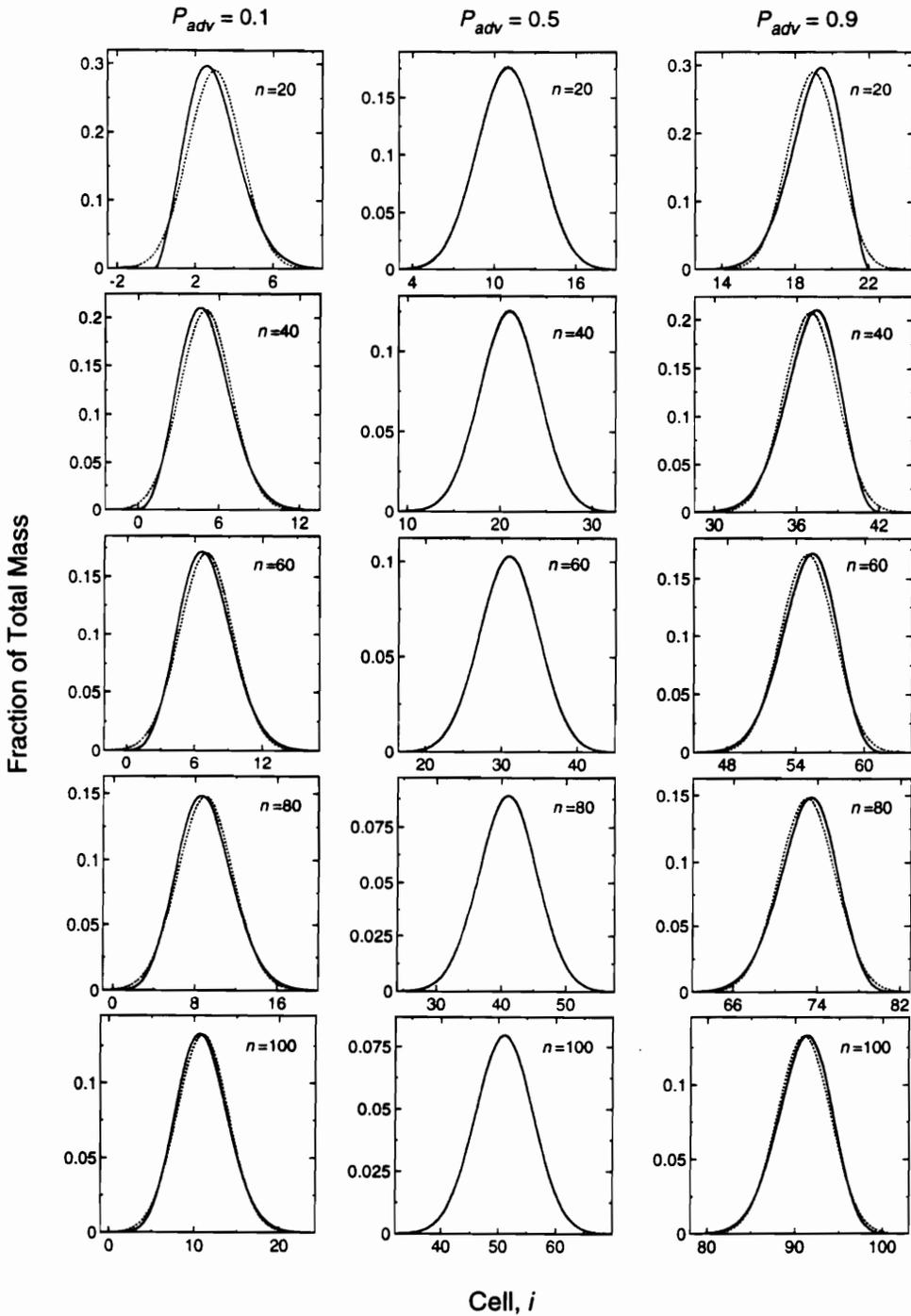


Figure A1 Comparison between the mass distributions obtained for the CA advection surrogate method (solid line) and the analytical solution for the advection-dispersion differential equation for an instantaneous input (dotted line). The distributions are shown for different advection probability values and at the end of various simulation steps.

probability density function similar to the solution of the advection-dispersion differential equation for an instantaneous input.

APPENDIX B: DERIVATION OF THE EXPRESSION FOR THE DISPERSION PROBABILITY

The following derivation is based on the concepts of random-walk discussed by Bear and Verruijt (1987). Using a particle tracking formulation, a particle can be considered to travel by discrete steps in a direction x . This particle movement has two components: an advection component in which a particle moves by a deterministic amount A ; and a dispersion component in which a particle moves by a random amount of maximum magnitude (amplitude) B . It is assumed that the deterministic and random components of the movement are independent. Furthermore, in the following discussion only the random component of the movement, i.e., the dispersion component is considered.

Assuming that the distribution of the random component of the movement is uniform, the distribution function characterizing a step movement of the particle is given as:

$$P(x) = 0 \quad \text{if } x < -B \quad \text{(B1a)}$$

$$P(x) = \frac{1}{2B} \quad \text{if } -B < x < B \quad \text{(B1b)}$$

$$P(x) = 0 \quad \text{if } x > B \quad \text{(B1c)}$$

It can be shown that the mean, m , of this distribution is equal to zero. This is the average distance traveled by a particle in each step. Also, the standard deviation, s , of the distribution is given by $B/\sqrt{3}$.

The probability distribution of the particle movement for a large number of independent steps (n) is expected to follow a normal distribution of the form:

$$P(x) = \frac{1}{\sqrt{2\pi S^2}} \exp\left(-\frac{(x-M)^2}{2S^2}\right) \quad (\text{B2})$$

where

$$M = nm = 0, \quad (\text{B3})$$

$$S^2 = ns^2 = n\frac{B^2}{3}. \quad (\text{B4})$$

This probability distribution representing the random-walk can be compared with the one-dimensional dispersion differential equation

$$\frac{\partial C}{\partial t} = E \frac{\partial^2 C}{\partial x^2} \quad (\text{B5})$$

which has the following solution for an instantaneous spill of a unit mass of material in a channel of unit cross sectional area:

$$C = \frac{1}{\sqrt{4\pi Et}} \exp\left(-\frac{x^2}{4Et}\right). \quad (\text{B6})$$

Expressions (B2) and (B6) are similar and they become identical if

$$S^2 = 2Et. \quad (\text{B7})$$

Also, combining equations (B4) with (B7) leads to:

$$n\frac{B^2}{3} = 2Et. \quad (\text{B8})$$

Rearranging equation (B8), one obtains:

$$B = \sqrt{6E \frac{t}{n}}. \quad (\text{B9})$$

Since $\Delta t_{dis} = \frac{t}{n}$ it follows that

$$B = \sqrt{6E\Delta t_{dis}}. \quad (\text{B10})$$

Equation (B10) thus provides a useful relation between the amplitude of particle movement during a dispersion time step and the dispersion coefficient. This amplitude can be expressed in a dimensionless form $B / \Delta x$, which can be viewed as the amplitude of the dispersion probability required by the CA:

$$P_{dis,amp} = \frac{B}{\Delta x} = \frac{\sqrt{6E\Delta t_{dis}}}{\Delta x} \quad (\text{B11})$$

where $P_{dis,amp}$ is the dispersion probability amplitude, i.e., the maximum value allowed for the absolute value of the dispersion probability P_{dis} . All values for P_{dis} are therefore in the interval $[-P_{dis,amp}, +P_{dis,amp}]$. Given the previous assumption of a uniform distribution for the random component of particle movement, an expression for the dispersion probability is:

$$P_{dis} = (2q - 1)P_{dis,amp} \quad (\text{B12})$$

where q is a uniformly distributed random number between 0 and 1.

REFERENCES

Ackerer, P., "Random-Walk Method to Simulate Pollutant Transport in Aluvial Aquifers or Fractured Rocks." in *Groundwater Flow and Quality Modelling*, E. Custodio, A. Gurgui, and J. P. Lobo Ferreira, eds., NATO ASI Series C, vol. 224, D. Reidel, Dordrecht, p. 475-486 (1988).

- Ahlstrom, S., H. Foote, R. Arnett, C. Cole, and R. Serne, "Multicomponent Mass Transport Model: Theory and Numerical Implementation." Technical Report No. BNWL 2127, Battelle Pacific Northwest Laboratories, Richland, WA (1977).
- Allen, C. M., "Numerical Simulation of Contaminant Dispersion in Estuary Flows." in *Proc. R. Soc. Lond.*, A381, p. 179-194 (1982).
- Bear, J., and A. Verruijt, *Modeling Groundwater Flow and Pollution*. D. Reidel, Dordrecht, Holland, 414 p. (1987).
- Bernardin, D., O. E. Sero-Guillaume, and C. H. Sun, "Multispecies 2D Lattice Gas with Energy Levels: Diffusive Properties." *Physica D*, 47, 169-188 (1991).
- Boghosian, B. M., "Lattice Gases." in *1989 Lectures in Complex Systems*, Erica Jen, ed., vol. II, Addison-Wesley, Redwood City, CA (1990).
- Bogle, G. V., T. A. Smith, and F. I. Chung, "Particle Tracking Model for the Sacramento-San Joaquin Delta." in *Proceedings of the 1993 Conference on Hydraulic Engineering*, H. W. Shen, S. T. Su, and F. Wen, eds., ASCE, p. 827-832 (1993).
- Boon, J. P., "Statistical Mechanics and Hydrodynamics of Lattice Gas Automata: An Overview." *Physica D*, 47, 3-8 (1991).
- Brieger, L., and E. Bonomi, "A Stochastic Cellular Automaton Simulation of the Non-Linear Diffusion Equation." *Physica D*, 47, 159-168 (1991).
- Chen, S., K. Diemer, G. D. Doolen, K. Eggert, C. Fu, S. Gutman, and B. J. Travis, "Lattice Gas Automata for Flow Through Porous Media." *Physica D*, 47, 72-84 (1991).
- Cliffe, K. A., R. D. Kingdon, P. Schofield, and P. J. Stopford, "Lattice Gas Simulation of Free-Boundary Flows." *Physica D*, 47, 275-280 (1991).
- Dimou, K. N., and E. E. Adams, "A Random-Walk, Particle Tracking Model for Well-Mixed Estuaries and Coastal Waters." *Estuarine, Coastal and Shelf Science*, 37, 99-110 (1993).

- Doolen, G. D., U. Frisch, B. Hasslacher, S. Orszag, and S. Wolfram, *Lattice Gas Methods for Partial Differential Equations*. Proceedings of the Workshop on Large Nonlinear Systems, Santa Fe Institute Studies in the Sciences of Complexity, Vol. IV, Addison-Wesley, Redwood City, CA, 554 p. (1990).
- Dougherty, D. E., and A. F. B. Tompson, "Implementing the Particle Method on the iPSC/2." in *Computational Methods in Surface Hydrology*, G. Gambolati, A. Rinaldo, C. A. Brebbia, W. G. Gray, and G. F. Pinder, eds., Springer-Verlag, New York, p. 473-478 (1990).
- Dougherty, D. E., "Hydrologic Applications of the Connection Machine CM-2." *Water Resources Research*, 27, 3137-3147 (1991).
- Duarte, J. A. M. S., and U. Brosa, "Viscous Drag by Cellular Automata." *Journal of Statistical Physics*, 59, 501-508 (1990).
- Eli, R. N., "Cellular-Automata for Hydrodynamic Modeling." in *Proceedings of the 1987 National Conference on Hydraulic Engineering*, R. M. Ragan, ed., ASCE, New York, p. 552-557 (1987).
- Fogelman, F., Y. Robert, and M. Tchente, *Automata Networks in Computer Science: Theory and Applications*. Princeton University Press, Princeton, NJ, 264 p. (1987).
- Fox, G. C., D. R. Williams, and P. C. Messina, *Parallel Computing Works!* Morgan Kaufmann, San Francisco, CA, 977 p. (1994).
- Hockney, R. W., and J. W. Eastwood, *Computer Simulation Using Particles*. Adam Hilger, Bristol, 540 p. (1988).
- Józsa, J., "2-D Particle Model for Predicting Depth-Integrated Pollutant and Surface Oil Slick Transport in Rivers." in *Proceedings of the International Conference on Hydraulic and Environmental Modelling of Coastal, Estuarine and River Waters*, R. A. Falconer, P. Goodwin, and R. G. S. Matthew, eds., Gower Technical, Brookfield, Vermont, p. 332-340 (1989).

- Kinzelbach, W., "The Random Walk Method in Pollutant Transport Simulation." in *Groundwater Flow and Quality Modelling*, E. Custodio, A. Gurgui, and J. P. Lobo Ferreira, eds., NATO ASI Series C, vol. 224, D. Reidel, Dordrecht, p. 227-245 (1988).
- Kleinschmidt, D. G., and B. R. Pearce, "A Microcomputer-Based Tool for the Investigation of Coastal Hydrodynamic Systems." in *Proceedings of the Second International Conference on Hydraulic and Environmental Modelling of Coastal, Estuarine and River Waters*, R. A. Falconer, K. Shiono, and R. G. S. Matthew, eds., vol. 2, Ashgate, Brookfield, Vermont, p. 491-500 (1992).
- Kong, X. P., and E. G. D. Cohen, "A Kinetic Theorist's Look at Lattice Gas Cellular Automata." *Physica D*, 47, 9-18 (1991).
- Kougias, C. F., G. Krause, and A. Rauh, "Simulation of River-Discharge Fronts with Lattice Gas Automata." in *Discrete Models of Fluid Dynamics*, A. S. Alves, ed., Series on Advances in Mathematics for Applied Sciences, vol. 2, World Scientific, Singapore, p. 156-161 (1991).
- Mahinthakumar, G., and A. J. Valocchi, "Application of the Connection Machine to Flow and Transport Problems in Three-Dimensional Heterogeneous Aquifers." *Advances in Water Resources*, 15, 289-302 (1992).
- Prickett, T. A., T. G. Naymik, and C. G. Lonquist, "A 'Random-Walk' Solute Transport Model for Selected Groundwater Quality Evaluations." *Bulletin 65, ISWS/BUL-65/81*, Illinois Department of Energy and Natural Resources, Illinois State Water Survey, Champaign, IL, 103 p. (1981).
- Rothman, D. H., "Macroscopic Laws for Immiscible Two-Phase Flow in Porous Media: Results from Numerical Experiments." *Journal of Geophysical Research*, 95, 8663-8674 (1990).
- Shen, H. T., P. D. Yapa, and M. E. Petroski, "A Simulation Model for Oil Slick Transport in Lakes." *Water Resources Research*, 23, 1949-1957 (1987).
- Shen, H. T., and P. D. Yapa, "Oil Slick Transport in Rivers." *Journal of Hydraulic Engineering*, 114, 529-543 (1988).

- Thomann, R. V., and J. A. Mueller, *Principles of Surface Water Quality Modeling and Control*. Harper and Row, New York, N. Y., 644 p. (1987).
- Toffoli, T., and N. Margolus, *Cellular Automata Machines: a New Environment for Modeling*. MIT Press, Cambridge, MA, 259 p. (1987).
- Tompson, A. F. B., and L. W. Gelhar, "Numerical Simulation of Solute Transport in Three-Dimensional, Randomly Heterogeneous Porous Media." *Water Resources Research*, 26, 2541-2562 (1990).
- Tompson, A. F. B., and D. E. Dougherty, "Particle-Grid Methods for Reacting Flows in Porous Media with Application to Fisher's Equation." *Applied Mathematical Modelling*, 16, 374-383 (1992).
- Uffink, G. J. M., "Modeling of Solute Transport with the Random Walk Method." in *Groundwater Flow and Quality Modelling*, E. Custodio, A. Gurgui, and J. P. Lobo Ferreira, eds., NATO ASI Series C, vol. 224, D. Reidel, Dordrecht, p. 247-265 (1988).
- Valocchi, A. J., and H. A. M. Quinodoz, "Application of the Random Walk Method to Simulate the Transport of Kinetically Adsorbing Solutes." in *Groundwater Contamination*, Proceedings of the Symposium held at the Third IAHS Scientific Assembly, Baltimore, MD, L. M. Abriola, ed., IAHS Pub. no. 185, p. 35-42 (1989).
- von Neumann, J., *Theory of Self-Reproducing Automata*. A. W. Burks, ed., University of Illinois Press, Urbana, IL, 388 p. (1966).
- Wheeler, M. F., and C. N. Dawson, "An Operator-Splitting Method for Advection-Diffusion-Reaction Problems." in *The Mathematics of Finite Elements and Applications VI MAFELAP 1987*, J. R. Whiteman, ed., Academic Press, London, p. 463-482 (1988).
- Williams, B. J., and J. B. Hinwood, "Two-Dimensional Mathematical Water Quality Model." *Journal of the Environmental Engineering Division, EE1*, 149-163 (1976).

Wolfram, S., "Cellular Automata as Models of Complexity." *Nature*, 311, 419-424 (1984).

Zannetti, P., *Air Pollution Modeling: Theories, Computational Methods and Available Software*. Van Nostrand Reinhold, New York, 444 p. (1990).

Zeigler, B. P., *Multifaceted Modelling and Discrete Event Simulation*. Academic Press, London, 372 p. (1984).

3 DYNAMIC WATER QUALITY MODELING USING CELLULAR AUTOMATA: MODEL APPLICATION USING PARALLEL PROCESSORS

3.1 INTRODUCTION

In chapter 2 a new methodology based on cellular automata (CA) was developed which successfully represents fundamental water quality process, namely advection, dispersion, and first-order decay. This chapter tests the validity of the CA methodology as an integrated water quality model. Due to inherent parallelism, CA models are well suited to implementation on parallel processors (Toffoli and Margolus, 1987; Amato, 1991; Fox *et al.*, 1994). This characteristic of CA opens the possibility of a more detailed and efficient dynamic modeling of water resources systems. Consequently the CA water quality model presented here was implemented on parallel processors.

3.1.1 Parallel Computing

Since the advent of the electronic computer in the 1950's a typical ten-fold improvement in computational speed has occurred every five years, mainly as a result of considerably advances in electronic integrate circuitry. However, such technological progress has not been sufficient to satisfy the increasing computational demand from scientific and engineering applications. Thus, parallel computation appeared as an alternative approach to increase computer performance. This approach involves incorporating multiple computational units in a single computer and operating them concurrently, thereby substantially increasing system performance (Green, 1991; Messina, 1991).

There are several classification schemes for parallel computer architectures based on design and functional characteristics. Existing parallel computer architectures can be distinguished in two categories based on the relation between the sequence of instructions executed, and the sequence of data operated on (Ortega, 1988; Duncan, 1990; Fox, 1991; Green, 1991): single instruction stream, multiple data stream (SIMD), and multiple instruction stream, multiple data stream (MIMD).

In a SIMD machine, a controller processor broadcasts a single instruction to all the individual processors which synchronously execute the instruction on different data (Ortega, 1988; Duncan, 1990; Green, 1991). Typical examples of SIMD computers are provided by the array processors, and the Connection Machine (Ortega, 1988; Green, 1991). Pipelined vector processors (e.g. the Cray supercomputers) can also be viewed as SIMD machines (Ortega, 1988).

Computers of the MIMD category consist of multiple autonomous processors, each executing asynchronously a particular set of instructions on a particular set of data (Ortega, 1988; Duncan, 1990; Green, 1991). Most existing parallel computers are MIMD (Ortega, 1988; Green, 1991). Examples of MIMD machines include the Sequent, Ncube, and Intel iPSC (Green, 1991; Messina, 1991). The autonomy associated with the individual processors in MIMD machines provides them with greater flexibility than the SIMD systems (Ortega, 1988; Messina, 1991).

Parallel computers are also classified, based on how memory is available to the different processors, as shared versus distributed memory (Ortega, 1988; Duncan, 1990; Green, 1991; Messina, 1991). While in a shared memory system all processors have access to a common memory, in a distributed memory system each processor has only its own (local) memory (Ortega, 1988; Green, 1991). Thus communication between different processors is done through the common memory for shared memory systems, and through message passing for distributed memory systems (Ortega, 1988; Green, 1991).

Another important aspect of parallel computers is the type of interconnection scheme providing communication between the different processors, and between processors and memory (Ortega, 1988). Several successful interconnection schemes are currently in use, such as meshes, switches, hypercubes, and hybrid schemes (Ortega, 1988; Duncan, 1990; Fox, 1991).

Parallel computers have evolved substantially during the last decade and that trend is expected to continue (Messina, 1991; Fox *et al.*, 1994). The possibility of successful scaling to a large number of processors is shown by the testimony of high performance machines now operational (Messina, 1991; Fox *et al.*, 1994). Many parallel computer architectures have proved to be reliable for engineering and scientific applications involving large-scale computations (Fox *et al.*, 1988; Fox, 1991; Messina, 1991; Camp *et al.*, 1994; Dabdub and Seinfeld, 1994; Fox *et al.*, 1994). In particular, models based on CA have been successfully implemented in parallel processors (Toffoli and Margolus, 1987; Fox *et al.*, 1994).

However, current use of parallel computers is still limited to some extent by the availability of software (Fox, 1991; Fox *et al.*, 1994). Substantial software development is needed in several areas such as standardized programming languages and compilers with support for parallelism, debuggers, libraries, performance evaluation, data visualization, and multi-user system management (Fox, 1991; Messina, 1991; Fox *et al.*, 1994).

In relation to the performance of today's parallel computers, their limitations typically are not in the internal computational speed of their microprocessors but in the performance of input and output (I/O) systems and slow communication between processors in distributed memory systems (Fox *et al.*, 1994).

3.2 METHODOLOGY

3.2.1 General Model

The basic equation describing the concentration distribution in time and space of a water quality constituent subject to decay in a one-dimensional river or estuary is (Thomann and Mueller, 1987):

$$\frac{\partial C}{\partial t} = -u \frac{\partial C}{\partial x} + E \frac{\partial^2 C}{\partial x^2} - k_d C \quad (3.1)$$

where C (M/L^3) is the constituent concentration, t (T) is the time, x (L) is the distance in the longitudinal direction, u (L/T) is the advective velocity, E (L^2/T) is the longitudinal dispersion coefficient, and k_d (T^{-1}) is the first-order decay rate constant.

This equation takes into account three essential processes affecting the constituent distribution: advection (represented by the velocity term), longitudinal dispersion, and decay. It is typically solved using finite difference or finite element schemes. The use of CA principles is an alternative method to represent and solve the same problem.

The development of a CA water quality model for a one-dimensional river or estuary has been presented in the previous chapter. It used a representation consisting of a line of cells. The amount of constituent is represented by the number of particles, with a defined mass, present in each cell. In the computer implementation of the CA algorithm, the number of particles present in each cell is stored in computer memory. Constituent concentration at a given cell is derived by dividing the product of the respective number of particles and the particle mass by the volume of that cell. This volume is defined by the cell length and cross-sectional area. Since the physical entity represented by a particle (a finite amount of constituent mass) is typically orders of magnitude smaller than the cell

volume, it is convenient to allow a cell to have more than a single particle. Each cell has no defined limit on the number of particles it can contain. This does not mean, however, that an unlimited number of states per cell are considered. The transition rules are iteratively applied to each particle, and they result in particles either moving to adjacent cells or disappearing from the system. This process corresponds to a two-state (particle or 1, and no particle or 0) cellular automaton.

These rules are defined in such a way that the local neighborhood of a cell is defined by just that specific cell. The rules account for the advection, dispersion, and decay processes. Each rule (representing a process) is considered independent of other rules (representing other processes). Cellular automata evolve as a result of applying a sequence of independent rules. However, in this methodology a particular rule can be applied more than once before the rule for the next process is called in. Synchronization of the various rules in the sequence is obtained by applying proportionally more often those rules using smaller time steps.

Each cell in the CA model can be individually assigned any specific set of coefficient values (such as velocity, dispersion, and decay coefficients), which may or may not be time dependent. This translates into the possibility of having varying probability values for the same transition rule among different cells at a given time and/or among different times for a given cell. Values for velocity, dispersion, and decay coefficients, for instance, must be specified at user selected locations and times as part of the model input data. The model then assigns coefficient values at each time step for every cell, based on temporal and then spatial interpolation, typically using a linear interpolation algorithm. To satisfy the continuity equation, spatial interpolation of velocity involves converting velocity values to flow rates, based on cross-sectional areas, followed by flow rate interpolation, and finally conversion of flow rates back to velocities.

The transition rules are not necessarily applied to individual particles present in a cell, as suggested for simplicity above, but to groups or packets of particles which then become the units for computation. The rationale underlying this approach is given next.

It has been shown for the Random-Walk Particle Method (RWPM) that the random noise associated with model results, measured as standard concentration error, is a function of the inverse of the square root of the number of particles used in the simulation (Ahlstrom *et al.*, 1977; Bagtzoglou *et al.*, 1992). However, computation time is a linear function of the number of particles (Ahlstrom *et al.*, 1977). Finally, the noise to signal ratio, measured as standard error of concentration over concentration, is inversely related to the square root of concentration (Bagtzoglou *et al.*, 1992). Assuming these results can be extended to the present CA model, they suggest an approach in which smaller mass particles are considered for cells having lower concentration, and larger mass particles are considered for cells with higher concentration. Model resolution can then be improved where most needed (where concentrations are lower) without a significant increase in computing time.

To simplify the implementation of this approach, all particles are assumed to have the same mass. Preceding the application of a rule, particles in a cell are grouped in packets and each packet is then treated as a single particle. The number of particles included in a packet is the total number of particles in a cell multiplied by the packet fraction f , the relative size of the packets or groups of particles to which the CA rules are applied.

When the number of particles included in a packet is a non integer, a procedure is used to guarantee conservation of particles. This involves allowing an additional packet, having more or less particles than the others. For instance, if $f=0.1$ and a cell contains 1027 particles then there will be 10 packets (i.e., $1/f$) of 102 particles each (i.e., the integer component of $1027 \cdot f=102.7$) plus an additional packet which includes 7 particles

(i.e., $10^{27} \cdot 10 \cdot 10^2 = 7$). Although this approach was used throughout this chapter, a better approach is to assign the extra particles to the regular packets by adding one additional particle to each packet until all extra particles have been assigned. This approach has the advantage of guaranteeing that the number of packets in cells is fixed ($1/f$) and that no large differences in the number of particles among the packets in a cell will result (packets differ by a single particle at the most).

The particle mass selected should be small enough to allow the higher concentration cells to have each a number of particles several orders of magnitude higher than $1/f$. This will assure that only a few lower concentration cells will have a number of particles on the order of $1/f$. When the number of particles in a cell is $1/f$, the limit of maximum packet resolution is reached. The setting up of the packets in each cell is an efficient procedure since as mentioned earlier what is stored in computer memory is the number of particles in every cell.

So far the CA model was described as one-dimensional. However, the model concept easily extends to higher dimensions. In the two-dimensional case the CA are represented by a grid of cells and the geometry of the modeled system dictates the shape of the grid.

In the current two-dimensional implementation of the model, advection is still considered as a one-dimensional process, which is a common assumption in many models. This implies that the cell length in any spatial direction perpendicular to the direction in which advection occurs, does not affect the advection induced numerical dispersion. Therefore, this cell length can be chosen by the model user and is constrained only by spatial resolution requirements.

Dispersion and decay, however, are two-dimensional. This involves having two independent rules for dispersion, each one defined along one of the two perpendicular directions of the cell grid. Therefore, particles are initially dispersed in one direction then

dispersed in a second direction. The simplicity of this approach arises from the fact that the dispersion rule in any direction is defined exactly the same way as described previously for the one-dimensional case. The decay rule is applied to every cell as in the one-dimensional case. The fact that the cells are organized as a two-dimensional grid naturally leads to a two-dimensional representation of decay.

Independently of the number of dimensions incorporated in the model, initial conditions specifying the number of particles in each cell are required at the beginning of the simulation. This implies having in the model input data the constituent initial concentration at user specified locations. Based on spatial interpolation of these values, concentrations are obtained for every cell. Each concentration is then converted to a corresponding number of particles, based on the volume of the cell and the particle mass. When this conversion leads to a non integer number, the number of particles assigned to a cell is the integer component of the number, plus an extra particle if a generated uniformly distributed random number (between 0 and 1) does not exceed its fractional component.

Dirichlet boundary conditions are handled in a similar way. Concentrations are specified for the upstream and downstream boundaries of the system, such as a river or estuary. Those concentrations can eventually be interpolated in time and/or space leading to concentration values at the boundary cells. Then these are converted to corresponding number of particles. At each time step the boundary conditions have to be specified, i.e., the number of particles in boundary cells has to be reset, prior to each repetition of the advection and longitudinal dispersion rules. Dirichlet boundary conditions do not affect the outcome of the decay rule, since this rule does not involve any interaction between different cells. Given the way boundary conditions are implemented in the current model, only at upstream and downstream cells, the outcome of non longitudinal dispersion is not affected.

The implementation of gradient or Neumann boundary conditions in the CA model involves adjusting the number of particles in adjacent cells on the boundary to match a specified concentration gradient across those cells. To adjust the number of particles requires an algorithm that redistributes particles among the adjacent cells until the specified gradient is met. However, the existing particle movement at boundary cells makes the incorporation of gradient boundary conditions slightly more complicated than just described. First, the advection rule drains particles out of cells on the upstream boundary (this will occur at both upstream and downstream boundaries, although intermittently, when tidal conditions exist) simply because these particles are moved downstream while no particles enter these cells to replace them. Second, the dispersion rule produces localized particle deficits at the upstream and downstream boundaries as a result of particles leaking out of the system on both upstream and downstream cells. Contrary to Dirichlet boundary conditions which naturally counteract these effects, gradient boundary conditions do nothing to prevent it. Thus, the specification of gradient boundary conditions requires the simultaneous incorporation of particle fluxes on those boundary cells in such a way as to add new particles into the system. The purpose of these fluxes is to balance long-term particle deficits that would otherwise develop at those boundaries.

The discussion above suggests that successful incorporation of gradient boundary conditions in the CA model is possible if additional measures are taken to counteract particle deficit problems. Although this research did not deal with the gradient boundary condition problem, particle deficits at boundary cells were observed during simple preliminary simulations testing the behavior of the CA methodology when using gradient boundary conditions.

No-flow boundary conditions are included in the CA model to represent soil-water and air-water interfaces. These boundary conditions are incorporated in the rules

involving particle movement between cells, namely the advection and dispersion rules. This involves overriding the normal rule behavior when particles are selected to move to a neighbor cell which does not exist since it would lie past the system boundary. These particles are not allowed to move and stay in the original cell.

Constituent discharges into the system are handled by adding particles to cells during a certain number of time steps depending on the duration of the discharge. Any nonpoint and point discharge can be considered. A rate of mass discharge is converted to number of particles based on the length of the simulation time step, the particle mass, and perhaps the size of the cell. When the number of particles to be added is found to be a non integer, the procedure described previously is used.

3.2.2 BOD/DO Model

A typical water quality model for biochemical oxygen demand (BOD) and dissolved oxygen (DO) follows a Streeter and Phelps formulation (Thomann and Mueller, 1987), in which the BOD and DO behavior is represented by two different equations. Due to the dependence of the DO on the BOD, the two equations must be solved sequentially, with the BOD equation being solved first.

Similarly, in the CA BOD/DO model, one cellular automaton is used for each of the two constituents. A stack of two one-dimensional (or two-dimensional) CA can then be visualized. Interaction between BOD and DO occurs only in the BOD decay process. A particle representing DO is removed from a cell in the DO cellular automaton each time a particle of BOD is removed from the corresponding cell of the BOD cellular automaton. In this way, BOD decay controls DO directly.

A BOD/DO model also includes a source/sink term representing the change in DO due to exchange with the atmosphere (reaeration). In the CA model this process is

conceptually similar to the first-order decay (as presented in chapter 2), in effect being a decay of deficit particles. This deficit is just the difference between the number of particles corresponding to oxygen saturation conditions and the actual number of particles present in a cell. A non-zero deficit can be either a positive (undersaturation) or negative quantity (supersaturation conditions).

Therefore, the reaeration probability, P_{aer} , is defined as:

$$P_{aer} = \frac{k_{aer} \Delta t_{aer}}{-0.003282(k_{aer} \Delta t_{aer})^3 + 0.065914(k_{aer} \Delta t_{aer})^2 + 0.563833(k_{aer} \Delta t_{aer}) + 0.973541}$$

$$0 \leq P_{aer} \leq 1 \quad (3.2)$$

where k_{aer} is the first-order reaeration rate constant (T^{-1}), and Δt_{aer} is the time step for the reaeration process. The value of $P_{aer} \leq 1$. As suggested above, the rule is designed to work for undersaturated and supersaturated DO conditions. When the DO deficit is positive, particles can be added to a cell; when negative, particles can be removed.

As mentioned previously at each main time step the rules for the various processes are applied in sequence, and for simplicity that sequence is fixed. In the BOD/DO model, advection is applied to BOD and DO (and repeated φ_{adv} times), followed by dispersion also applied to BOD and DO (and repeated φ_{dis} times). Then, decay is applied to BOD (and repeated φ_{dec} times). Finally, reaeration is applied to DO (and repeated φ_{aer} times).

3.2.3 Simulation Scenarios

3.2.3.1 One-Dimensional Line Pulse Input

This simulation is included solely to illustrate the impact of using the packet fraction approach on the model results and validation through comparison with other

models is not pursued. The simulation includes advection, dispersion, and decay in a one-dimensional river of uniform cross-section subject to a pulse discharge of a non-conservative constituent. The discharge occurs along a line in the longitudinal direction for a fixed distance. The longitudinal concentration profile just after the time of discharge follows a symmetric trapezoidal shape, with the concentration linearly increasing from zero to a constant value, staying at that value for most of the longitudinal distance, and then decreasing back to zero. This concentration profile includes a range of concentration values therefore allowing for observation of the effects on model noise-to-signal ratios as a function of concentration. The coefficients used in this simulation are space and time invariant. Table 3.1 summarizes parameter values for this simulation.

3.2.3.2 Two-Dimensional Pulse Input

The main objective of this simulation is to evaluate the behavior of a two-dimensional version of the CA model. A pulse input of a conservative constituent is considered in a river with a uniform rectangular cross-section. The model dimensions represent the longitudinal (x) and vertical (z) directions. The input discharge occurs at the top of the water column, i.e., at zero depth, and it is considered laterally well mixed. The simulation includes advection in the longitudinal direction, and dispersion in the longitudinal and vertical directions. Since decay is naturally extended from one to higher dimensions it is not considered here for simplicity reasons. Following advection,

Table 3.1 Parameter values used in the one-dimensional line pulse input simulation.

CA Modeling Parameters	
General	$A = 50.0 \text{ m}^2$ ^a $M = 480.0 \text{ g}$ ^b $C_0 = 0.0 \text{ mg/L}$ ^c $C_{ubc} = C_{dbc} = 0.0 \text{ mg/L}$ ^d $\Delta t = 40.0 \text{ sec}$ $f = 0; 0.0001; 0.001; 0.01; 0.1$ ^e $m_p = 0.01; 0.1; 1.0; 10.0 \text{ g/part}$ ^f
Advection	$u = 0.05 \text{ m/sec}$ $u_{max} = 0.1 \text{ m/sec}$ $E_{num,max} = 0.5 \text{ m}^2/\text{sec}$ ^g $\Delta t_{adv,max} = 400.0 \text{ sec}$ $\Delta x_{max} = 40.0 \text{ m}$ $\Delta t_{adv} = 40.0 \text{ sec}$ $\phi_{adv} = 1$ $\Delta x = 4.0 \text{ m}$
Dispersion	$E = 4.5 \text{ m}^2/\text{sec}$ $E_{max} = 4.5 \text{ m}^2/\text{sec}$ $\Delta t_{dis,max} = 0.885 \text{ sec}$ $\Delta t_{dis} = 0.870 \text{ sec}$ $\phi_{dis} = 46$
Decay	$k_{dec} = 300.0 \text{ day}^{-1}$ $k_{dec,max} = 300.0 \text{ day}^{-1}$ $\Delta t_{dec,max} = 2556.452 \text{ sec}$ $\Delta t_{dec} = 40.0 \text{ sec}$ $\phi_{dec} = 1$

^a Cross sectional area;

^b Constituent mass instantaneously injected in each of 250 contiguous cells; the 100 cells just upstream and downstream from those 250 cells are injected an amount of mass interpolated between 0 and 480 g and between 480 and 0 g, respectively;

^c Constituent initial condition;

^d Constituent upstream and downstream boundary conditions;

^e A value $f = 0$ means packets were not used, i.e., particles were treated individually;

^f Constituent particle mass;

^g This is the user specified $E_{num,max}$ value. Since $\Delta t_{adv} < \Delta t_{adv,max}$ (therefore $\Delta x < \Delta x_{max}$) the actual value for $E_{num,max}$ is only $0.05 \text{ m}^2/\text{sec}$ (from equation (2.9)).

dispersion in the x direction is applied first (and repeated φ_{dis}^x times) then followed by dispersion in the z direction (and repeated φ_{dis}^z times).

The results from the CA model are validated through comparison with the analytical solution for an instantaneous input of the two-dimensional advection-dispersion differential equation (Hemond and Fechner, 1994). The coefficient values used in this simulation are space and time invariant. Table 3.2 summarizes parameter values for this simulation.

3.2.3.3 One-Dimensional Steady-State BOD/DO

To illustrate the application of the CA model to a multiple constituent system, simple one-dimensional steady-state BOD and DO simulations are considered. First, a single BOD continuous discharge in a river with uniform cross-section is assumed. A second simulation incorporates multiple BOD and DO continuous discharges in a river with variable cross-section. In both cases longitudinal advection and dispersion, decay and reaeration are simulated as described previously. The results from the CA model are validated through comparison with the Streeter and Phelps model (Thomann and Mueller, 1987).

The coefficients used in the simulation for the uniform cross-section river are space and time invariant. Table 3.3 summarizes parameter values for this simulation. The coefficient values used in the simulation for the variable cross-section river are time invariant but vary with space. Table 3.4 summarizes parameter values for this simulation. Some of the parameter values are also depicted in the river system layout shown in Figure 3.1.

Table 3.2 Parameter values used in the two-dimensional pulse input simulation.

	Modeling Parameters	CA Only Parameters
General	$W = 10.0 \text{ m}^a$ $M = 1.0 \text{ kg}^b$ $C_0 = 0.0 \text{ mg/L}^c$ $C_{abc} = C_{dbc} = 0.0 \text{ mg/L}^d$	$\Delta t = 50.0 \text{ sec}$ $f = 0.1$ $m_p = 0.002 \text{ g/part}^e$ $\Delta z = 0.5 \text{ m}^f$
Advection	$u = 1.0 \text{ m/sec}$	$u_{max} = 1.6 \text{ m/sec}$ $E_{num,max} = 0.5 \text{ m}^2/\text{sec}^g$ $\Delta t_{adv,max} = 1.562 \text{ sec}$ $\Delta x_{max} = 2.499 \text{ m}$ $\Delta t_{adv} = 1.515 \text{ sec}$ $\phi_{adv} = 33$ $\Delta x = 2.424 \text{ m}$
Longitudinal Dispersion	$E^x = 4.955 \text{ m}^2/\text{sec}$	$E^x = 4.5 \text{ m}^2/\text{sec}$ $E_{max}^x = 4.5 \text{ m}^2/\text{sec}$ $\Delta t_{dis,max}^x = 0.325 \text{ sec}$ $\Delta t_{dis}^x = 0.325 \text{ sec}$ $\phi_{dis}^x = 154$
Vertical Dispersion	$E^z = 0.005 \text{ m}^2/\text{sec}$	$E_{max}^z = 0.005 \text{ m}^2/\text{sec}$ $\Delta t_{dis,max}^z = 12.442 \text{ sec}$ $\Delta t_{dis}^z = 10.0 \text{ sec}$ $\phi_{dis}^z = 5$

^a Channel width;

^b Constituent mass instantaneously injected at a longitudinal distance of 250 m and zero depth;

^c Constituent initial condition;

^d Constituent upstream and downstream boundary conditions;

^e Constituent particle mass;

^f Cell size in the vertical direction;

^g This is the user specified $E_{num,max}$ value. Since $\Delta t_{adv} < \Delta t_{adv,max}$ (therefore $\Delta x < \Delta x_{max}$) the actual value for $E_{num,max}$ is only $0.485 \text{ m}^2/\text{sec}$ (from equation (2.9)).

Table 3.3 Parameter values used in the one-dimensional steady-state BOD/DO simulation for a river with uniform cross-section and a single continuous discharge.

	Modeling Parameters	CA Only Parameters
General	$A = 30.0 \text{ m}^2$ ^a $W_{BOD} = 750.0 \text{ g/sec}$ ^b $BOD_0 = 0.0 \text{ mg/L}$ ^c $DO_0 = 10.0 \text{ mg/L}$ ^c $BOD_{ubc} = BOD_{dbc} = 0.0 \text{ mg/L}$ ^d $DO_{ubc} = DO_{dbc} = 10.0 \text{ mg/L}$ ^d $DO_{sat} = 10.0 \text{ mg/L}$ ^e	$\Delta t = 100.0 \text{ sec}$ $f = 0.1$ $m_p^{BOD} = m_p^{DO} = 0.01 \text{ g/part}$ ^f
Advection	$u = 0.5 \text{ m/sec}$	$u_{max} = 1.0 \text{ m/sec}$ $E_{num,max} = 2.5 \text{ m}^2/\text{sec}$ $\Delta t_{adv,max} = 20.0 \text{ sec}$ $\Delta x_{max} = 20.0 \text{ m}$ $\Delta t_{adv} = 20.0 \text{ sec}$ $\varphi_{adv} = 5$ $\Delta x = 20.0 \text{ m}$
Dispersion ^g		$E = 2.5 \text{ m}^2/\text{sec}$ $E_{max} = 2.5 \text{ m}^2/\text{sec}$ $\Delta t_{dis,max} = 39.813 \text{ sec}$ $\Delta t_{dis} = 33.333 \text{ sec}$ $\varphi_{dis} = 3$
BOD Decay	$k_{dec} = 0.7 \text{ day}^{-1}$	$k_{dec,max} = 0.7 \text{ day}^{-1}$ $\Delta t_{dec,max} = 1095531.993 \text{ sec}$ $\Delta t_{dec} = 100.0 \text{ sec}$ $\varphi_{dec} = 1$
DO Reaeration	$k_{aer} = 4.0 \text{ day}^{-1}$	$k_{aer,max} = 4.0 \text{ day}^{-1}$ $\Delta t_{aer,max} = 191706.265 \text{ sec}$ $\Delta t_{aer} = 100.0 \text{ sec}$ $\varphi_{aer} = 1$

^a Cross sectional area;

^b BOD loading rate, discharged at a distance of 20 km;

^c BOD and DO initial conditions;

^d BOD and DO upstream and downstream boundary conditions;

^e DO saturation concentration;

^f Particle mass for BOD and DO;

^g Although the Streeter and Phelps model does not include a dispersion term, dispersion is still included in the CA model simulation.

Table 3.4 Parameter values used in the one-dimensional steady-state BOD/DO simulation for a river with variable cross-section and multiple continuous discharges.

Modeling Parameters ^a	CA Only Parameters ^a
General	
$L_r = 70.0$ (I); 70.0 (II); 20.0 (III); 80.0 km (IV) ^b	$\Delta t = 100.0$ sec
$Q = 15.0$ (I); 45.0 (II); 70.0 (III); 70.0 m ³ /sec (IV) ^c	$f = 0.1$
$A = 30.0$ (I); 45.0 (II); 87.5 (III); 140.0 m ² (IV) ^d	$m_p^{BOD} = m_p^{DO} = 0.01$ g/part ^j
$W_{BOD} = 650.0$ (I); 850.0 (II); 375.0 g/sec (III) ^e	
$W_{DO} = 240.0$ (II); 125.0 g/sec (III) ^f	
$BOD_0 = 0.0$ mg/L ^g	
$DO_0 = 10.0$ mg/L ^g	
$BOD_{abc} = BOD_{bc} = 0.0$ mg/L ^h	
$DO_{abc} = DO_{bc} = 10.0$ mg/L ^h	
$DO_{sat} = 10.0$ mg/L ⁱ	
Advection	
$u = 0.5$ (I); 1.0 (II); 0.8 (III); 0.5 m/sec (IV)	$u_{max} = 1.0$ m/sec
	$E_{nem,max} = 2.5$ m ² /sec
	$\Delta t_{adv,max} = 20.0$ sec
	$\Delta x_{max} = 20.0$ m
	$\Delta t_{adv} = 20.0$ sec
	$\varphi_{adv} = 5$
	$\Delta x = 20.0$ m
Dispersion ^k	
	$E = 2.5$ (I); 7.0 (II); 7.4 (III); 1.5 m ² /sec (IV)
	$E_{max} = 7.4$ m ² /sec
	$\Delta t_{dis,max} = 13.450$ sec
	$\Delta t_{dis} = 12.5$ sec
	$\varphi_{dis} = 8$
BOD Decay	
$k_{dec} = 0.7$ (I); 1.5 (II); 1.0 day ⁻¹ (III and IV)	$k_{dec,max} = 1.0$ day ⁻¹
	$\Delta t_{dec,max} = 511290.319$ sec
	$\Delta t_{dec} = 100.0$ sec
	$\varphi_{dec} = 1$
DO Reaeration	
$k_{aer} = 3.5$ (I); 3.0 (II); 4.0 (III); 2.0 day ⁻¹ (IV)	$k_{aer,max} = 4.0$ day ⁻¹
	$\Delta t_{aer,max} = 191706.265$ sec
	$\Delta t_{aer} = 100.0$ sec
	$\varphi_{aer} = 1$

^a The Roman numerals in parenthesis indicate the river section for which parameter values refer;

^b River section length;

^c Flow rate;

^d Cross sectional area;

^e BOD loading rates, discharged at a distance of 20.0 (I), 100.0 (II), and 140.0 km (III);

^f DO loading rates, discharged at a distance of 70.0 (II) and 140.0 km (III);

^g BOD and DO initial conditions;

^h BOD and DO upstream and downstream boundary conditions;

ⁱ DO saturation concentration;

^j Particle mass for BOD and DO;

^k Although the Streeter and Phelps model does not include a dispersion term, dispersion is still included in the CA model simulation.

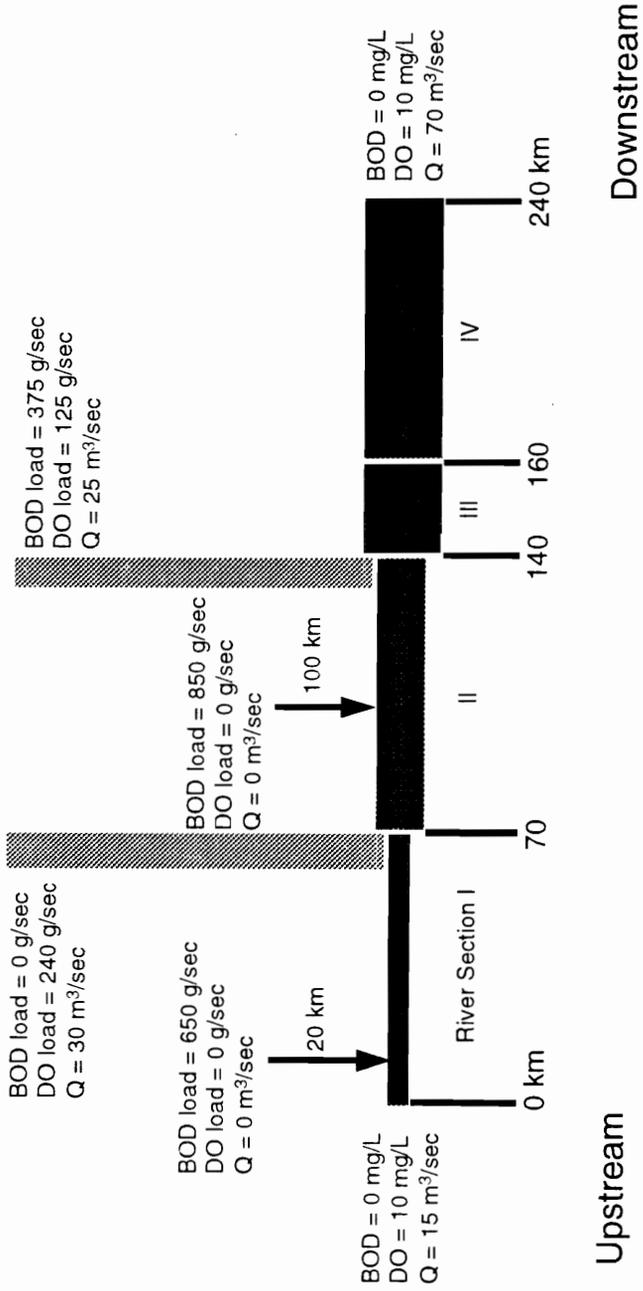


Figure 3.1 Layout of the variable cross-section river system corresponding to the model input parameters of Table 3.4.

3.2.3.4 One-Dimensional Tidal BOD/DO

To test the validity of the CA model under dynamic conditions, BOD and DO profiles resulting from a single continuous discharge of BOD in a simple tidal river system are simulated. The results are compared with a model solving the one-dimensional differential equations for the coupled BOD/DO system on an oscillating flow for a single continuous BOD discharge (Holley, 1969; Macdonald and Weisman, 1977; Giles, 1995). The cross-sectionally averaged longitudinal advective velocity, u , is time dependent and represented through an oscillating tidal velocity superimposed on a constant freshwater velocity (Holley, 1969):

$$u = u_f + u_t \sin\left(\frac{2\pi}{T_p} t\right) \quad (3.3)$$

where positive velocity denotes flow towards the sea, u_f is the freshwater velocity associated with the river flow, u_t is the amplitude of the tidal velocity, T_p is the tidal period, and t is the time. Given this velocity function the CA model parameter u_{max} is given by

$$u_{max} = u_f + u_t. \quad (3.4)$$

The coefficient values used in this simulation are space and time invariant with the exception of the time variant velocity. Sine interpolation is used in the CA model to calculate velocity at any time based on flood, ebb, and slack values of velocity for the entire simulation period. At each time step in the simulation, a time averaged velocity over the entire length of the time step is obtained through 3-point numerical integration using Simpson's rule.

The scenario values used in the CA simulation and differential equation model follow a combination of those reported by Holley (1969) and Giles (1995). Table 3.5 summarizes parameter values for this simulation. Notice in Table 3.5 that the value for the dispersion coefficient in the CA model is $58.68 \text{ m}^2/\text{sec}$, instead of $60.0 \text{ m}^2/\text{sec}$, to account for numerical dispersion. However, the difference between these two values is less than the $E_{num\ max}$ value of $2.03 \text{ m}^2/\text{sec}$ because as shown below, the actual numerical dispersion is less than $E_{num\ max}$ (which just represents the maximum possible value for numerical dispersion). These considerations also apply to Table 3.2.

To get a measure of the actual numerical dispersion introduced by the model, an average value \bar{E}_{num} during a time period $(t_2 - t_1)$ for the actual numerical dispersion is derived from equation (2.8) as:

$$\bar{E}_{num} = \frac{\int_{t_1}^{t_2} 0.5\mu|\Delta x \left(1 - \frac{|u|}{u_{max}}\right) dt}{t_2 - t_1}. \quad (3.5)$$

With the values from Table 3.5 and if equation (3.3) is substituted for u in equation (3.5) and the integral is evaluated over a tidal period, \bar{E}_{num} is $1.32 \text{ m}^2/\text{sec}$. This means the average numerical dispersion introduced by the CA model during a tidal cycle is only about 65% of the maximum allowed numerical dispersion.

This exercise leads to the conclusion that equation (3.5) can be incorporated in the CA model to quantify numerical dispersion introduced during a simulation time step. Then this amount would be subtracted from the original dispersion coefficient value (representing the real dispersion to be simulated at that time step) before applying the dispersion rule. As long as numerical dispersion does not exceed the real dispersion, this approach will mask numerical dispersion making it virtually absent.

Table 3.5 Parameter values used in the one-dimensional tidal BOD/DO simulation.

Modeling Parameters	CA Only Parameters
General	
$A = 90.0 \text{ m}^2$ ^a	$\Delta t = 20.0 \text{ sec}$
$W_{BOD} = 1050.0 \text{ g/sec}$ ^b	$f = 0.1$
$T_p = 12.4 \text{ hr}$	$m_p^{BOD} = m_p^{DO} = 0.05 \text{ g/part}$ ^f
$BOD_0 = 0.0 \text{ mg/L}$ ^c	
$DO_0 = 10.0 \text{ mg/L}$ ^c	
$BOD_{ubc} = BOD_{dbc} = 0.0 \text{ mg/L}$ ^d	
$DO_{ubc} = DO_{dbc} = 10.0 \text{ mg/L}$ ^d	
$DO_{sat} = 10.0 \text{ mg/L}$ ^e	
Advection	
$u_f = 0.1 \text{ m/sec}$	$u_{max} = 0.9 \text{ m/sec}$
$u_t = 0.8 \text{ m/sec}$	$E_{num,max} = 5.0 \text{ m}^2/\text{sec}$ ^g
$u = u_f + u_t \sin\left(\frac{2\pi}{T_p} t\right)$ [equation (3.3)]	$\Delta t_{adv,max} = 49.383 \text{ sec}$
	$\Delta x_{max} = 44.4 \text{ m}$
	$\Delta t_{adv} = 20.0 \text{ sec}$
	$\varphi_{adv} = 1$
	$\Delta x = 18.0 \text{ m}$
Dispersion	
$E = 60.0 \text{ m}^2/\text{sec}$	$E = 58.68 \text{ m}^2/\text{sec}$
	$E_{max} = 58.68 \text{ m}^2/\text{sec}$
	$\Delta t_{dis,max} = 1.374 \text{ sec}$
	$\Delta t_{dis} = 1.333 \text{ sec}$
	$\varphi_{dis} = 15$
BOD Decay	
$k_{dec} = 0.3 \text{ day}^{-1}$	$k_{dec,max} = 0.3 \text{ day}^{-1}$
	$\Delta t_{dec,max} = 2556451.694 \text{ sec}$
	$\Delta t_{dec} = 20.0 \text{ sec}$
	$\varphi_{dec} = 1$
DO Reaeration	
$k_{aer} = 1.0 \text{ day}^{-1}$	$k_{aer,max} = 1.0 \text{ day}^{-1}$
	$\Delta t_{aer,max} = 765172.407 \text{ sec}$
	$\Delta t_{aer} = 20.0 \text{ sec}$
	$\varphi_{aer} = 1$

^a Cross sectional area;

^b BOD loading rate, discharged at a distance of 15 km;

^c BOD and DO initial conditions;

^d BOD and DO upstream and downstream boundary conditions;

^e DO saturation concentration;

^f Particle mass for BOD and DO;

^g This is the user specified $E_{num,max}$ value. Since $\Delta t_{adv} < \Delta t_{adv,max}$ (therefore $\Delta x < \Delta x_{max}$) the actual value for $E_{num,max}$ is only $2.03 \text{ m}^2/\text{sec}$ (from equation (2.9)).

3.2.4 Parallel Computer Implementation

A CA model is computationally intensive. A large number of simple computations must be done to update the CA at each time step of the simulation. Computer architectures exploiting the high degree of parallelism associated with the CA structure are desired.

During the development of the CA model as presented in chapter 2, the algorithms were implemented in a serial (von Neumann) computer. Although this architecture is satisfactory for initial testing of the behavior of the model, it becomes severely time limiting when trying to simulate real water quality problems.

To improve performance, an implementation on a parallel (or concurrent) machine was pursued (Intel Paragon). This machine has MIMD architecture, distributed (local) memory, and a two-dimensional mesh topology (Fox *et al.*, 1994). The Intel Paragon has a configuration of up to 4096 second generation Intel i860 processors (nodes) (Fox *et al.*, 1994). Thus, each processor has its own memory and no direct knowledge of the work being done by other processors. This is well suited to the simulation of CA since the mechanism of updating a cell is typically local. To update its boundary cells, however, a processor may need to receive data from its adjacent processors, and this is accomplished by simple message passing. The use of a MIMD distributed memory implementation has also the added advantage of facilitating code portability to a network of workstations (Dabdub and Seinfeld, 1994).

Therefore, the parallelism associated with the structure of the CA is exploited here by distributing the work load for the CA among different processors. Since the number of computations associated with each processor is large, the overall savings in computation

time are expected to typically offset the increase in overhead due to any necessary communication between processors.

Based on the above considerations, a domain or geometric decomposition strategy is used to divide the computing load among the different processors (Wilson, 1993). Regardless of the number of dimensions involved in the simulation a one-dimensional (longitudinal) domain decomposition is here used. A two-dimensional decomposition was not pursued since it would lead to a greater message traffic (Dabdub and Seinfeld, 1994). Therefore, the domain is divided in a certain number of vertical slices, with all slices having about the same number of cells. The number of slices is given by the number of worker nodes available for the simulation. Each domain slice (subdomain) is then assigned to a particular worker node. The worker nodes are the processors responsible for executing the CA simulation in parallel.

There is also an additional processor, the manager node, responsible for setting up the simulation, supplying the input data to the different worker nodes, and collecting their output. However, the existence of a manager node is not a requirement in the implementation of the CA model. The model can be implemented using only worker nodes. Then the tasks which otherwise would be the responsibility of a manager node are instead performed by the worker nodes themselves. This leads to larger code and memory requirements for the worker nodes while reducing communication overhead (by eliminating communication with a manager node).

To be more specific, the manager node is initially responsible for: loading to memory the input datasets; setting up the grid domain; domain decomposition and assignment of the resulting subdomains to the available worker nodes; computing model parameters such as rule time steps and rule repetitions; and sending parameter values and initial conditions to the appropriate worker nodes. Then at each simulation time step, the manager node is responsible for: temporal interpolation of model coefficients, including

velocity, dispersion, decay, and source (discharge) coefficients, and of boundary conditions; sending those values to the worker nodes; waiting for completion of the simulation time step by all worker nodes; and receiving simulation results from all worker nodes at specific output times and writing them to disk.

The worker nodes are responsible for receiving the initial data from the manager node, namely the subdomain, parameter values, and initial conditions. Then at each time step they are responsible for: receiving the updated model coefficients and boundary conditions from the manager node; spatial interpolation of those coefficients and of boundary conditions; applying the CA rules to their respective subdomains; notifying the manager node of the completion of the simulation time step; and sending simulation results to the manager node at specific output times.

Communication between worker nodes is required during each execution of the advection and longitudinal dispersion rules. The larger the number of repetitions of these rules at each time step (i.e., the larger φ_{adv} and φ_{dis}^x), the greater the number of communications involved. A benefit of the domain decomposition being solely along the longitudinal direction is that the vertical dispersion rule does not entail any communication. Since decay and reaeration rules do not involve any interaction between adjacent cells they require no communication whatsoever.

3.3 RESULTS AND DISCUSSION

3.3.1 One-Dimensional Line Pulse Input

The analysis of the results from this simulation concentrates on snapshots of model behavior obtained at the end of 10 time steps (i.e., at 400 sec simulation time). A total of eight snapshots are presented corresponding to different combinations of packet fraction values and particle mass. By varying the particle mass, one controls the number

of particles in the cells. Decreasing the particle mass increases the number of particles and vice-versa.

The simulation snapshots are organized in four groups, with each group showing the results for two simulations. The first simulation in each group has a packet fraction of zero (which simply means particles are allowed to behave independently of each other and thus no packets are used) and an average number of particles per cell, \bar{N}_p , of approximately a power of ten. The average value, \bar{N}_p , is based solely on the cells for which the concentration is on the higher plateau portion of the concentration curve. This value of \bar{N}_p is intended to be approximately the same as the number of packets per cell (i.e., the inverse of the packet fraction) for the second simulation. In reality the \bar{N}_p for the first simulation ended up being slightly higher (by a factor of 1.17 to 1.20) than the inverse of the packet fraction for the second simulation. The second simulation in each group always has an average number of particles per cell of about 10^4 (in reality 1.15 to 1.22 times that value). The purpose is to compare the quality of results obtained from identical simulations except for the use of the packet fraction approach.

Figure 3.2 shows these results. Smoothed concentrations are obtained through 5 passes of a 5-point moving average algorithm. Noise to signal ratios are obtained for the concentrations before smoothing using the smoothed concentration for the simulation with $f=0$ and $\bar{N}_p \approx 10^4$ as a measure of the true signal. The computation time refers to the model iteration corresponding to the 10th time step. It represents the summation of the individual computation times for the 12 worker nodes used in the simulation.

Comparison of the results from three of the groups clearly shows that the packet fraction approach leads to a notable reduction in the noise to signal ratio at the lower concentration range. Moreover the noise to signal ratio increases, although only slightly, at the higher concentration range. This slight increase is likely an artifact resulting from \bar{N}_p for the first simulation being slightly higher than the inverse of the packet fraction for

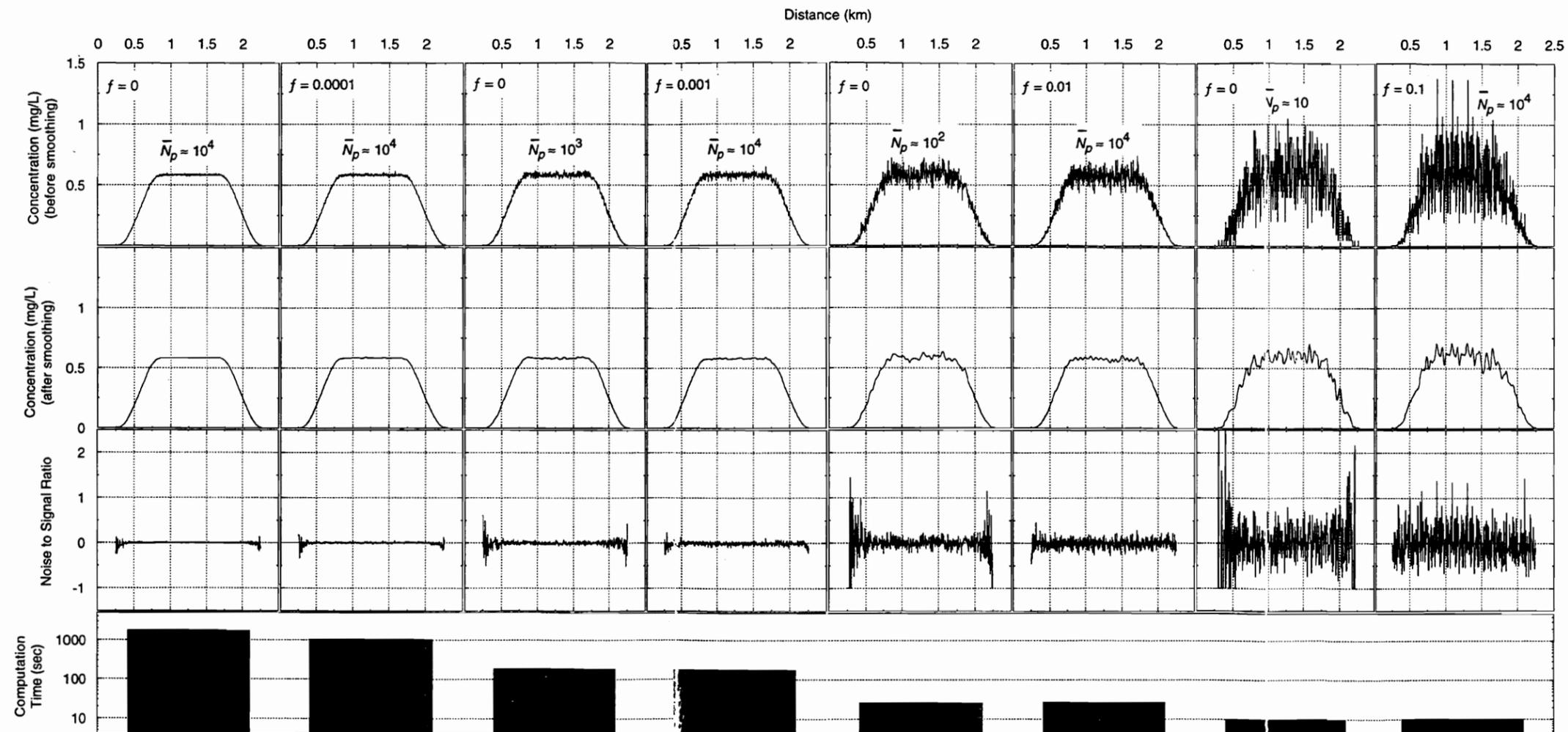


Figure 3.2 Comparison of concentration profiles, noise to signal ratio, and computation time from CA model simulations for evaluation of the effects of the packet fraction approach. (Model input parameters from Table 3.1.)

the second simulation as noted previously. For the remaining group, the pair $f=0$, $\bar{N}_p \approx 10^4$ and $f=0.0001$, $\bar{N}_p \approx 10^4$, a reduction in the noise to signal ratio at the lower concentration range is absent because $\bar{N}_p \approx 10^4$ is about the same as $1/f$. This represents the conditions in which the maximum packet resolution has been reached.

The results from each group consistently show that the use of the packet fraction approach does not incur any significant penalty in terms of computation time. The data suggest that the packet fraction approach has the benefit of reducing the noise to signal ratio at lower concentrations without a significant tradeoff in computation time.

One additional benefit from using the packet fraction approach is the resulting equalization of the work load of cells having different number of particles (as long as cells have at least $1/f$ particles). This is important when domain decomposition is used to split the work among various processors and thus the cell entity represents the limit on the decomposition procedure. In addition it provides some dynamic load balancing as the number of particles in the cells changes during the course of the simulation.

Decreasing the value for the packet fraction clearly increases the computation time while significantly reducing the variability of the results. This suggests that criteria for the selection of desirable values of the packet fraction must consider a compromise between precision of model results and execution time.

3.3.2 Two-Dimensional Pulse Input

Figure 3.3 shows the conservative constituent concentration contour lines at three different times in the simulation. The CA results depicted correspond to an average of the results of two different simulations followed by smoothing with 3 passes of a two-dimensional 9-point moving average algorithm (Fortner, 1992). The CA results show good agreement with the results from the differential equation.

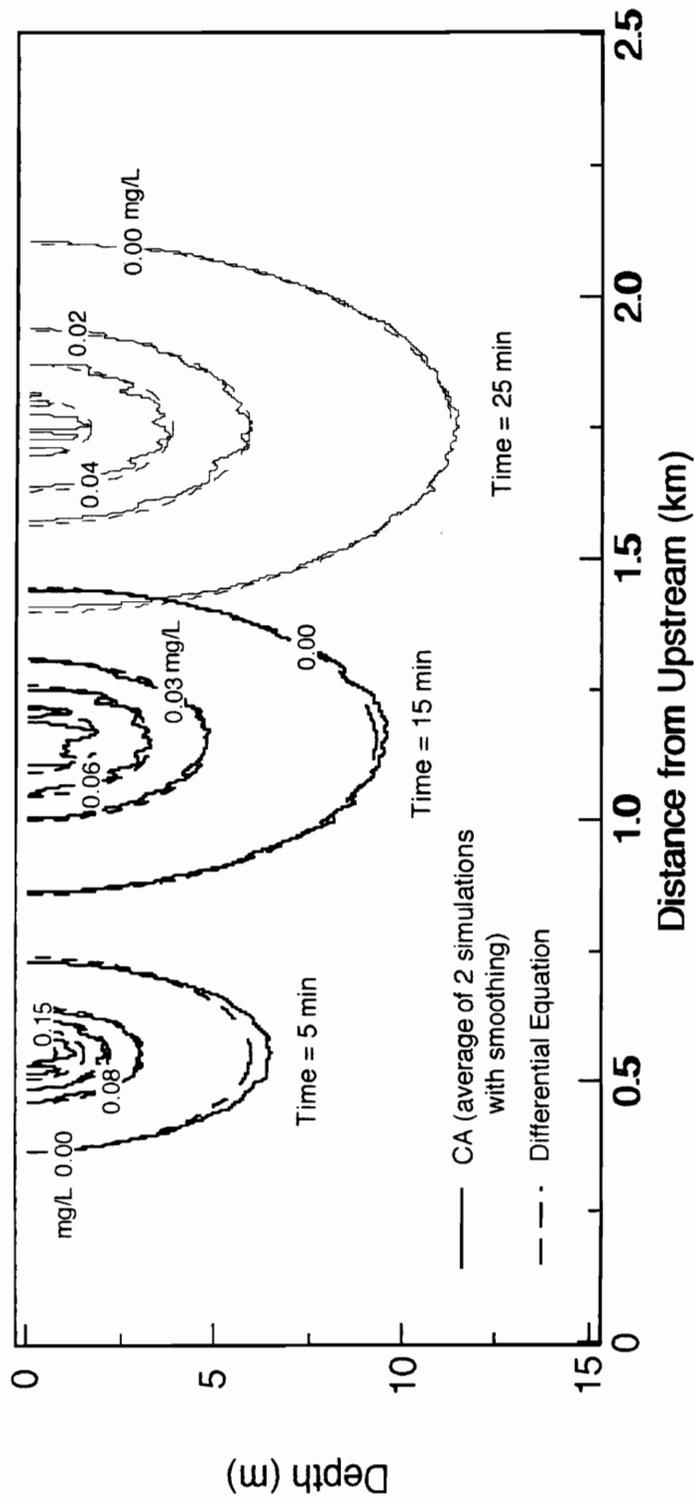


Figure 3.3 Concentration plume of a conservative constituent at successive times after a pulse discharge at a distance of 0.25 km and zero depth. Comparison between the CA model and the two-dimensional advection-dispersion differential equation. (Model input parameters from Table 3.2.)

To complement these results Figure 3.4 is presented. It shows concentration profiles for a longitudinal and a vertical transect at a simulation time of 25 minutes. Both transects are through the point of maximum plume concentration. Overall, these results suggest that the process of simply averaging different simulations does not lead to a decrease in the scattering of the CA results. This finding is unexpected since in the previous chapter, averages of 100 different simulations did produce smooth CA results. Furthermore, it shows that a smoothing algorithm effectively smoothes the CA results. The combination of averaging different simulations followed by smoothing does not seem to lead to a substantial improvement in predictions compared to just smoothing the results for a single simulation. In addition, the results in Figure 3.4 confirm the good agreement between the CA model and the differential equation.

3.3.3 One-Dimensional Steady-State BOD/DO

Simulation results for the uniform cross-section river are shown in Figure 3.5 in which the BOD/DO CA model is compared with the Streeter and Phelps model. These CA model results are from a single simulation and correspond to a simulation time of 5.79 days at which time steady-state conditions are assumed to exist. From these comparisons the CA model is clearly capable of describing the BOD behavior and the DO sag curve as accurately as the traditional approach based upon differential equations.

These conclusions are also supported by the simulation results for the variable cross-section river shown in Figure 3.6. These CA model results are also from a single simulation and correspond to a simulation time of 5.78 days at which time steady-state conditions are assumed to exist. These results also show that the CA model can handle spatial changes in model coefficients very well.

Although dispersion is included in these CA simulations it obviously has no significant influence in the results since during steady-state conditions the effect of the

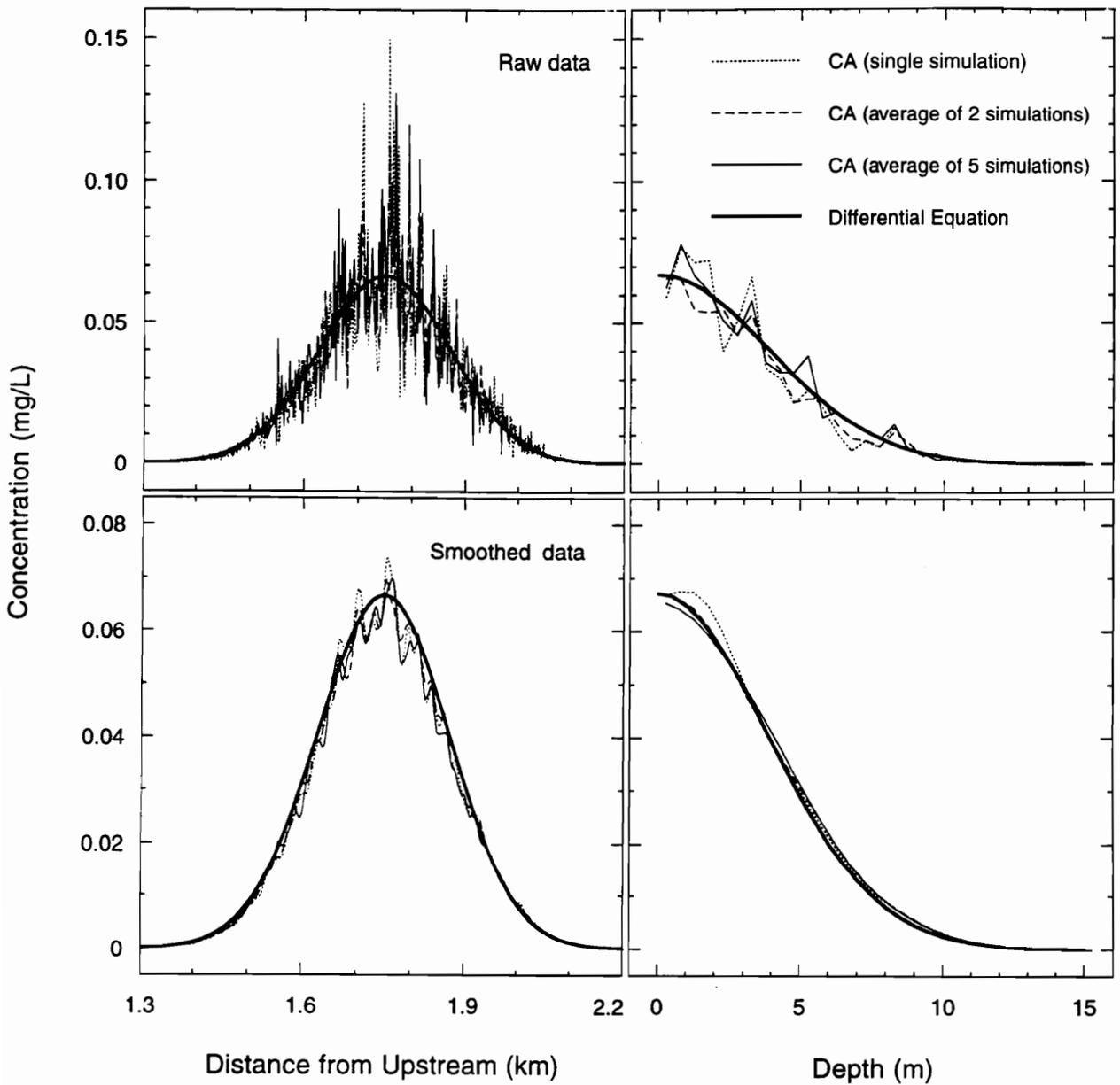


Figure 3.4 Concentration profiles corresponding to longitudinal and vertical transects passing through the maximum concentration point of the plume in Figure 3.3 at a time of 25 minutes after the discharge. Comparison between the CA model and the two-dimensional advection-dispersion differential equation.

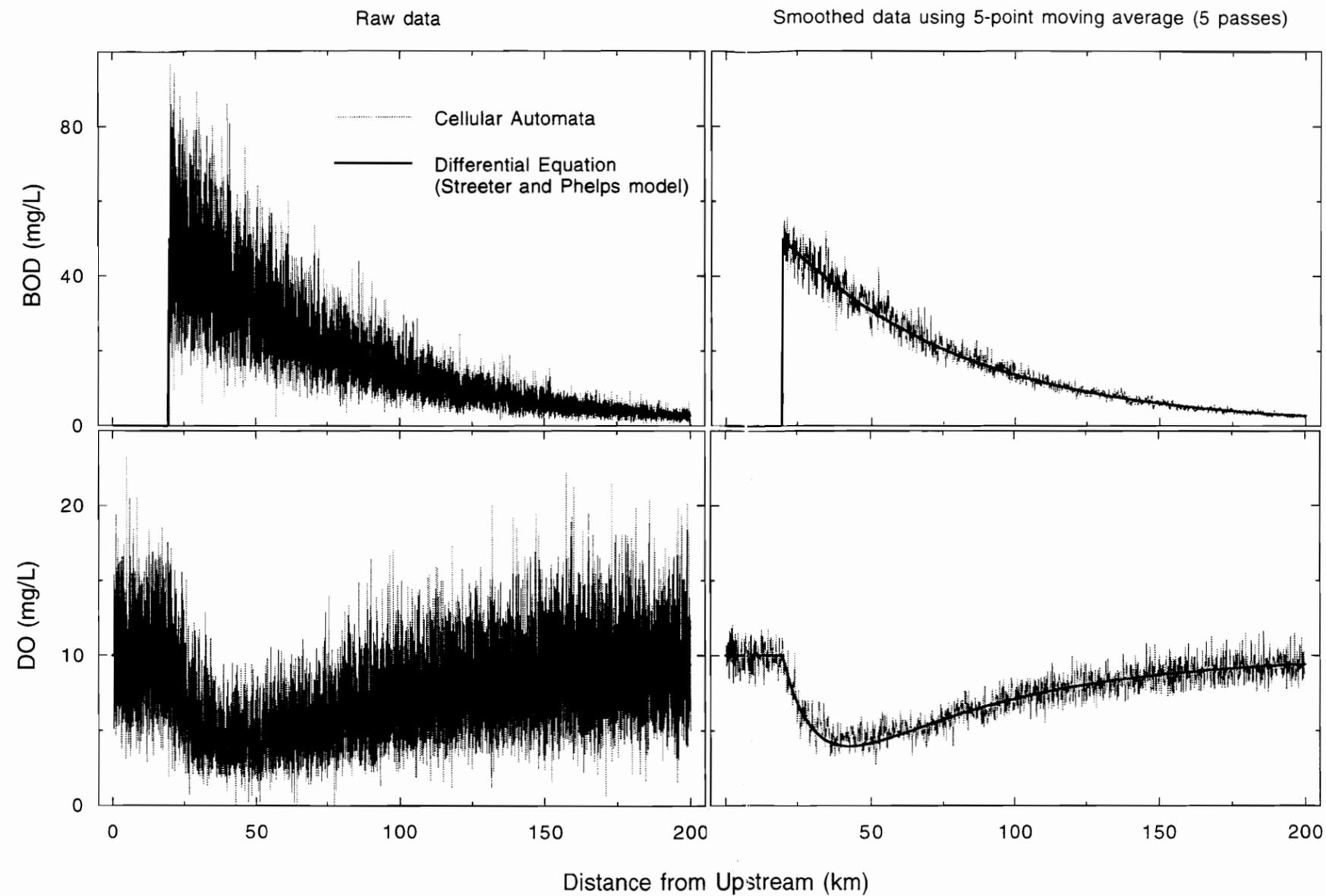


Figure 3.5 Concentration profiles for BOD and DO at steady-state for a river with uniform cross-section and a single continuous discharge. The BOD is being continuously discharged at a distance of 20 km. Comparison between the CA model and the Streeter and Phelps model. The CA results are from a single model simulation. (Model input parameters from Table 3.3.)

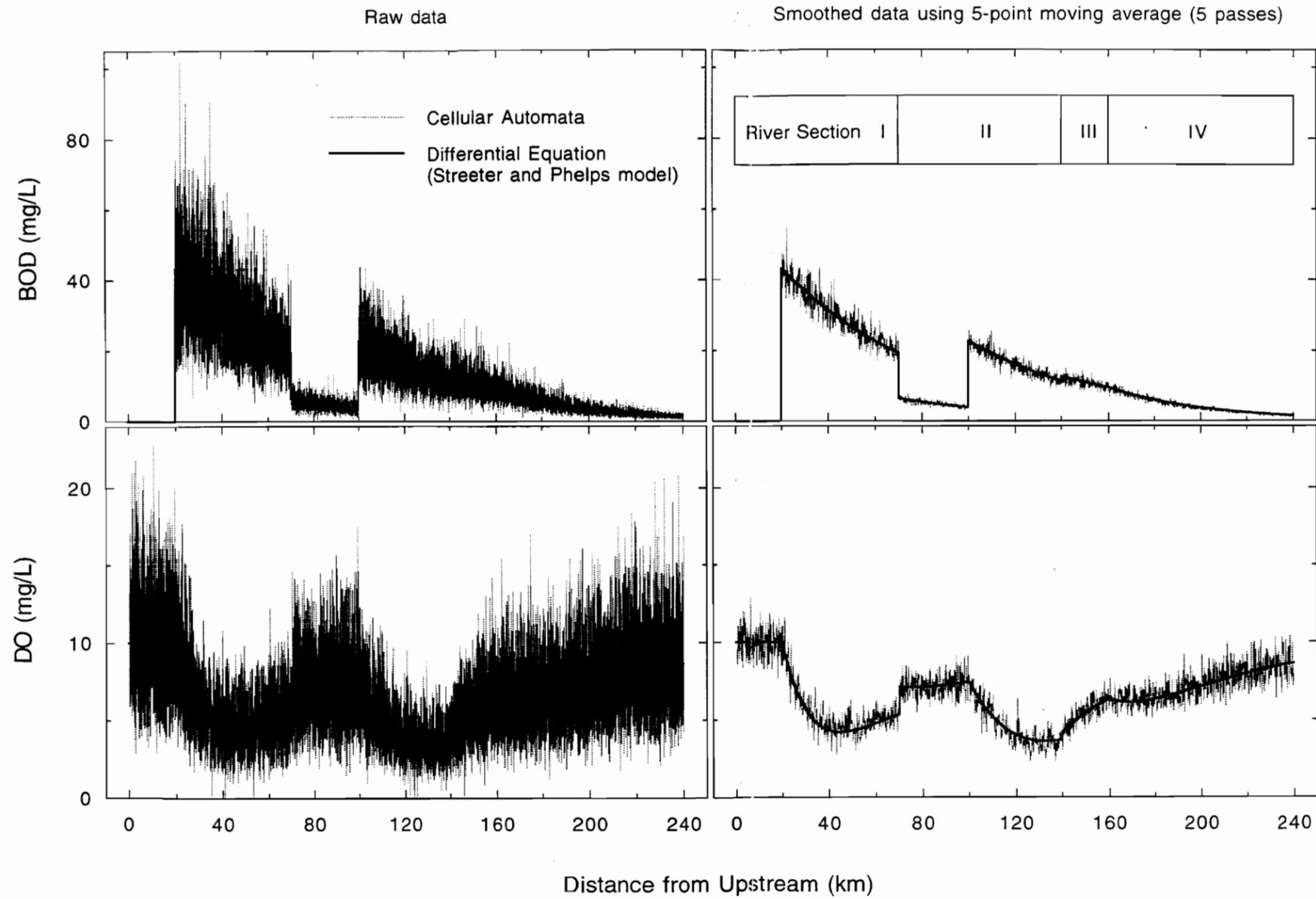


Figure 3.6 Concentration profiles for BOD and DO at steady-state for a river with variable cross-section and multiple continuous discharges. Comparison between the CA model and the Streeter and Phelps model. The CA results are from a single model simulation. (Model input parameters from Table 3.4.)

dispersion process is typically minimal. The results above illustrate once again that a smoothing algorithm substantially reduces the variability associated with the CA results.

Figure 3.7 shows, for the uniform cross-section river simulation, the number of BOD and DO particles contained in all subdomain cells of each worker node. Also shown is the computation time for each worker node corresponding to a single model iteration. All values refer to the iteration corresponding to a simulation time of 5.79 days.

The results illustrate the load balancing associated with the packet fraction approach. Despite the fact that most worker nodes show substantial differences in the number of BOD and DO particles being processed, their computation times are similar. That is not the case for node 1 since its subdomain cells have no BOD particles. And for node 2 since only a minute fraction of its subdomain cells (the most downstream cells) actually contain BOD particles. As expected, the computation time for these two nodes is substantially smaller than for the other nodes despite the use of the packet fraction approach.

Similar considerations can be drawn from Figure 3.8 which shows, for the variable cross-section river simulation, the number of BOD and DO particles contained in all subdomain cells of each worker node. Also shown as before is the computation time for each worker node corresponding to a single model iteration. All values refer to the iteration corresponding to a simulation time of 5.78 days. The load balancing property of the packet fraction approach can easily be viewed as dynamic load balancing since it keeps balancing the work load across nodes as the number of particles in the nodes changes through time.

The close resemblance of the profiles of number of particles and the profiles of concentration for both BOD and DO which can be seen between Figures 3.5 and 3.7 is absent in Figures 3.6 and 3.8. This is because in the variable cross-section river the volume of the cells vary accordingly and therefore identical concentrations can be

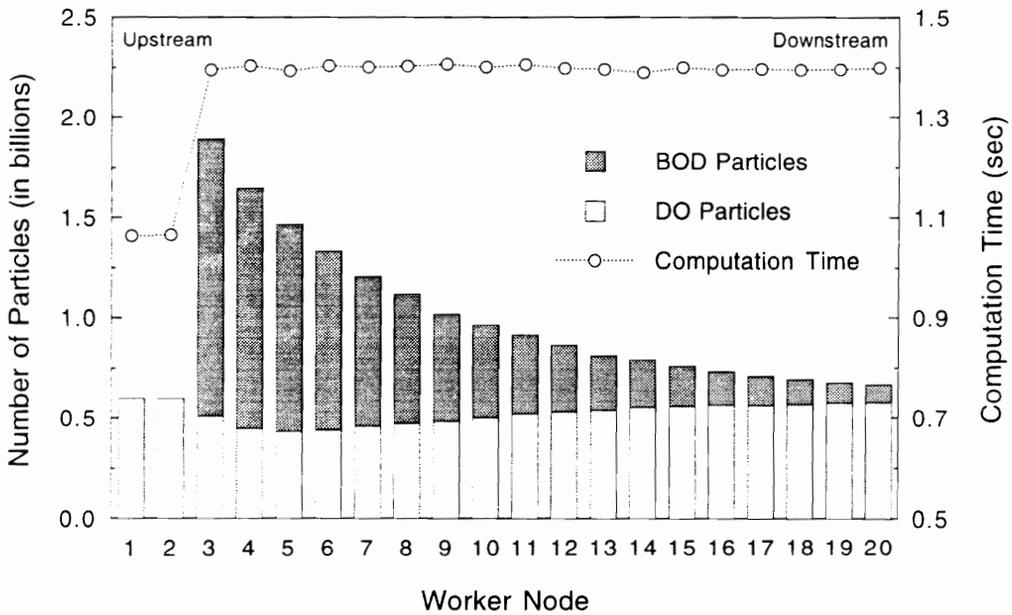


Figure 3.7 Distribution of the number of BOD and DO particles and the computation time among the worker nodes, corresponding to the CA simulation results of Figure 3.5. The worker nodes are numbered based on an upstream to downstream ordering of their subdomains.

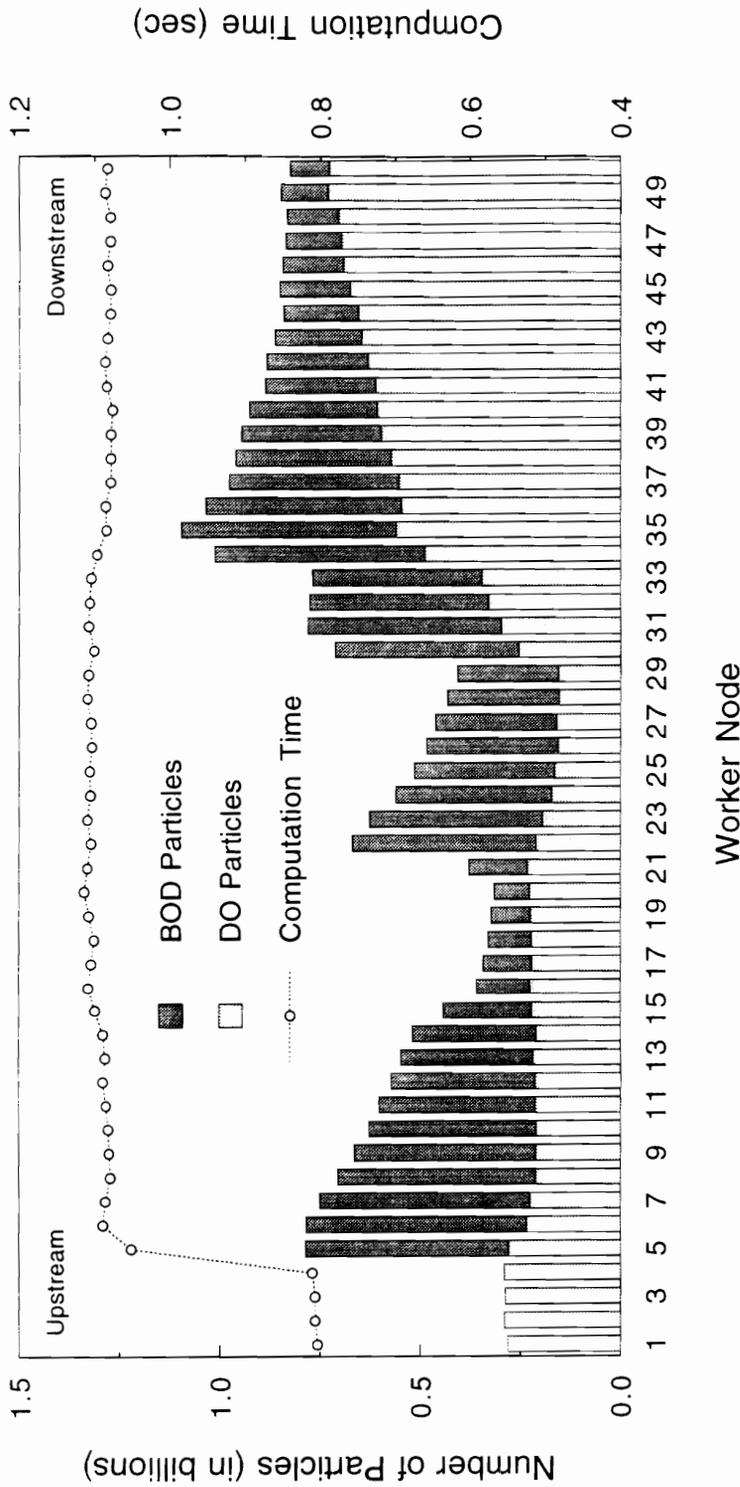


Figure 3.8 Distribution of the number of BOD and DO particles and the computation time among the worker nodes, corresponding to the CA simulation results of Figure 3.6. The worker nodes are numbered based on an upstream to downstream ordering of their subdomains.

associated with a different number of particles. Thus, and depending on the relation between the volumetric size of the cells, the load balancing effect of the packet fraction approach can still be of real significance when constituent concentrations are identical across cells.

Figure 3.9 shows execution times as a function of the number of worker nodes used in the uniform cross-section river simulation. The measured times are CPU times. Each node measures its own execution times using the built in function `mclock()` which returns relative time in milliseconds. Subtraction of the values returned by this function at two different points during code execution leads to an elapsed time.

This procedure is used in each worker node to measure elapsed times corresponding to the computation portions of the code. By accumulating all these elapsed times a value for the computation time is obtained. The same procedure is also used to obtain a time representing the sum of the computation and worker-worker communication (communication between a worker node and other worker nodes) portions of the code. And similarly to obtain a time representing the sum of the computation, worker-worker communication, and manager-worker communication (communication between the manager node and the worker node) portions of the code. This time accounting methodology is performed for each model iteration, corresponding to each simulation time step. The manager node uses the same basic procedure to obtain an overall execution time for the 5.79 days of model simulation. This includes the time for computation in all nodes, communication among worker nodes and between worker nodes and the manager node, and I/O for all nodes.

The execution times obtained from the worker nodes are shown in Figure 3.9, where each value is an average of the times for all worker nodes during 99 consecutive iterations obtained from a single simulation. These iterations are the 99 iterations immediately preceding the iteration corresponding to a simulation time of 5.79 days and

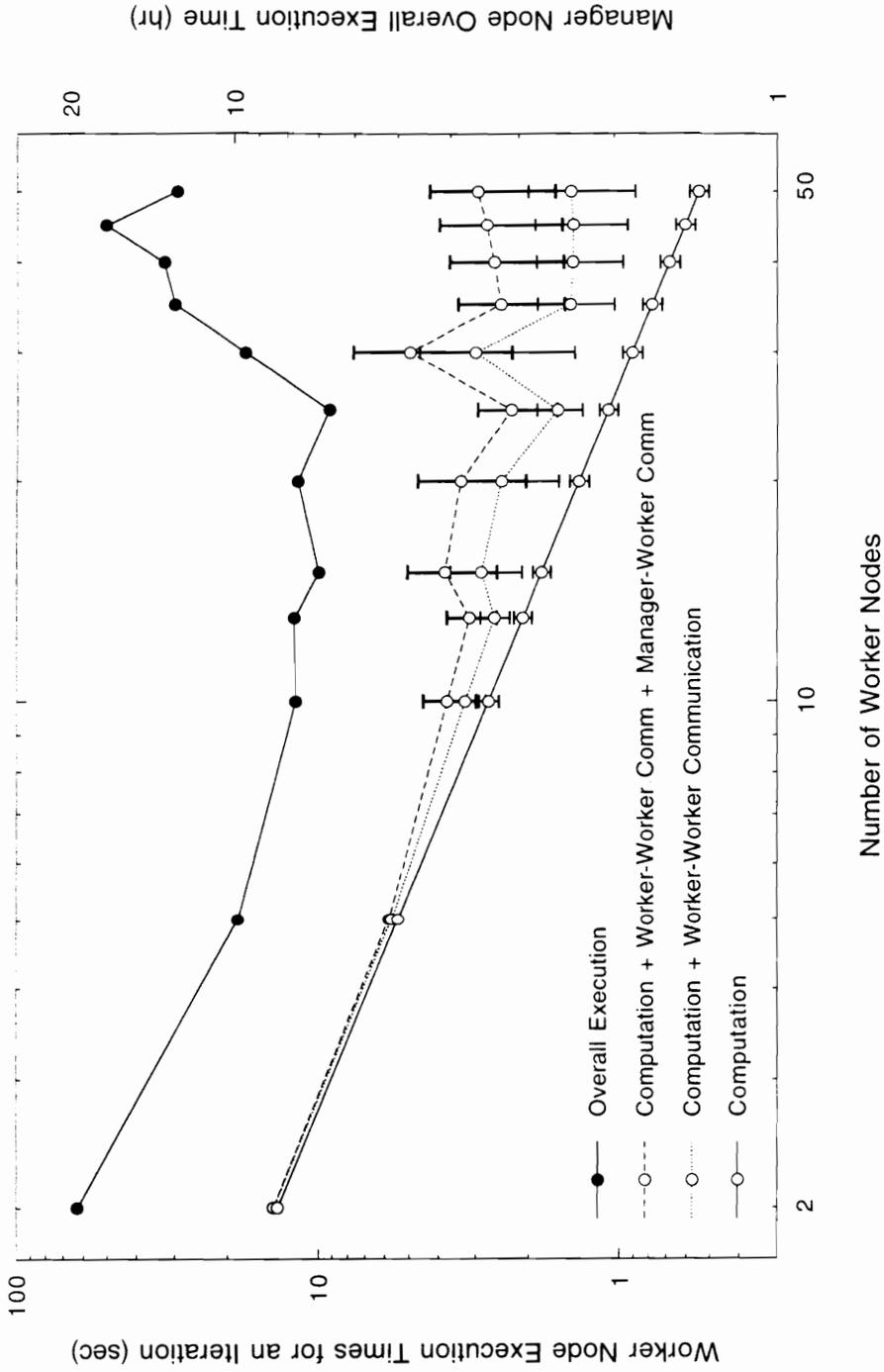


Figure 3.9 Execution times for the uniform cross-section steady-state BOD/DO simulation as a function of the number of worker nodes used in the simulation. (Model input parameters from Table 3.3.)

are expected to still represent steady-state conditions. Also shown is the standard deviation associated with those results which incorporates the variability in the execution times from different worker nodes as well as from iteration to iteration. (Due to the y-axis logarithmic scale some standard deviation values are too small to be visible in the plot.) On the other hand, each overall execution time from the manager node, shown in Figure 3.9, is a single measurement of the simulation performance using a particular number of worker nodes.

The results from Figure 3.9 show a decrease in the computation plus communication time as the number of worker nodes in the simulation increases. This represents the typical behavior observed during implementation of models on parallel processors (Crowl, 1994; Dabdub and Seinfeld, 1994). It also means that, although it is possible to significantly decrease the computation plus communication time by using more processors, this time tends to level off and will eventually increase if the number of processors further increases (Dabdub and Seinfeld, 1994). This is due to the communication overhead which offsets the decrease in computation time from using a larger number of processors. This typical behavior is also visible in the overall execution time from the manager node.

The number of processors at which the computation plus communication time begins to level off is machine dependent (through the processor and communication speeds) but also related to the specific problem being solved. In other words, it depends upon the relation between the computation and communication times. Multidimensional, multiple constituent problems with more computations per processor are likely to lead to greater decreases in computation plus communication time as more processors are made available.

The substantial variability associated with the computation plus communication time in Figure 3.9 is clearly due to variability associated with the communication, since

the standard deviation of the computation time is relatively small. The variability in the computation time is expected to be affected by the randomness associated with the CA model and the degree of load balancing, as well as by random events occurring in the computing system (Crowl, 1994). The variability in the communication time is also likely affected by such random events.

Based on these findings, the overall execution time from the manager node is expected to be characterized by substantial variability. However, the values presented in Figure 3.9 are a strong indication of the poor performance of the CA model. At an optimum number of processors the overall time for the execution of the 5.79 days of model simulation is still as high as 6.5 hours.

It is instructive to note that of the average computation time reported in Figure 3.9 about 70% represents time spent by the worker nodes tracking particles through the rules (which involves generating random numbers). The remaining 30% is time spent by a worker node on spatial interpolation of model coefficients and execution of the code loops to check to see if particles are present in every subdomain cell. Obviously, if a smaller packet fraction was used, the time spent on particle tracking would be a larger percentage of the computation time.

3.3.4 One-Dimensional Tidal BOD/DO

Figure 3.10 visualizes the temporal evolution of the advective velocity throughout the simulation, highlighting the points for which model results are presented. Results are only shown for slack water times to minimize the amount of plotted data while showing results over a complete tidal cycle.

Figure 3.11 shows those results for a single simulation of the BOD/DO CA model and the differential equation (numerically integrated). During the tidal cycle for which results are shown, the mass of constituent starts by moving upstream (during flood tide,

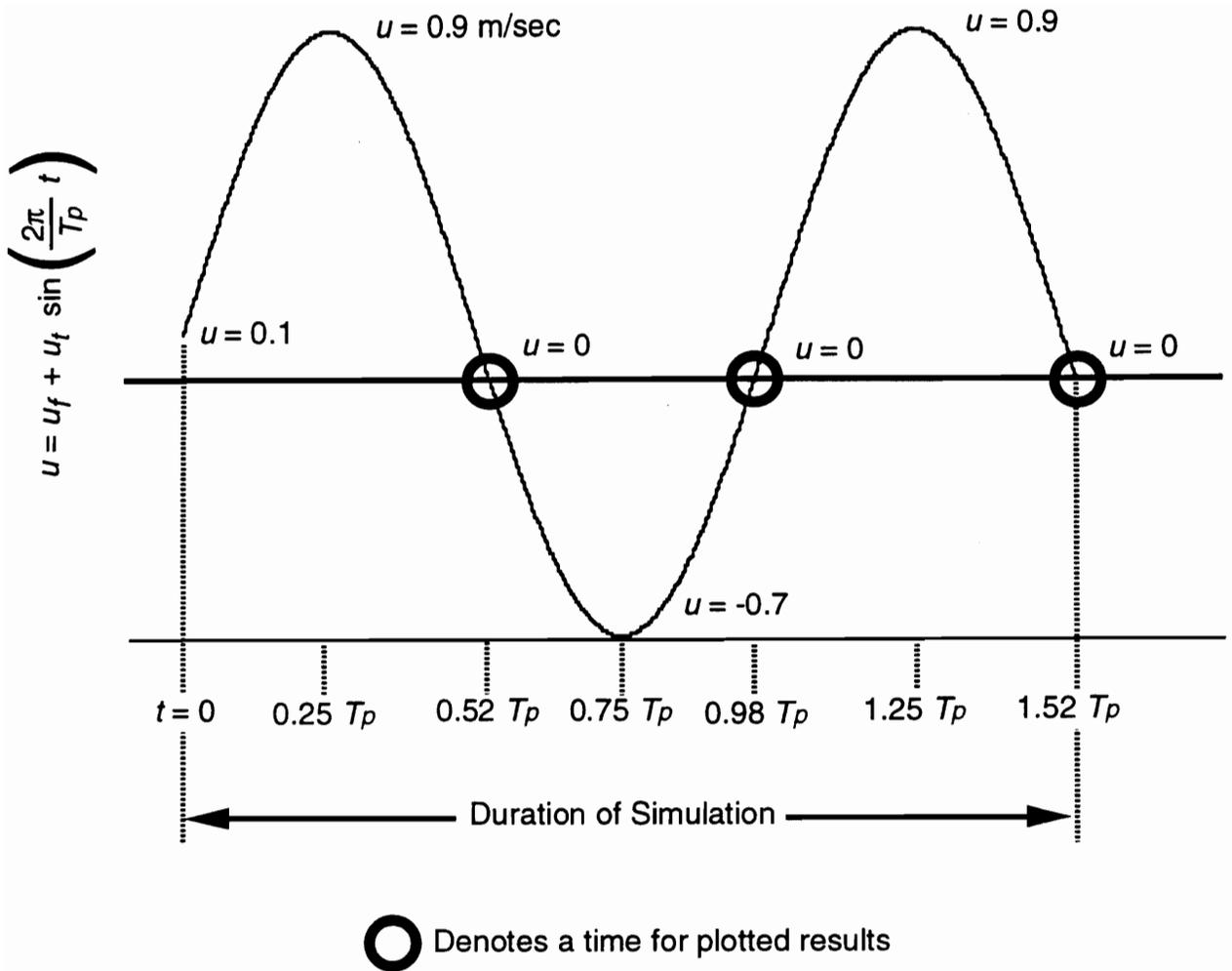


Figure 3.10 Graphical representation of the time evolution of the advective velocity for the simulation scenario corresponding to Table 3.5. The circles indicate the time at which model results are shown in Figure 3.11.

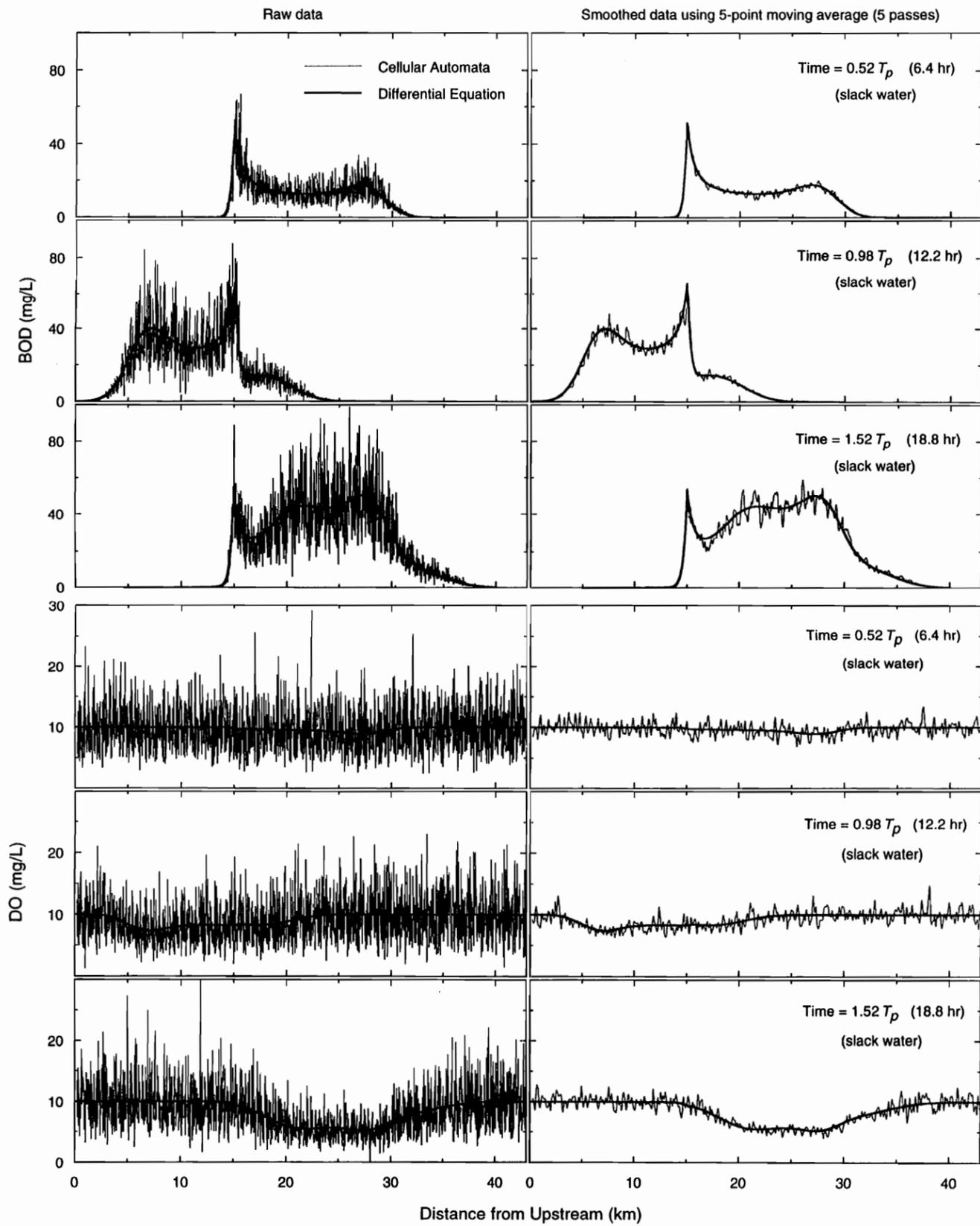


Figure 3.11 Concentration profiles for BOD and DO at successive slack water times over an entire tidal cycle. The BOD is being continuously discharged at a distance of 15 km. Comparison between the CA model and a differential equation model as described in the text. The CA results are from a single model simulation. (Model input parameters from Table 3.5.)

i.e., while the advective velocity is negative), then halts and changes direction (at slack water, i.e., when the advective velocity is zero), and moves back downstream (during ebb tide, i.e., while the advective velocity is positive). In comparison to flood tide, the amplitude of the advective velocity is largest and the tide duration is longest during ebb tide. Therefore, during a tidal cycle there is a net downstream movement of constituent mass. This is best seen for the BOD.

The results show a very good agreement for the BOD and DO between the two models over an entire tidal cycle, despite the expected noise associated with the CA solution. The noise is once again substantially reduced through the application of a smoothing algorithm. Although here results are only shown for slack water times the same good fit was obtained for the flood and ebb times as well.

These CA model results show that the model can handle temporal changes in the advective velocity very well, and suggests that a similar behavior could be expected from temporal changes in other model coefficients as well.

3.4 CONCLUSIONS

Cellular automata are a promising new approach for modeling water quality problems. The CA model represents fundamental water quality processes such as advection, dispersion, decay, and reaeration as reliably as the traditional approach of differential equations. It is capable of adequately simulating one and two-dimensional, single and multiple constituent, steady-state and transient, and spatially invariant and variant systems. Although the model is subject to advection induced numerical dispersion this dispersion can be minimized. In addition, it can easily be incorporated into real dispersion.

The use of the packet fraction approach leads to a significant reduction in the noise to signal ratio at lower constituent concentrations, and to an equalization of the work load among cells having different number of particles (as long as cells have at least $1/f$ particles) and thus to some dynamic load balancing. The approach does not seem to involve any significant tradeoffs. Moreover, decreasing the value for the packet fraction clearly increases the computation time while significantly reducing the variability of the CA model results. Also the application of smoothing algorithms substantially reduces that variability.

The CA model is easier to understand and implement than the traditional numerical models. Implementation of the CA model on parallel processors having a MIMD distributed memory configuration was feasible and posed no major difficulties. A large number of simple computations must be done to update the CA at each time step of the simulation making the CA model computationally intensive. Although model implementation was not optimized for performance, the model performed poorly even when using an optimum number of processors. Yet, it is possible that for more complex simulations, having higher computation-to-communication ratios, significant improvements in model performance could be attained with implementation on massively parallel computers.

REFERENCES

- Ahlstrom, S., H. Foote, R. Arnett, C. Cole, and R. Serne, "Multicomponent Mass Transport Model: Theory and Numerical Implementation." Technical Report No. BNWL 2127, Battelle Pacific Northwest Laboratories, Richland, WA (1977).
- Amato, I., "Speculating in Precious Computronium." *Science*, 253, 856-857 (1991).

- Bagtzoglou, A. C., A. F. B. Tompson, and D. E. Dougherty, "Projection Functions for Particle-Grid Methods." *Numerical Methods for Partial Differential Equations*, 8, 325-340 (1992).
- Camp, W. J., S. J. Plimpton, B. A. Hendrickson, and R. W. Leland, "Massively Parallel Methods for Engineering and Science Problems." *Communications of the ACM*, 37, 31-41 (1994).
- Crowl, L. A., "How to Measure, Present, and Compare Parallel Performance." *IEEE Parallel & Distributed Technology, Spring Issue*, 9-25 (1994).
- Dabdub, D., and J. H. Seinfeld, "Air Quality Modeling on Massively Parallel Computers." *Atmospheric Environment*, 28, 1679-1687 (1994).
- Duncan, R., "A Survey of Parallel Computer Architectures." *IEEE Computer Magazine*, Feb, 5-16 (1990).
- Fortner, B., *The Data Handbook: A Guide to Understanding the Organization and Visualization of Technical Data*. Spyglass, Champaign, IL (1992).
- Fox, G. C., M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors: General Techniques and Regular Problems, Volume I*. Prentice Hall, Englewood Cliffs, NJ, 592 p. (1988).
- Fox, G. C., "Achievements and Prospects for Parallel Computing." *Concurrency: Practice and Experience*, 3 (6), 725-739 (1991).
- Fox, G. C., D. R. Williams, and P. C. Messina, *Parallel Computing Works!*. Morgan Kaufmann, San Francisco, CA, 977 p. (1994).
- Giles, R. T., "Optimal Strategies for Discharging Pollutants Into Narrow Estuaries." *Water Research*, 29, 563-569 (1995).
- Green, S., *Parallel Processing for Computer Graphics*. MIT Press, Cambridge, MA, 233 p. (1991).
- Hemond, H. F., and E. J. Fechner, *Chemical Fate and Transport in the Environment*. Academic Press, San Diego, CA, 338 p. (1994).

- Holley, E. R., "Difference Modeling of Stream Pollution." *Journal of the Sanitary Engineering Division Proc. ASCE*, 95, 968-972 (1969).
- Macdonald, G. J., and R. N. Weisman, "Oxygen-Sag in a Tidal River." *Journal of the Environmental Engineering Division Proc. ASCE*, 103, 473-488 (1977).
- Messina, P. C., "Parallel Computing in the 1980s - One Person's View." *Concurrency: Practice and Experience*, 3 (6), 501-524 (1991).
- Ortega, J. M., *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, NY, 305 p. (1988).
- Thomann, R. V., and J. A. Mueller, *Principles of Surface Water Quality Modeling and Control*. Harper and Row, New York, NY, 644 p. (1987).
- Toffoli, T., and N. Margolus, *Cellular Automata Machines: a New Environment for Modeling*. MIT Press, Cambridge, MA, 259 p. (1987).
- Wilson, G. V., "A Glossary of Parallel Computing Terminology." *IEEE Parallel & Distributed Technology*, February Issue, 52-67 (1993).

4 EUTROPHICATION MODELING WITH CELLULAR AUTOMATA

The purpose of this chapter is to illustrate how the concepts presented in the previous chapters can be used to develop a more complex CA model. Eutrophication modeling constitutes an appropriate example since it involves multiple water quality constituents interacting through numerous processes. The development of such a model using the methodology already described will be illustrated. The goal of this chapter is solely to demonstrate that more complex water quality models are possible with the CA methodology. It does not attempt to provide an exhaustive representation of all water quality constituents and processes pertaining to eutrophication, nor does it attempt to fully examine the applicability of the CA model. Therefore, no CA simulations or comparisons with other models are presented in this chapter.

Figure 4.1 shows the constituents and processes included in a typical eutrophication model. Figure 4.2 illustrates external sources and sinks typically found in this kind of model. The remaining of this chapter shows how the CA methodology is used to represent each of the above processes in the context of the CA model discussed previously. As expected from the methodology presented in previous chapters the parameters required by the CA eutrophication model are also typical of parameters included in other water quality models.

4.1 WATER QUALITY CONSTITUENTS

As show in Figure 4.1 a total of eight water quality constituents are included in the CA eutrophication model. These are the CBOD (carbonaceous BOD), DO, phytoplankton as chlorophyll-a, organic nitrogen, ammonia, and nitrate all expressed as nitrogen, and organic and inorganic phosphorus both expressed as phosphorus.

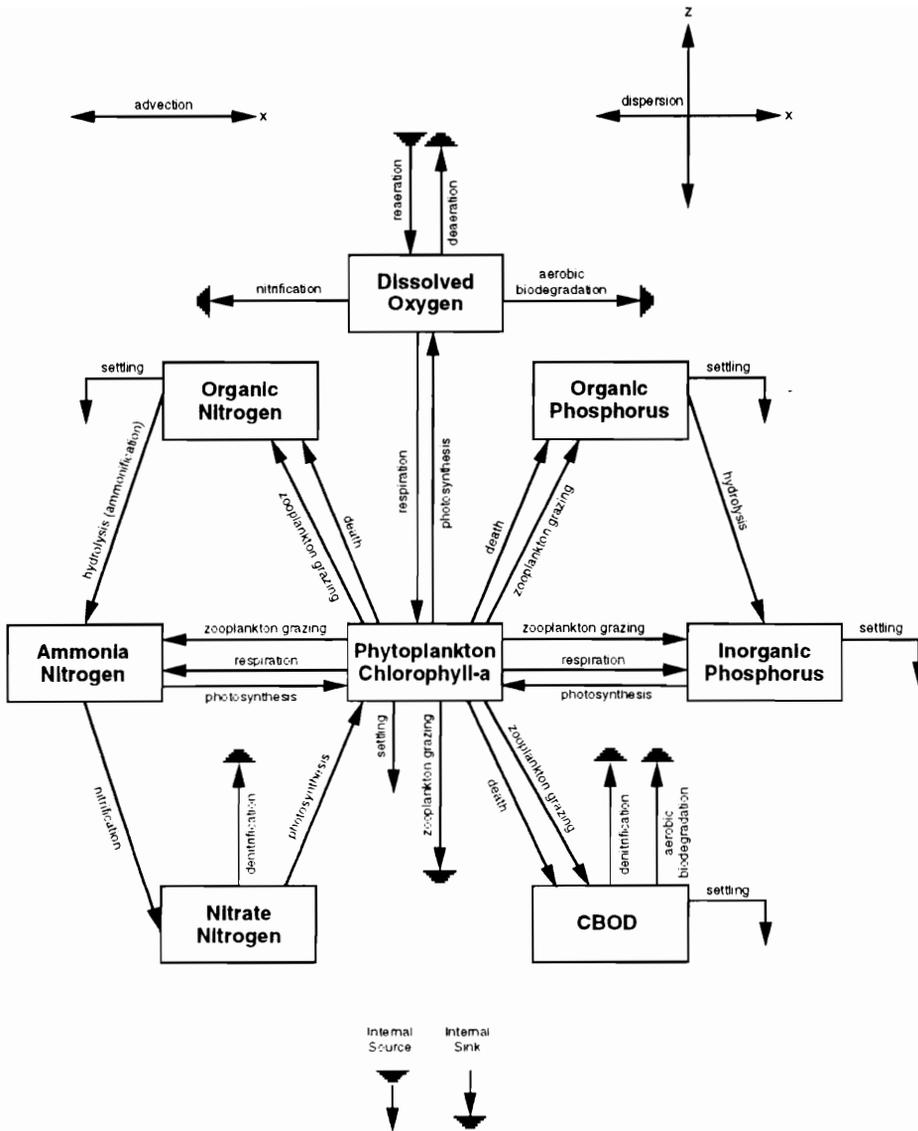


Figure 4.1 Water quality constituents and processes typically included in a eutrophication model.

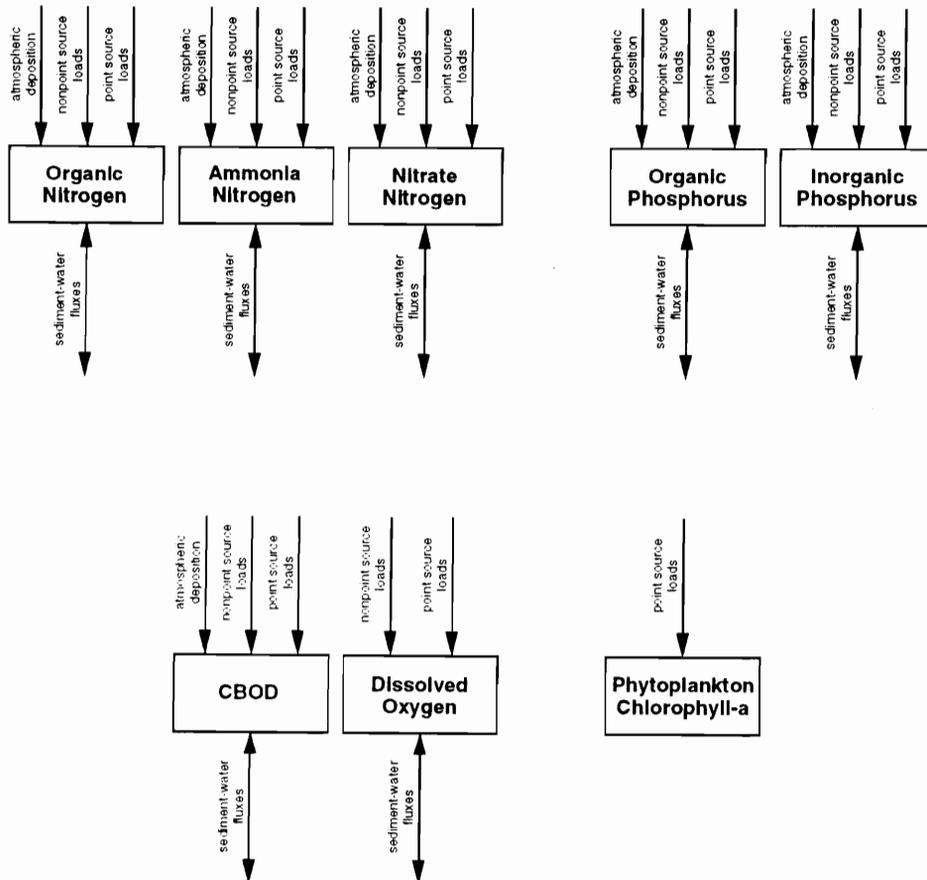


Figure 4.2 External Sources and sinks typically included in a eutrophication model.

Some constituents, such as CBOD, organic nitrogen, and organic and inorganic phosphorus, typically appear in particulate and dissolved forms. Although the present CA model does not attempt to simulate the particulate and dissolved forms of these constituents, it takes into account the particulate and dissolved fractions through model input parameters.

Similarly to the CA BOD/DO model already discussed, one cellular automaton is used to represent each of the water quality constituents. A stack of eight CA is then obtained. Some processes (such as dispersion and reaeration) affect many constituents independently or just a single constituent; while others (such as respiration and nitrification) represent interactions between constituents, therefore simultaneously affecting multiple layers of the CA stack.

The relation between the particle mass of the different constituents can be as diverse as one desires. The particle mass for the CBOD can be for instance twice as much as the one for the DO. In this case, when one particle of CBOD is removed by decay, two particles of DO are also removed. In another example the particle mass for the CBOD can be two and a half times the one for the DO. In this case when one particle of CBOD is removed by decay, two and a half particles of DO are also removed. This means removing two particles of DO, and then applying some criteria to decide if a third particle is or not removed. Obviously, a simpler approach is to define the particle mass of CBOD and DO as being the same. Thus, when one particle of CBOD is removed, one particle of DO is also removed.

These considerations suggest that a one-to-one relationship between the particle mass of different constituents seems to provide the simplest approach. In a model such as eutrophication, involving many constituents and processes, a relationship between the particle mass values can be obtained based on the stoichiometry of the photosynthesis/respiration processes and the chemical composition of one central

constituent, the phytoplankton. As seen in Figure 4.1, phytoplankton occupies a central position in the diagram, i.e., there is always at least one process relating phytoplankton with any of the other constituents.

First, the stoichiometry for the photosynthesis/respiration reactions allows one to relate the phytoplankton with the DO and CBOD. A typical chemical equation representing photosynthesis/respiration is given by Stumm and Morgan (1981):



This equation shows that for every mole of phytoplankton (which represents 1.272 kg of phytoplankton carbon, based on the formula $C_{106}H_{263}O_{110}N_{16}P$) formed by photosynthesis, 107 moles of molecular oxygen (or 3.424 kg of oxygen) are released; and for every mole of phytoplankton (or 1.272 kg of phytoplankton carbon) lost through respiration, 107 moles of molecular oxygen (3.424 kg of oxygen) are consumed. This also means that when phytoplankton cells, representing 1.272 kg of phytoplankton carbon, become part of the CBOD as a result of death or zooplankton grazing then that amount of CBOD is 3.424 kg. Therefore, equation (4.1) gives an oxygen-to-carbon ratio for photosynthesis/respiration, denoted simply as a_{OC}^{photos} , of 2.69 (i.e., 3.424/1.272).

Second, the information provided by ratios between the important chemical constituents making up the phytoplankton, such as the ratios of carbon-to-chl-a (a_{CChl-a}^{phyto}), nitrogen-to-carbon (a_{NC}^{phyto}), and phosphorus-to-carbon (a_{PC}^{phyto}), allows one to relate the phytoplankton chl-a with the remaining water quality constituents. Typical values for these ratios are 50 mg C/mg chl-a, 0.25 mg N/mg C, and 0.025 mg P/mg C, respectively (Thomann and Mueller, 1987; Ambrose *et al.*, 1988; Park *et al.*, 1993).

The values of particle mass for the various water quality constituents can then be obtained, based on the above discussed ratios and the user specified particle mass for the CBOD and DO, as follows:

$$m_p^{CBOD} = m_p^{DO} = \text{model input value} \quad (4.2a)$$

$$m_p^{Chla} = \frac{m_p^{DO}}{a_{OC}^{photos} a_{CChla}^{phyto}} \quad (4.2b)$$

$$m_p^{OrgN} = m_p^{Chla} a_{NC}^{phyto} a_{CChla}^{phyto} \quad (4.2c)$$

$$m_p^{NH_3} = m_p^{NO_3} = m_p^{OrgN} \quad (4.2d)$$

$$m_p^{OrgP} = m_p^{Chla} a_{PC}^{phyto} a_{CChla}^{phyto} \quad (4.2e)$$

$$m_p^{InorgP} = m_p^{OrgP} \quad (4.2f)$$

where m_p^{CBOD} , m_p^{DO} , m_p^{Chla} , m_p^{OrgN} , $m_p^{NH_3}$, $m_p^{NO_3}$, m_p^{OrgP} , and m_p^{InorgP} are, respectively, the particle mass for CBOD, DO, phytoplankton chl-a, organic nitrogen, ammonia nitrogen, nitrate nitrogen, organic phosphorus, and inorganic phosphorus.

4.2 WATER QUALITY PROCESSES

4.2.1 Advection and Dispersion

The advection and dispersion processes included in the CA eutrophication model follow the methodology for advection and dispersion already presented in previous chapters.

4.2.2 Aerobic Biodegradation

Aerobic biodegradation of CBOD is treated as a first-order decay process and thus follows the methodology for first-order decay for BOD presented earlier in the context of

the BOD/DO model. However, to consider the inhibitory effect of lower DO concentrations on the CBOD aerobic biodegradation rate, an additional factor representing the Michaelis-Menten kinetics is included (Ambrose *et al.*, 1988). The expression for the biodegradation probability, P_{deg} , is then given as:

$$P_{deg} = \frac{A_{deg}}{-0.003282A_{deg}^3 + 0.065914A_{deg}^2 + 0.563833A_{deg} + 0.973541} \quad 0 \leq P_{deg} \leq 1 \quad (4.3a)$$

$$A_{deg} = k_{deg} \left(\frac{C_{DO}}{K_{deg} + C_{DO}} \right) \Delta t_{deg} \quad (4.3b)$$

where k_{deg} is the first-order aerobic biodegradation rate constant (T^{-1}), C_{DO} is the DO concentration (ML^{-3}), K_{deg} is the Michaelis-Menten half-saturation constant for oxygen limitation on aerobic biodegradation (ML^{-3}), and Δt_{deg} is the time step for the biodegradation process.

At each simulation time step, the value of P_{deg} is updated prior to each repetition of the biodegradation rule. Its value thus reflects the amount of DO present just before the rule is (re)applied. If during the application of the biodegradation rule to a given cell the number of particles of DO becomes depleted before the rule is applied to all CBOD particles, then the remaining CBOD particles are not allowed to biodegrade.

4.2.3 Reaeration/Deaeration

The representation of the DO reaeration/deaeration process, usually known simply as reaeration, is identical to the reaeration methodology described earlier in the context of the BOD/DO model. In a multidimensional model the reaeration rule is only applied to the upper cells, i.e., the cells contacting the air-water interface.

4.2.4 Denitrification

Denitrification is treated as a nitrate first-order decay process and thus follows the general CA methodology for first-order decay. An additional factor, similar to the Michaelis-Menten expression, is included to represent the inhibitory effect of higher DO concentrations on the denitrification rate (Ambrose *et al.*, 1988; Park and Kuo, 1993; Park *et al.*, 1993). The denitrification probability, P_{den} , is given as:

$$P_{den} = \frac{A_{den}}{-0.003282A_{den}^3 + 0.065914A_{den}^2 + 0.563833A_{den} + 0.973541} \quad 0 \leq P_{den} \leq 1 \quad (4.4a)$$

$$A_{den} = k_{den} \left(\frac{K_{den}}{K_{den} + C_{DO}} \right) \Delta t_{den} \quad (4.4b)$$

where k_{den} is the first-order denitrification rate constant (T^{-1}), K_{den} is the Michaelis-Menten half-saturation constant for oxygen limitation on denitrification (ML^{-3}), and Δt_{den} is the time step for the denitrification process.

Since the DO concentration is not directly affected by denitrification, at each simulation time step, the denitrification probability is evaluated only once and independently of the number of repetitions for the rule. The denitrification rule involves applying the CA methodology for decay to all particles in each cell of the nitrate cellular automaton using the probability value from equation (4.4). In addition, each time a particle of nitrate is removed from a cell in the nitrate cellular automaton an equivalent number of particles of CBOD is also removed from the corresponding cell of the CBOD cellular automaton. If the number of particles of CBOD becomes less than the equivalent number of particles before the rule has been applied to all nitrate particles, then the remaining nitrate particles are not allowed to decay.

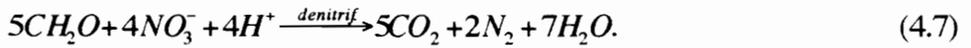
The equivalent number of particles of CBOD, $n_{CBOD:NO_3}^{den}$, is based on the stoichiometric ratio between CBOD and nitrate for the overall denitrification and respiration reaction, and is given by:

$$n_{CBOD:NO_3}^{den} = \frac{a_{OC}^{photos} m_p^{NO_3}}{a_{NC}^{den} m_p^{CBOD}} \quad (4.5)$$

where a_{NC}^{den} is the nitrogen-to-carbon ratio for the denitrification reaction. Equation (4.5) can be simplified by substituting equations 4.2(b) through 4.2(d). The resulting expression is:

$$n_{CBOD:NO_3}^{den} = \frac{a_{NC}^{phyto}}{a_{NC}^{den}} \quad (4.6)$$

To calculate the value of a_{NC}^{den} , the chemical equation for the denitrification reaction is needed. This equation is given by Stumm and Morgan (1981) as:



This equation shows that for every 4 moles of nitrate (or 56 g of nitrate nitrogen) denitrified, 5 moles of CH_2O (or 60 g of carbon) are consumed. Therefore, this gives a nitrogen-to-carbon ratio for denitrification a_{NC}^{den} of 0.93 (i.e., 56/60).

Since the equivalent number of particles $n_{CBOD:NO_3}^{den}$ is likely to be a non integer, a stochastic approach is used to determine how many particles of CBOD to actually remove. That number of particles is always the integer component of $n_{CBOD:NO_3}^{den}$, plus an extra particle when a randomly generated uniformly distributed number (between 0 and 1) does not exceed the fractional component of $n_{CBOD:NO_3}^{den}$.

4.2.5 Nitrification

Nitrification is treated as an ammonia first-order decay process and thus follows the general CA methodology for first-order decay. Additional factors representing Michaelis-Menten kinetics are included to consider the inhibitory effects of lower ammonia and/or DO concentrations on the nitrification rate (Ambrose *et al.*, 1988; Park and Kuo, 1993; Park *et al.*, 1993). The nitrification probability, P_{nit} , is given as:

$$P_{nit} = \frac{A_{nit}}{-0.003282A_{nit}^3 + 0.065914A_{nit}^2 + 0.563833A_{nit} + 0.973541} \quad 0 \leq P_{nit} \leq 1 \quad (4.8a)$$

$$A_{nit} = k_{nit} \left(\frac{C_{NH_3}}{K_{nit}^{NH_3} + C_{NH_3}} \right) \left(\frac{C_{DO}}{K_{nit}^{DO} + C_{DO}} \right) \Delta t_{nit} \quad (4.8b)$$

where k_{nit} is the first-order nitrification rate constant (T^{-1}), C_{NH_3} is the ammonia concentration (ML^{-3}), $K_{nit}^{NH_3}$ and K_{nit}^{DO} are the Michaelis-Menten half-saturation constants, respectively, for ammonia and oxygen limitation on nitrification (ML^{-3}), and Δt_{nit} is the time step for the nitrification process.

At each simulation time step, the value of P_{nit} is updated prior to each repetition of the nitrification rule. Its value thus reflects the amount of ammonia and DO present just before the rule is (re)applied. The nitrification rule involves applying the CA methodology for decay to all particles in each cell of the ammonia cellular automaton using the probability value from equation (4.8). In addition, each time a particle of ammonia is removed from a cell in the ammonia cellular automaton a particle of nitrate is added and an equivalent number of particles of DO is removed from the corresponding cell of their CA. If the number of particles of DO becomes less than the equivalent number of particles before the rule has been applied to all ammonia particles, then the remaining ammonia particles are not allowed to decay.

The equivalent number of particles of DO, $n_{DO:NH_3}^{nit}$, is based on the stoichiometric ratio between DO and ammonia for the nitrification reaction, and is given by:

$$n_{DO:NH_3}^{nit} = \frac{a_{ON}^{nit} m_p^{NH_3}}{m_p^{DO}} \quad (4.9)$$

where a_{ON}^{nit} is the oxygen-to-nitrogen ratio for the nitrification reaction. Equation (4.9) can be simplified by substituting equations 4.2(b) through 4.2(d). The resulting expression is:

$$n_{DO:NH_3}^{nit} = \frac{a_{ON}^{nit} a_{NC}^{phyto}}{a_{OC}^{photos}} \quad (4.10)$$

To calculate the value of a_{ON}^{nit} , the chemical equation for the nitrification reaction is needed. This equation is given by Stumm and Morgan (1981) as:



This equation shows that for every mole of ammonia (or 14 g of ammonia nitrogen) nitrified, 2 moles of molecular oxygen (or 64 g of oxygen) are consumed. Therefore, this gives an oxygen-to-nitrogen ratio for nitrification a_{ON}^{nit} of 4.57 (i.e., 64/14). Again, since the equivalent number of particles $n_{DO:NH_3}^{nit}$ is likely to be a non integer, the stochastic approach described previously is used to determine how many particles of DO are actually removed.

4.2.6 Hydrolysis

The model includes hydrolysis (mineralization) of the organic nitrogen and phosphorus. Hydrolysis is treated as a first-order decay process of organic nitrogen and phosphorus and thus follows the general CA methodology for first-order decay. An additional factor representing the Michaelis-Menten kinetics is included to consider the

inhibitory effect of lower concentrations of organic nitrogen (or phosphorus) on the hydrolysis rate (Park and Kuo, 1993; Park *et al.*, 1993). The hydrolysis probability, P_{hyd} , is given as:

$$P_{hyd} = \frac{A_{hyd}}{-0.003282A_{hyd}^3 + 0.065914A_{hyd}^2 + 0.563833A_{hyd} + 0.973541} \quad 0 \leq P_{hyd} \leq 1 \quad (4.12a)$$

$$A_{hyd} = k_{hyd} \left(\frac{C_{org}}{K_{hyd} + C_{org}} \right) \Delta t_{hyd} \quad (4.12b)$$

where k_{hyd} is the first-order hydrolysis rate constant (T^{-1}), C_{org} is the organic nitrogen (or phosphorus) concentration (ML^{-3}), K_{hyd} is the Michaelis-Menten half-saturation constant for organic nitrogen (or phosphorus) limitation on hydrolysis (ML^{-3}), and Δt_{hyd} is the time step for the hydrolysis process.

At each simulation time step, the value of P_{hyd} is updated prior to each repetition of the hydrolysis rule. Its value thus reflects the amount of organic nitrogen (or phosphorus) present just before the rule is (re)applied. The hydrolysis rule involves applying the CA methodology for decay to all particles in each cell of the organic nitrogen (or phosphorus) cellular automaton using the probability value from equation (4.12). In addition, each time a particle of organic nitrogen (or phosphorus) is removed from a cell in the organic nitrogen (or phosphorus) cellular automaton a particle of ammonia (or inorganic phosphorus) is added to the corresponding cell of the ammonia (or inorganic phosphorus) cellular automaton.

4.2.7 Photosynthesis

Photosynthesis is treated as a phytoplankton first-order growth process which is similar to the general CA methodology for first-order decay. Additional factors are included representing the effects of light intensity and nutrient limitation (through

Michaelis-Menten kinetics) on the photosynthetic rate (Thomann and Mueller, 1987; Ambrose *et al.*, 1988; Park and Kuo, 1993; Park *et al.*, 1993). The photosynthesis probability, P_{pho} , is given as:

$$P_{pho} = \frac{A_{pho}}{-0.003282A_{pho}^3 + 0.065914A_{pho}^2 + 0.563833A_{pho} + 0.973541} \quad 0 \leq P_{pho} \leq 1 \quad (4.13a)$$

$$A_{pho} = \mu \left[\frac{I}{I_s} e^{\left(1 - \frac{I}{I_s}\right)} \right] \left[\min \left\{ \left(\frac{C_{NH_3} + C_{NO_3}}{K_{pho}^{DIN} + C_{NH_3} + C_{NO_3}} \right), \left(\frac{C_{InorgP}(1 - f_p^{InorgP})}{K_{pho}^{DIP} + C_{InorgP}(1 - f_p^{InorgP})} \right) \right\} \right] \Delta t_{pho} \quad (4.13b)$$

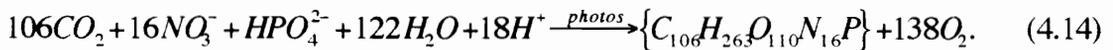
where μ is the phytoplankton first-order growth rate constant (T^{-1}), I is the light intensity, I_s is the phytoplankton photosynthesis saturating light intensity, C_{NO_3} is the nitrate concentration (ML^{-3}), C_{InorgP} is the inorganic phosphorus concentration (ML^{-3}), f_p^{InorgP} is the inorganic phosphorus particulate fraction, K_{pho}^{DIN} and K_{pho}^{DIP} are the Michaelis-Menten half-saturation constants, respectively, for dissolved inorganic nitrogen (ammonia plus nitrate) and dissolved inorganic phosphorus limitation on photosynthesis (ML^{-3}), and Δt_{pho} is the time step for the photosynthesis process.

At each simulation time step, the value of P_{pho} is updated prior to each repetition of the photosynthesis rule. Its value thus reflects the available light intensity (which changes as a function of the phytoplankton concentration due to the phytoplankton self-shading effect) and the amount of ammonia, nitrate, and inorganic phosphorus present just before the rule is (re)applied.

The photosynthesis rule involves applying the CA methodology for decay to all particles in each cell of the phytoplankton chl-a cellular automaton using the probability value from equation (4.13). However, since photosynthesis is a growth (negative decay) process, the rule adds phytoplankton chl-a particles instead of removing them. In addition, each time a particle of phytoplankton chl-a is added to a cell in the phytoplankton chl-a cellular automaton a particle of DO is added, and a particle of

ammonia and a particle of inorganic phosphorus are removed from the corresponding cell of their CA. This corresponds to the situation in which phytoplankton uses ammonia as a source of nitrogen.

However, phytoplankton can use nitrate, instead of ammonia, as the source of nitrogen (Ambrose *et al.*, 1988). In this case, the photosynthesis rule removes a particle of nitrate (instead of a particle of ammonia) and possibly adds more than just a single particle of DO. This larger amount of DO is the result of the different oxygen stoichiometries for the photosynthetic reactions using ammonia and nitrate. A typical chemical equation for photosynthesis using nitrate as a source of nitrogen is given by Stumm and Morgan (1981):



The ratio of the released oxygen given by equations (4.14) and (4.1) is 1.29 (i.e., 138/107). This means that when nitrate is the source of nitrogen the photosynthesis rule adds 1.29 particles of DO. This means one particle is added plus an extra particle when a randomly generated uniformly distributed number (between 0 and 1) does not exceed 0.29.

The ammonia preference factor (Ambrose *et al.*, 1988), which is a function of the ammonia and nitrate concentrations and varies between 0 and 1, is included in the photosynthesis rule to help determine which source of nitrogen to use in the photosynthesis process. For each newly photosynthesized particle of phytoplankton chl-a, a uniformly distributed random number (between 0 and 1) is generated and compared with the ammonia preference factor. If this random number does not exceed the ammonia preference factor then ammonia is used; otherwise nitrate is used instead.

When the number of particles of ammonia and nitrate, and/or inorganic phosphorus in a given cell become depleted before the photosynthesis rule has been

applied to all phytoplankton chl-a particles then the remaining phytoplankton chl-a particles are not allowed to 'grow'.

4.2.8 Respiration

Phytoplankton endogenous respiration is treated as a phytoplankton first-order decay process (Thomann and Mueller, 1987; Ambrose *et al.*, 1988; Park and Kuo, 1993; Park *et al.*, 1993) and thus follows the general CA methodology for first-order decay. The expression for the respiration probability, P_{res} , is then given as:

$$P_{res} = \frac{A_{res}}{-0.003282A_{res}^3 + 0.065914A_{res}^2 + 0.563833A_{res} + 0.973541} \quad 0 \leq P_{res} \leq 1 \quad (4.15a)$$

$$A_{res} = k_{res} \Delta t_{res} \quad (4.15b)$$

where k_{res} is the first-order phytoplankton respiration rate constant (T^{-1}) and Δt_{res} is the time step for the respiration process.

Since the value of the respiration probability is not affected by the application of the rule itself, at each simulation time step the respiration probability is evaluated only once and independently of the number of repetitions for the rule. The respiration rule involves applying the CA methodology for decay to all particles in each cell of the phytoplankton chl-a cellular automaton using the probability value from equation (4.15). In addition, each time a particle of phytoplankton chl-a is removed from a cell in the phytoplankton chl-a cellular automaton a particle of ammonia and a particle of inorganic phosphorus are added to, and a particle of DO is removed from the corresponding cell of their CA. If the number of particles of DO in a given cell becomes depleted before the rule is applied to all phytoplankton chl-a particles then the remaining phytoplankton chl-a particles are not allowed to decay through respiration.

4.2.9 Death

Phytoplankton death (due to various causes such as parasitism, infection, and toxicity) is treated as a phytoplankton first-order decay process (Thomann and Mueller, 1987; Ambrose *et al.*, 1988) and thus follows the general CA methodology for first-order decay. The expression for the death probability, P_{dea} , is then given as:

$$P_{dea} = \frac{A_{dea}}{-0.003282A_{dea}^3 + 0.065914A_{dea}^2 + 0.563833A_{dea} + 0.973541} \quad 0 \leq P_{dea} \leq 1 \quad (4.16a)$$

$$A_{dea} = k_{dea} \Delta t_{dea} \quad (4.16b)$$

where k_{dea} is the first-order phytoplankton death rate constant (T^{-1}) and Δt_{dea} is the time step for the death process.

Since the value of the death probability is not affected by the application of the rule itself, at each simulation time step the death probability is evaluated only once and independently of the number of repetitions for the rule. The death rule involves applying the CA methodology for decay to all particles in each cell of the phytoplankton chl-a cellular automaton using the probability value from equation (4.16). In addition, each time a particle of phytoplankton chl-a is removed from a cell in the phytoplankton chl-a cellular automaton a particle of CBOD, a particle of organic nitrogen, and a particle of organic phosphorus are added to the corresponding cell of their CA.

4.2.10 Grazing

The effect of zooplankton grazing on the phytoplankton is treated as a phytoplankton first-order decay process (Thomann and Mueller, 1987; Ambrose *et al.*, 1988; Park and Kuo, 1993; Park *et al.*, 1993) and thus follows the general CA

methodology for first-order decay. The expression for the grazing probability, P_{gra} , is then given as:

$$P_{gra} = \frac{A_{gra}}{-0.003282A_{gra}^3 + 0.065914A_{gra}^2 + 0.563833A_{gra} + 0.973541} \quad 0 \leq P_{gra} \leq 1 \quad (4.17a)$$

$$A_{gra} = k_{gra} \Delta t_{gra} \quad (4.17b)$$

where k_{gra} is the first-order grazing rate constant (T^{-1}) and Δt_{gra} is the time step for the grazing process.

Since the value of the grazing probability is not affected by the application of the rule itself, at each simulation time step the grazing probability is evaluated only once and independently of the number of repetitions for the rule. The grazing rule involves applying the CA methodology for decay to all particles in each cell of the phytoplankton chl-a cellular automaton using the probability value from equation (4.17). In addition, each time a particle of phytoplankton chl-a is removed from a cell in the phytoplankton chl-a cellular automaton, it implies the following: (1) not a single particle from other constituents is added or removed; or (2) a particle of CBOD, a particle of organic nitrogen or ammonia, and a particle of organic or inorganic phosphorus are added to the corresponding cell of their CA.

The decision between options (1) and (2) above is made based on the parameter representing the efficiency of assimilation or conversion of phytoplankton biomass to zooplankton biomass (Thomann and Mueller, 1987). For each particle of grazed phytoplankton chl-a, a uniformly distributed random number (between 0 and 1) is generated and compared with the efficiency of assimilation. If this random number does not exceed the efficiency of assimilation then option (1) is selected; otherwise option (2) is used. Note that, since the model does not attempt to simulate the zooplankton biomass,

any phytoplankton assimilated or converted into zooplankton biomass is in fact phytoplankton that simply disappears from the system.

When option (2) is selected, meaning the grazed phytoplankton is not assimilated by the zooplankton, then the nitrogen content of the former phytoplankton can still be in organic form or already converted to ammonia. Similarly, the phosphorus can be in an organic or inorganic form. The decision between organic and inorganic forms for the nitrogen and phosphorus is made based on the fractions of not assimilated grazed phytoplankton nitrogen and phosphorus recycled to the organic pool (Ambrose *et al.*, 1988). For each particle of grazed but not assimilated phytoplankton chl-a, two uniformly distributed random numbers (between 0 and 1) are generated and compared with the above fractions. If a random number does not exceed a fraction value then the respective constituent organic form is considered; otherwise the inorganic form of the constituent is used.

4.2.11 Settling

Settling is typically treated as a first-order decay process (Thomann and Mueller, 1987; Ambrose *et al.*, 1988; Park and Kuo, 1993; Park *et al.*, 1993) and thus follows the general CA methodology for first-order decay. The expression for the settling probability, P_{set} , is then given as:

$$P_{set} = \left(\frac{A_{set}}{-0.003282A_{set}^3 + 0.065914A_{set}^2 + 0.563833A_{set} + 0.973541} \right) f_p \quad 0 \leq P_{set} \leq 1 \quad (4.18a)$$

$$A_{set} = k_{set} \Delta t_{set} \quad (4.18b)$$

$$k_{set} = \frac{v_s}{\Delta z} \quad (4.18c)$$

where, for a given water quality constituent, k_{set} is the first-order settling rate constant (T^{-1}), v_s is the settling velocity (LT^{-1}), f_p is the particulate fraction, and Δt_{set} is the time step for the settling process. The Δz represents, as mentioned in the previous chapter, the cell size in the vertical direction (L).

The parameter f_p is included in equation (4.18) since settling affects only the particulate component of a constituent (Ambrose *et al.*, 1988). The settling process applies solely to some of the water quality constituents namely the CBOD, phytoplankton, organic nitrogen, and organic and inorganic phosphorus. The values for the particulate fraction f_p are between 0 and 1 for those constituents, with the obvious exception of the phytoplankton for which f_p is equal to 1.

Since the value of the settling probability is not affected by the application of the rule itself, at each simulation time step the settling probability is evaluated only once and independently of the number of repetitions for the rule. In a model without a vertical dimension, the settling rule involves applying the CA methodology for decay to all particles in each cell of the constituent cellular automaton using the probability value from equation (4.18). However, in a model including a vertical dimension each time a particle of constituent is removed from a cell of its cellular automaton (with the exception of the bottom cells) an identical particle is added to the cell located just below the cell from which the particle was removed.

The modeling approach just described obviously does not attempt to include more complicated processes involved in sediment dynamics.

4.3 EXTERNAL SOURCES AND SINKS

As mentioned earlier, typical external sources and sinks included in a eutrophication model are shown in Figure 4.2. The way these sources are handled in the CA model has been already discussed in the previous chapter.

Sediment-water fluxes which are both a source and sink are handled in a similar way. During a simulation time step the constituent mass flux is converted to a number of particles to be added to or removed from a sediment-water boundary cell of the constituent cellular automaton based on the length of the simulation time step, the boundary area of the cell, and the constituent particle mass.

4.4 IMPLEMENTATION ON PARALLEL PROCESSORS

The methodology just presented shows that a more complex CA model, such as for eutrophication, follows the same general approach of the simpler CA water quality model discussed in the previous chapters. Therefore, its implementation on parallel processors is straightforward.

The new rules included in the eutrophication model are all based on the first-order decay rule even though in some instances, when Michaelis-Menten kinetics is used, they require an additional step of evaluating one or more constituent concentrations. (If second-order decay rules were present it would also require evaluating constituent concentration(s) but nevertheless these rules would again be similar to first-order decay.) Since concentrations are readily available from the number of particles in cells this extra step does not add a considerable computation penalty.

As a final note, the new rules included in the CA eutrophication model (with the possible exception of the settling rule) do not require communication between processors,

therefore increasing model computation-to-communication ratio. This will lead to larger performance gains from model implementation on parallel processors.

REFERENCES

- Ambrose, R. B., T. A. Wool, J. P. Connolly, and R. W. Schanz, "WASP4, A Hydrodynamic and Water Quality Model - Model Theory, User's Manual, and Programmer's Guide." U.S. Environmental Protection Agency, EPA/600/3-87/039, 297 p. (1988).
- Stumm, W., and J. J. Morgan, *Aquatic Chemistry*. Wiley-Interscience, New York, NY, 780 p. (1981).
- Thomann, R. V., and J. A. Mueller, *Principles of Surface Water Quality Modeling and Control*. Harper and Row, New York, NY, 644 p. (1987).
- Park, K., and A. Y. Kuo, "A Vertical Two-Dimensional Model of Estuarine Hydrodynamics and Water Quality." Special Report in Applied Marine Science and Ocean Engineering No. 321, Virginia Institute of Marine Science, Gloucester Point, VA (1993).
- Park, K., A. Y. Kuo, and B. J. Neilson, "A Modeling Study of Hypoxia and Eutrophication in the Tidal Rappahannock River, Virginia." Special Report in Applied Marine Science and Ocean Engineering No. 322, Virginia Institute of Marine Science, Gloucester Point, VA, 158 p. (1993).

5 CONCLUSIONS

Parallel computing has recently appeared as an alternative approach to increase computing performance. In the world of engineering and scientific computing the efficient use of parallel computers is dependent on the availability of methodologies capable of exploiting the new computing environment. The research presented here focused on a modeling approach, known as cellular automata (CA), which is characterized by a high degree of parallelism, and thus is well suited to implementation on parallel processors. The inherent degree of parallelism also exhibited by the random-walk particle method provided a suitable basis for the development of a CA water quality model. The random-walk particle method is shown to be successfully represented using a CA approach.

The simulation results in this research prove that it is possible to replace traditional differential equations by CA formulations in water quality modeling. However, they are only simple illustrations of the potential of these new methods and resources to solve complex water quality management problems.

One major advantage with the CA model is the level of mathematics required to teach and understand water quality modeling. The model was found to be simpler to understand and implement than the traditional numerical models. The CA focus can be on the physical and chemical mechanisms at a microscopic level. The resulting transition rules can be understood by anyone with a basic algebra and statistics background. An understanding of calculus and numerical methods is no longer required to fully understand the modeling process.

In relation to the specific objectives of this research, the conclusions are:

- CA methodology can be used to develop model representations of the more common water quality processes, namely advection, dispersion, and first-order decay;
- those CA representations were shown to be accurate in spite of the discrete nature of the model. Numerical dispersion was quantified and procedures incorporated to minimize or eliminate its effects;
- due to the independence between CA rules for different processes, these rules were easily integrated into water quality models, even in the case of the relatively more complex eutrophication model;
- the CA model results for typical water quality modeling scenarios were successfully validated through visual comparison with existing analytical and numerical solutions. The substantial noise associated with the results of the CA model did not pose a significant difficulty during model comparisons since simple smoothing algorithms were successful in removing most of that variability;
- the CA model was easily implemented on parallel processors having a MIMD distributed memory configuration. A large number of simple computations must be done to update the CA at each time step of the simulation making the CA model computationally intensive. Although model implementation was not optimized for performance, the model performed poorly even when using an optimum number of processors. It is possible, however, that for more complex simulations, having higher computation-to-communication ratios, significant improvements in model performance could be attained with implementation on massively parallel computers;
- the packet fraction approach leads to a significant reduction in the noise to signal ratio at lower constituent concentrations, and allows for an equalization

of the work load among cells having different number of particles thereby leading to some dynamic load balancing. Using this approach does not seem to involve any significant tradeoffs. Moreover, decreasing the value of the packet fraction parameter clearly increases the computation time while significantly reducing the variability in the results of the CA model.

APPENDIX: LISTING OF THE C SOURCE CODE FOR THE MAIN COMPONENTS OF THE CA WATER QUALITY MODEL

Global variables used (alphabetically):

advectiveVelocity[1..numCellsLongit][1..numCellsVert] = u {floating-point} (array containing the advective velocity; model input value) [m/sec]

atmosphDeposition[1..numCellsLongit].cbod = {floating-point} (vector containing the atmospheric deposition rate of CBOD; model input value) [$\text{g/m}^2 \cdot \text{sec}$]

biodegradationCoef[1..numCellsLongit][1..numCellsVert] = k_{deg} {floating-point} (array containing the aerobic biodegradation coefficient; model input value) [sec^{-1}]

boundCondDownstream[1..numCellsVert].cbod = {floating-point} (vector containing the downstream concentration boundary condition for CBOD; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].chla = {floating-point} (vector containing the downstream concentration boundary condition for phytoplankton chl-a; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].inorgp = {floating-point} (vector containing the downstream concentration boundary condition for inorganic phosphorus; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].nh3 = {floating-point} (vector containing the downstream concentration boundary condition for ammonia; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].no3 = {floating-point} (vector containing the downstream concentration boundary condition for nitrate; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].o2 = {floating-point} (vector containing the downstream concentration boundary condition for DO; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].orgn = {floating-point} (vector containing the downstream concentration boundary condition for organic nitrogen; model input value) [g/m^3]

boundCondDownstream[1..numCellsVert].orgp = {floating-point} (vector containing the downstream concentration boundary condition for organic phosphorus; model input value) [g/m^3]

boundCondUpstream[1..numCellsVert].cbod = {floating-point} (vector containing the upstream concentration boundary condition for CBOD; model input value) [g/m^3]

boundCondUpstream[1..numCellsVert].chla = {floating-point} (vector containing the upstream concentration boundary condition for phytoplankton chl-a; model input value) [g/m^3]

boundCondUpstream[1..numCellsVert].inorgp = {floating-point} (vector containing the upstream concentration boundary condition for inorganic phosphorus; model input value) [g/m^3]
 boundCondUpstream[1..numCellsVert].nh3 = {floating-point} (vector containing the upstream concentration boundary condition for ammonia; model input value) [g/m^3]
 boundCondUpstream[1..numCellsVert].no3 = {floating-point} (vector containing the upstream concentration boundary condition for nitrate; model input value) [g/m^3]
 boundCondUpstream[1..numCellsVert].o2 = {floating-point} (vector containing the upstream concentration boundary condition for DO; model input value) [g/m^3]
 boundCondUpstream[1..numCellsVert].orgn = {floating-point} (vector containing the upstream concentration boundary condition for organic nitrogen; model input value) [g/m^3]
 boundCondUpstream[1..numCellsVert].orgp = {floating-point} (vector containing the upstream concentration boundary condition for organic phosphorus; model input value) [g/m^3]
 cellSize.x = Δx {floating-point} (cell size in the longitudinal direction) [m]
 cellSize.z = Δz {floating-point} (cell size in the vertical direction; model input value) [m]
 cellSizeY[1..numCellsLongit][1..numCellsVert] = {floating-point} (array containing the cell size in the lateral direction or cell width; model input value) [m]
 denitrificationCoef[1..numCellsLongit][1..numCellsVert] = k_{den} {floating-point} (array containing the denitrification coefficient; model input value) [sec^{-1}]
 dispersionCoefX[1..numCellsLongit][1..numCellsVert] = E^x {floating-point} (array containing the longitudinal dispersion coefficient; model input value) [m^2/sec]
 dispersionCoefZ[1..numCellsLongit][1..numCellsVert] = E^z {floating-point} (array containing the vertical dispersion coefficient; model input value) [m^2/sec]
 dispersionProbAmp[1..numCellsLongit][1..numCellsVert] = $P_{dis\ amp}$ {floating-point} (array containing the dispersion rule probability amplitude) [unitless]
 downstreamBuffer[1..numCellsVert] = {integer} (vector containing the particles that are to be sent to the next worker node)
 edgeCellBotNextNode = {integer} (index of the lower cell for the upstream boundary of the subdomain of the next worker node) [unitless]
 edgeCellBotPrevNode = {integer} (index of the lower cell for the downstream boundary of the subdomain of the previous worker node) [unitless]
 edgeCellTopNextNode = {integer} (index of the upper cell for the upstream boundary of the subdomain of the next worker node) [unitless]
 edgeCellTopPrevNode = {integer} (index of the upper cell for the downstream boundary of the subdomain of the previous worker node) [unitless]
 edgeCell[1.. numCellsLongit].top = {integer} (array containing the index of the upper cells of the subdomain, i.e., the surface elevation in terms of cells; model input value) [unitless]
 edgeCell[1..numCellsLongit].bot = {integer} (array containing the index of the lower cells of the subdomain, i.e., the bottom elevation in terms of cells; model input value) [unitless]
 firstNode = {integer} (ID number of the worker node dealing with the most upstream subdomain) [unitless]

fractionGrazing.orgn = {floating-point} (fraction of not assimilated grazed phytoplankton nitrogen recycled to the organic pool; model input value) [unitless]
 fractionGrazing.orgp = {floating-point} (fraction of not assimilated grazed phytoplankton phosphorus recycled to the organic pool; model input value) [unitless]
 fractionNotAssimilatedGrazing = {floating-point} (1 - efficiency of assimilation by the zooplankton; model input value) [unitless]
 generalProb[1..numCellsLongit][1..numCellsVert] = P {floating-point} (array containing the rule probability) [unitless]
 grazingCoef[1..numCellsLongit][1..numCellsVert] = k_{gra} {floating-point} (array containing the grazing coefficient; model input value) [sec^{-1}]
 halfSatConst.biodegradation = K_{deg} {floating-point} (Michaelis-Menten half-saturation constant for oxygen limitation on aerobic biodegradation; model input value) [g/m^3]
 halfSatConst.denitrification = K_{den} {floating-point} (Michaelis-Menten half-saturation constant for oxygen limitation on denitrification; model input value) [g/m^3]
 halfSatConst.hydrolysisOrgN = K_{hyd}^{OrgN} {floating-point} (Michaelis-Menten half-saturation constant for organic nitrogen limitation on hydrolysis; model input value) [g/m^3]
 halfSatConst.hydrolysisOrgP = K_{hyd}^{OrgP} {floating-point} (Michaelis-Menten half-saturation constant for organic phosphorus limitation on hydrolysis; model input value) [g/m^3]
 halfSatConst.nitrificationNH3 = K_{den} {floating-point} (Michaelis-Menten half-saturation constant for ammonia limitation on nitrification; model input value) [g/m^3]
 halfSatConst.nitrificationO2 = K_{den} {floating-point} (Michaelis-Menten half-saturation constant for oxygen limitation on nitrification; model input value) [g/m^3]
 halfSatConst.photosynDIN = K_{pho}^{DIN} {floating-point} (Michaelis-Menten half-saturation constant for dissolved inorganic nitrogen (DIN) limitation on photosynthesis; model input value) [g/m^3]
 halfSatConst.photosynDIP = K_{pho}^{DIP} {floating-point} (Michaelis-Menten half-saturation constant for dissolved inorganic phosphorus (DIP) limitation on photosynthesis; model input value) [g/m^3]
 hydrolysisOrgNCoef[1..numCellsLongit][1..numCellsVert] = k_{hyd}^{OrgN} {floating-point} (array containing the hydrolysis coefficient for organic nitrogen; model input value) [sec^{-1}]
 hydrolysisOrgPCoef[1..numCellsLongit][1..numCellsVert] = k_{hyd}^{OrgP} {floating-point} (array containing the hydrolysis coefficient for organic phosphorus)
 initialConcentration[1..numCellsLongit][1..numCellsVert].cbod = {floating-point} (array containing the CBOD concentration initial condition; model input value) [g/m^3]
 lastNode = {integer} (ID number of the worker node dealing with the most downstream subdomain) [unitless]
 lightExtinctCoefNonAlgal[1..numCellsLongit] = {floating-point} (light extinction coefficient; model input value) [m^{-1}]

$\text{lightExtinctCoefSelfShade} = \{\text{floating-point}\}$ (light extinction coefficient due to phytoplankton self-shading; model input value) [m^2/g]
 $\text{mainTimeStep} = \{\text{floating-point}\}$ (main time step) [sec]
 $\text{mainTimeStepInput} = \{\text{floating-point}\}$ (user selected main time step; model input value) [sec]
 $\text{maxAdvecVelocity} = u_{\text{max}}$ {floating-point} (maximum absolute value for the advective velocity which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m/sec]
 $\text{maxBiodegradationCoef} = k_{\text{deg,max}}$ {floating-point} (maximum value for the aerobic biodegradation coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxDeathCoef} = k_{\text{dea,max}}$ {floating-point} (maximum value for the phytoplankton death coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxDenitrificationCoef} = k_{\text{den,max}}$ {floating-point} (maximum value for the denitrification coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxDispersionCoefX} = E_{\text{max}}^x$ {floating-point} (maximum value for the longitudinal dispersion coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m^2/sec]
 $\text{maxDispersionCoefZ} = E_{\text{max}}^z$ {floating-point} (maximum value for the vertical dispersion coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m^2/sec]
 $\text{maxGrazingCoef} = k_{\text{gra,max}}$ {floating-point} (maximum value for the grazing coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxHydrolysisOrgNCoef} = k_{\text{hyd,max}}^{\text{OrgN}}$ {floating-point} (maximum value for the organic nitrogen hydrolysis coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxHydrolysisOrgPCoef} = k_{\text{hyd,max}}^{\text{OrgP}}$ {floating-point} (maximum value for the organic phosphorus hydrolysis coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxNitrificationCoef} = k_{\text{nit,max}}$ {floating-point} (maximum value for the nitrification coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxNumericalDisp} = E_{\text{num,max}}$ {floating-point} (maximum accepted value for the advection induced numerical dispersion; model input value) [m^2/sec]
 $\text{maxPhotosynthesisCoef} = k_{\text{pho,max}}$ {floating-point} (maximum value for the phytoplankton photosynthesis coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]
 $\text{maxReaerationCoef} = k_{\text{aer,max}}$ {floating-point} (maximum value for the reaeration coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec^{-1}]

$\text{maxRespirationCoef} = k_{res,max}$ {floating-point} (maximum value for the phytoplankton respiration coefficient which will not be exceeded anywhere in the system during the entire model simulation; model input value) [sec⁻¹]

$\text{maxSettlingVelCBOD} = k_{set,max}^{CBOD}$ {floating-point} (maximum value for the particulate CBOD settling velocity which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m/sec]

$\text{maxSettlingVelChla} = k_{set,max}^{Chla}$ {floating-point} (maximum value for the phytoplankton chl-a settling velocity which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m/sec]

$\text{maxSettlingVelInorgP} = k_{set,max}^{InorgP}$ {floating-point} (maximum value for the particulate inorganic phosphorus settling velocity which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m/sec]

$\text{maxSettlingVelOrgN} = k_{set,max}^{OrgN}$ {floating-point} (maximum value for the particulate organic nitrogen settling velocity which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m/sec]

$\text{maxSettlingVelOrgP} = k_{set,max}^{OrgP}$ {floating-point} (maximum value for the particulate organic phosphorus settling velocity which will not be exceeded anywhere in the system during the entire model simulation; model input value) [m/sec]

$\text{maxTimeStep.advec} = \Delta t_{adv,max}$ {floating-point} (maximum time step allowed for the advection rule) [sec]

$\text{maxTimeStep.biodegradation} = \Delta t_{deg,max}$ {floating-point} (maximum time step allowed for the aerobic biodegradation rule) [sec]

$\text{maxTimeStep.death} = \Delta t_{dea,max}$ {floating-point} (maximum time step allowed for the phytoplankton death rule) [sec]

$\text{maxTimeStep.denitrification} = \Delta t_{den,max}$ {floating-point} (maximum time step allowed for the denitrification rule) [sec]

$\text{maxTimeStep.dispX} = \Delta t_{dis,max}^x$ {floating-point} (maximum time step allowed for the longitudinal dispersion rule) [sec]

$\text{maxTimeStep.dispZ} = \Delta t_{dis,max}^z$ {floating-point} (maximum time step allowed for the vertical dispersion rule) [sec]

$\text{maxTimeStep.grazing} = \Delta t_{gra,max}$ {floating-point} (maximum time step allowed for the grazing rule) [sec]

$\text{maxTimeStep.hydrolysisOrgN} = \Delta t_{hyd,max}^{OrgN}$ {floating-point} (maximum time step allowed for the organic nitrogen hydrolysis rule) [sec]

$\text{maxTimeStep.hydrolysisOrgP} = \Delta t_{hyd,max}^{OrgP}$ {floating-point} (maximum time step allowed for the organic phosphorus hydrolysis rule) [sec]

$\text{maxTimeStep.nitrification} = \Delta t_{nit,max}$ {floating-point} (maximum time step allowed for the nitrification rule) [sec]

$\text{maxTimeStep.photosynthesis} = \Delta t_{pho,max}$ {floating-point} (maximum time step allowed for the phytoplankton photosynthesis rule) [sec]

$\text{maxTimeStep.reaeration} = \Delta t_{aer,max}$ {floating-point} (maximum time step allowed for the reaeration rule) [sec]

$\text{maxTimeStep.respiration} = \Delta t_{res,max}$ {floating-point} (maximum time step allowed for the phytoplankton respiration rule) [sec]

$\text{maxTimeStep.settlingCBOD} = \Delta t_{set,max}^{CBOD}$ {floating-point} (maximum time step allowed for the CBOD settling rule) [sec]

$\text{maxTimeStep.settlingChla} = \Delta t_{set,max}^{Chla}$ {floating-point} (maximum time step allowed for the phytoplankton chl-a settling rule) [sec]

$\text{maxTimeStep.settlingInorgP} = \Delta t_{set,max}^{InorgP}$ {floating-point} (maximum time step allowed for the inorganic phosphorus settling rule) [sec]

$\text{maxTimeStep.settlingOrgN} = \Delta t_{set,max}^{OrgN}$ {floating-point} (maximum time step allowed for the organic nitrogen settling rule) [sec]

$\text{maxTimeStep.settlingOrgP} = \Delta t_{set,max}^{OrgP}$ {floating-point} (maximum time step allowed for the organic phosphorus settling rule) [sec]

$\text{myNode} = \{\text{integer}\}$ (ID number identifying a particular worker node) [unitless]

$\text{newBuffer}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{integer}\}$ (array that stores the new configuration of particles in cells as the rule is applied) [particles/cell]

$\text{nextNode} = \{\text{integer}\}$ (ID number of the worker node dealing with the adjacent subdomain in the downstream direction) [unitless]

$\text{nitrificationCoef}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = k_{nit}$ {floating-point} (array containing the nitrification coefficient; model input value) [sec^{-1}]

$\text{nonpointLoad}[1..\text{numCellsLongit}].\text{cbod} = \{\text{floating-point}\}$ (vector containing the nonpoint source load for CBOD; model input value) [$\text{g/m}\cdot\text{sec}$]

$\text{numberOfPart_cbod}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of CBOD per cell) [particles/cell]

$\text{numberOfPart_chla}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of phytoplankton chl-a per cell) [particles/cell]

$\text{numberOfPart_inorgp}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of inorganic phosphorus per cell) [particles/cell]

$\text{numberOfPart_nh3}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of ammonia per cell) [particles/cell]

$\text{numberOfPart_no3}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of nitrate per cell) [particles/cell]

$\text{numberOfPart_o2}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of DO per cell) [particles/cell]

$\text{numberOfPart_orgn}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of organic nitrogen per cell) [particles/cell]

$\text{numberOfPart_orgp}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{floating-point}\}$ (array containing the number of particles of organic phosphorus per cell) [particles/cell]

$\text{numCellsLongit} = \{\text{integer}\}$ (number of cells in the longitudinal direction of the subdomain) [unitless]

$\text{numCellsVert} = \{\text{integer}\}$ (number of cells in the vertical direction of the domain) [unitless]

$\text{oldBuffer}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \{\text{integer}\}$ (array that stores the old configuration of particles in cells just before a rule is applied) [particles/cell]

$\text{packetFraction} = f$ {floating-point} (packet fraction; model input value) [unitless]

$\text{partCBODPerPartNO3} = n_{CBOD:NO3}^{den}$ {floating-point} (number of particles of CBOD removed for each particle of NO3 removed during denitrification) [unitless]

$\text{partFraction.cbod} = f_p^{CBOD}$ {floating-point} (particulate fraction for CBOD; model input value) [unitless]

$\text{partFraction.inorgp} = f_p^{InorgP}$ {floating-point} (particulate fraction for inorganic phosphorus; model input value) [unitless]

$\text{partFraction.orgn} = f_p^{OrgN}$ {floating-point} (particulate fraction for organic nitrogen; model input value) [unitless]

$\text{partFraction.orgp} = f_p^{OrgP}$ {floating-point} (particulate fraction for organic phosphorus; model input value) [unitless]

$\text{particleMassCBODO2} = m_p^{CBOD} = m_p^{DO}$ {floating-point} (particle mass for CBOD and DO; model input value) [g/particle]

$\text{particleMassChla} = m_p^{Chla}$ {floating-point} (particle mass for phytoplankton chl-a) [g/particle]

$\text{particleMassInorgP} = m_p^{InorgP}$ {floating-point} (particle mass for inorganic phosphorus) [g/particle]

$\text{particleMassNH3} = m_p^{NH_3}$ {floating-point} (particle mass for ammonia) [g/particle]

$\text{particleMassNO3} = m_p^{NO_3}$ {floating-point} (particle mass for nitrate) [g/particle]

$\text{particleMassOrgN} = m_p^{OrgN}$ {floating-point} (particle mass for organic nitrogen) [g/particle]

$\text{particleMassOrgP} = m_p^{OrgP}$ {floating-point} (particle mass for organic phosphorus) [g/particle]

$\text{partO2PerPartNH3} = n_{DO, NH_3}^{nit}$ {floating-point} (number of particles of DO removed for each particle of ammonia removed during nitrification) [unitless]

$\text{phytoDeathCoef}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = k_{dea}$ {floating-point} (array containing the phytoplankton death coefficient; model input value) [sec^{-1}]

$\text{phytoGrowthCoef}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = \mu$ {floating-point} (array containing the phytoplankton growth coefficient; model input value) [sec^{-1}]

$\text{phytoRespirationCoef}[1..\text{numCellsLongit}][1..\text{numCellsVert}] = k_{res}$ {floating-point} (array containing the phytoplankton respiration coefficient; model input value) [sec^{-1}]

$\text{pointLoad}[1..\text{numCellsLongit}].\text{cbod} = \{\text{floating-point}\}$ (vector containing the point source load for CBOD; model input value) [g/sec]

$\text{prevNode} = \{\text{integer}\}$ (ID number of the worker node dealing with the adjacent subdomain in the upstream direction) [unitless]

$\text{ratioOxygenToCarbon_photosynNH3} = a_{OC}^{photos}$ {floating-point} (oxygen-to-carbon ratio for photosynthesis using ammonia as the source of nitrogen -- see equation (4.1); model input value)

$\text{ratioOxygenToCarbon_photosynNO3} = a_{OC}^{photos}$ {floating-point} (oxygen-to-carbon ratio for photosynthesis using nitrate as the source of nitrogen -- see equation (4.14); model input value)

$\text{reaerationCoef}[1..\text{numCellsLongit}] = k_{aer}$ {floating-point} (vector containing the reaeration coefficient; model input value) [sec^{-1}]

$\text{ruleReps.advec} = \varphi_{adv} \{\text{integer}\}$ (number of repetitions of the advection rule during a main time step) [unitless]

$\text{ruleReps.biodegradation} = \varphi_{deg} \{\text{integer}\}$ (number of repetitions of the aerobic biodegradation rule during a main time step) [unitless]

$\text{ruleReps.death} = \varphi_{dea} \{\text{integer}\}$ (number of repetitions of the phytoplankton death rule during a main time step) [unitless]

$\text{ruleReps.denitrification} = \varphi_{den} \{\text{integer}\}$ (number of repetitions of the denitrification rule during a main time step) [unitless]

$\text{ruleReps.dispX} = \varphi_{dis}^x \{\text{integer}\}$ (number of repetitions of the longitudinal dispersion rule during a main time step) [unitless]

$\text{ruleReps.dispZ} = \varphi_{dis}^z \{\text{integer}\}$ (number of repetitions of the vertical dispersion rule during a main time step) [unitless]

$\text{ruleReps.grazing} = \varphi_{gra} \{\text{integer}\}$ (number of repetitions of the grazing rule during a main time step) [unitless]

$\text{ruleReps.hydrolysisOrgN} = \varphi_{hyd}^{OrgN} \{\text{integer}\}$ (number of repetitions of the organic nitrogen hydrolysis rule during a main time step) [unitless]

$\text{ruleReps.hydrolysisOrgP} = \varphi_{hyd}^{OrgP} \{\text{integer}\}$ (number of repetitions of the organic phosphorus hydrolysis rule during a main time step) [unitless]

$\text{ruleReps.nitrification} = \varphi_{nu} \{\text{integer}\}$ (number of repetitions of the nitrification rule during a main time step) [unitless]

$\text{ruleReps.photosynthesis} = \varphi_{pho} \{\text{integer}\}$ (number of repetitions of the phytoplankton photosynthesis rule during a main time step) [unitless]

$\text{ruleReps.reaeration} = \varphi_{aer} \{\text{integer}\}$ (number of repetitions of the reaeration rule during a main time step) [unitless]

$\text{ruleReps.respiration} = \varphi_{res} \{\text{integer}\}$ (number of repetitions of the phytoplankton respiration rule during a main time step) [unitless]

$\text{ruleReps.settlingCBOD} = \varphi_{set}^{CBOD} \{\text{integer}\}$ (number of repetitions of the CBOD settling rule during a main time step) [unitless]

$\text{ruleReps.settlingChla} = \varphi_{set}^{Chla} \{\text{integer}\}$ (number of repetitions of the phytoplankton chl-a settling rule during a main time step) [unitless]

$\text{ruleReps.settlingInorgP} = \varphi_{set}^{InorgP} \{\text{integer}\}$ (number of repetitions of the inorganic phosphorus settling rule during a main time step) [unitless]

$\text{ruleReps.settlingOrgN} = \varphi_{set}^{OrgN} \{\text{integer}\}$ (number of repetitions of the organic nitrogen settling rule during a main time step) [unitless]

$\text{ruleReps.settlingOrgP} = \varphi_{set}^{OrgP} \{\text{integer}\}$ (number of repetitions of the organic phosphorus settling rule during a main time step) [unitless]

$\text{satConcDO} = DO_{sat} \{\text{floating-point}\}$ (DO saturation concentration; model-input value) [g/m^3]

$\text{satLightIntensity} = I_s \{\text{floating-point}\}$ (phytoplankton photosynthesis saturating light intensity; model input value) [langley/day]

$\text{sedimentWaterFlux}[1..\text{numCellsLongit}].\text{cbod} = \{\text{floating-point}\}$ (vector containing the sediment-water flux for CBOD; model input value) [$\text{g/m}^2 \cdot \text{sec}$]

settlingVel[1..numCellsLongit][1..numCellsVert].cbod = v_s^{CBOD} {floating-point} (array containing the settling velocity for particulate CBOD; model input value) [m/sec]
 settlingVel[1..numCellsLongit][1..numCellsVert].chla = v_s^{Chla} {floating-point} (array containing the settling velocity for phytoplankton chl-a; model input value) [m/sec]
 settlingVel[1..numCellsLongit][1..numCellsVert].inorgp = v_s^{InorgP} {floating-point} (array containing the settling velocity for particulate inorganic phosphorus; model input value) [m/sec]
 settlingVel[1..numCellsLongit][1..numCellsVert].orgn = v_s^{OrgN} {floating-point} (array containing the settling velocity for particulate organic nitrogen; model input value) [m/sec]
 settlingVel[1..numCellsLongit][1..numCellsVert].orgp = v_s^{OrgP} {floating-point} (array containing the settling velocity for particulate organic phosphorus; model input value) [m/sec]
 settToLowerCell[1..numCellsLongit][1..numCellsVert] = {floating-point} (array containing the fraction of settling that effectively goes to the lower cell, to incorporate the effect of unequal cell widths) [unitless]
 simulationTime = {floating-point} (keeps track of the simulation time) [sec]
 solarRadiation[1..numCellsLongit] = {floating-point} (solar radiation just reaching the surface of the water column; model input value) [langley/day]
 timeSimulationBegin = {floating-point} (time simulation begins; model input value) [sec]
 timeSimulationEnd = {floating-point} (time simulation ends; model input value) [sec]
 timeStep.advec = Δt_{adv} {floating-point} (time step for the advection rule) [sec]
 timeStep.biodegradation = Δt_{deg} {floating-point} (time step for the aerobic biodegradation rule) [sec]
 timeStep.death = Δt_{dea} {floating-point} (time step for the phytoplankton death rule) [sec]
 timeStep.denitrification = Δt_{den} {floating-point} (time step for the denitrification rule) [sec]
 timeStep.dispX = Δt_{dis}^x {floating-point} (time step for the longitudinal dispersion rule) [sec]
 timeStep.dispZ = Δt_{dis}^z {floating-point} (time step for the vertical dispersion rule) [sec]
 timeStep.grazing = Δt_{gra} {floating-point} (time step for the grazing rule) [sec]
 timeStep.hydrolysisOrgN = Δt_{hyd}^{OrgN} {floating-point} (time step for the organic nitrogen hydrolysis rule) [sec]
 timeStep.hydrolysisOrgP = Δt_{hyd}^{OrgP} {floating-point} (time step for the organic phosphorus hydrolysis rule) [sec]
 timeStep.nitrification = Δt_{nit} {floating-point} (time step for the nitrification rule) [sec]
 timeStep.photosynthesis = Δt_{pho} {floating-point} (time step for the phytoplankton photosynthesis rule) [sec]
 timeStep.reaeration = Δt_{aer} {floating-point} (time step for the reaeration rule) [sec]
 timeStep.respiration = Δt_{res} {floating-point} (time step for the phytoplankton respiration rule) [sec]

timeStep.settlingCBOD = Δt_{set}^{CBOD} {floating-point} (time step for the CBOD settling rule) [sec]
timeStep.settlingChla = Δt_{set}^{Chla} {floating-point} (time step for the phytoplankton chl-a settling rule) [sec]
timeStep.settlingInorgP = Δt_{set}^{InorgP} {floating-point} (time step for the inorganic phosphorus settling rule) [sec]
timeStep.settlingOrgN = Δt_{set}^{OrgN} {floating-point} (time step for the organic nitrogen settling rule) [sec]
timeStep.settlingOrgP = Δt_{set}^{OrgP} {floating-point} (time step for the organic phosphorus settling rule) [sec]
upstreamBuffer[1..numCellsVert] = {integer} (vector containing the particles that are to be sent to the previous worker node)

```

void calculateTimeStep()
{
  /*
  This function calculates the main time step and Δx, and the time step and number of
  repetitions for each rule. It can be included in the manager node (if one exists) or
  alternatively in the worker nodes. This function needs to be executed only once.
  */
  if (maxAdvecVelocity != 0.0)
  {
    maxTimeStep.advec = (8.0 * maxNumericalDisp) / (maxAdvecVelocity)
    * maxAdvecVelocity);
  }
  else
  {
    maxTimeStep.advec = 0.0;
    cellSize.x = 1.0; /* when advection does not occur
    Δx becomes some user specified value; in this case 1.0 */
  }

  if (maxDispersionCoefZ != 0.0)
    maxTimeStep.dispZ = (1.493 * cellSize.z * cellSize.z) / (6.0 *
    maxDispersionCoefZ);
  else
    maxTimeStep.dispZ = 0.0;

  if (maxBiodegradationCoef != 0.0)
    maxTimeStep.biodegradation = (8.876 / maxBiodegradationCoef);
  else
    maxTimeStep.biodegradation = 0.0;

  if (maxReaerationCoef != 0.0)
    maxTimeStep.reaeration = (8.876 / maxReaerationCoef);
  else
    maxTimeStep.reaeration = 0.0;

  if (maxNitrificationCoef != 0.0)
    maxTimeStep.nitrification = (8.876 / maxNitrificationCoef);
  else
    maxTimeStep.nitrification = 0.0;

  if (maxDenitrificationCoef != 0.0)
    maxTimeStep.denitrification = (8.876 / maxDenitrificationCoef);
  else
    maxTimeStep.denitrification = 0.0;

  if (maxHydrolysisOrgNCoef != 0.0)
    maxTimeStep.hydrolysisOrgN = (8.876 / maxHydrolysisOrgNCoef);
  else
    maxTimeStep.hydrolysisOrgN = 0.0;

  if (maxHydrolysisOrgPCoef != 0.0)
    maxTimeStep.hydrolysisOrgP = (8.876 / maxHydrolysisOrgPCoef);
  else
    maxTimeStep.hydrolysisOrgP = 0.0;

  if (maxPhotosynthesisCoef != 0.0)
    maxTimeStep.photosynthesis = (8.876 / maxPhotosynthesisCoef);
  else
    maxTimeStep.photosynthesis = 0.0;

  if (maxRespirationCoef != 0.0)
    maxTimeStep.respiration = (8.876 / maxRespirationCoef);
  else
    maxTimeStep.respiration = 0.0;

  if (maxDeathCoef != 0.0)
    maxTimeStep.death = (8.876 / maxDeathCoef);
  else
    maxTimeStep.death = 0.0;

  if (maxGrazingCoef != 0.0)
    maxTimeStep.grazing = (8.876 / maxGrazingCoef);
  else
    maxTimeStep.grazing = 0.0;

  if (maxSettlingVelCBOD != 0.0)
    maxTimeStep.settlingCBOD = (8.876 / (maxSettlingVelCBOD / cellSize.z));
  else
    maxTimeStep.settlingCBOD = 0.0;

  if (maxSettlingVelOrgN != 0.0)
    maxTimeStep.settlingOrgN = (8.876 / (maxSettlingVelOrgN / cellSize.z));
  else
    maxTimeStep.settlingOrgN = 0.0;

  if (maxSettlingVelOrgP != 0.0)
    maxTimeStep.settlingOrgP = (8.876 / (maxSettlingVelOrgP / cellSize.z));
  else
    maxTimeStep.settlingOrgP = 0.0;

  if (maxSettlingVelInorgP != 0.0)
    maxTimeStep.settlingInorgP = (8.876 / (maxSettlingVelInorgP / cellSize.z));
  else
    maxTimeStep.settlingInorgP = 0.0;

  if (maxSettlingVelChla != 0.0)
    maxTimeStep.settlingChla = (8.876 / (maxSettlingVelChla / cellSize.z));
  else
    maxTimeStep.settlingChla = 0.0;

  /* Setting up the main time step */
  if (mainTimeStepInput == 0.0)
    /* when true it means the user does not impose a main time step value which therefore needs
    to be calculated by the program;
    when false it means the user imposes a main time step which is then simply assigned to the
    appropriate variable */

```

```

{
    mainTimeStep = (maxTimeStep.advec > maxTimeStep.dispZ) ?
    maxTimeStep.advec : maxTimeStep.dispZ;
    mainTimeStep = (mainTimeStep > maxTimeStep.biodegradation) ?
    mainTimeStep : maxTimeStep.biodegradation;
    mainTimeStep = (mainTimeStep > maxTimeStep.reaeration) ?
    mainTimeStep : maxTimeStep.reaeration;
    mainTimeStep = (mainTimeStep > maxTimeStep.nitrification) ?
    mainTimeStep : maxTimeStep.nitrification;
    mainTimeStep = (mainTimeStep > maxTimeStep.denitrification) ?
    mainTimeStep : maxTimeStep.denitrification;
    mainTimeStep = (mainTimeStep > maxTimeStep.hydrolysisOrgN) ?
    mainTimeStep : maxTimeStep.hydrolysisOrgN;
    mainTimeStep = (mainTimeStep > maxTimeStep.hydrolysisOrgP) ?
    mainTimeStep : maxTimeStep.hydrolysisOrgP;
    mainTimeStep = (mainTimeStep > maxTimeStep.photosynthesis) ?
    mainTimeStep : maxTimeStep.photosynthesis;
    mainTimeStep = (mainTimeStep > maxTimeStep.respiration) ?
    mainTimeStep : maxTimeStep.respiration;
    mainTimeStep = (mainTimeStep > maxTimeStep.death) ?
    mainTimeStep : maxTimeStep.death;
    mainTimeStep = (mainTimeStep > maxTimeStep.grazing) ?
    mainTimeStep : maxTimeStep.grazing;
    mainTimeStep = (mainTimeStep > maxTimeStep.settlngBOD) ?
    mainTimeStep : maxTimeStep.settlngBOD;
    mainTimeStep = (mainTimeStep > maxTimeStep.settlngOrgN) ?
    mainTimeStep : maxTimeStep.settlngOrgN;
    mainTimeStep = (mainTimeStep > maxTimeStep.settlngOrgP) ?
    mainTimeStep : maxTimeStep.settlngOrgP;
    mainTimeStep = (mainTimeStep > maxTimeStep.settlngInorgP) ?
    mainTimeStep : maxTimeStep.settlngInorgP;
    mainTimeStep = (mainTimeStep > maxTimeStep.settlngChla) ?
    mainTimeStep : maxTimeStep.settlngChla;
}
else
    mainTimeStep = mainTimeStepInput;

if (maxTimeStep.advec != 0.0)
    ruleReps.advec = roundUp(mainTimeStep, maxTimeStep.advec);
else
    ruleReps.advec = 0;
if (ruleReps.advec != 0)
{
    timeStep.advec = mainTimeStep / ruleReps.advec;
    cellSize.x = maxAdvecVelocity * timeStep.advec;
}
else
    timeStep.advec = 0.0;

if (maxDispersionCoefX != 0.0)
    maxTimeStep.dispX = (1.493 * cellSize.x * cellSize.x) / (6.0 * maxDispersionCoefX);
else
    maxTimeStep.dispX = 0.0;
}
if (maxTimeStep.dispX != 0.0)
    ruleReps.dispX = roundUp(mainTimeStep, maxTimeStep.dispX);
else
    ruleReps.dispX = 0;
if (ruleReps.dispX != 0)
    timeStep.dispX = mainTimeStep / ruleReps.dispX;
else
    timeStep.dispX = 0.0;

if (maxTimeStep.dispZ != 0.0)
    ruleReps.dispZ = roundUp(mainTimeStep, maxTimeStep.dispZ);
else
    ruleReps.dispZ = 0;
if (ruleReps.dispZ != 0)
    timeStep.dispZ = mainTimeStep / ruleReps.dispZ;
else
    timeStep.dispZ = 0.0;

if (maxTimeStep.biodegradation != 0.0)
    ruleReps.biodegradation = roundUp(mainTimeStep, maxTimeStep.biodegradation);
else
    ruleReps.biodegradation = 0;
if (ruleReps.biodegradation != 0)
    timeStep.biodegradation = mainTimeStep / ruleReps.biodegradation;
else
    timeStep.biodegradation = 0.0;

if (maxTimeStep.reaeration != 0.0)
    ruleReps.reaeration = roundUp(mainTimeStep, maxTimeStep.reaeration);
else
    ruleReps.reaeration = 0;
if (ruleReps.reaeration != 0)
    timeStep.reaeration = mainTimeStep / ruleReps.reaeration;
else
    timeStep.reaeration = 0.0;

if (maxTimeStep.nitrification != 0.0)
    ruleReps.nitrification = roundUp(mainTimeStep, maxTimeStep.nitrification);
else
    ruleReps.nitrification = 0;
if (ruleReps.nitrification != 0)
    timeStep.nitrification = mainTimeStep / ruleReps.nitrification;
else
    timeStep.nitrification = 0.0;

if (maxTimeStep.denitrification != 0.0)
    ruleReps.denitrification = roundUp(mainTimeStep, maxTimeStep.denitrification);
else
    ruleReps.denitrification = 0;
if (ruleReps.denitrification != 0)
    timeStep.denitrification = mainTimeStep / ruleReps.denitrification;
else
    timeStep.denitrification = 0.0;
}

```

```

if (maxTimeStep.hydrolysisOrgN != 0.0)
  ruleReps.hydrolysisOrgN = roundUp(mainTimeStep, maxTimeStep.hydrolysisOrgN);
else
  ruleReps.hydrolysisOrgN = 0;
if (ruleReps.hydrolysisOrgN != 0)
  timeStep.hydrolysisOrgN = mainTimeStep / ruleReps.hydrolysisOrgN;
else
  timeStep.hydrolysisOrgN = 0.0;

if (maxTimeStep.hydrolysisOrgP != 0.0)
  ruleReps.hydrolysisOrgP = roundUp(mainTimeStep, maxTimeStep.hydrolysisOrgP);
else
  ruleReps.hydrolysisOrgP = 0;
if (ruleReps.hydrolysisOrgP != 0)
  timeStep.hydrolysisOrgP = mainTimeStep / ruleReps.hydrolysisOrgP;
else
  timeStep.hydrolysisOrgP = 0.0;

if (maxTimeStep.photosynthesis != 0.0)
  ruleReps.photosynthesis = roundUp(mainTimeStep, maxTimeStep.photosynthesis);
else
  ruleReps.photosynthesis = 0;
if (ruleReps.photosynthesis != 0)
  timeStep.photosynthesis = mainTimeStep / ruleReps.photosynthesis;
else
  timeStep.photosynthesis = 0.0;

if (maxTimeStep.respiration != 0.0)
  ruleReps.respiration = roundUp(mainTimeStep, maxTimeStep.respiration);
else
  ruleReps.respiration = 0;
if (ruleReps.respiration != 0)
  timeStep.respiration = mainTimeStep / ruleReps.respiration;
else
  timeStep.respiration = 0.0;

if (maxTimeStep.death != 0.0)
  ruleReps.death = roundUp(mainTimeStep, maxTimeStep.death);
else
  ruleReps.death = 0;
if (ruleReps.death != 0)
  timeStep.death = mainTimeStep / ruleReps.death;
else
  timeStep.death = 0.0;

if (maxTimeStep.grazing != 0.0)
  ruleReps.grazing = roundUp(mainTimeStep, maxTimeStep.grazing);
else
  ruleReps.grazing = 0;
if (ruleReps.grazing != 0)
  timeStep.grazing = mainTimeStep / ruleReps.grazing;
else
  timeStep.grazing = 0.0;

```

```

if (maxTimeStep.settlingCBOD != 0.0)
  ruleReps.settlingCBOD = roundUp(mainTimeStep, maxTimeStep.settlingCBOD);
else
  ruleReps.settlingCBOD = 0;
if (ruleReps.settlingCBOD != 0)
  timeStep.settlingCBOD = mainTimeStep / ruleReps.settlingCBOD;
else
  timeStep.settlingCBOD = 0.0;

if (maxTimeStep.settlingOrgN != 0.0)
  ruleReps.settlingOrgN = roundUp(mainTimeStep, maxTimeStep.settlingOrgN);
else
  ruleReps.settlingOrgN = 0;
if (ruleReps.settlingOrgN != 0)
  timeStep.settlingOrgN = mainTimeStep / ruleReps.settlingOrgN;
else
  timeStep.settlingOrgN = 0.0;

if (maxTimeStep.settlingOrgP != 0.0)
  ruleReps.settlingOrgP = roundUp(mainTimeStep, maxTimeStep.settlingOrgP);
else
  ruleReps.settlingOrgP = 0;
if (ruleReps.settlingOrgP != 0)
  timeStep.settlingOrgP = mainTimeStep / ruleReps.settlingOrgP;
else
  timeStep.settlingOrgP = 0.0;

if (maxTimeStep.settlingInorgP != 0.0)
  ruleReps.settlingInorgP = roundUp(mainTimeStep, maxTimeStep.settlingInorgP);
else
  ruleReps.settlingInorgP = 0;
if (ruleReps.settlingInorgP != 0)
  timeStep.settlingInorgP = mainTimeStep / ruleReps.settlingInorgP;
else
  timeStep.settlingInorgP = 0.0;

if (maxTimeStep.settlingChla != 0.0)
  ruleReps.settlingChla = roundUp(mainTimeStep, maxTimeStep.settlingChla);
else
  ruleReps.settlingChla = 0;
if (ruleReps.settlingChla != 0)
  timeStep.settlingChla = mainTimeStep / ruleReps.settlingChla;
else
  timeStep.settlingChla = 0.0;
}

roundUp(x, y)
* This function finds the quotient (an integer) of x and the maximum submultiple of x which
does not exceed y.

```

```

while (simulationTime <= timeSimulationEnd)
{
simulationTime += mainTimeStep;

/*
External sources and sinks for CBOD
(for other constituents the implementation of external sources and sinks is similar)
*/

/* Atmospheric Deposition for CBOD */
for (h = 1, h <= numCellsLongit, h++)
{
if (atmosphDeposition[h].cbod != 0.0)
{
numberOfPart_cbod[h][edgeCell[h].top] +=
randomRound((atmosphDeposition[h].cbod * mainTimeStep * cellSize.x *
cellSize.Y [h][edgeCell[h].topNew]) / particleMassCBODO2);
}
}

/* Nonpoint Load for CBOD */
for (h = 1, h <= numCellsLongit, h++)
{
if (nonpointLoad[h].cbod != 0.0)
{
numberOfPart_cbod[h][edgeCell[h].top] +=
randomRound((nonpointLoad[h].cbod * mainTimeStep * cellSize.x) /
particleMassCBODO2);
}
}

/* Point Load for CBOD */
for (h = 1, h <= numCellsLongit, h++)
{
if (pointLoad[h].cbod != 0.0)
{
numberOfPart_cbod[h][edgeCell[h].top] +=
randomRound((pointLoad[h].cbod * mainTimeStep) / particleMassCBODO2);
}
}

/* Sediment-Water Flux for CBOD */
for (h = 1, h <= numCellsLongit, h++)
{
if (sedimentWaterFlux[h].cbod > 0.0) /* source */
for (i = edgeCell[h].bot, i <= edgeCell[h].top, i++)
{
if (i == edgeCell[h].bot)

```

```

*/
float x, y;
{
int i;
if (x < y)
i = 1;
else
{
if (((int)(x / y) * y) == x)
i = (int)(x / y);
else
i = ((int)(x / y)) + 1;
}
return (i);
}

IterationLoop()
{
/*
This function loops the execution of a model iteration corresponding to a main time step. It
is included in each of the worker nodes.
*/

int h, i, j;
double concentrationO2, p;

/* initial condition for CBOD
(for other constituents the implementation of initial conditions is similar) */
for (h = 1, h <= numCellsLongit, h++)
{
for (i = edgeCell[h].bot, i <= edgeCell[h].top, i++)
{
numberOfPart_cbod[h][i] = randomRound((initialConcentration[h][i].cbod *
cellSize.x * cellSize.z * cellSize.Y [h][i]) / particleMassCBODO2);
}
}

/*
Fraction of settling that effectively goes to the lower cell, to reflect the effect of unequal cell
widths.
*/
for (h = 1, h <= numCellsLongit, h++)
for (i = edgeCell[h].bot, i <= edgeCell[h].top, i++)
setFolowerCell[h][i] = (((cellSize.Y [h][i-1] / cellSize.Y [h][i]) < 1.0) ?
(cellSize.Y [h][i-1] / cellSize.Y [h][i]) : 1.0);

simulationTime = timeSimulationBegin;

```



```

for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
p = phytoRespirationCoef [h][i] * timeStep.respiration;
generalProb[h][i] = p / ((-0.003282 * p * p) + (0.065914 * p * p)
+ (0.563833 * p) + 0.973541);
}
}

for (j = 1; j <= ruleReps.respiration; j++)
respiration();

/* Death of Phytoplankton */
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
p = phytoDeathCoef [h][i] * timeStep.death;
generalProb[h][i] = p / ((-0.003282 * p * p) + (0.065914 * p * p)
+ (0.563833 * p) + 0.973541);
}
}

for (j = 1; j <= ruleReps.death; j++)
death();

/* Grazing of Phytoplankton */
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
p = grazingCoef [h][i] * timeStep.grazing;
generalProb[h][i] = p / ((-0.003282 * p * p) + (0.065914 * p * p)
+ (0.563833 * p) + 0.973541);
}
}

for (j = 1; j <= ruleReps.grazing; j++)
grazing();

/* Settling of CBOD */
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
p = (settlingVel[h][i].cbod / cellSize.z) * timeStep.settlingCBOD;
generalProb[h][i] = p / ((-0.003282 * p * p) + (0.065914 * p * p)
+ (0.563833 * p) + 0.973541);
}
}

```

```

(cellSize.x * cellSize.z * cellSizeY[h][i]);
denitrificationCoef[h][i] *= (halfSatConst.denitrification /
(halfSatConst.denitrification + concentrationO2));
p = denitrification Coef[h][i] * timeStep.denitrification;
generalProb[h][i] = p / ((-0.003282 * p * p) + (0.065914 * p * p)
+ (0.563833 * p) + 0.973541);
}
}

for (j = 1; j <= ruleReps.denitrification; j++)
denitrification();

/* Nitrification */
for (j = 1; j <= ruleReps.nitrification; j++)
nitrification();

/* Reaeration */
for (h = 1; h <= numCellsLongit; h++)
{
p = reaerationCoef[h] * timeStep.reaeration;
generalProb[h][i] = p / ((-0.003282 * p * p) + (0.065914 * p * p)
+ (0.563833 * p) + 0.973541);
}
}

for (j = 1; j <= ruleReps.reaeration; j++)
reaeration();

/* Hydrolysis of Organic Nitrogen */
for (j = 1; j <= ruleReps.hydrolysisOrgN; j++)
hydrolysisOrgN();

/* Hydrolysis of Organic Phosphorus */
for (j = 1; j <= ruleReps.hydrolysisOrgP; j++)
hydrolysisOrgP();

/* Photosynthesis for Phytoplankton */
for (j = 1; j <= ruleReps.photosynthesis; j++)
photosynthesis();

/* Respiration for Phytoplankton */
for (h = 1; h <= numCellsLongit; h++)
{

```

```

generalProb[h][i] *= partFraction.cbod;
}
}
for (j = 1; j <= ruleReps.settlingCBOD; j++)
    settling3BOD();

/* Settling of Organic Nitrogen */
for (h = 1; h <= numCellsLongit; h++)
{
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
        p = (settlingVel[h][i].orgn / cellSize.z) * timeStep.settlingOrgN;
        generalProb[h][i] = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
            + (0.563833 * p) + 0.973541);
    }
    generalProb[h][i] *= partFraction.orgn[h][i];
}

for (j = 1; j <= ruleReps.settlingOrgN; j++)
    settlingOrgN();

/* Settling of Organic Phosphorus */
for (h = 1; h <= numCellsLongit; h++)
{
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
        p = (settlingVel[h][i].orgp / cellSize.z) * timeStep.settlingOrgP;
        generalProb[h][i] = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
            + (0.563833 * p) + 0.973541);
    }
    generalProb[h][i] *= partFraction.orgp[h][i];
}

for (j = 1; j <= ruleReps.settlingOrgP; j++)
    settlingOrgP();

/* Settling of Inorganic Phosphorus */
for (h = 1; h <= numCellsLongit; h++)
{
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
        p = (settlingVel[h][i].inorgp / cellSize.z) * timeStep.settlingInorgP;
        generalProb[h][i] = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
            + (0.563833 * p) + 0.973541);
    }
}

```

```

generalProb[h][i] *= partFraction.inorgp;
}
}
for (j = 1; j <= ruleReps.settlingInorgP; j++)
    settlingInorgP();

/* Settling of Phytoplankton Chl-a */
for (h = 1; h <= numCellsLongit; h++)
{
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
        p = (settlingVel[h][i].chla / cellSize.z) * timeStep.settlingChla;
        generalProb[h][i] = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
            + (0.563833 * p) + 0.973541);
    }
}
for (j = 1; j <= ruleReps.settlingChla; j++)
    settlingChla();

/* Apply Upstream Boundary Condition */
if (myNode == firstNode)
    applyUpstreamBoundCond();

/* Apply Downstream Boundary Condition */
if (myNode == lastNode)
    applyDownstreamBoundCond();
} /* end while */
}

void applyUpstreamBoundCond()
{
    /* This function applies Dirichlet boundary conditions to the cells on the upstream boundary of
    the subdomain. It is included in the worker nodes (although only used by the first worker
    node, i. e., the node dealing with the most upstream subdomain).
    */
    int i;
    for (i = edgeCell[1].bot; i <= edgeCell[1].top; i++)
    {
        numberOfPart_cbod[1][i] = randomRound((boundCond[1].cbod *
            cellSize.x * cellSize.z * cellSizeY[1][i]) / particleMassCBOD2);
        numberOfPart_o2[1][i] = randomRound((boundCond[1].o2 *
            cellSize.x * cellSize.z * cellSizeY[1][i]) / particleMassCBOD2);
        numberOfPart_orgn[1][i] = randomRound((boundCond[1].orgn *

```

```

cellSize.x * cellSize.z * cellSize.Y[1][i] / particleMassOrgN);
numberOfPart_nh3[1][i] = randomRound((boundCond[1] * stream[i].nh3 *
cellSize.x * cellSize.z * cellSize.Y[1][i] / particleMassNH3);
numberOfPart_no3[1][i] = randomRound((boundCond[1] * stream[i].no3 *
cellSize.x * cellSize.z * cellSize.Y[1][i] / particleMassNO3);
numberOfPart_organ[1][i] = randomRound((boundCond[1] * stream[i].orgp *
cellSize.x * cellSize.z * cellSize.Y[1][i] / particleMassOrgP);
numberOfPart_inorgp[1][i] = randomRound((boundCond[1] * stream[i].inorgp *
cellSize.x * cellSize.z * cellSize.Y[1][i] / particleMassInorgP);
numberOfPart_chla[1][i] = randomRound((boundCond[1] * stream[i].chla *
cellSize.x * cellSize.z * cellSize.Y[1][i] / particleMassChla);
}
}

void applyDownstreamBoundCond()
{
/*
This function applies Dirichlet boundary conditions to the cells on the downstream boundary
of the subdomain. It is included in the worker nodes (although only used by the last worker
node, i. e., the node dealing with the most downstream subdomain).
*/
int i, k;
k = numCellsLongit;
for (i = edgeCell[k].bot; i <= edgeCell[k].top; i++)
{
numberOfPart_cbod[k][i] = randomRound((boundCondDownstream[i].cbod *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassCBOD2);
numberOfPart_o2[k][i] = randomRound((boundCondDownstream[i].o2 *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassCBOD2);
numberOfPart_organ[k][i] = randomRound((boundCondDownstream[i].orgn *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassOrgN);
numberOfPart_nh3[k][i] = randomRound((boundCondDownstream[i].nh3 *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassNH3);
numberOfPart_no3[k][i] = randomRound((boundCondDownstream[i].no3 *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassNO3);
numberOfPart_organp[k][i] = randomRound((boundCondDownstream[i].orgp *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassOrgP);
numberOfPart_inorgp[k][i] = randomRound((boundCondDownstream[i].inorgp *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassInorgP);
numberOfPart_chla[k][i] = randomRound((boundCondDownstream[i].chla *
cellSize.x * cellSize.z * cellSize.Y[k][i] / particleMassChla);
}
}

void advection()
{
/*
This function implements the rule for advection. It is included in the worker nodes

```

```

*/
int g, h, i, j;
/*
numPackets = (number of packets in a cell)
packetSize = (number of particles per packet)
lastPacket = (number of particles in the last packet -- to guarantee conservation of particles)
*/
int numPackets, packetSize, lastPacket;
for (g = 1; g <= 8; g++)
{
for (i = 1; i <= numCellsVert; i++)
{
upstreamBuffer[i] = 0;
downstreamBuffer[i] = 0;
}
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
if (g == 1) oldBuffer[h][i] = numberOfPart_cbod[h][i];
if (g == 2) oldBuffer[h][i] = numberOfPart_o2[h][i];
if (g == 3) oldBuffer[h][i] = numberOfPart_organ[h][i];
if (g == 4) oldBuffer[h][i] = numberOfPart_nh3[h][i];
if (g == 5) oldBuffer[h][i] = numberOfPart_no3[h][i];
if (g == 6) oldBuffer[h][i] = numberOfPart_organp[h][i];
if (g == 7) oldBuffer[h][i] = numberOfPart_inorgp[h][i];
if (g == 8) oldBuffer[h][i] = numberOfPart_chla[h][i];
newBuffer[h][i] = 0;
}
}
}
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
if (oldBuffer[h][i] != 0)
{
packetSize = xroundUp(packetFraction * (float) oldBuffer[h][i]);
if (packetSize == 0)
packetSize = 1;
if ((float) (int) ((float) oldBuffer[h][i] / (float) packetSize) <
(float) oldBuffer[h][i] / (float) packetSize)
lastPacket = oldBuffer[h][i] - (((int) ((float) oldBuffer[h][i] / (float)
packetSize)) * packetSize);
else
lastPacket = packetSize;
numPackets = xroundUp((float) oldBuffer[h][i] / (float) packetSize);

```

```

}
}
if (generalProb[h][i] < 0.0)
/*
When advection probability is negative -- as a result of negative velocity --
particles (packets) can only move upstream
*/
{
for (j = 1; j <= numPackets; j++)
{
if ((generalProb[h][i]) >= drand48())
{
if (h != 1)
{
if (i >= edgeCell[h-1].bot && i <= edgeCell[h-1].top)
{
if (j != numPackets)
newBuffer[h-1][i] += packetSize;
else
newBuffer[h-1][i] += lastPacket;
}
else
{
if (j != numPackets)
newBuffer[h][i] += packetSize;
else
newBuffer[h][i] += lastPacket;
}
}
else
{
if (i >= edgeCellBotPrevNode && i <= edgeCellTopPrevNode)
{
if (j != numPackets)
upstreamBuffer[i] += packetSize;
else
upstreamBuffer[i] += lastPacket;
}
else
{
if (j != numPackets)
newBuffer[h][i] += packetSize;
else
newBuffer[h][i] += lastPacket;
}
}
}
else
{
if (j != numPackets)
newBuffer[h][i] += packetSize;
else
newBuffer[h][i] += lastPacket;
}
}
}
}
}

```

```

if (generalProb[h][i] > 0.0)
/*
When advection probability is positive -- as a result of positive velocity --
particles (packets) can only move downstream
*/
{
for (j = 1; j <= numPackets; j++)
{
if ((generalProb[h][i]) >= drand48())
{
if (h != numCellsLongit)
{
if (i >= edgeCell[h+1].bot && i <= edgeCell[h+1].top)
{
if (j != numPackets)
newBuffer[h+1][i] += packetSize;
else
newBuffer[h+1][i] += lastPacket;
}
else
{
if (j != numPackets)
newBuffer[h][i] += packetSize;
else
newBuffer[h][i] += lastPacket;
}
}
}
else
{
if (i >= edgeCellBotNextNode && i <= edgeCellTopNextNode)
{
if (j != numPackets)
downstreamBuffer[i] += packetSize;
else
downstreamBuffer[i] += lastPacket;
}
else
{
if (j != numPackets)
newBuffer[h][i] += packetSize;
else
newBuffer[h][i] += lastPacket;
}
}
}
}
else
{
if (j != numPackets)
newBuffer[h][i] += packetSize;
else
newBuffer[h][i] += lastPacket;
}
}
}
}

```

```

if (g == 7) numberOPart_morgp[h][i] = newBuffer[h][i];
if (g == 8) numberOPart_chla[h][i] = newBuffer[h][i];
}
}
}
}

void dispersionX()
{
/*
This function implements the rule for dispersion in the longitudinal direction. It is included
in the worker nodes.
*/
int g, h, i, j;
double prob;
int numPackets, packetSize, lastPacket;
for (g = 1; g <= 8; g++)
{
for (i = 1; i <= numCellsVert; i++)
{
upstreamBuffer[i] = 0;
downstreamBuffer[i] = 0;
}
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
if (g == 1) oldBuffer[h][i] = numberOPart_cbood[h][i];
if (g == 2) oldBuffer[h][i] = numberOPart_o2[h][i];
if (g == 3) oldBuffer[h][i] = numberOPart_organ[h][i];
if (g == 4) oldBuffer[h][i] = numberOPart_nh3[h][i];
if (g == 5) oldBuffer[h][i] = numberOPart_no3[h][i];
if (g == 6) oldBuffer[h][i] = numberOPart_organp[h][i];
if (g == 7) oldBuffer[h][i] = numberOPart_morgp[h][i];
if (g == 8) oldBuffer[h][i] = numberOPart_chla[h][i];
newBuffer[h][i] = 0;
}
}
}
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
packetSize = xroundUp(packetFraction * (float) oldBuffer[h][i]);
if (packetSize == 0)
packetSize = 1;
if ((float) ((int) (oldBuffer[h][i] / (float) packetSize)) <

```

```

}
}
}
}
if (generalProb[h][i] == 0.0)
/*
When advection probability is zero -- as a result of zero velocity -- particles
(packets) do not move
*/
newBuffer[h][i] += oldBuffer[h][i];
}
}
}
}
/*
Send particles to adjacent neighbor worker nodes
*/
if (myNode != firstNode)
csend(10, upstreamBuffer, sizeof(upstreamBuffer), prevNode, NODEPID);
if (myNode != lastNode)
csend(20, downstreamBuffer, sizeof(downstreamBuffer), nextNode, NODEPID);
/*
Receive particles from adjacent neighbor worker nodes
*/
if (myNode != firstNode)
crecv(20, upstreamBuffer, sizeof(upstreamBuffer));
if (myNode != lastNode)
crecv(10, downstreamBuffer, sizeof(downstreamBuffer));
/*
Update number of particles at the upstream and downstream boundaries of
subdomain by adding particles received from adjacent neighbor worker nodes
*/
if (myNode != firstNode)
for (i = edgeCell[1].bot; i <= edgeCell[1].top; i++)
newBuffer[1][i] += upstreamBuffer[i];
if (myNode != lastNode)
for (i = edgeCell[numCellsLongit].bot; i <= edgeCell[numCellsLongit].top; i++)
newBuffer[numCellsLongit][i] += downstreamBuffer[i];
for (h = 1; h <= numCellsLongit; h++)
{
for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
{
if (g == 1) numberOPart_cbood[h][i] = newBuffer[h][i];
if (g == 2) numberOPart_o2[h][i] = newBuffer[h][i];
if (g == 3) numberOPart_organ[h][i] = newBuffer[h][i];
if (g == 4) numberOPart_nh3[h][i] = newBuffer[h][i];
if (g == 5) numberOPart_no3[h][i] = newBuffer[h][i];
if (g == 6) numberOPart_organp[h][i] = newBuffer[h][i];

```



```

    }
  }
  else
  {
    if (j != numPackets)
      newBuffer[h][i] += packetSize;
    else
      newBuffer[h][i] += lastPacket;
  }
}

if (prob == 0.0)
  /*
  */
  {
    if (j != numPackets)
      newBuffer[h][i] += packetSize;
    else
      newBuffer[h][i] += lastPacket;
  }
}

}
}
}

/*
*/
Send particles to adjacent neighbor worker nodes
*/
if (myNode != firstNode)
  csend(10, upstreamBuffer, sizeof(upstreamBuffer), prevNode, NODEPID);
if (myNode != lastNode)
  csend(20, downstreamBuffer, sizeof(downstreamBuffer), nextNode, NODEPID);

/*
*/
Receive particles from adjacent neighbor worker nodes
*/
if (myNode != firstNode)
  crecv(20, upstreamBuffer, sizeof(upstreamBuffer));
if (myNode != lastNode)
  crecv(10, downstreamBuffer, sizeof(downstreamBuffer));

/*
*/
Update number of particles at the upstream and downstream boundaries of
subdomain by adding particles received from adjacent neighbor worker nodes
*/
for (i = edgeCell[1].bot; i <= edgeCell[1].top; i++)
  newBuffer[1][i] += upstreamBuffer[i];

if (myNode != lastNode)
  for (i = edgeCell[numCellsLongit].bot; i <= edgeCell[numCellsLongit].top; i++)
    newBuffer[numCellsLongit][i] += downstreamBuffer[i];

```

```

for (h = 1; h <= numCellsLongit; h++)
  {
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
      {
        if (g == 1) numberOfPart_cboxd[h][i] = newBuffer[h][i];
        if (g == 2) numberOfPart_o2[h][i] = newBuffer[h][i];
        if (g == 3) numberOfPart_orgn[h][i] = newBuffer[h][i];
        if (g == 4) numberOfPart_ni3[h][i] = newBuffer[h][i];
        if (g == 5) numberOfPart_no3[h][i] = newBuffer[h][i];
        if (g == 6) numberOfPart_orgph[h][i] = newBuffer[h][i];
        if (g == 7) numberOfPart_morgp[h][i] = newBuffer[h][i];
        if (g == 8) numberOfPart_chiaph[h][i] = newBuffer[h][i];
      }
    }
  }

void dispersionZ()
{
  /*
  */
  This function implements the rule for dispersion in the vertical direction. It is included in the
  worker nodes.
  */
  int g, h, i, j;
  double prob;
  int numPackets, packetSize, lastPacket;
  int a, b;

  if (myNode == firstNode)
    a = 2;
  else
    a = 1;

  if (myNode == lastNode)
    b = numCellsLongit - 1;
  else
    b = numCellsLongit;

  for (g = 1; g <= 8; g++)
    {
      for (h = a; h <= b; h++)
        {
          for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
            {
              if (g == 1) oldBuffer[h][i] = numberOfPart_cboxd[h][i];
              if (g == 2) oldBuffer[h][i] = numberOfPart_o2[h][i];
              if (g == 3) oldBuffer[h][i] = numberOfPart_orgn[h][i];
            }
          }
        }
    }

```



```

    }
    }
    }
}

for (h = a; h <= b; h++)
{
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
        if (g == 1) numberOfPart_cbod[h][i] = newBuffer[h][i];
        if (g == 2) numberOfPart_o2[h][i] = newBuffer[h][i];
        if (g == 3) numberOfPart_ovgn[h][i] = newBuffer[h][i];
        if (g == 4) numberOfPart_nh3[h][i] = newBuffer[h][i];
        if (g == 5) numberOfPart_no3[h][i] = newBuffer[h][i];
        if (g == 6) numberOfPart_ovgp[h][i] = newBuffer[h][i];
        if (g == 7) numberOfPart_inorgp[h][i] = newBuffer[h][i];
        if (g == 8) numberOfPart_chla[h][i] = newBuffer[h][i];
    }
}

}

void biodegradation()
{
    /*
    This function implements the rule for aerobic biodegradation. It is included in the worker
    nodes.
    */
    int h, i, j;
    int numPart;
    double concentrationO2;
    double p, prob;
    int numPackets, packetSize, lastPacket;
    int a, b;

    if (myNode == firstNode)
        a = 2;
    else
        a = 1;

    if (myNode == lastNode)
        b = numCellsLongit - 1;
    else
        b = numCellsLongit;

    for (h = a; h <= b; h++)
    {

```

```

        for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
        {
            if (numberOfPart_cbod[h][i] != 0 && numberOfPart_o2[h][i] != 0)
            {
                concentrationO2 = (numberOfPart_o2[h][i] * particleMassCBO(DO2) /
                (cellSize.x * cellSize.z * cellSize.y[h][i]));
                p = biodegradationCoef[h][i] * (concentrationO2 /
                (halfSatConst.biodegradation + concentrationO2));
                p *= timeStep.biodegradation;
                prob = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
                + (0.563833 * p) + 0.973541);
                if (prob != 0.0)
                {
                    numPart = numberOfPart_cbod[h][i];
                    packetSize = xroundl(p(packetFraction * (float) numPart);
                    if (packetSize == 0)
                        packetSize = 1;
                    else
                        lastPacket = packetSize;
                    numPackets = xroundl(p(float) numPart / (float) packetSize) <
                    ((float) numPart / (float) packetSize)
                    lastPacket = numPart - ((int) ((float) numPart / (float) packetSize)) *
                    packetSize);
                    if ((float) ((int) ((float) numPart / (float) packetSize)) <
                    ((float) numPart / (float) packetSize))
                    {
                        lastPacket = numPart - ((int) ((float) numPart / (float) packetSize)) *
                        packetSize;
                    }
                    else
                        lastPacket = packetSize;
                    numPackets = xroundl(p(float) numPart / (float) packetSize);

                    for (j = 1; j <= numPackets; j++)
                    {
                        if (prob >= drand48())
                        {
                            if (j != numPackets)
                            {
                                if (numberOfPart_o2[h][i] >= packetSize)
                                {
                                    numberOfPart_cbod[h][i] -= packetSize;
                                    numberOfPart_o2[h][i] -= packetSize;
                                }
                                else
                                {
                                    packetSize = numberOfPart_o2[h][i];
                                    numberOfPart_cbod[h][i] -= packetSize;
                                    numberOfPart_o2[h][i] -= packetSize;
                                    break;
                                }
                            }
                            else
                            {
                                if (numberOfPart_o2[h][i] >= lastPacket)
                                {
                                    numberOfPart_cbod[h][i] -= lastPacket;

```



```

}
}
}
}
}
}
}
}
}
}
}

void nitrification()
{
/*
*/
This function implements the rule for nitrification. It is included in the worker nodes.
*/
int h, i, j;
int numPart;
double concentrationNH3, concentrationO2;
double p, prob;
int numPackets, packetSize, lastPacket;
int a, b;

if (myNode == firstNode)
  a = 2;
else
  a = 1;

if (myNode == lastNode)
  b = numCellsLongit - 1;
else
  b = numCellsLongit;

for (h = a; h <= b; h++)
{
  for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
  {
    if (numberOfPart_nh3[h][i] != 0)
    {
      concentrationNH3 = (numberOfPart_nh3[h][i] * particleMassNH3) /
        (cellSize.x * cellSize.z * cellSizeY[h][i]);
      p = nitrificationCoef[h][i] * (concentrationNH3 /
        (halfSatConst.nitrificationNH3 + concentrationNH3));
      concentrationO2 = (numberOfPart_o2[h][i] * particleMassCO2O2) /
        (cellSize.x * cellSize.z * cellSizeY[h][i]);
      p *= (concentrationO2 / (halfSatConst.nitrificationO2 + concentrationO2));

```

```

p *= timeStep.nitrification;
prob = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
  + (0.563833 * p) + 0.973541);
if (prob != 0.0)
{
  numPart = numberOfPart_nh3[h][i];
  packetSize = xroundUp(packetFraction * (float) numPart);
  if (packetSize == 0)
    packetSize = 1;
  if ((float) (int) ((float) numPart / (float) packetSize) <
    ((float) numPart / (float) packetSize))
    lastPacket = numPart - ((int) ((float) numPart / (float) packetSize)) *
      packetSize;
  else
    lastPacket = packetSize;
  numPackets = xroundUp((float) numPart / (float) packetSize);
  for (j = 1; j <= numPackets; j++)
  {
    if (prob >= drand48())
    {
      if (j != numPackets)
      {
        if (numberOfPart_o2[h][i] >= (xroundUp(partOf2PerPartNH3) *
          packetSize))
        {
          numberOfPart_nh3[h][i] -= packetSize;
          numberOfPart_o2[h][i] -= packetSize;
          randomRound(partOf2PerPartNH3);
          numberOfPart_nh3[h][i] += packetSize;
        }
        else
        {
          packetSize = (int) (numberOfPart_o2[h][i] /
            xroundUp(partOf2PerPartNH3));
          numberOfPart_nh3[h][i] -= packetSize;
          numberOfPart_o2[h][i] -= packetSize *
            randomRound(partOf2PerPartNH3);
          numberOfPart_nh3[h][i] += packetSize;
          break;
        }
      }
      else
      {
        if (numberOfPart_o2[h][i] >= (xroundUp(partOf2PerPartNH3) *
          lastPacket))
        {
          numberOfPart_nh3[h][i] -= lastPacket;
          numberOfPart_o2[h][i] -= lastPacket *

```



```

    }
    }
}

void hydrolysisOrgN()
{
    /*
    This function implements the rule for hydrolysis of organic nitrogen. It is included in the
    worker nodes.
    */
    int h, i, j;
    int numPart;
    double concentrationOrgN;
    double p, prob;
    int numPackets, packetSize, lastPacket;
    int a, b;

    if (myNode == firstNode)
        a = 2;
    else
        a = 1;

    if (myNode == lastNode)
        b = numCellsLongit - 1;
    else
        b = numCellsLongit;

    for (h = a; h <= b; h++)
    {
        for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
        {
            if (numberOfPart_orgn[h][i] != 0)
            {
                concentrationOrgN = (numberOfPart_orgn[h][i] * particleMassOrgN) /
                    (cellSize.x * cellSize.z * cellSizeY[h][i]);
                p = hydrolysisOrgNCoef[h][i] * (concentrationOrgN /
                    (halfSatConst.hydrolysisOrgN + concentrationOrgN));
                p *= timesStep.hydrolysisOrgN;
                prob = p / ((-0.003282 * p * p * p) + (0.065914 * p * p)
                    + (0.563833 * p) + 0.973541);
                if (prob != 0.0)
                {
                    numPart = numberOfPart_orgn[h][i];

```

```

                packetSize = xroundUp(packetFraction * (float) numPart);
                if (packetSize == 0)
                    packetSize = 1;

                if ((float) ((int) ((float) numPart / (float) packetSize)) <
                    ((float) numPart / (float) packetSize))
                    lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
                        packetSize);
                else
                    lastPacket = packetSize;

                numPackets = xroundUp((float) numPart / (float) packetSize);

                for (j = 1; j <= numPackets; j++)
                {
                    if (prob >= drand48())
                    {
                        if (j != numPackets)
                        {
                            numberOfPart_orgn[h][i] -= packetSize;
                            numberOfPart_nh3[h][i] += packetSize;
                        }
                        else
                        {
                            numberOfPart_orgn[h][i] = lastPacket;
                            numberOfPart_nh3[h][i] += lastPacket;
                        }
                    }
                }
            }
        }
    }
}

```

void hydrolysisOrgP()

```

{
    /*
    This function implements the rule for hydrolysis of organic phosphorus. It is included in the
    worker nodes.
    */
    int h, i, j;
    int numPart;
    double concentrationOrgP;
    double p, prob;
    int numPackets, packetSize, lastPacket;
    int a, b;

```

```

numberOfPart_inorgp[h][i] += packetSize;
    }
    else
    {
        numberOFPart_organp[h][i] = lastPacket;
        numberOFPart_inorgp[h][i] += lastPacket;
    }
}
}
}
}
}
}
}

void photosynthesis()
{
    /*
    * This function implements the rule for phytoplankton photosynthesis. It is included in the
    * worker nodes.
    */
    int h, i, j;
    int numPart, numPartDIP;
    lightIntensity[1..numCellsVert] = I (vector containing light intensity at various cell depths)
    [langley/day]
    /*
    double lightIntensity[numCellsVert + 1];
    double concentrationChla, concentrationDIN, concentrationDIP;
    double concentrationNH3, concentrationNO3;
    double nutrientFactorDIN, nutrientFactorDIP;
    /*
    preferenceForNH3 = (ammonia preference factor) [unitless]
    /*
    double preferenceForNH3;
    double fractionDIP;
    double p, prob;
    int numPackets, packetSize, lastPacket;
    int a, b;

    if (myNode == firstNode)
        a = 2;
    else
        a = 1;
    if (myNode == lastNode)
        b = numCells_ongit - 1;
    else
        b = numCells_ongit;
}

```

```

if (myNode == firstNode)
    a = 2;
else
    a = 1;
if (myNode == lastNode)
    b = numCells_ongit - 1;
else
    b = numCells_ongit;

for (h = a; h <= b; h++)
{
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
        if (numberOFPart_organp[h][i] != 0)
        {
            concentrationOrgP = (numberOFPart_organp[h][i] * particleMassOrgP) /
            (cellSize.x * cellSize.z * cellSize.y[h][i]);
            p = hydrolysisOrgP * coef [h][i] * (concentrationOrgP /
            (halfSatConst hydrolysisOrgP + concentrationOrgP));
            p * = timeStep hydrolysisOrgP;
            prob = p / ((-0.003282 * p * p * p) + (0.065914 * p * p) + (0.563833 * p)
            + 0.973541);
            if (prob != 0.0)
            {
                numPart = numberOFPart_organp[h][i];
                packetSize = xroundUp(packetFraction * (float) numPart);
                if (packetSize == 0)
                    packetSize = 1;
                if ((float) (int) ((float) numPart / (float) packetSize)) <
                ((float) numPart / (float) packetSize))
                    lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
                    packetSize);
                else
                    lastPacket = packetSize;
                numPackets = xroundUp((float) numPart / (float) packetSize);

                for (j = 1; j <= numPackets; j++)
                {
                    if (prob >= drand48())
                    {
                        if (j != numPackets)
                        {
                            numberOFPart_organp[h][i] -= packetSize;

```



```

void settlingOrgP()
{
    /*
    This function implements the rule for settling of organic phosphorus. It is included in the
    worker nodes.
    */
    int h, i, j;
    int numPart;
    int numPackets, packetSize, lastPacket;
    int a, b;
    double prob;

    if (myNode == firstNode)
        a = 2;
        a = 1;

    if (myNode == lastNode)
        b = numCells.orgit - 1;
        else
        b = numCells.orgit;

    for (h = a; h <= b; h++)
    {
        for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
        {
            /* Settling to lower cell */
            if (i > edgeCell[h].bot)
            {
                if (numberOfPart_orgp[h][i] != 0)
                {
                    numPart = numberOFPart_orgp[h][i];
                    packetSize = xroundUp(packetFraction * (float) numPart);
                    if (packetSize == 0)
                        packetSize = 1;

                    if ((float) ((int) ((float) numPart / (float) packetSize)) <
                        ((float) numPart / (float) packetSize))
                        lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
                            packetSize);
                    else
                        lastPacket = packetSize;

                    numPackets = xroundUp((float) numPart / (float) packetSize);
                }
            }
        }
    }
}

```

```

prob = generalProb[h][i] * setToLowerCell[h][i];
if (prob != 0.0)
{
    for (j = 1; j <= numPackets; j++)
    {
        if (prob >= drand48())
        {
            if (j != numPackets)
            {
                numberOFPart_orgp[h][i] -= packetSize;
                numberOFPart_orgp[h][i - 1] += packetSize;
            }
            else
            {
                numberOFPart_orgp[h][i] -= lastPacket;
                numberOFPart_orgp[h][i - 1] += lastPacket;
            }
        }
    }
}

/* Settling to cell bottom */
if (numberOFPart_orgp[h][i] != 0)
{
    numPart = numberOFPart_orgp[h][i];
    packetSize = xroundUp(packetFraction * (float) numPart);
    if (packetSize == 0)
        packetSize = 1;

    if ((float) ((int) ((float) numPart / (float) packetSize)) <
        ((float) numPart / (float) packetSize))
        lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
            packetSize);
    else
        lastPacket = packetSize;

    numPackets = xroundUp((float) numPart / (float) packetSize);

    if (i == edgeCell[h].bot)
        generalProb[h][i];
    else
        prob = generalProb[h][i] * (1.0 - setToLowerCell[h][i]);

    if (prob != 0.0)

```

```

{
  for (j = 1; j <= numPackets; j++)
  {
    if (prob >= drand48())
    {
      if (j != numPackets)
        numberOfPart_inorgp[h][i] -= packetSize;
      else
        numberOfPart_inorgp[h][i] = lastPacket;
    }
  }
}
}
}
}

void settlingInorgP()
{
  /* This function implements the rule for settling of inorganic phosphorus. It is included in the
  worker nodes.
  */
  int h, i, j;
  int numPart;
  int numPackets, packetSize, lastPacket;
  int a, b;
  double prob;

  if (myNode == firstNode)
    a = 2;
  else
    a = 1;

  if (myNode == lastNode)
    b = numCellsLongit - 1;
  else
    b = numCellsLongit;

  for (h = a; h <= b; h++)
  {
    for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
    {
      /* Settling to lower cell */

```

```

      if (i > edgeCell[h].bot)
      {
        if (numberOfPart_inorgp[h][i] != 0)
        {
          numPart = numberOfPart_inorgp[h][i];
          packetSize = xroundUp(packetFraction * (float) numPart);
          if (packetSize == 0)
            packetSize = 1;
          if ((float) (int) ((float) numPart / (float) packetSize) <
              ((float) numPart / (float) packetSize))
            lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
                                   packetSize);
          else
            lastPacket = packetSize;
          numPackets = xroundUp((float) numPart / (float) packetSize);
          prob = generalProb[h][i] * setTolLowerCell[h][i];
          if (prob != 0.0)
          {
            for (j = 1; j <= numPackets; j++)
            {
              if (prob >= drand48())
              {
                if (j != numPackets)
                {
                  numberOfPart_inorgp[h][i] -= packetSize;
                  numberOfPart_inorgp[h][i - 1] += packetSize;
                }
                else
                {
                  numberOfPart_inorgp[h][i] = lastPacket;
                  numberOfPart_inorgp[h][i - 1] += lastPacket;
                }
              }
            }
          }
          /* Settling to cell bottom */
          if (numberOfPart_inorgp[h][i] != 0)
          {
            numPart = numberOfPart_inorgp[h][i];

```

```

packetSize = xroundUp(packetFraction * (float) numPart);
if (packetSize == 0)
    packetSize = 1;
if ((float) ((int) ((float) numPart / (float) packetSize)) <
    ((float) numPart / (float) packetSize))
    lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
        packetSize);
else
    lastPacket = packetSize;
numPackets = xroundUp((float) numPart / (float) packetSize);
if (i == edgeCell[h].bot)
    prob = generalProb[h][i];
else
    prob = generalProb[h][i] * (1.0 - settToLowerCell[h][i]);
if (prob != 0.0)
    {
        for (j = 1; j <= numPackets; j++)
            if (prob >= drand48())
                {
                    if (j != numPackets)
                        numberOPart_inorgp[h][i] -= packetSize;
                    else
                        numberOPart_inorgp[h][i] -= lastPacket;
                }
            }
    }
}

void settlingChla()
{
    /* This function implements the rule for settling of phytoplankton chl-a. It is included in the
    worker nodes.
    */
    int h, i, j;
    int numPart;
    int numPackets, packetSize, lastPacket;
    int a, b;

```

```

double prob;
if (myNode == firstNode)
    a = 2;
else
    a = 1;
if (myNode == lastNode)
    b = numCells.longit - 1;
else
    b = numCells.longit;
for (h = a; h <= b; h++)
    {
        for (i = edgeCell[h].bot; i <= edgeCell[h].top; i++)
            /* Settling to lower cell */
            if (i > edgeCell[h].bot)
                {
                    if (numberOPart_chla[h][i] != 0)
                        numPart = numberOPart_chla[h][i];
                    packetSize = xroundUp(packetFraction * (float) numPart);
                    if (packetSize == 0)
                        packetSize = 1;
                    if ((float) ((int) ((float) numPart / (float) packetSize)) <
                        ((float) numPart / (float) packetSize))
                        lastPacket = numPart - (((int) ((float) numPart / (float) packetSize)) *
                            packetSize);
                    else
                        lastPacket = packetSize;
                    numPackets = xroundUp((float) numPart / (float) packetSize);
                    prob = generalProb[h][i] * settToLowerCell[h][i];
                    if (prob != 0.0)
                        {
                            for (j = 1; j <= numPackets; j++)
                                if (prob >= drand48())
                                    {
                                        if (j != numPackets)
                                            {
                                                numberOPart_chla[h][i] -= packetSize;
                                                numberOPart_chla[h][i - 1] += packetSize;

```



```
}  
drand48()  
/*  
Built-in function returning a uniformly distributed random number between 0 and 1.  
*/
```

VITA

Paulo Castro entered this world at the very early hours of February 12, 1963, in the coastal town of Caldas da Rainha, Portugal. He found himself surrounded by caring parents, many brothers and sisters, and a great dog. He spent most of his school years with his family in Oeiras, a coastal town on the outskirts of Lisbon. After finishing high-school, he enrolled in a Bachelor's degree in Environmental Engineering at the New University of Lisbon, just across the Tagus estuary. Then he pursued the opportunity of studying in the US, and was fortunate to enroll in a Master's degree in Environmental Systems Engineering at Clemson University, South Carolina. From there he moved on to a Ph.D. in Civil/Environmental Engineering at Virginia Tech, which he just completed.

A handwritten signature in black ink that reads "Paulo Castro". The signature is written in a cursive style with a horizontal line underneath it.

Paulo Castro