Parallel Processing for Modeling Reactive Transport in Groundwater

Jennifer L. Wright

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

ENVIRONMENTAL ENGINEERING

Dr. Mark A. Widdowson, Chair
Dr. Panos Diplas, Committee Member
Dr. G.V. Loganathan, Committee Member

1 May 2006
Blacksburg, Virginia

Key Words: Parallel processing, groundwater modeling, reactive transport

Parallel Processing for Modeling Reactive Transport in Groundwater

Jennifer L. Wright

(ABSTRACT)

Natural attenuation and biotransformation are processes that can potentially control the transport and enhance the remediation of contaminants in groundwater. It is necessary to develop computer simulations that not only model the physical transport (advection and dispersion) of contaminants, but that can also accurately depict chemical reactions and some of these more complex processes, in order to determine the type and extent of contaminant plumes and to analyze potential remediation strategies. Modeling these systems effectively is becoming possible with a growing understanding of the chemical and biological processes that occur in groundwater. However, more accurate and more involved models come with much higher memory and computational requirements. Parallel processing provides the computational resources needed to employ reactive transport simulations effectively and more efficiently.

N2D-H2 is a FORTRAN code that simulates two-dimensional reactive solute transport in groundwater. More specifically, it simulates the biotransformation of nitrate into the end products of denitrification. A parallel version of the N2D-H2 code is developed using the Message-Passing Interface (MPI), a library of sequences and routines that can be called from FORTRAN programs. Using MPI to develop the parallel version of the code involves decomposing the computational domain among processors, defining the computational roles of each processor, and implementing the required communication between processors by using the message-passing procedures that allow the processors to exchange data.

Several test problems are developed to analyze the performance of the parallel code. The test problems are used in the benchmarking procedure to demonstrate that the parallel code returns results identical to the sequential code. The CPU time required and the speedup achieved by running the simulation on parallel processors is presented for multiple test problems with varying physical processes and computational grid sizes. For a two-dimensional plume simulation of five solutes, with a finite difference grid of 490 nodes x 99 nodes, the total CPU time is decreased from 410 seconds on one processor to 220 seconds on two processors, and 75

seconds on ten processors. The speedup achieved gets closer to the ideal speedup as the problem size increases. Although the speedup observed with the parallel version of N2D-H2 is not 100% of the ideal speedup because of communication requirements, the parallel simulation demonstrates the benefits of parallel processing and the possibility of expansion that it provides for modeling reactive transport in groundwater.

Table of Contents

List of Figures

List of Tables

1 Introduction

Computer models are often used to interpret data and predict the effectiveness of different remediation strategies in cleaning up groundwater pollution. In order for these computer simulations to be accurate and give useful results, conceptual and mathematical models should include all of the relevant complexities in the subsurface. Simplifying assumptions about the subsurface properties, flow behavior, or the physical and chemical processes affecting contaminant transport are appropriate only when it is certain that these assumptions will not affect the outcome of a simulation (Zheng et al. 1995). Sufficient detail must be used when employing numerical solution methods so that the results are accurate and reflect all of the complexities of the flow and contamination in the region of interest. To achieve numerical accuracy in large-scale multiple solute simulations, a very large number of computational nodes must be used (Tsai et al. 1999). Depending on the modeling objective and quality of data, simplified mathematical models may not always be capable of accurately defining the transport of contamination, and therefore, any remediation strategies based on the results of the simulation will likely be ineffective and costly (Ashby et al. 1996).

Avoiding simplifying assumptions in groundwater pollution simulations results in complex three-dimensional problems. Sources of complexity include heterogeneous parameters, multiple contaminants, geochemical interactions, and reactive transport processes. Obtaining accurate results from reactive transport models is becoming more important as natural attenuation becomes widely recognized as an effective groundwater remediation strategy (Schwartz et al. 2003). These models must correctly identify the chemical and biological species in the groundwater and their potential interactions in order to determine the extent and type of possible contamination plumes. For example, under the right conditions, groundwater containing the prevalent contaminants perchloroethene (PCE) or trichloroethene (TCE) could biodegrade the contaminants and produce the hazardous chemicals dichloroethene (DCE) and vinyl chloride (VC) through the process of reductive dechlorination. As this process continues, the vinyl chloride could degrade into the less hazardous chemical ethene (Wiedemeier et al. 1999). A simulation of this process will require a very high load of computational effort, compounded by the number of chemical and biological species that must be considered.

As advances in groundwater simulations are made, parallel computing will become necessary in order for modeling to remain a feasible tool in designing remediation and clean-up strategies. Parallel computing will allow models with much greater detail and accuracy to be simulated on computers within a time-frame that makes their use more practical.

This report is a study of parallel processing concepts and the potential application of parallel processing to groundwater pollution simulations. The basic concepts involved with writing parallel computer code are explained and some existing attempts at using parallel processing for groundwater simulations are reviewed. Using this background information, a parallel computer code is developed that models the reactive transport of nitrate contamination in groundwater. Several problems are developed to test this code and to conduct a performance analysis on the code. Based on the performance of the code, it is determined that developers and users of reactive transport simulations could greatly benefit from the application of parallel processing to these models.

2 Background

Parallel computation is a widely growing area of high technology and is becoming an important trend for numerical simulation in many applications. It is helpful for use in engineering applications that operate with large data domains and take a significant amount of time to simulate (Madabhushi et al. 1998). The application of complex three-dimensional groundwater contaminant transport models to heterogeneous sites with complex hydrogeology is computationally challenging. Implementation of parallel processing enables the developers of these models to obtain accurate numerical solutions while speeding up the computational time, and promoting the computational efficiency (Tsai et al. 1999).

2.1 Parallel processing

Parallel processing is an appropriate computational tool if the application has enough parallelism to make good use of multiple processors. Parallelism is the idea that there is a high volume of repetitive computation that takes place during a simulation. This often comes in the form of do-loops in computer code. There must be portions of the program that can execute independently and simultaneously on separate processors, which is an inherent feature of solving partial differential equations with the finite difference method. Computer code that solves a partial differential equation with any numerical method usually has a time-loop and several loops to move through the computational grid. The time-loop is inherently sequential and obviously can not be solved in parallel. However, a do-loop that performs identical computations at every node in the domain lends itself well to parallelism. The problem must also be large enough so that the simulation time is lengthy enough on one processor that significant speedup would be achieved by using multiple processors (Dietz 1998). The background information given here describes some of the tools and methods available for developing parallel code, and demonstrates that reactive transport models could potentially make good used of parallel computing resources.

All of the parallel programming in this report uses the Single Program Multiple Data (SPMD) model, meaning that there is only one executable program that is read by all processors (Aoyama et al. 1999). Tasks are defined for different processors by branches in the code. The Multiple Data part of this model implies that each processor works on its own data set. The processing occurs on a distributed-memory system, as opposed to a shared-memory system, meaning that each processor has only local memory and there is no access to any shared address

space. The processors can share the information in their local memories by communicating with other processors by sending and receiving messages. This is known as the message-passing parallel computational model. Parallelization using the message-passing model is achieved through the use of the Message-Passing Interface (MPI), which is a library that specifies the names, calling sequences, and results of subroutines to be called from FORTRAN programs (Gropp et al. 1999). A few of the most important parallel programming concepts and specific MPI tools are discussed here to provide the appropriate background for understanding the issues in writing parallel code.

2.1.1 Domain decomposition

In order to parallelize an algorithm, the data in the problem must be distributed across multiple processors, e.g. a finite difference solution grid must be divided into subdomains, and then the programmer must decide how to assign processes to each part of the decomposed domain. Once the decomposition has been determined, the most efficient way to assign subdomains to different processors depends on the details of the underlying hardware (Aoyama et al. 1999).

The description of how the processes in a parallel computer are connected to one another is called the topology of the computer, or interconnection network (Gropp et al. 1999). In most parallel programs, each process communicates mostly with only a few other processes. Therefore, it is important for the performance of the program that the subdomains are mapped to processes in such a way that makes sense with the topology of the network. There are functions built in to MPI that will automatically assign subdomains to processes once the program has defined the overall structure of the decomposition.

There are several methods of decomposing the domain. Figure 1 illustrates a one-dimensional column-based decomposition, a one-dimensional row-based decomposition, and a two-dimensional decomposition. For a finite difference solution grid of this size, some decomposition methods may lead to a more efficient parallel simulation than others. For example, the row-based decomposition method illustrated here will require a high volume of message-passing between processes as compared to the computational load on each processor. This concept reflects the ratio of the surface area of a subdomain (representing required communication) to the volume of the subdomain (representing the computational load). An

efficient decomposition method will minimize this ratio. Therefore, in this case, the column-based decomposition or the two-dimensional decomposition may be more efficient.

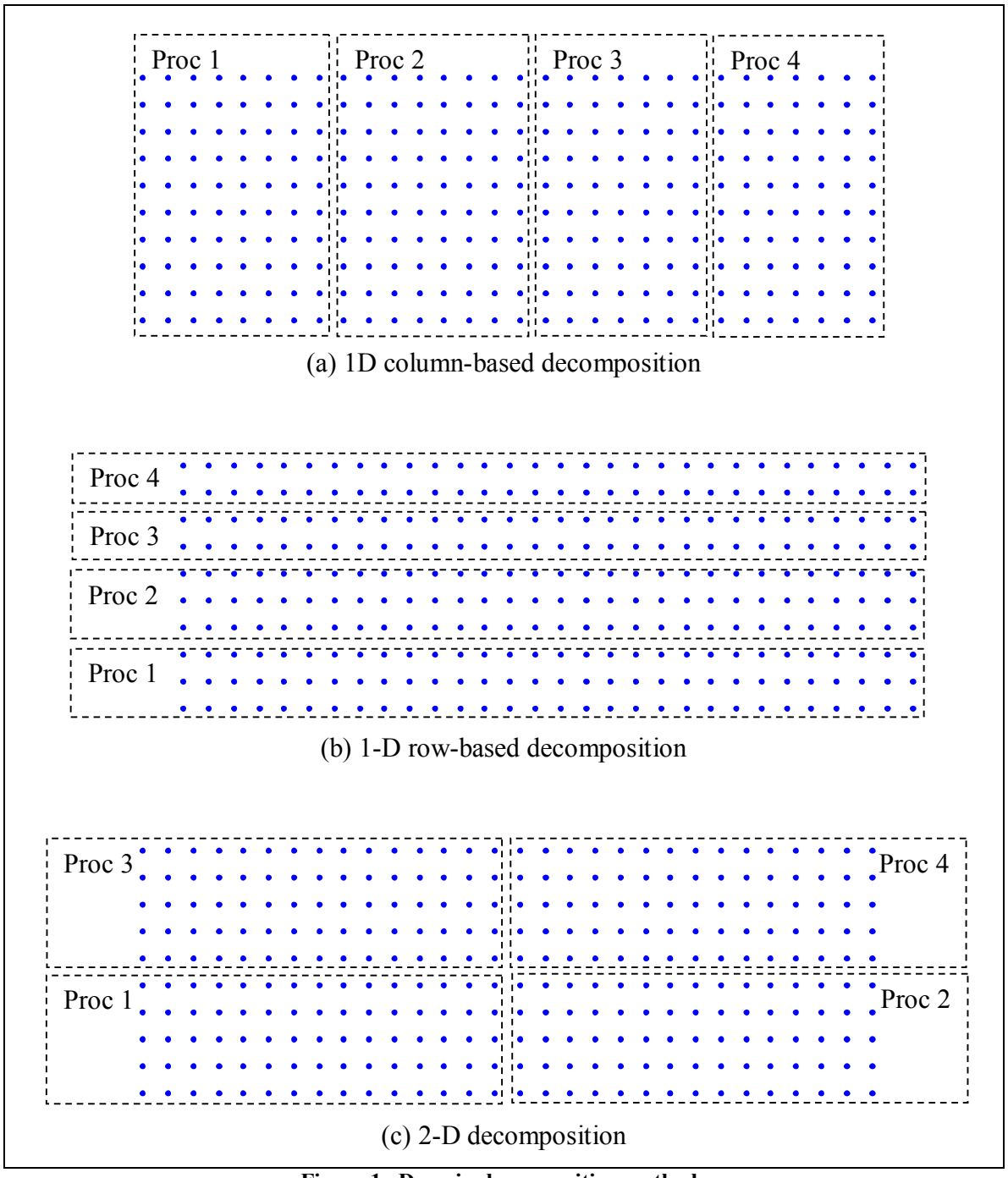

Figure 1. Domain decomposition methods

For the application of parallel computation to groundwater simulations, another concept that should be considered in determining the decomposition method is the direction of groundwater flow. The direction of flow could influence the solution method and the amount of data that needs to be shared among adjacent nodes. For example, in order to calculate concentration due to advective transport for strictly horizontal flow in the x-direction, data will need to be shared only among nodes in the same row. Therefore, the row-based decomposition would require less communication than a column-based decomposition, and the row-based decomposition would lead to a higher computation to communication ratio for each processor.

The domain decomposition is the initial step in developing a parallel simulation, and it will ultimately determine the performance of the parallel code. Therefore, it is important that the concepts discussed here are considered and the best method is chosen depending on the particular application.

2.1.2 Communication

2.1.2.1 Communicators

A communicator is a collection of processes that can send messages to each other. *MPI_COMM_WORLD* is a predefined communicator that consists of all the processes running the program when the execution begins (Pacheco et al. 1997). It is possible to create additional communicators within the program in order to define a subset of the processes that will be sending messages to each other for a specific purpose, so that these messages do not get confused with messages sent for other purposes.

2.1.2.2 Standard communication

Under the message-passing model for parallel computation, each of the processes executing in parallel have separate address spaces. Communication occurs when a portion of one process's address space is copied into another process's address space. This operation is cooperative and occurs only when the first process executes a send operation and the second process executes a receive operation (Gropp et al. 1999). The syntax of the MPI send and receive commands are as follows:

MPI_SEND(message, count, datatype, dest, tag, comm, ierror)

MPI_RECV(message, count, datatype, source, tag, comm, status, ierror)

The contents of the message are stored in the block of memory referenced by the argument *message*. The message consists of *count* elements of the MPI type *datatype*. For example, *count* = 3 and *datatype* = mpi_integer implies that the messages consists of an array of three integers. The data type can be one of the predefined MPI types (which generally correspond to the data types in Fortran). The arguments *dest* and *source* are the ranks of the receiving and the sending processes. The *tag* argument is an integer used so that processes receiving several messages can identify which message is to be used for which purpose. The communicator *comm* identifies the collection of processes that is being used to communicate, as discussed previously. Finally, the *status* argument returns information on the data that was actually received (Pacheco et al. 1997).

2.1.2.3 Collective communication

The communication described in the previous section involved direct communication from one processor to another, known as point-to-point communication. There is also a set of commands in MPI that all allow communication to take place among all of the processors in a communicator. This is collective communication. A collective operation is performed by all processes in a computation, and can be of two types: 1) data movement operations, which are used to rearrange data among processes, and 2) collective computation operations. An example of a data movement operation is the *BROADCAST* command. This command is used to send a message from one process to all of the other processes in the communicator, and is complete once all of the processes have called the *broadcast* command. An example of a collective computation is the *ALL_REDUCE* command, which can be used when each process holds a section of an array and the sum of all of the values in the array need to be found. Each process will find the sum of its local data and then call the *ALL_REDUCE* command, which will essentially distribute all of the local sums to all of the processes, so that each process can compute the overall sum (Gropp et al. 1999).

This section has described some of the basic tools and concepts underlying the message-passing method for parallel computation. It is not all inclusive, but will hopefully provide

enough background for the reader to understand some of the challenges that will be faced in developing a program to execute in parallel.

2.2 Challenges in writing parallel code

Parallel code can be written by finding existing divisions in the computation among the problem domain or by reworking the solution method into one that involves more inherent parallel computation. In the first case, groups of independent calculations will be performed on individual processors. For example, the computational domain will be partitioned among processors so that each processor will be performing the calculations required for a subset of the entire computational grid. However, data dependencies exist in all numerical solution methods that will prevent the parallelization of a sequential algorithm without requiring communication among processors. Communication among processors degrades the performance of the parallel code.

In parallelizing any groundwater modeling code, three major ideas must be considered: 1) computational load balancing, 2) minimizing communication, and 3) scalability. These three considerations are all highly related and dependent on the domain decomposition method.

2.2.1 Computational load balancing

Achieving computational load balancing is an important aspect of writing parallel computer code. In order for the parallel code to be efficient, the domain decomposition method should lead to a collection of subdomains that are mapped to different processors in such a way that all processors have approximately the same amount of work to do throughout the course of the simulation (Lingen 2000). The computation time of parallel sub-tasks should be relatively uniform across the processors in order to avoid some processors being idle in waiting for others to finish their sub-tasks (Dou 1998). Leaving processors idle means that the parallel simulation is inefficient and does not make full use of the computational resources.

In order to achieve load balancing, one would normally like to partition the computational domain into subdomains of about equal size, and map the subdomains to an equal number of processors. However, some variation is required in the domain decomposition method depending on the application. For example, consider a problem that uses a variably sized solution grid. A variably sized grid may be used in a large-scale problem to capture localized phenomena, like observing drawdown in a region near a pumping well. In order to

balance work loads among different processors, small subdomains with fine grids should be used in the rapidly changing solution areas (e.g. the region near the well), and large subdomains with coarse grids should be used in the slowly changing solution areas (Yang 1997).

There are two methods to assign work to processors and achieve load balance when writing parallel code: static load balancing and dynamic load balancing. Static load balancing assumes that the amount of computation required in each subdomain can be determined before the simulation is performed. In this case, the division of computation is written in to the code after analysis of the computation requirements. Static load balancing is largely a result of the domain decomposition method, and the best strategy to achieve this can vary, as discussed above. When it is unknown exactly how much communication will be required by a particular simulation, then work is assigned to processors that are currently idle throughout the simulation process; this is dynamic load balancing (Lingen 2000).

In order to demonstrate these load balancing concepts, consider a problem that models the reactive transport of five solutes in groundwater. One domain decomposition method considered for such a problem is a division of computation according to solute. However, the concentrations of each solute are dependent on their initial concentrations, the concentrations of the other solutes, and the conditions of the site-specific problem. It is possible that, at a certain point in the simulation, the concentration of a solute could equal zero, or reach some steady state. Thus, at future time steps, no computation would be required to determine the concentration of such a solute, or fewer iterations would be required to calculate the concentration of the solute. Consequently, the processor that was assigned to do the work for that solute would become idle for the remainder of the simulation, which suggests that the parallel simulation is inefficient. Dynamic load balancing could possibly mediate this problem by assigning more work to the idle processor when it finishes the work required to calculate the concentration of the solute it was initially assigned.

Dynamic load balancing is often performed using the master-slave paradigm for parallel simulations. Under this approach, one processor is considered the master, and all other processors are the slaves. The master reads in input data and grid information, initializes variables, and sends all of the data and parameters to the slaves as needed. The slaves compute relevant information, such as coefficients of the equations being solved, and compute the solution to the problem for a specific block of the computational domain, as specified by the

master. The solution is then sent back to the master and this processor waits for the next task (Dou et al. 1998). Computational tasks are assigned to each processor throughout the simulation to ensure that no processors become idle; thus, the master-slave approach ensures load balancing. However, as problem size increases, at a certain point this approach becomes inefficient and does not show good scalability (Yang 1997). One processor holds all of the data and has to send and receive messages from all of the other processors, so the total simulation time will be dominated by the CPU time required by the master processor. This introduces another challenge in writing parallel code, which is the essential need to minimize communication.

2.2.2 Minimizing communication

Along with computational load imbalance, the main cause of poor parallel performance is high communication overhead. Communication is closely related to load balancing, and both are determined by the domain decomposition method. The best domain decomposition method can be determined by analyzing the following factors: (1) computation load balance (as discussed in the previous section), (2) average (or total) communication volume, (3) communication volume load balance, (4) average (or total) number of messages, and (5) load balance in the number of messages (Elmroth 2000). The factors relating to communication will be discussed here.

The average communication volume is a function of the average surface area of each subdomain. This is because, if a subdomain consists of a section of a finite difference solution grid, then this subdomain will need to communicate with all bordering subdomains. It will need to send and receive messages whose sizes correspond to the length of the domain boundaries with other subdomains. Therefore, in order to minimize communication, the average number of external elements per processor should be minimized (external elements are finite difference nodes that are not part of the subdomain, but are needed in the computations performed by that subdomain).

Communication volume load balance is important in much the same way that computational load balance is important. Full use should be made of the computing resources, and processors should not be idle waiting for other processors to complete high volumes of communication. In order to achieve communication load balance, parallel code should minimize the difference between the maximum number of external elements for any processor and the average number of external elements per processor (Elmroth 2000).

The number of messages that a processor needs to send and receive in order to obtain the pertinent data to its computation depends on the number of neighboring processors. Therefore, the domain decomposition method needs to minimize the average number of neighboring processors per processor and also minimize the difference between the maximum number of neighbors for any processor and the average number of neighbors per processor. This will result in an efficient average number of messages per processor and a load balance among processors in the number of messages being sent and received (Elmroth 2000).

As discussed, computational load balancing and minimizing communication overhead are two important factors in achieving good parallel performance. These two factors are very closely related and often trade-offs need to be made between the two in order to achieve the best results. In order to determine the domain decomposition method and computational algorithms that lead to the most efficient code, performance analysis must be executed. Scalability is an important measure of the performance of any parallel code.

2.2.3 Scalability

A common problem with parallel computer code is poor scalability. Poor scalability implies that increasing the number or processors used to simulate an increasing problem size does not achieve the same relative speedup that was observed when the problem size was only slightly increased or only a few processors were used. A parallel algorithm is said to be scalable if the solution time remains constant when the number of computer processors increases linearly with the number of unknown parameters in the problem to be solved, i.e., the efficiency should remain constant for a constant ratio of processors to problem size (Dou et al. 1998). The performance of a parallel algorithm is based on scalability and reflects the parallel efficiency and the speedup up the code. The speedup factor and efficiency are defined, respectively, as:

$$S(n) = \frac{T_s}{T(n)} \text{ and } E(n) = \frac{S(n)}{n},$$

where T_s is the CPU time for the best serial algorithm, $T(n)$ is the CPU time for the parallel algorithm using n processors, $S(n)$ is the total speed-up factor for the parallel computation, and $E(n)$ is the total efficiency for the parallel algorithm (Dou et al. 1998). The parallel efficiency

represents the effectiveness of the parallel program on n processors as compared to one processor, and therefore takes into account the loss of efficiency due to communication and data management requirements resulting from the domain decomposition method.

In developing parallel computer code, the related issues of computational load balancing, communication requirements, and scalability, and how these issues are affected by the domain decomposition method, all need to be considered and reconciled in order to have a simulation that performs well. As parallel groundwater modeling codes are discussed, all of these issues will be prevalent.

2.3 Parallel codes used for groundwater modeling

Relatively few attempts have been made at developing parallel code for groundwater flow and contaminant transport models. Parallel code has been developed for the purpose of modeling groundwater flow, single solute transport, and multiple-component reactive transport. The issues discussed above are the common challenges faced by the developers of these parallel codes. However, these challenges become more difficult to overcome for more complex codes. At the same time, it is the comprehensive numerical three-dimensional models that benefit the most from well written parallel code. Therefore, the focus here will be on the development of parallel code for multi-component reactive transport models.

Tsai et al. (1999) developed a parallel code that solves the advection-dispersion equation for a single solute in two dimensions. The program solves the transport equation using the finite analytic method. The focus in developing this parallel code was on the domain decomposition method. The dominant direction of groundwater flow and the processing requirements of the solution method were the two biggest factors in determining the most efficient domain partitioning method. A single solute transport problem in a two-dimensional flow field was developed to test the code. The test problem was simulated on 2, 4, and 8 processors. These simulations achieved speedup values of 83%, 91%, and 69% respectively. This reveals that the optimal efficiency of the parallel computation occurs when 4 processors are used. The low efficiency of 69% on 8 processors suggests that the performance will degrade even further when additional processors are used. Therefore, the benefits of extending the code to the three-

dimensional problem or other more complex simulations are limited by the poor efficiency of the code (Tsai et al. 1999).

Wu et al. (2002) developed TOUGH2: a numerical simulation for modeling multidimensional, multiphase, multi-component fluid and heat flows in porous and fractured media. The code was used to simulate three-dimensional fluid flow in the unsaturated zone of Yucca Mountain, Nevada. Various levels of grid refinement were used in order to test the performance of the code as the problem size increased. The results of these tests showed that the one-million-cell models produced better resolution results and revealed flow patterns that could not be obtained using modeling results from a coarser grid. This simulation revealed the benefits and necessity of using parallel processing in complex groundwater flow problems. The size of the computational domain alone was enough to make parallel processing beneficial. When contaminant transport and chemical reactions are modeled, the increased computational requirements will make parallel processing even more valuable (Wu et al. 2002; Zhang et al. 2003).

Gwo et al. (2001) developed HBGC123D: a high-performance computer model of coupled hydrogeological and biogeochemical processes. This code uses a hybrid Eulerian-Lagrangian finite element method to solve the solute transport equations and a Newton's method to solve the system of reaction equations. One application used to test the code was a hypothetical three dimensional bioremediation problem. The finite element mesh is made up of 14,637 nodes and 12,800 elements, and there are 17 reaction and 24 species in the simulation. The simulation time was 10 years with 7300 time steps of 0.5 days. The performance tests for this large-scale problem revealed a speedup of 20 when the simulation was run with up to 32 processors. Beyond 32 processors the code shows poor scalability because the speedup does not increase, but the program does demonstrate the potential for using parallel computation in modeling bioremediation problems (Gwo et al. 2001).

Hammond et al. (2002) developed PARTRAN: a parallel reactive transport model. PARTRAN is a three-dimensional finite-volume model that solves the fully coupled nonlinear system of equations representing the complex geochemical transport processes in the subsurface. It is one of the first parallel codes developed that simulates reactive contaminant transport in groundwater. The developers of PARTRAN point out that while parallel high-performance computers provide the computational power, memory, and processing speed necessary to

simulate problems with greater efficiency in less time, it is difficult to overcome the complexity involved in programming parallel code. It is difficult to write scalable parallel code that maximizes computational efficiency (Hammond et al. 2002). Parallel code often provides greater computational power when used on a few processors, but becomes inefficient when a larger number of processors are used. All parallel simulations require some amount of communication among processors. Therefore, using a higher number of processors requires a greater amount of communication. The challenge is preventing the total communication time from exceeding the computation time on a single processor.

PARTRAN uses an operator-split preconditioned, Jacobian-free, Newton-Krylov method to solve the nonlinear reactive transport equations. The problem used to demonstrate the effectiveness of the code was the transport and sequential degradation of PCE and chlorinated ethenes in a three-dimensional heterogeneous flow field. The preconditioning and solution methods used in PARTRAN scaled much better on parallel computers than the other methods that were tested. The methods used reduced the memory and processing demands, and kept the communication among processors to a minimum. The scalability of the program was 97% when their test problem was solved on 32 processors, and 95% when solved on 64 processors. This reflects the efficiency of the program and suggests that when an even greater number of processors are used to run the model in parallel, the efficiency of the solution method will still remain high. Other preconditioning methods and solution methods tested either showed poor parallel performance, or did not show good scalability (Hammond et al. 2002).

The PARTRAN code showed good parallel performance in terms of simulation time and efficiency for the test problem presented. However, the performance of the code was only demonstrated for one test problem, and scalability could be poor for different test problems. Different transport mechanisms, different chemical reactions, or further geochemical complexities could lead to a great variety in the memory and processing requirements, and the performance of the solution method, and therefore could affect the performance and efficiency of the parallel code. Parallel processing techniques will be explored for additional multi-component reactive transport codes in order to further establish the benefits of developing parallel code in this area.

3 N2D-H2

N2D-H2 is a two-dimensional contaminant transport code for simulation of the autotrophic denitrification process in groundwater. The physical and chemical processes governing the transport and biotransformation of solutes due to denitrification are explained before describing the governing equations and outlining the computer code.

3.1 Denitrification

Nitrate is one of the most common groundwater contaminants found in freshwater aquifers. It is particularly prevalent in shallow unconfined aquifers in rural areas (Nolan et al. 1997). The most common source of nitrate contamination is agricultural activity and fertilizer use. Nitrate contamination also originates from sewage-disposal and industrial practices (Smith et al. 2001). The EPA's federal standard for the maximum contaminant level (MCL) of nitrate in drinking water is 10 mg/L as nitrogen. The MCL for nitrite is 1 mg N/L. In many agricultural areas within the United States, the nitrate levels in domestic wells exceed the 10 mg N/L standard; therefore, strategies have been developed to remediate the nitrate contamination.

The most effective method of removing nitrates from subsurface water is the naturally occurring process of denitrification. Denitrification is a process carried out by microorganisms in groundwater that converts nitrate into a gaseous form of nitrogen. Denitrification can take place in groundwater when the soil is deficient in oxygen, or anaerobic, and sufficient carbon for energy is available from organic materials (Fedkiw 1991). Alternatively, hydrogen-enhanced denitrification is a potential treatment process that eliminates the need for an organic electron donor. Denitrifying bacteria can autotrophically reduce nitrate, using hydrogen as an electron donor (Smith et al. 2001). In this situation, the stoichiometric relationship for the reduction of nitrate to nitrogen gas is as follows:

$$2NO_3^-(aq) + 5H_2 \rightarrow N_2(g) + 4H_2O + 2OH^-$$

(Killingstad et al. 2002). Hydrogen is used as an electron donor to reduce nitrate and produce water and nitrogen gas, two innocuous end products (Smith et al. 2001). During this process, nitrate (NO_3^-) reduction results in nitrite (NO_2^-) production. Nitrite is then used along with

nitrate as an additional electron acceptor and reduced to nitrogen gas. Growth and decay of the nitrate/nitrite-reducing microbial population also occurs.

N2D-H2 is a two-dimensional contaminant transport code that simulates this autotrophic denitrification process in groundwater and can be applied to groundwater systems that are contaminated with nitrate. For denitrification to occur, the presence of nitrate or nitrite as an electron acceptor, denitrifying bacteria, anaerobic conditions, and an adequate supply of electron donors are required. The N2D-H2 code simulates the transport and biodegradation of five solutes: hydrogen, oxygen, nitrate, nitrite, and gaseous denitrification end products, as well as the growth and decay of the nitrate/nitrite-reducing microbial population, which is assumed to be present in the aquifer system. Hydrogen acts as the electron donor, and oxygen, nitrate, and nitrite are the electron acceptors. The N2D-H2 code solves the system of six differential equations that results from defining the concentration of the five solutes plus the microbial population concentration.

3.2 Governing equations

The general form of the two-dimensional advection-dispersion equation including retardation and biological reactions is given by

$$R\frac{\partial C}{\partial t} = D_{xx}\frac{\partial^2 C}{\partial x^2} + D_{yy}\frac{\partial^2 C}{\partial y^2} - v_x\frac{\partial C}{\partial x} - v_y\frac{\partial C}{\partial y} + \frac{q_s}{\theta}C_s - R_{bio,C}^{\sin k} + R_{bio,C}^{source} \qquad \textbf{Equation 1}$$

where R = coefficient of retardation [dimensionless]

C = solute concentration [M/L^3]

t = time [T]

D_{xx} = longitudinal dispersion coefficient [L^2/T]

D_{yy} = transverse dispersion coefficient [L^2/T]

x = distance along flow path in the longitudinal direction [L]

y = distance along flow path in the transverse direction [L]

v_x = average groundwater velocity in the longitudinal direciton [L/T]

v_y = average groundwater velocity in the transverse direciton [L/T]

q_s = volumetric flux of water per unit volume of aquifer [1/T]

C_s = solute point source concentration [M/L^3]

$R_{bio,C}^{source}$ = production of mass resulting from microbial processes [M/L^3 per T]

$R_{bio,C}^{\sin k}$ = mass loss due to biological utilization [M/L^3 per T]

Table 1. Reaction term variable definitions

H	hydrogen concentration (mg/L)
O	oxygen concentration (mg/L)
$NO3$	nitrate concentration (mg N/L)
$NO2$	nitrite concentration (mg N/L)
$N2$	nitrogen concentration (mg N/L)
M	biomass concentration (mg/L)
θ	porosity
η_O	rate of H_2 utilization with oxygen as electron acceptor
η_{NO3}	rate of H_2 utilization with nitrate as electron acceptor
η_{NO2}	rate of H_2 utilization with nitrite as electron acceptor
γ_O	oxygen use coefficient (mg O/mg H)
γ_{NO3}	nitrate use coefficient (mg NO3/mg H)
γ_{NO2}	nitrite use coefficient (mg NO2/mg H)
ζ_{NO3}	nitrite production coefficient
ζ_{NO2}	nitrogen production coefficient
$K_{H,O}$	hydrogen saturation constant for oxygen reduction
$K_{H,NO3}$	hydrogen saturation constant for nitrate reduction
$K_{H,NO2}$	hydrogen saturation constant for nitrite reduction
K_O	oxygen saturation constant
K_{NO3}	nitrate saturation constant
K_{NO2}	nitrite saturation constant
$K_{O2\text{-}NO3}$	oxygen inhibition constant
$K_{NO3\text{-}NO2}$	nitrate inhibition constant

3.3 Reaction terms

The advection-dispersion equation is solved for each solute. In the denitrification process considered in the N2D-H2 code, there are five solutes: hydrogen, oxygen, nitrate, nitrite, and gaseous nitrogen. Therefore, this equation is solved for C representing five different solutes.

The concentrations are related by the chemical reaction terms. The reaction terms are given below. These equations were derived from Killingstad et al. (2002), and Waddill et al. (1998). There are biological sink terms for hydrogen, oxygen, nitrate, and nitrite. Hydrogen gets utilized as an electron donor. Oxygen, nitrate, and nitrite are utilized as electron acceptors. The three electron acceptors get utilized sequentially, which is represented by the inhibition terms in the utilization terms for nitrate and nitrite (Waddill et al. 1998). Oxygen is used as an electron acceptor preferentially before nitrate, and nitrate is used preferentially before nitrite. There are biological source terms for nitrite and N_2, which are produced during the utilization of nitrate. The production of nitrite and N_2 resulting from denitrification are proportional to the amounts of nitrate and nitrite consumed, respectively (Killingstad et al. 2002). The parameters used in the reaction terms are defined in Table 1.

Hydrogen Utilization

The total hydrogen utilization is equal to utilization due to oxygen-based reduction + utilization due to nitrate-based reduction + utilization due to nitrite based reduction (Killingstad et al. 2002).

$$R_{bio,H}^{\sin k} = \frac{M}{\theta}\eta_O\left[\frac{H}{K_{H,O}+H}\right]\left[\frac{O}{K_O+O}\right]$$
$$+\frac{M}{\theta}\eta_{NO3}\left[\frac{H}{K_{H,NO3}+H}\right]\left[\frac{NO3}{K_{NO3}+NO3}\right]\left[\frac{K_{O2-NO3}}{K_{O2-NO3}+O}\right]$$
$$+\frac{M}{\theta}\eta_{NO2}\left[\frac{H}{K_{H,NO2}+H}\right]\left[\frac{NO2}{K_{NO2}+NO2}\right]\left[\frac{K_{O2-NO3}}{K_{O2-NO3}+O}\right]\left[\frac{K_{NO3-NO2}}{K_{NO3-NO2}+NO3}\right]$$

Oxygen Utilization

$$R_{bio,O}^{\sin k} = \frac{M}{\theta}\eta_O\gamma_O\left[\frac{H}{K_{H,O}+H}\right]\left[\frac{O}{K_O+O}\right]$$

Nitrate Utilization

$$R_{bio,NO3}^{\sin k} = \frac{M}{\theta}\eta_{NO3}\gamma_{NO3}\left[\frac{H}{K_{H,NO3}+H}\right]\left[\frac{NO3}{K_{NO3}+NO3}\right]\left[\frac{K_{O2-NO3}}{K_{O2-NO3}+O}\right]$$

Nitrite Utilization

$$R_{bio,NO2}^{\sin k} = \frac{M}{\theta}\eta_{NO2}\gamma_{NO2}\left[\frac{H}{K_{H,NO2}+H}\right]\left[\frac{NO2}{K_{NO2}+NO2}\right]\left[\frac{K_{O2-NO3}}{K_{O2-NO3}+O}\right]\left[\frac{K_{NO3-NO2}}{K_{NO3-NO2}+NO3}\right]$$

Nitrite Production

$R_{bio,NO2}^{source} = \zeta_{NO3} R_{bio,NO3}^{\sin k}$, where ζ_{NO3} = nitrite production constant (M of NO_2^-/M of NO_3^-)

N2 Production

$R_{bio,N2}^{source} = \zeta_{NO2} R_{bio,NO2}^{\sin k}$, where ζ_{NO2} = N_2 production constant (M of N_2/M of NO_2^-)

3.4 Solution method

The advection-diffusion equation is solved in two phases for each time step using the split-operator approach, in which the advection part of the transport is solved separately. During each time step, the solution of the concentration arrays consists of two steps:

1) Transport of each solute is solved assuming advection is the only transport mechanism. Thus, for one-dimensional flow in the x-direction, the partial differential equation solved here is:

$$\frac{\partial C}{\partial t} = -v_x \frac{\partial C}{\partial x}$$

This equation is solved with an explicit finite difference approximation. In order to limit the numerical instability of the explicit method, the advection time step is determined from a Courant number equal to one, i.e. $\frac{v_x \Delta t}{\Delta x} = 1.0$ (Anderson et al. 1995). The spatial derivative is approximated with a backward difference and the time derivative is approximated with a forward difference. Therefore, the finite difference form of the advective transport equation is:

$$\frac{C_{i,j}^{n+1} - C_{i,j}^{n}}{\Delta t} = -\frac{v_x}{\Delta x}\left(C_{i,j}^{n} - C_{i-1,j}^{n}\right),$$

where i = the x-node and j = the y-node, and n indicates the time step. Because the Courant number is equal to one, the solution to this equation is:

$$C_{i,j}^{n+1} = C_{i-1,j}^{n},$$

i.e. in one time step, the solute travels a distance equal to Δx.

2) Next, the diffusion-reaction transport is solved. The equation describing diffusion and reactions is:

$$R\frac{\partial C}{\partial t} = D_{xx}\frac{\partial^2 C}{\partial x^2} + D_{yy}\frac{\partial^2 C}{\partial y^2} + \frac{q_s}{\theta}C_s - R_{bio,C}^{\sin k} + R_{bio,C}^{source}$$

The biotransformation terms are solved first using the kinetic equations defined previously. The solute concentrations used to calculate the biotransformation terms are the new concentrations after advection has occurred during this time step. The finite difference form of the dispersion-reaction equation that is used in the N2D-H2 code is given as follows:

$$R\frac{C_{i,j}^{n+1} - C_{i,j}^{n}}{\Delta t} =$$

$$D_{xx}\frac{C_{i-1,j}^{n+1} - 2C_{i,j}^{n+1} + C_{i+1,j}^{n+1}}{(\Delta x)^2} + D_{yy}\frac{C_{i,j-1}^{n+1} - 2C_{i,j}^{n+1} + C_{i,j+1}^{n+1}}{(\Delta y)^2} + (q_s C_s)_{i,j}^{n+1} + \left(R_{bio,C}^{source}\right)_{i,j}^{n+1} - \left(R_{bio,C}^{\sin k}\right)_{i,j}^{n+1},$$

where i = the x-node and j = the y-node, and n indicates the time step. The spatial derivatives are evaluated implicitly, at the advanced time step. The time derivative is evaluated with a forward difference. The form of the equation that is solved in the N2D-H2 code is given as follows:

$$-\left(\frac{D_{xx}\Delta t}{R(\Delta x)^2}\right)C_{i-1,j}^{n+1} + \left(1 + 2\frac{D_{xx}\Delta t}{R(\Delta x)^2} + 2\frac{D_{yy}\Delta t}{R(\Delta y)^2}\right)C_{i,j}^{n+1} - \left(\frac{D_{xx}\Delta t}{R(\Delta x)^2}\right)C_{i+1,j}^{n+1} =$$

$$C_{i,j}^{n} + \left(\frac{D_{yy}\Delta t}{R(\Delta y)^2}\right)\left(C_{i,j-1}^{n+1} - C_{i,j+1}^{n+1}\right) + (q_s C_s)_{i,j}^{n+1} + \left(R_{bio,C}^{source}\right)_{i,j}^{n+1} - \left(R_{bio,C}^{\sin k}\right)_{i,j}^{n+1}$$

This finite difference equation takes the form [A]{x}={f}, which is solved by the Thomas algorithm. The matrix [A] represents the coefficients of the terms on the left-hand side of the equation, {x} represents the vector of concentrations at the new time step for a row of nodes, and {f} is the right-hand side of the equation, which can be calculated from the concentrations at the old time step. The values $\left(C_{i,j-1}^{n+1}\right)$ and $\left(C_{i,j+1}^{n+1}\right)$ on the right-hand side of the equation are updated with each iteration of the solution. For the problem considered in this study, the source term is always zero for all solutes except for hydrogen. It is greater than zero at the nodes representing the locations of the membranes that introduce the hydrogen to the porous media. The source concentration is determined from the physical characteristics of the membranes.

The disadvantage to the split-operator approach is the introduction of error in the solution that results from the approximation of advective transport as a discrete process from the diffusion-reaction transport, when these physical processes are in fact occurring simultaneously. However, the error introduced by the split-operator approach is not substantial and can be minimized by the numerical procedures used to solve the split problem (Stefanovic et al. 2001).

3.5 Sequential code

N2D-H2 solves the finite difference form of the advection-dispersion equation for each solute. The code consists of a main program and four primary subroutines. A flowchart for the main program is provided in Figure 2. The user must input the initial concentrations, boundary conditions, time and spatial discretization information, locations and concentrations of sources and sinks, various aquifer parameters, and parameters defining the biotransformation processes. The changes in concentration due to advection, dispersion, sources and sinks, and biotransformation are solved in independent subroutines. The new concentration values at each time step are then solved in the TRIDIA subroutine. A flowchart for the TRIDIA subroutine is provided in Figure 3. The following section describes each of the main subroutines in the N2D-H2 code.

Advection subroutine – ADVECTION

The advection subroutine calculates the change in concentration of each species due to advection. The assumption is made that the transport time steps are identical to the flow time

steps, which simplifies the solution of the concentration array due to advective transport. The transport time step is determined by a Courant Number equal to one, so that the concentration of a solute at a node during the current time step is equal to the concentration at the node directly upstream at the previous time step.

Biotransformation subroutine – BIOTRANS

The biotransformation subroutine solves the terms for biotransformation and generation using microbial population growth dynamics. The definitions of the utilization and production terms that are solved in this subroutine are given in the above section on the governing equations. It also calculates the biomass concentration at the current transport step.

Solve subroutine – TRIDIA

The subroutine TRIDIA solves the matrix form of the equations for dispersive-reactive transport iteratively. A coefficient matrix is set up for each row of the grid and the Thomas algorithm is used to solve for the concentration array row-by-row. When all rows have been solved, the error from the previous iteration at each grid node is compared to the error tolerance to determine whether an additional iteration is needed. When the solution has converged at the current time step, the concentration arrays are then prepared for the next time step.

Mass Balance subroutine – MASBAL

The mass balance subroutine calculates the mass of each constituent into the system and out of the system during the current time step.

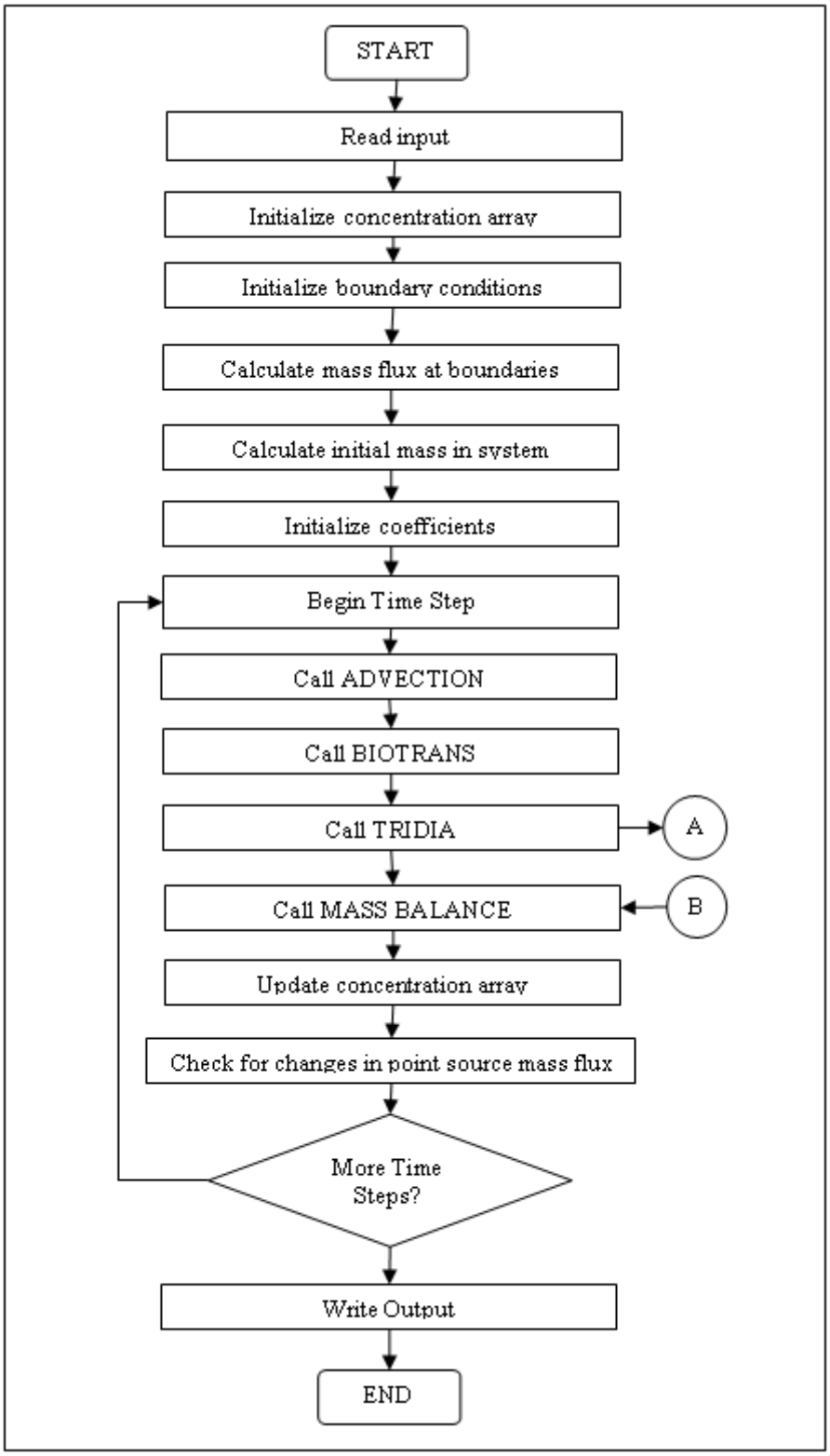

Figure 2. Flowchart for the main program of the N2D-H2 code

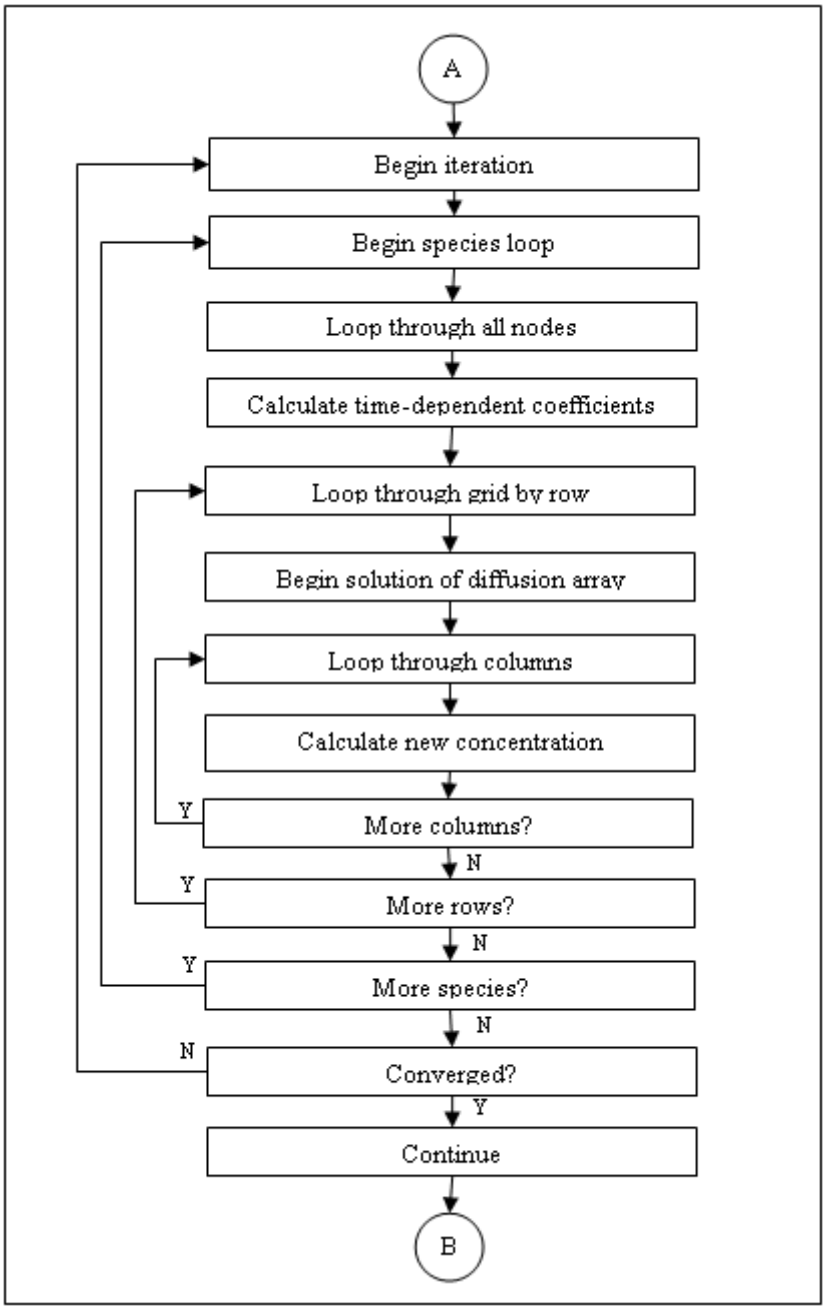

Figure 3. Flowchart for TRIDIA subroutine

3.6 Parallel code

The N2D-H2 code was used as a test code to determine the feasibility and benefits of parallelizing a larger multi-component solute transport code. N2D-H2 is a smaller scale model that uses some simplifying assumptions in solving the transport equations. However, the overall format of the code is similar to commonly used contaminant transport models, in that the same

governing equations are being considered and a standard finite-difference form of the equations is solved. Therefore, developing a parallel version of N2D-H2 serves as a reliable example of the process of parallelizing transport codes such as MT3DMS or SEAM3D. MT3DMS (A Modular Three-Dimensional Multispecies Transport Model for Simulation of Advection, Dispersion, and Chemical Reactions of Contaminants in Groundwater Systems) is the most widely used code for contaminant transport modeling and remediation assessment studies, and was developed by Zheng and Wang (1999). The multispecies aspect of the code was an expansion of the original model and indicates the capability for accommodating add-on reaction packages (Zheng 1999). The original code MT3D contains packages for advection, dispersion, sink and source mixing, and chemical reactions. SEAM3D (Sequential Electron Acceptor Model, Three-Dimensional) is a collection of add-on packages including biodegradation, NAPL dissolution, cometabilism, reductive dechlorination, and phytoremediation. SEAM3D was developed by Widdowson and Waddill (1998). Once the feasibility of writing parallel versions of contaminant transport models has been demonstrated with the N2D code, the parallelization of MT3D and SEAM3D will be explored.

The parallelization process for the N2D-H2 code included decomposing the problem domain and implementing the communication required among processors to preserve the original solution method.

3.6.1 Domain decomposition

The domain decomposition involves figuring out how the computational effort should be divided among multiple processors. The computational effort must be equally divided among the processors, and the communication required between processors must be kept to a minimum. Computational effort implies higher CPU time, so keeping the work load balanced among the processors is essential. Also, communication among processors is very time consuming, so dividing the domain in such a way as to minimize communication is also important.

In order to determine the best way to decompose the problem domain, the solution technique must be considered. This includes consideration of where most of the computational effort occurs and for different decomposition techniques, where communication would be required. Depending on the solution method, this inspection may reveal that it would be more efficient for a single processor to perform the computations for a long narrow section of the grid,

or it may reveal that more square-like partitions would better meet the decomposition criteria. Thus, in two dimensions, the problem grid could be divided along rows, or along columns. The decomposition could be one-dimensional, two-dimensional, or three-dimensional. In the case of solute transport, an additional dimension for decomposition is possible (the three-dimensional concentration arrays have a fourth dimension indicating which solute is being considered).

The parallel version of the N2D-H2 code employs a one-dimensional domain decomposition. The bulk of the parallel computation occurs in the TRIDIA subroutine where the concentration arrays are solved iteratively. The solution method calculates the new concentration values row by row, and, within a single iteration, no communication is required between rows. Therefore, the domain was decomposed by groups of rows. Figure 4 shows a hypothetical one-dimensional decomposition of a finite difference grid among three processors.

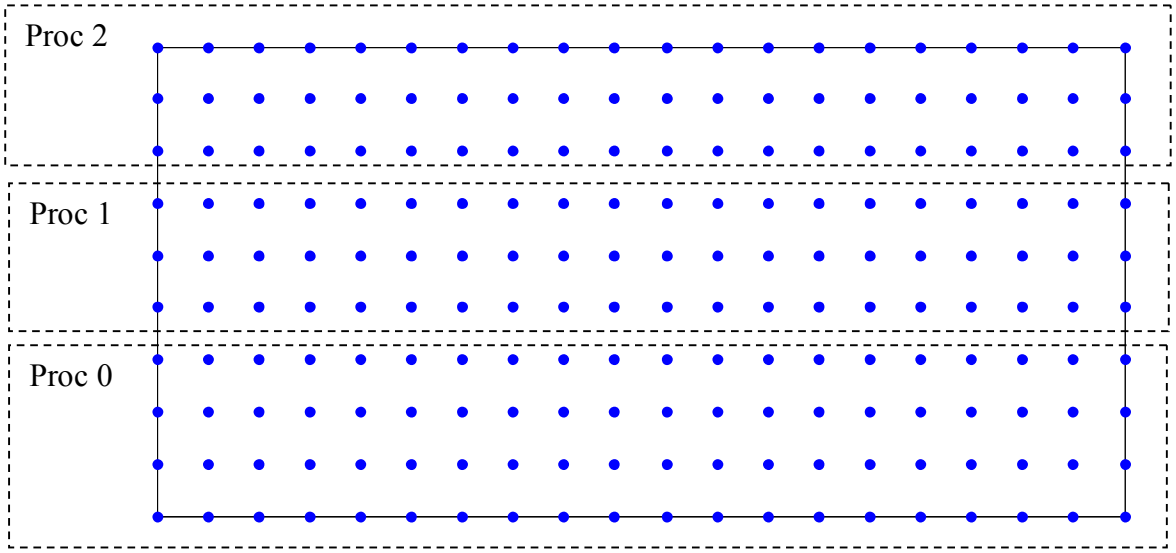

Figure 4. One-dimensional domain decomposition

3.6.2 Distribution of data

Each processor calculates the new concentration values at all of the nodes in its local domain. The total number of rows is divided evenly among the number of processors used to run the simulation, which is defined by the user at runtime. Each processor holds its own set of data for the entire problem grid, so all processors initialize variables, read all of the input data, set the initial boundary conditions, and calculate all parameters defining the problem. The

parallelization occurs in the ADVECT and TRIDIA subroutines, where the concentration arrays at the new time step are solved.

3.6.3 Initial parallel analysis

For a transport problem involving 5 solutes, consider a grid consisting of 50 nodes in the x-direction and 10 nodes in the y-direction. These discretization parameters require 50×10×5 = 2,500 calculations to be performed for each iteration at every time step in order to calculate new concentration values. A typical problem using the N2D-H2 code could consist of 250 time steps. A problem of this size was tested and took an average of 7 iterations to converge. This leads to a total of 2,500×250×7 = 4,375,000 times that the transport equation must be solved. The total required CPU time is determined by the time it takes for the machine to do a floating point calculation and the number of floating point calculations required to solve the transport equation at one node. This analysis illustrates the bulk of computation that occurs in the TRIDIA subroutine. With the domain decomposition described above, if two processors are used, then the total computing time required in TRIDIA will ideally be halved.

A flowchart of the parallel algorithm for the TRIDIA subroutine is provided in Figure 5. The shaded boxes indicate the communication required for the solution to be computed in parallel.

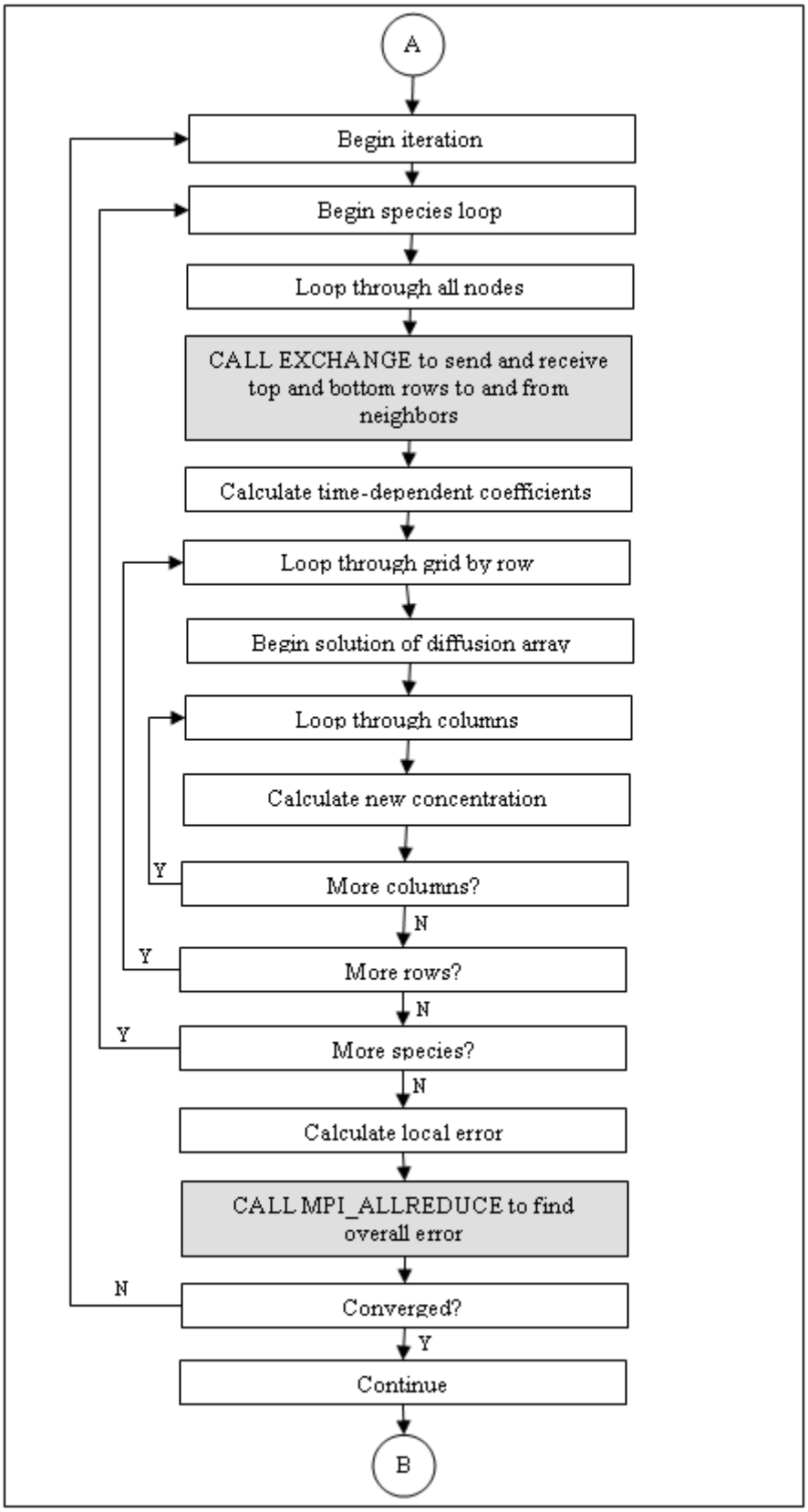

Figure 5. Flowchart for TRIDIA subroutine in parallel

3.7 Input/Output

Input to the N2D program includes all of the information that defines a particular simulation. This information includes space discretization parameters, timing discretization parameters, the number of solutes that are simulated, and coefficients and constants that define the interactions of the solutes. Figure 6 lists all of the numerical and transport parameters that are read as input to the N2D-H2 program, and Figure 7 lists the biodegradation parameters.

The N2D-H2 program writes the simulation results to several output files. The output files are named and contain the information as follows:

- OUTPUT.DAT – Simulation parameters, Number of iterations for each time step
- CONC.DAT – Concentration of solutes at every point in domain at specified output times
- CVSX.DAT – Concentration of all solutes vs. distance at specified output times
- CVSXY.DAT – Concentration of solutes at each point in domain at specified output times
- OBS.DAT – Concentration of each solute vs. time at each observation point
- BALANCE.DAT – Mass balance information

Title-line1 of simulation
Title-line2 of simulation
Number of x-nodes, Number of y-nodes, Number of solutes, Number of biological species
Node spacing in x-direction, Node spacing in y-direction, Vertical thickness
Solution option, Restart option
Error tolerance
Number of time steps, Time step size
Velocity in x-direction, Longitudinal dispersivity, Transverse dispersivity, Porosity, Bulk density
For each solute:
    Solute name
    Initial concentration, Boundary source concentration, Minimum concentration
    Boundary concentration option, Initial concentration option
    Molecular diffusion coefficient, Distribution coefficient

Figure 6. N2D-H2 Input (numerical and transport parameters)

For each microbial population:

Microbial population name

Initial concentration, Minimum concentration

Biomass yield coefficient, Biomass decay coefficient

Initial concentration option

H2 rate of utilization (for Oxygen, Nitrate, Nitrite)

H2 saturation constant (for Oxygen, Nitrate, Nitrite)

Oxygen use coefficient, Oxygen saturation constant

Inhibition coefficient (O2-NO3), Inhibition coefficient (O2-NO2)

Nitrate use coefficient, Nitrate saturation constant, Nitrate production coefficient

Inhibition coefficient (NO3-NO2)

Nitrite use coefficient, Nitrite saturation constant, Nitrite production coefficient

Number of point sources, Number of flux periods

Henry's law coefficient, H2 molecular weight, Kinematic viscosity of water

H2 mass transfer constant, H2 flux zone porosity

For each point source:

Membrane surface area, Membrane diameter

For each point source:

Location of source (x-node), Location of source (y-node), Membrane pressure

Number of observation points

For each observation point

Location of obs. point (x-node), Location of obs. point (y-node)

Frequency of output

Figure 7. N2D-H2 Input (biodegradation and source/sink parameters)

4 Test Problems

Several test problems were developed to test the N2D-H2 code and were used to compare the results of the parallel code to the results obtained using the original version of the code. Once the parallel code produced results identical to the original results, the simulation was used to perform various timing tests on the code. This section describes the test problems before presenting the results.

4.1 One-dimensional problem

4.1.1 Conceptual model

The problem used to test and time the parallel N2D code involves replicating a lab experiment that observes the denitrification process in a hypothetical aquifer contaminated with nitrate. The area considered is 1.2 m long x 0.6 m wide. The left-hand boundary of the model has a constant nitrate contamination that is transported into the rest of the problem domain through advection. Hydrogen is introduced into the porous media by membranes located 0.2 m from the left-hand boundary, and the denitrification process is observed along the length of the problem domain. The problem is simulated in two dimensions, but because the nitrate contamination is constant along the width of the domain, and all other parameters are constant throughout the domain, this is essentially a one dimensional problem. There is a microbial population present in the aquifer responsible for autotrophic denitrification. The N2D code is used to observe the concentration changes of hydrogen, dissolved oxygen, nitrate, nitrite, nitrogen, and the microbial population for 50 days after the hydrogen injection begins. Oxygen, nitrate, and nitrite are electron acceptors; hydrogen is the electron donor; the microbial population is autotrophic denitrifiers; and, gaseous nitrogen is the end product of the denitrification process. The flow conditions are assumed to be constant throughout the domain for the length of the simulation, and the constant velocity is read as input to the model. Figure 8 gives a diagram of the problem and shows the 49x25 node finite difference grid used to solve the problem.

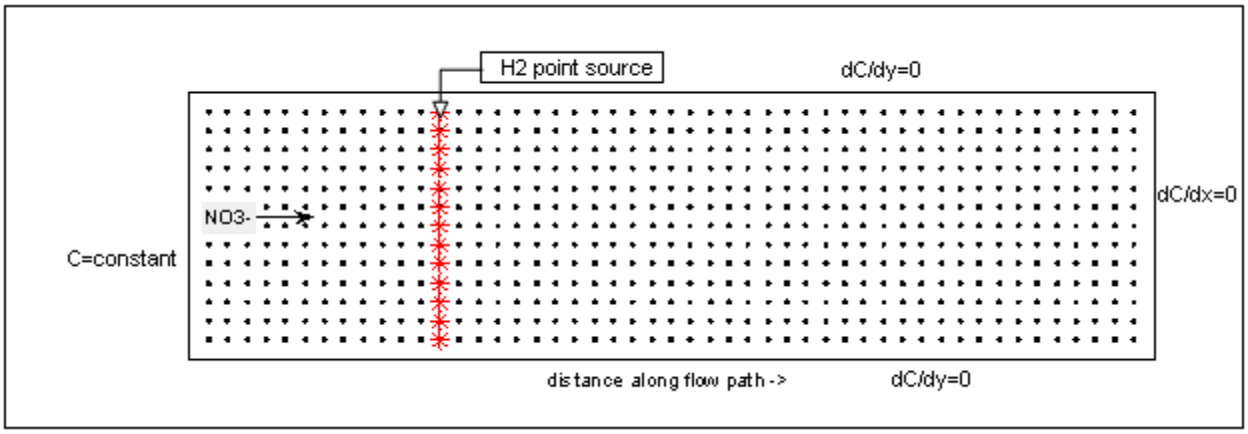

Figure 8. Test problem conceptual model

4.1.2 Finite difference grid

The finite difference grid is shown in Figure 8. The program uses a block-centered finite difference grid to solve the finite difference form of the partial differential equations. The grid is equally spaced throughout the problem domain with dx = 0.025 m and dy = 0.025 m.

4.1.3 Time discretization

To limit the effects of numerical dispersion in the simulation results, the time step size must be limited based on the Courant number (Anderson et al. 1992). The condition for determining a reasonable time step size is defined as follows:

$$\text{Courant number} = \frac{v_x \Delta t}{\Delta x} \approx 1.0 ,$$

which leads to a time step size of 0.083 days for this simulation. The total length of the simulation is 50 days.

4.1.4 Initial conditions

The initial concentrations of the five solutes and the microbial population specified in the test problem are given in

Table 2. The initial concentrations given are constant throughout the entire problem domain.

Table 2. Test problem initial conditions

Solute	Hydrogen	Oxygen	Nitrate	Nitrite	N_2
Initial concentration (mg/L)	0.0001	1.00E-05	1.00E-08	1.00E-08	1.00E-08
Microbial population	Autotrophic denitrifiers				
Initial concentration (mg/L)	2.5				

4.1.5 Boundary conditions

The left boundary of the problem domain shown in Figure 8 is a constant concentration boundary for all solutes. The concentrations of each of the solutes at the boundary are given in Table 3.

Table 3. Boundary concentrations

Solute	Hydrogen	Oxygen	Nitrate	Nitrite	N_2
Concentration at left boundary (mg/L)	0.0001	2	10.2	1.00E-08	1.00E-08

The top, bottom, and right-hand boundaries of the problem domain are no mass flux boundaries, corresponding to the boundaries of the laboratory experiment.

4.1.6 Sources/Sinks

Hydrogen is injected into the aquifer through a line of membranes acting as point sources. The location of the membranes are at column 8 of the grid (x = 0.2 m), along the entire width of the aquifer. The hydrogen sources are constant throughout the entire simulation (until t = 50 days).

4.1.7 Parameters

Table 4 through Table 8 give the values of all of the remaining parameters necessary for the simulation.

Table 4. Simulation parameters

Number of x-nodes	*nxn*	49
Number of y-nodes	*nyn*	25
Cell width along rows (m)	Δx	0.0255
Cell width along columns (m)	Δy	0.0255
Layer thickness (m)	Δz	0.025
Longitudinal velocity (m/day)	v_x	0.306
Transverse velocity (m/day)	v_y	0
Longitudinal dispersivity (m)	α_L	0.01
Transverse dispersivity (m)	α_T	0.01
Porosity	θ	0.35
Bulk density (kg/m)	ρ_b	1.9e-06
Advection time step size (days)	Δt	0.08333
Simulation time (days)		50

Table 5. Microbial population parameters

Microbial population	Autotrophic denitrifiers
Biomass yield coefficient	0.5
Biomass decay coefficient	0

Table 6. H2 source parameters

Henry's law coefficient	1.23
H2 molecular weight	2.016
Kinematic viscosity of water (m^2/day)	0.08667
H2 flux zone porosity	0.35
H2 mass transfer constant (a)	0.824
H2 mass transfer constant (b)	0.39
H2 mass transfer constant (c)	0.33
Number of flux periods	1
Duration of flux period (days)	50

Table 7. Solute parameters

Solute	Hydrogen	Oxygen	Nitrate	Nitrite	N2
Molecular diffusion coeff (m^2/day)	0.3456e-03	0	0	0	0
Distribution coefficient (m^3/kg)	0	0	0	0	0
Retardation factor	1	1	1	1	1
Longitudinal dispersion coeff (m^2/day)	0.3406e-02	0.306e-02	0.306e-02	0.306e-02	0.306e-02
Transverse dispersion coeff (m^2/day)	0.3406e-02	0.306e-02	0.306e-02	0.306e-02	0.306e-02

Table 8. Electron acceptor parameters

Electron Acceptor	Oxygen	Nitrate	Nitrite
H2 rate of utilization (mg/L/day)	8	0.5	0.25
Use coefficient	8	23	1.533
H2 saturation constant (mg/m^3)	0.01	0.1	1
EA saturation constant (mg/m^3)	11	20	5
Inhibition coefficient (mg/m^3)	0.9e+10 (O2-NO3) 0.9e+10 (O2-NO2)	0.9e+10 (NO3-NO2)	-
Production coefficient (mg N/L/day)	-	0.7419	0.3043

4.2 Two-dimensional problem

A two-dimensional problem was developed to test the code. This problem is identical to the one-dimensional problem except for the location of the nitrate source. Instead of having nitrate contamination flowing in to the problem domain along the entire width of the upstream boundary, the source is concentrated into an area of 0.1 m at the center of the boundary. This creates a plume of nitrate contamination in the domain. Because the concentration is no longer constant across the width of the domain, this requires a two-dimensional simulation. The hydrogen membranes are again introduced at 0.2 m from the left boundary, but they do not extend along the width of the domain as they did in the one dimensional problem. The hydrogen sources are only added to intercept the nitrate contamination. The problem domain with the nitrate sources and hydrogen sources is shown in the conceptual model in Figure 9.

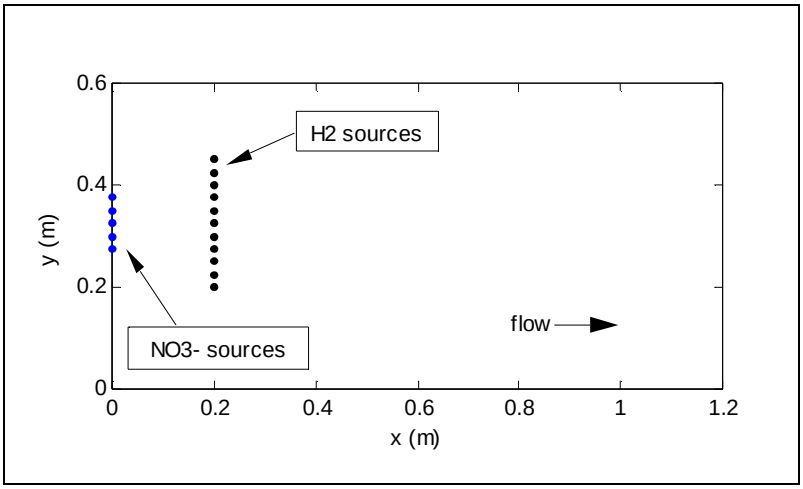

Figure 9. Two-dimensional conceptual model; one hydrogen source

The nitrate concentration along the remaining part of the left-hand boundary is equal to the initial nitrate concentration in the rest of the problem domain. The rest of the conceptual model is identical to the one-dimensional problem. The boundary conditions are the same, and all input parameters are the same.

A second two-dimensional problem was developed by adding two additional lines of hydrogen sources to the domain. In the previous problem, the nitrate was reduced by the autotrophic denitrifiers, but the nitrate reduction results in nitrite production. The second and

third sets of hydrogen sources will ideally be used to then reduce the nitrite contamination as it migrates further downstream. The conceptual model for this problem is shown in Figure 10.

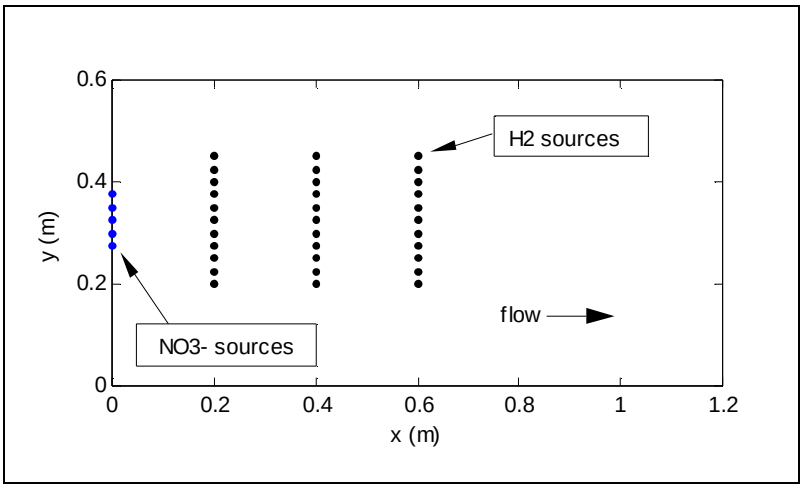

Figure 10. Two-dimensional conceptual model; two hydrogen sources

Before three lines of hydrogen sources were used, the simulation was performed with just two lines of hydrogen sources. It was thought that the addition of one line of sources would reduce the nitrite plume, however the change in the nitrite plume was not significant compared to the model using one line of hydrogen sources. In order to observe a reduction of the nitrite plume, three lines of hydrogen sources were needed.

5 Results and Discussion

5.1 Test problem results

The results of the previously defined simulations are presented in this section. For both the one-dimensional and two-dimensional problems, an initial simulation was performed without any hydrogen sources in order to see the nitrate contamination spread throughout the domain. The simulations with the hydrogen sources then showed the effects of the denitrification process.

Both the sequential version of the code and the parallel version of the code were used to simulate all of the test problems. The results presented here were identical for both the sequential and parallel simulations.

5.1.1 One-dimensional results

The model was first developed without the hydrogen source in order to observe the transport of the nitrate contamination when denitrification was not taking place. Figure 12 shows the nitrate concentration within the problem domain at 1, 3, and 5 days from the beginning of the simulation. The nitrate contamination has spread to 10 mg N/L in the entire domain.

Figure 11 shows the nitrate concentration with time at an observation point on the right boundary of the problem domain. These two figures demonstrate that without a hydrogen source, the nitrate contamination is transported throughout the entire domain within 5 days.

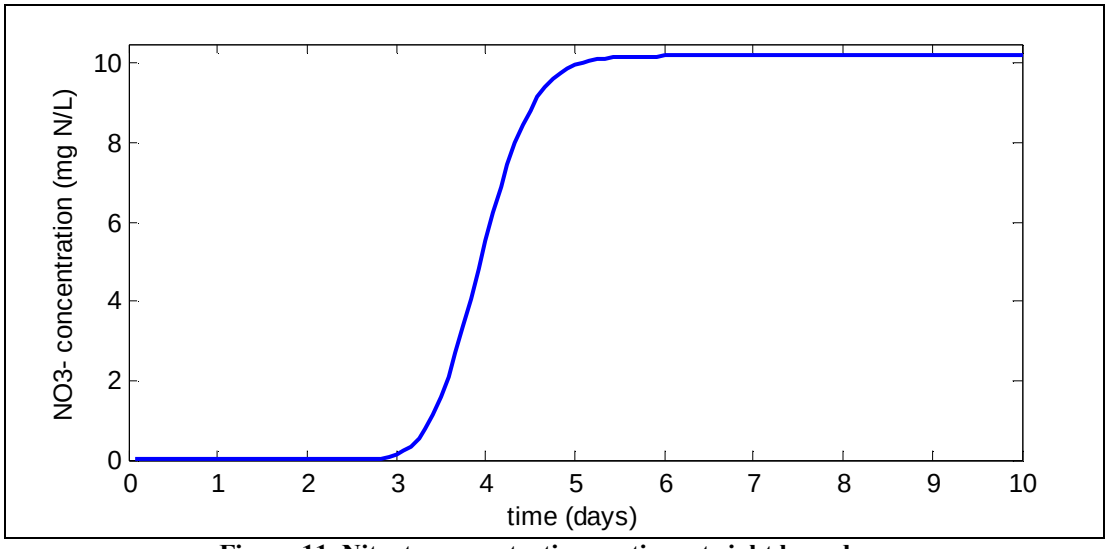

Figure 11. Nitrate concentration vs. time at right boundary

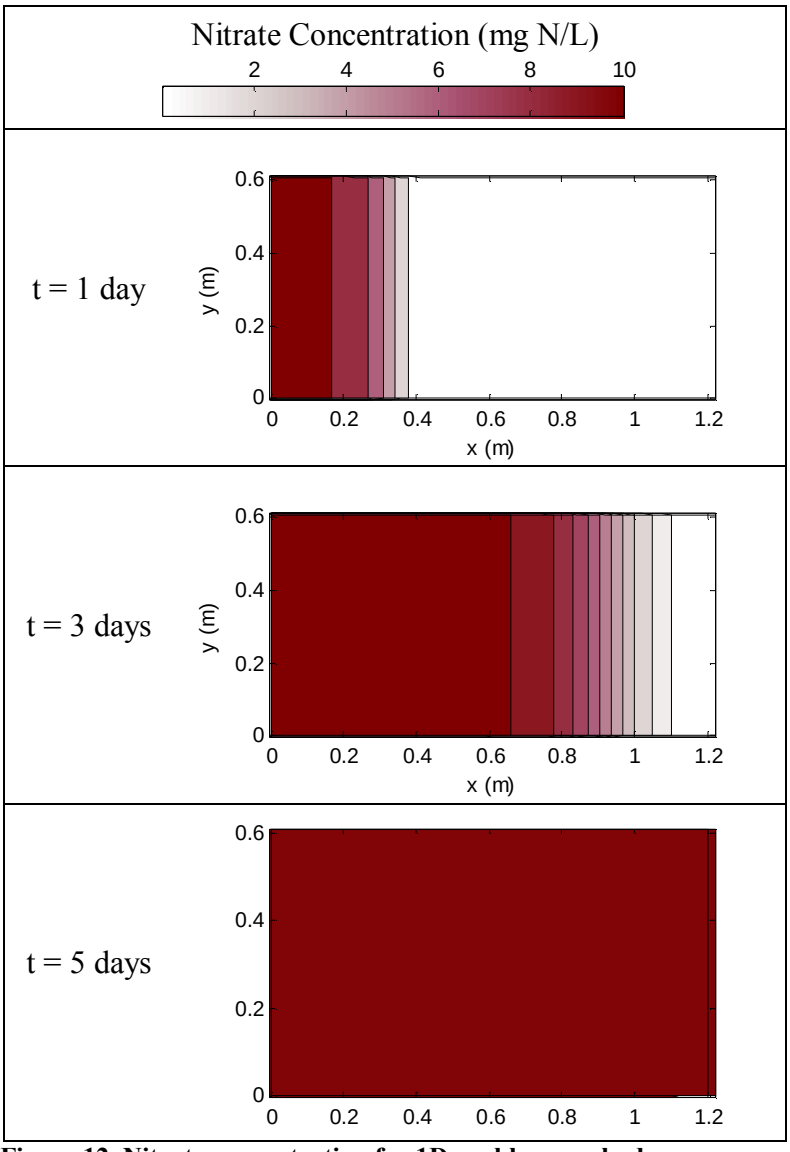

Figure 12. Nitrate concentration for 1D problem; no hydrogen source

Now a hydrogen source is added to the system as described in Chapter 4. The source is added at each node along the width of the domain at 0.2 m from the left boundary. Figure 13 shows the concentration of nitrate within the problem domain at 5 and 50 days after the start of the simulation. The hydrogen source was added at $x = 0.2$ m, beyond which there is now minimal nitrate contamination. The nitrate has been reduced by the microbial population, which uses the hydrogen as an electron donor.

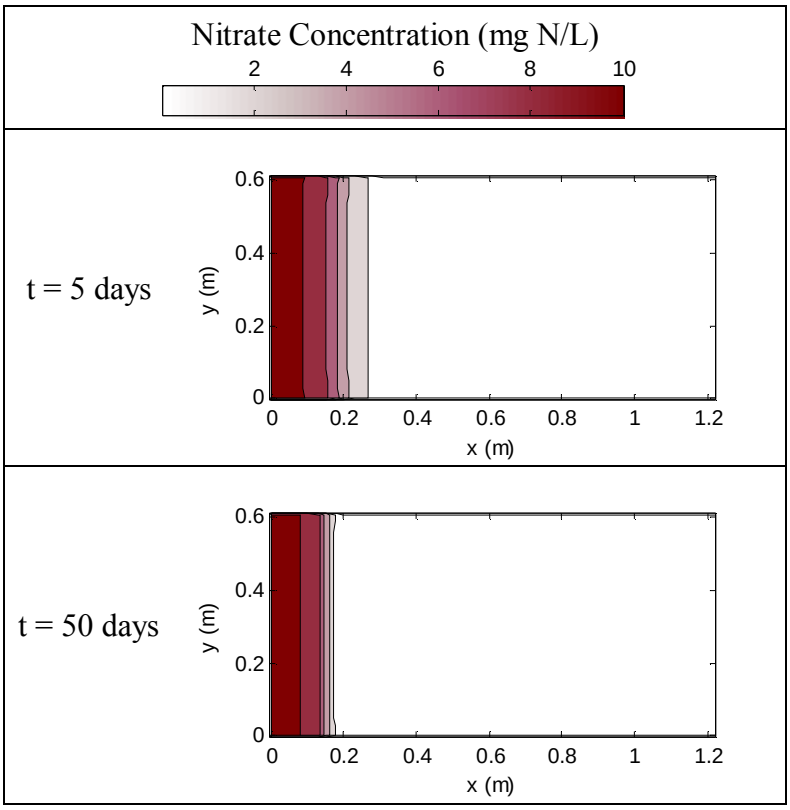

Figure 13. Nitrate concentration for 1D problem; with hydrogen source

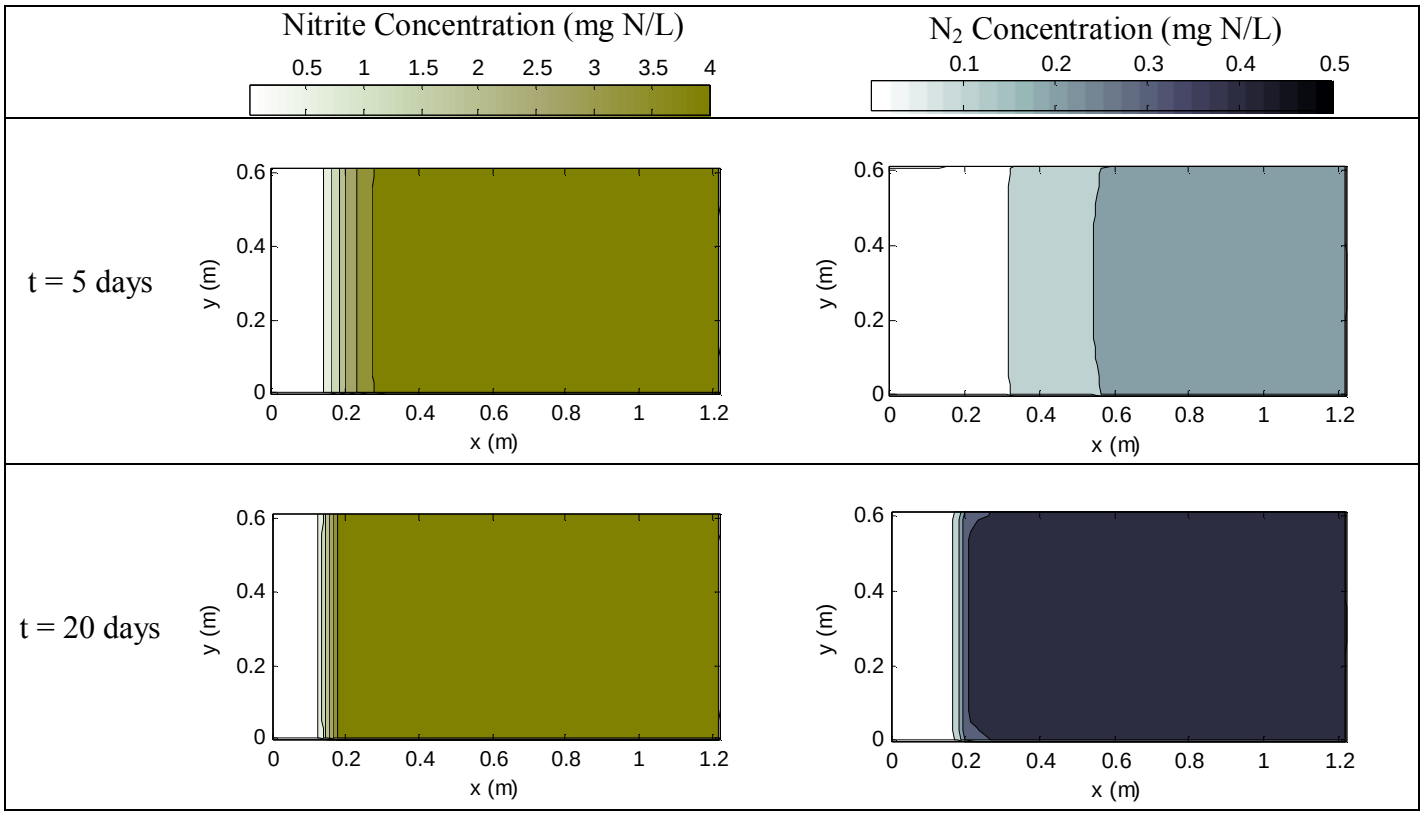

Figure 14. Nitrite and N_2 concentration for 1D problem; with hydrogen source

The reduction of nitrate results in the production of nitrite and gaseous nitrogen. Figure 14 shows the resulting concentrations of nitrite and gaseous nitrogen within the problem domain at 5 days and 50 days from the beginning of the simulation. The concentration of gaseous nitrogen is greatest at the end of the simulation. At this point, both nitrate and nitrite are being used as electron acceptors and are reduced to gaseous nitrogen.

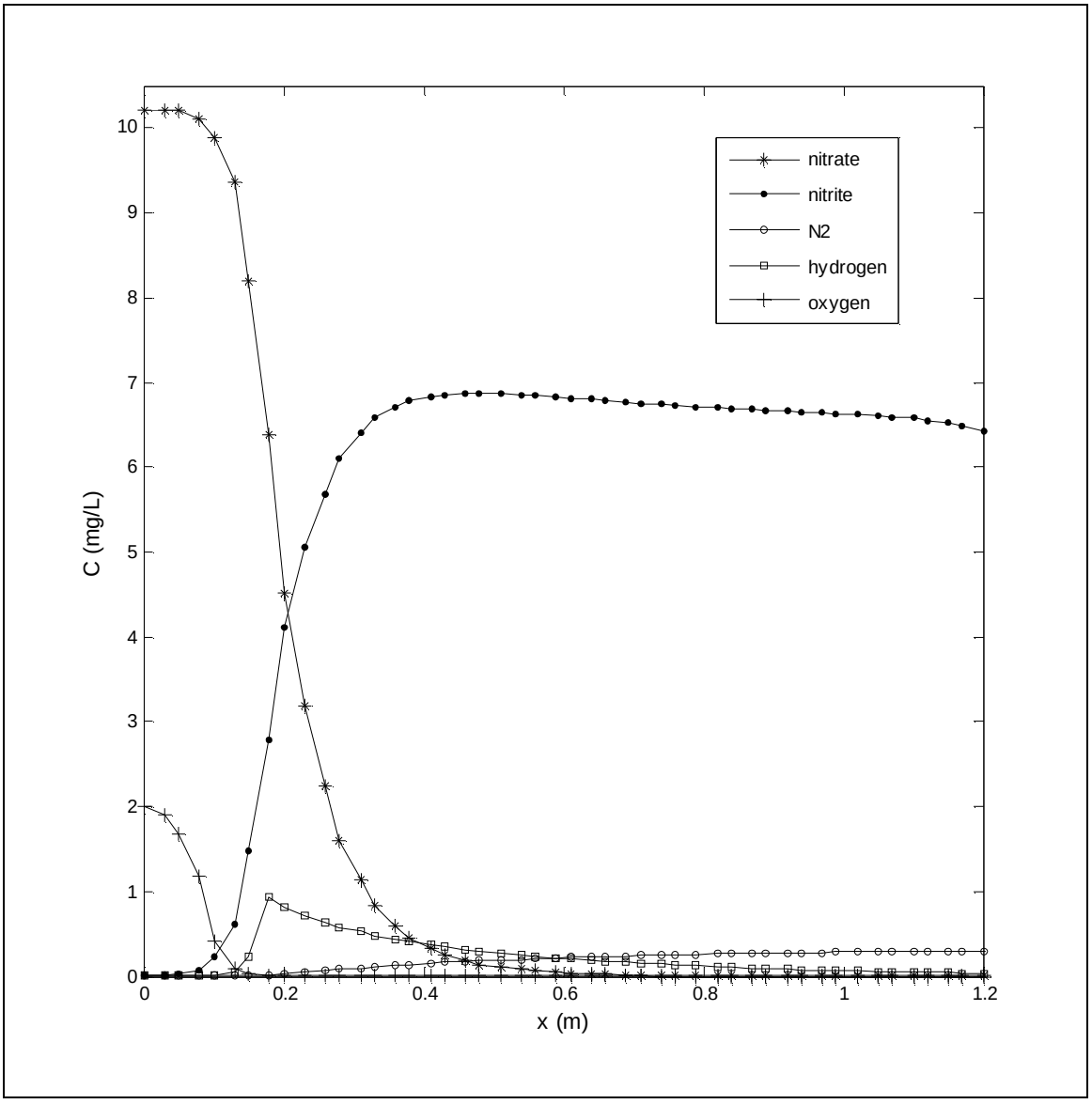

Figure 15. Solute centerline concentration vs. distance at t = 5 days

Figure 15 shows the centerline concentrations of all of the solutes with distance from the left boundary at 5 days from the beginning of the simulation. This plot shows how the

denitrification process works as a transport mechanism throughout the domain. Ultimately, it shows the reduction of the nitrate contamination, the production and reduction of nitrite contamination, and the production of gaseous nitrogen. The hydrogen concentration peaks at the location of the membranes that introduce the hydrogen in to the system.

5.1.2 Two-dimensional results

The two-dimensional problem was first simulated without any hydrogen source in order to observe the resulting nitrate contamination plume in the problem domain. Figure 16 shows the concentration profile of nitrate at 5 days and 50 days.

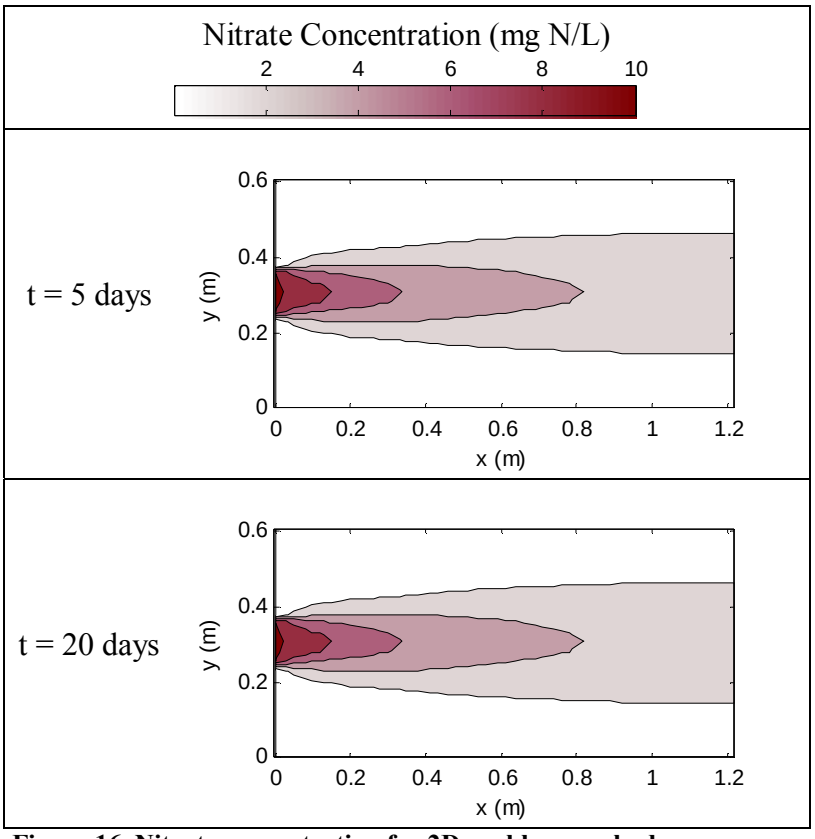

Figure 16. Nitrate concentration for 2D problem; no hydrogen source

Now one line of hydrogen sources is added to intercept the nitrate plume. The resulting nitrate concentration profiles are shown in Figure 17. Nitrite and gaseous nitrogen are produced as shown in Figure 18. These figures show that the hydrogen source has been effective in cutting back the nitrate contamination, but a large nitrite plume still covers the problem area.

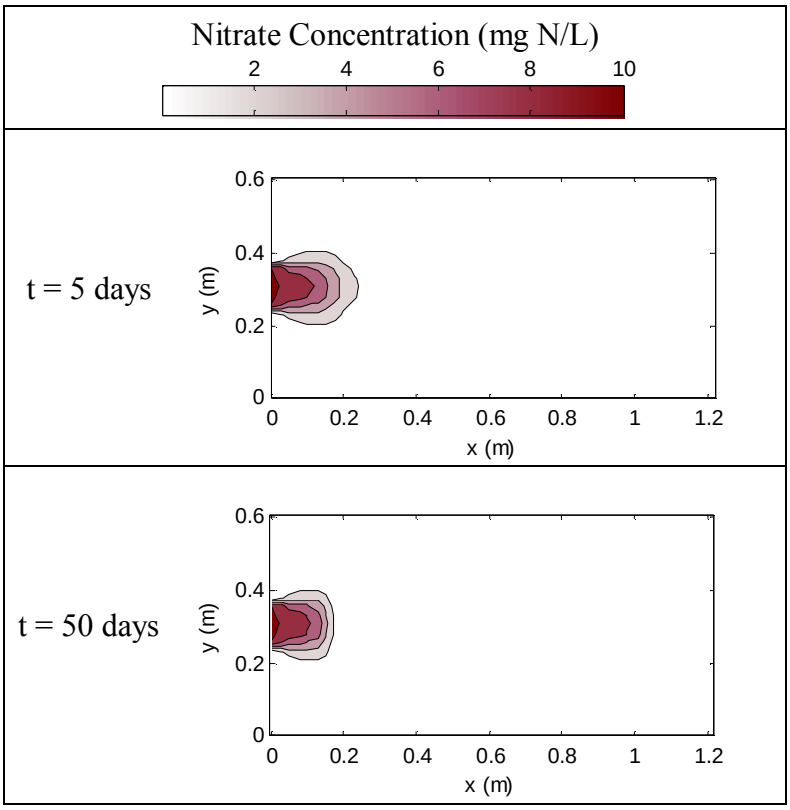

Figure 17. Nitrate concentration for 2D problem; with one line of hydrogen sources

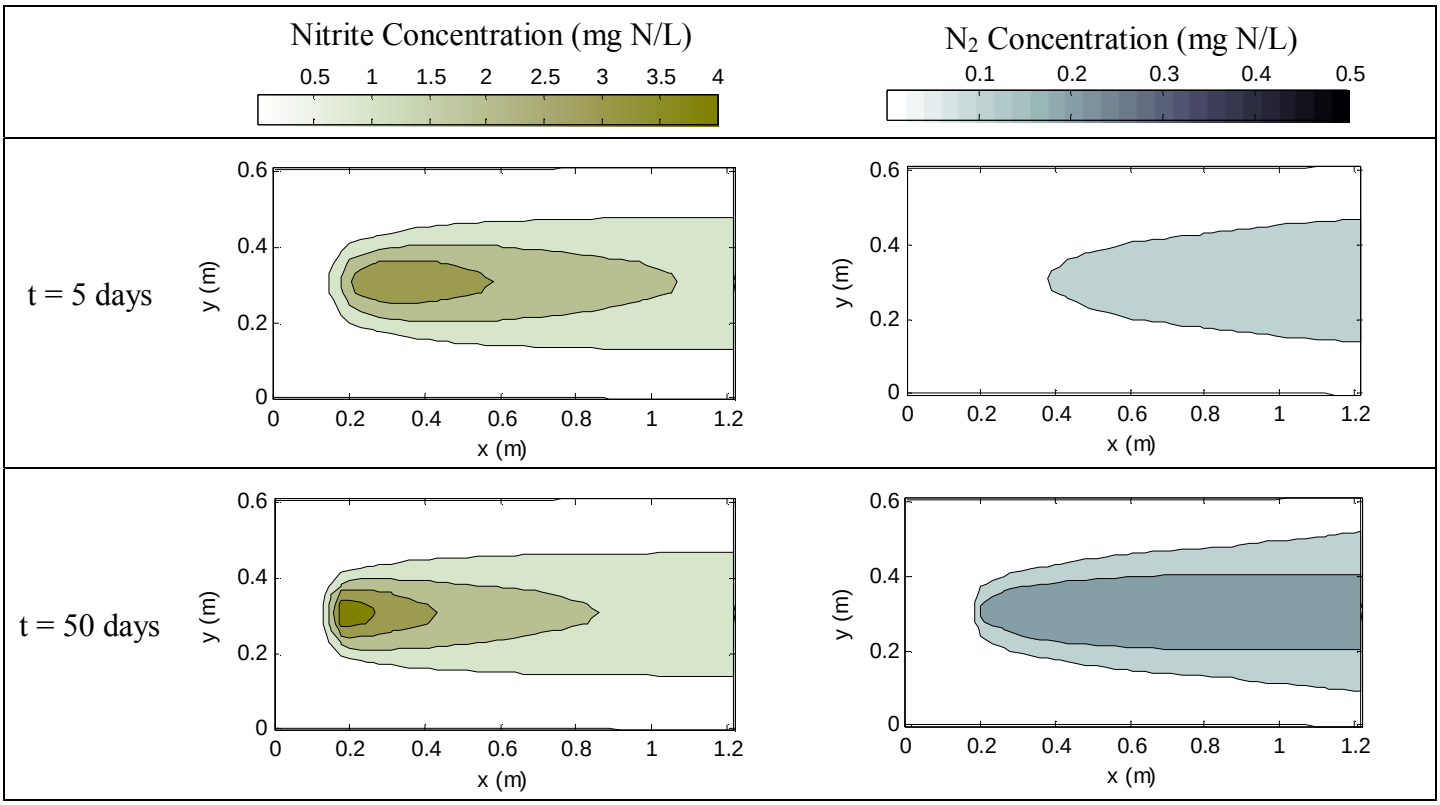

Figure 18. Nitrite and N_2 concentration for 2D problem; with one line of hydrogen sources

A second and third line of hydrogen sources were added to intercept the nitrite plume in order to reduce both the nitrate and nitrite contamination. Figure 19 shows the resulting nitrate concentrations. The nitrate profiles are identical to the case where only one line of hydrogen sources was used. The first line of sources provided enough hydrogen to reduce the nitrate plume so that it never reached far enough downstream to be affected by the second set of hydrogen sources.

Figure 20 shows the resulting concentration profiles of nitrite and gaseous nitrogen. The second and third sets of hydrogen sources resulted in a significant reduction in the nitrite plume from the initial case by the end of the simulation (t = 50 days). More gaseous nitrogen was produced in this case because the additional hydrogen sources allowed more nitrite to be reduced to gaseous nitrogen, resulting in higher gaseous nitrogen concentrations throughout the problem domain.

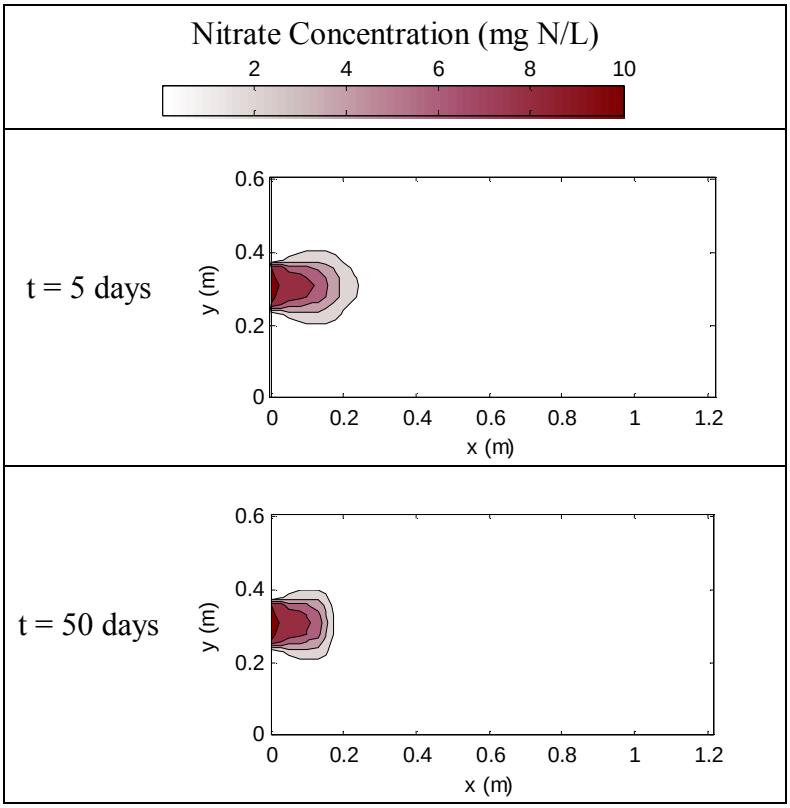

Figure 19. Nitrate concentration for 2D problem; with three lines of hydrogen sources

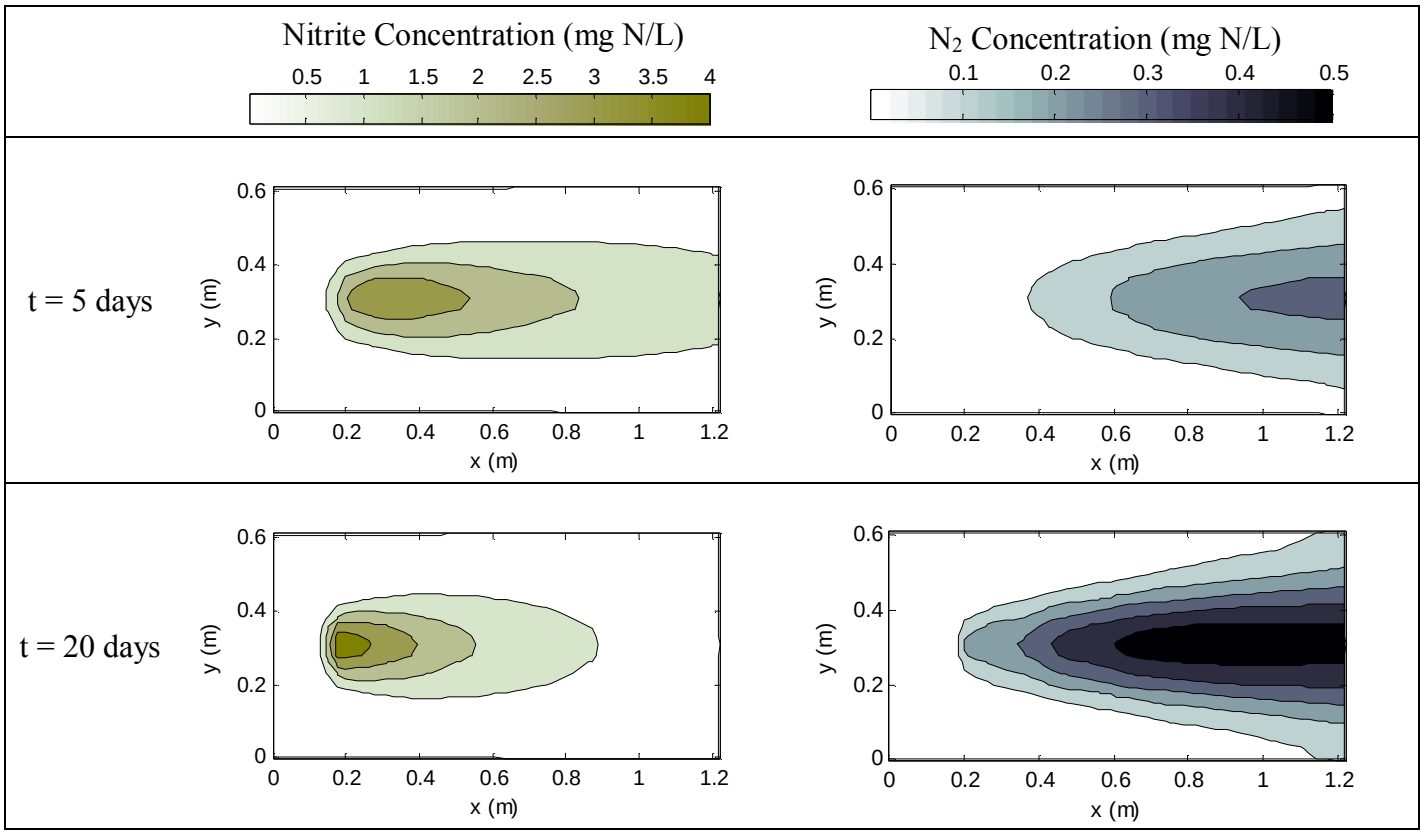

Figure 20. Nitrite and N_2 concentration for 2D problem; with three lines of hydrogen sources

5.2 Parallel performance analysis

The parallel N2D-H2 code was tested with problems of various sizes. CPU time was computed for each simulation. Because the parallelization only occurs within the time step loop, the CPU times presented in the following results include only the CPU time spent in the time step loop. This provides a good comparison of the performance of the parallel simulation versus the performance of the sequential simulation. Perfect speedup is not achieved because of the communication required between processors at the beginning and end of each iteration of the solution. Another time consuming operation was the communication at the end of each time step that had the processor of rank 0 collect the entire updated concentration array from all of the other processors, in order to write the results to the output files. This communication step was not included in the timing tests because it was only used to write output, not to perform the actual computations. All of the initial reading of data and initial computation of variables and boundary conditions is also left out of the timing tests. All processors do these initial steps individually, so each should take the same amount of time as one processor, requiring no additional time other than that required in the sequential version of the code.

The first problem tested had a relatively small computational grid (49 nodes × 25 nodes). The CPU time required to run the parallel version of N2D-H2 for this problem on a various number of processors is shown in Figure 21. This figure illustrates two important ideas. First, if the problem size is small, then there are no significant benefits of solving the problem in parallel because the time spent communicating with other processors is substantial compared to the time spent solving the problem. Second, the two data sets illustrate where a lot of the CPU time is spent. The exchange time is the time that one processor spent sending and receiving rows of data from the processor above and the processor below at the beginning of each iteration of the solution. The exchange time is almost 20% of the simulation time required to solve the problem on one processor. This illustrates the expense of message-passing.

The sample problem used to obtain these timing results was the one-dimensional problem with the hydrogen source. All of the sample problems described in the previous section had similar performance results. The dominant factor in determining the performance of the program was the size of the problem domain.

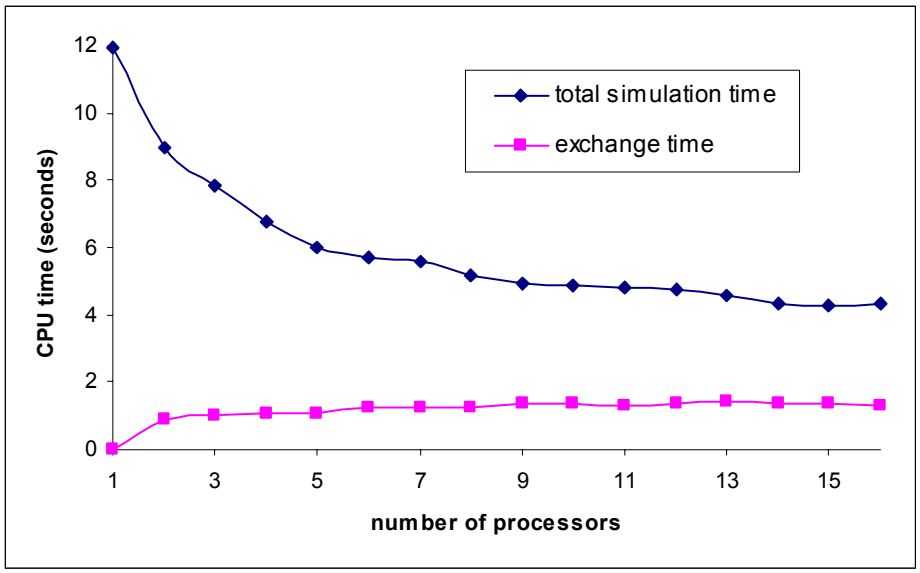

Figure 21. CPU time for a problem size of 49x25 nodes

Various additional problem sizes were tested. Increasing the problem size involved increasing the number of nodes in the finite difference solution grid. The spacing in the x-direction and the y-direction remained constant when the problem sized was increased. This allowed the same time discretization to be used for all of the problems. When a larger number of nodes were used in the y-direction, the hydrogen point source extended to include every node in the y-direction.

For problems with a greater number of nodes, parallel processing was beneficial. The time spent exchanging data still remains the most expensive operation, and this time remains relatively constant no matter how many processors are used. Therefore, if the original simulation time required to run N2D-H2 on one processor exceeds the time it takes a processor to exchange two rows of data with its neighbors, then using additional processors will offer some speedup. The total simulation time and the time spent exchanging data in a problem with a larger computational grid (490 nodes × 99 nodes) is shown in Figure 22.

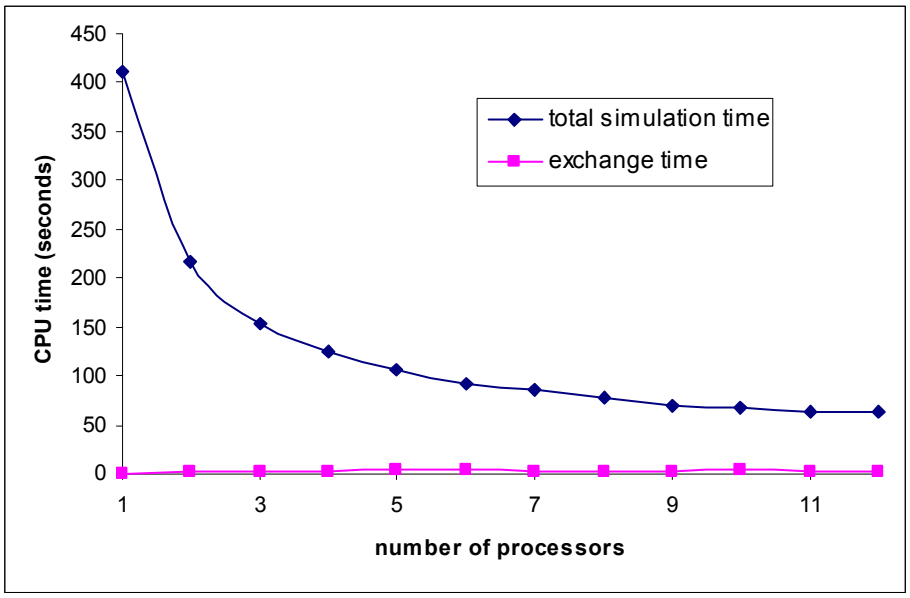

Figure 22. CPU time for a problem size of 490x99 nodes

Figure 22 reveals that parallel processing is beneficial for a problem of this size because the time required to exchange rows of data is significantly less than the time required to run the simulation on one processor. The CPU time required to run the parallel N2D-H2 code for varying problem sizes is shown in Figure 23.

Speedup is the ratio between the elapsed time for the best single processor algorithm and that of a parallel domain decomposition algorithm (Gropp et al. 1999). Speedup for various problem sizes is shown in Figure 24. Parallelization offers minimal speedup when the problem size is small. When the problem size is larger (490 nodes in the x-direction × 99 nodes in the y-direction), speedup is achieved with multiple processors. The large problem simulated in parallel achieves a speedup of about 90% of the ideal speedup on two processors and about 60% of the ideal speedup on eleven processors (ideal speedup for a simulation is equal to the number of processors used to perform the simulation) . There is significant improvement up to about ten processors, at which point using additional processors becomes inefficient. Also note that greater speedup occurs for larger problem sizes. When the finite-difference solution grid contains more nodes, each processor will perform a greater number of computations in the iteration loop, while the same amount of communication is required. This demonstrates that the benefits of parallel processing are more evident for a finer finite-difference grid or for a larger problem domain.

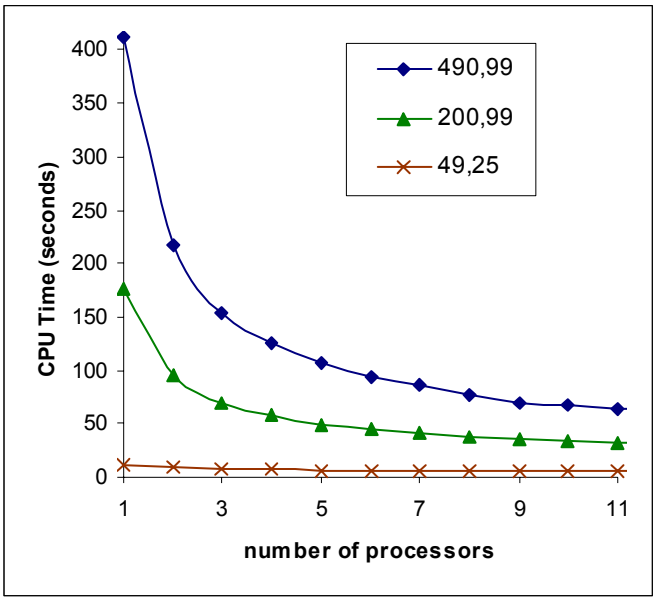

Figure 23. CPU time for various problem sizes

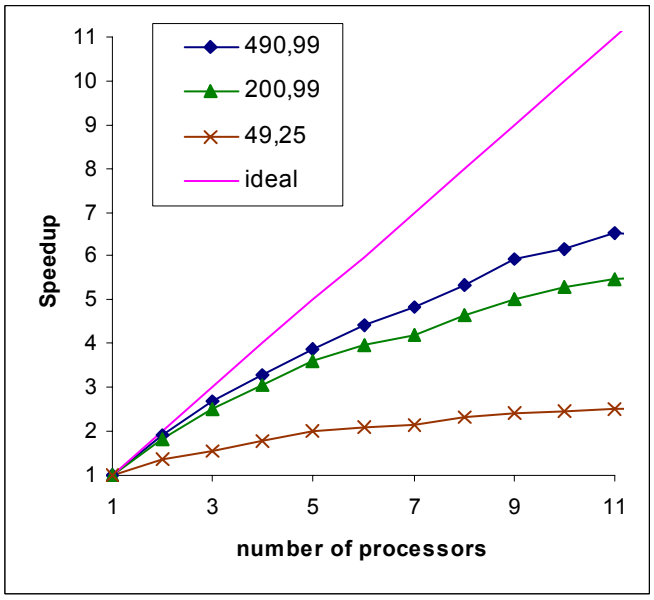

Figure 24. Speedup for various problem sizes

The purpose of developing several test problems for the N2D-H2 code was to determine what effects different aspects of the problem would have on the performance of the parallel code. The factors of the problem that have the most influence on the conceptual model are the nature of the nitrate contamination and the locations and quantities of the hydrogen point sources. Therefore, both one-dimensional and plume problems were developed, and the hydrogen point sources were varied. Performance tests on the code should reveal whether these aspects of the problem have an impact on the parallel simulation and whether or not different domain decomposition methods should be used depending on the nature of the solute sources.

Figure 25 shows the speedup for the various test problems. It shows the speedup observed for the one-dimensional problem with the hydrogen source, the plume problem without any hydrogen sources, and the plume problem with three hydrogen sources. For each of these problems, the speedup is shown for the case when a 49x25 node grid is used and the case when a 200x99 node grid is used. This figure reveals that the overriding factor that determines the performance of the parallel version of the N2D-H2 code is the problem size. As long as the problem size was kept constant, the nature of the problem did not have a significant impact on the speedup observed by running the program in parallel. The speedup observed for the plume

problem was the same whether or not hydrogen sources were simulated. This suggests that having the processors that do the computations for the section of the domain containing the hydrogen sources hold smaller sections of the domain, or other such possible adjustments to the domain decomposition method, would not play a significant role in speeding up the parallel program.

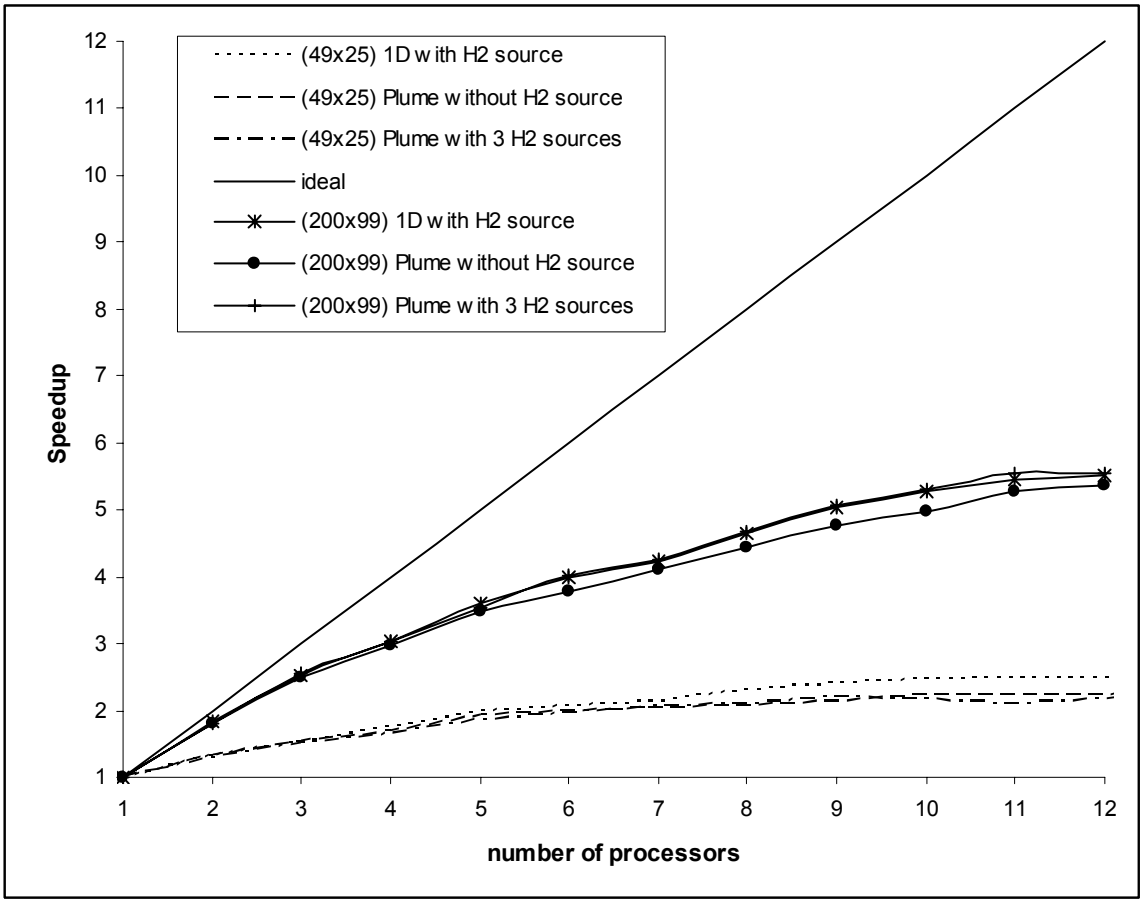

Figure 25. Speedup for various test problems

In addition to the parallelism that results from solving an equation at a large number of nodes, reactive transport modeling lends itself to additional parallelism resulting from the multiple species aspect of the simulations. Therefore, instead of decomposing the computational domain according to sections of the finite difference grid, the domain could be decomposed by solute. This domain decomposition method was explored in order to determine if it was a reasonable method to approach parallel processing in reactive transport and to see if it would improve the performance of the parallel code. The N2D-H2 code considers five solutes.

Therefore, five processors were used to simulate the reactive transport problem in parallel. Figure 26 shows the speedup achieved by the parallel code using the solute decomposition method compared to the speedup achieved by the standard decomposition method described in the previous sections of this report. The results from several of the test problems with two different problems sizes are shown here.

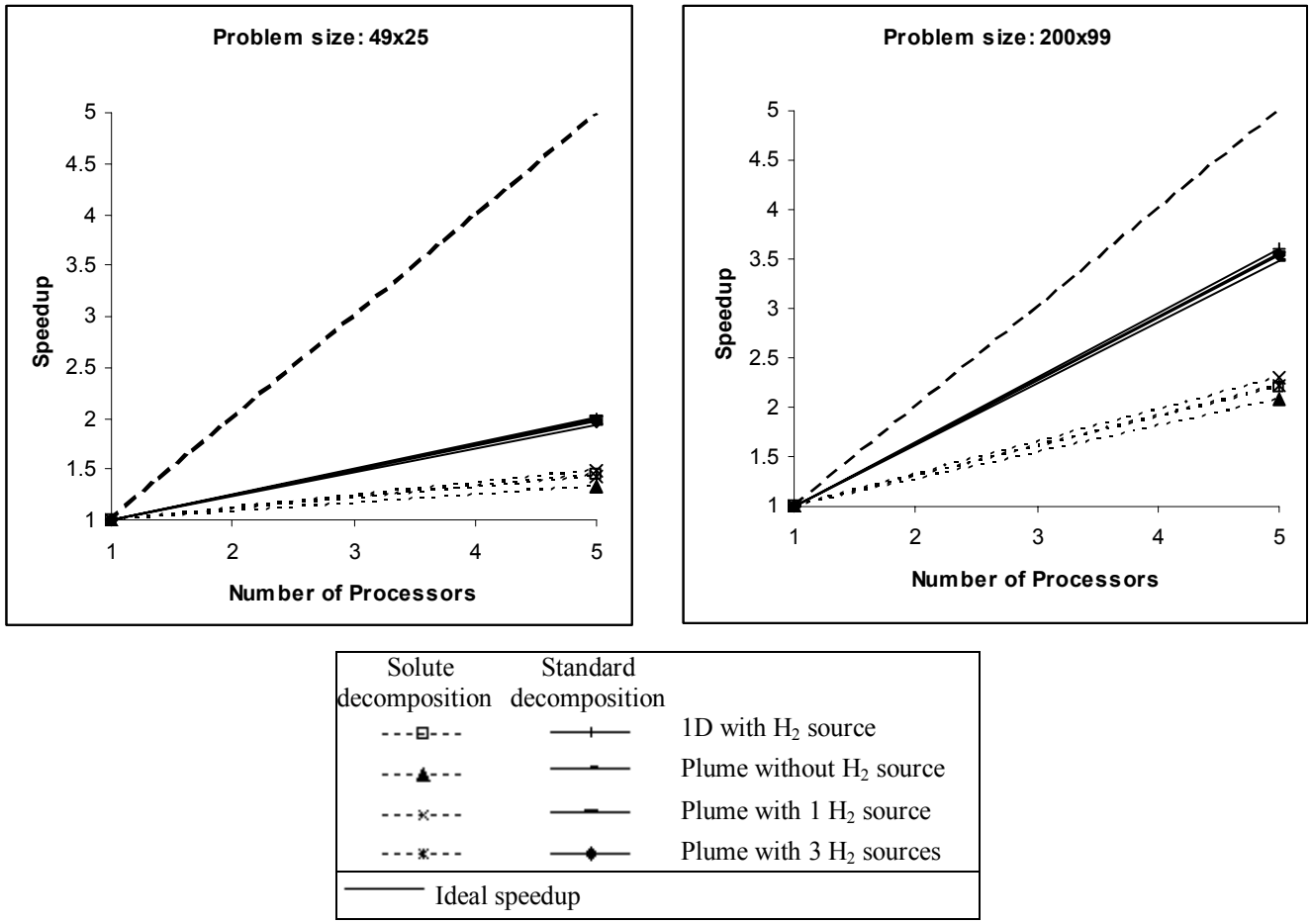

Figure 26. Speedup for standard decomposition vs. decomposition by solute

This figure demonstrates that the standard decomposition method performs significantly better than the solute decomposition method. There are several problems with the idea of decomposing the computational domain by solute. First, in this problem there are five solutes, so that decomposition method limits the number of processors to five. While this problem is small in comparison to some reactive transport simulations, even if ten to twenty solutes are simulated, this is still a limited division of computational volume compared to the possible division that is

possible when the grid is divided among processors. The bulk of computation is a result of the number of nodes in the grid, and not as much a result of the number of solutes being simulated. Second, by nature of reactive transport, the concentrations of the multiple solutes depend on each other. Therefore, at the beginning of each time step, each processor must broadcast its full set of concentration data to all of the other processors. This adds a very high volume of communication to the simulation, which degrades the performance of the parallel code. And finally, it is possible that certain solutes could take fewer iterations to converge to a solution during each time step than other solutes. When that solute converges, the processor becomes idle for the rest of the time step until the other solutes converge. This leads to inefficiency and an imbalance in the computational load among processors.

Further investigation could be done into determining more efficient methods of using a domain decomposition by solute in reactive transport simulations. Some of the more advanced MPI tools could be employed, such as overlapping communication and computation, in order to improve the performance of the code. The two decomposition method discussed could also be combined in order to eliminate the problem of limiting the number of processors to the number of solutes in the simulation. However, during the initial investigation into parallel processing, it seems that the standard decomposition method will always provide the greatest increase in performance.

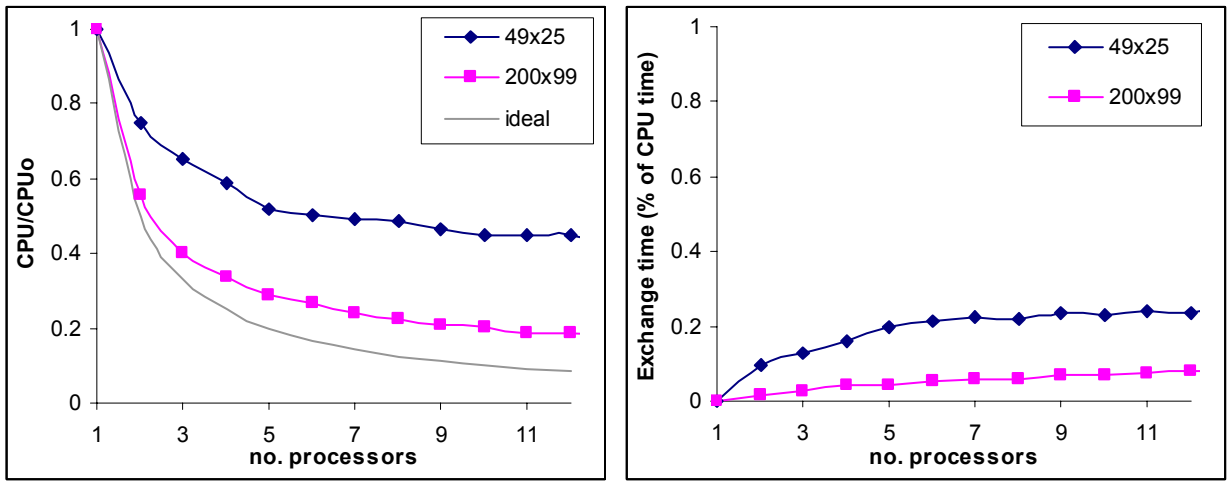

Figure 27. CPU time and exchange time for increasing problem size

6 Conclusions

This project was meant to be a study of the feasibility of writing parallel code for the purpose of reactive transport modeling. The parallel N2D code will not necessarily be used in practice, other than for academic reasons, but the purpose of generating the code and conducting the performance analysis was to derive lessons that could be considered when parallel processing is potentially applied to larger, more complex reactive transport models. The conclusions of this study are in the form of the lessons described here.

First, the parallel N2D code showed that the problem size, or the number finite difference nodes, has more influence on the parallel performance of the code than any other factor considered in this study. Larger problems make better, more efficient use of parallel processing resources. Therefore, before parallel code is developed, it is important that the size of the problem is considered, and it must be determined that the application does in fact warrant the use of parallel processors. If the problem size is not sufficiently large, the high-performance computational resources will be used inefficiently, and the application will not necessarily benefit from the use of these resources.

Finally, the N2D code also showed that reactive transport modeling is potentially a good application for the use of parallel computing resources. Simulating multiple contaminants in effect just increases the problem size. For example, simulating ten contaminants has the same effect on the amount of computation required as does increasing the size of the finite difference grid by a factor of ten. Because it was demonstrated that an increasing problem size shows better speedup, complex reactive transport models simulating multiple solutes should perform well on parallel processors. However, the N2D code also demonstrated that decomposing the computation by solute is not necessarily the most efficient approach. The grid-based decomposition remained the most efficient method even though reactive transport modeling has other sources of parallelism.

7 Future Work

The study of parallel processing as it applies to the N2D-H2 code revealed that the time to simulate problems with a high volume of computation resulting from a large grid and several solutes would be greatly reduced by decomposing the computational domain among several processors. In reactive transport simulations, the CPU time that is required is compounded by the number of solutes being modeled. Therefore, the results that have been observed here could be extended to other, more complex reactive transport models.

SEAM3D (Sequential Electron Acceptor Model, Three-Dimensional) models multi-component reactive transport in groundwater. It is based on the code MT3DMS (A Modular Three-Dimensional Multispecies Transport Model for Simulation of Advection, Dispersion, and Chemical Reactions of Contaminants in Groundwater Systems), and consists of a collection of add-on packages that model biodegradation, NAPL dissolution, cometabilism, reductive dechlorination, and phytoremediation. The complexities in this model require a high volume of memory and computational power, therefore its performance would theoretically greatly improve through the use of parallel processing.

If the parallel processing techniques studied in this report are extended to more complex codes such as SEAM3D, then several additional concepts should be explored. The next steps in the development of the parallel code for larger, more complex problems would include implementing a two-dimensional or three-dimensional domain decomposition, implementing parallel input and output procedures, and programming the code to provide overlapping communication and computation. Some of these more advanced features of parallel computing will become more important as more complex reactive transport codes are adjusted to make use of parallel processors.

For the N2D-H2 code, the level of complexity in the model did not warrant the use of a two-dimensional decomposition. However, for a significantly larger, three-dimensional model of about ten solutes, for example, employing a two-dimensional decomposition would allow the computation to be separated among an even greater number of processors, making more efficient use of the parallel computing resources.

In a code such as SEAM3D, a large part of the CPU time required for some simulations is the time required to read input files and write the results to output files. The input and output

procedures in N2D-H2 did not have a significant impact on the overall CPU time, so these procedures were performed sequentially, by all processors. However, MPI provides a set of tools to allow I/O operations to be performed in parallel. Implementing these operations could provide significant additional speedup.

Finally, some of the more advanced message-passing procedures could be explored in order to develop the most efficient code possible. The ultimate purpose of employing parallel processors is to reduce the time required to perform simulations, providing the capacity to expand existing models and implement more accurate modeling concepts. Therefore, when implementing a parallel simulation, the high-performance computing resources should be used to the full extent.

8 References

Anderson, M.P., Woessner, W.W. (2002). *Applied groundwater modeling: Simulation of Flow and Advective Transport*, Academic Press, San Diego.

Aoyama, Y., Nakano, J. (1999). "RS/6000 SP: Practical MPI Programming." *International Technical Support Organization,* IBM Corporation, www.redbooks.ibm.com.

Ashby, S., Falgout, R., Smith, S., Baldwin, C., Bosl, W., Thompson, A. (1996). "The ParFlow Project: Modeling Subsurface Flow and Chemical Migration on High Performance Computers," http://www.llnl.gov/CASC/ParFlow/.

Dietz, H. (1998). "Linux Parallel Processing HOWTO."
Website: http://yara.ecn.purdue.edu/~pplinux/PPHOWTO/pphowto.html#toc6

Dou, H.S., Phan-Thien, N. (1998). "On the scalability of parallel computations on a network of workstations." *Computational Mechanics*, 22, 344-354.

Elmroth, E. (2000). "On grid partitioning for a high performance groundwater simulation software." *Lecture notes in Computational Science and Engineering*, 13, 221-33.

Fedkiw, J. (1991). "Nitrate occurrence in U.S. waters (and related questions)." *USDA Working Group on Water Quality.*

Gropp, W., Lusk, E., Skjellum, A. (1999). *Using MPI: Portable Parallel Programming with the Message-Passing Interface, second edition*, The MIT Press, Cambridge, Massachusetts.

Gwo, J.P., D'Azevedo, E.F., Frenzel, H., Mayes, M., Yeh, G.T. (2001). "HBGC123D: a high-performance computer model of coupled hydrogeological and biogeochemical processes." *Computers & Geosciences*, 27, 1231-1242.

Hammond, G., Valocchi, A., Lichtner, P. (2002). "Modeling multicomponent reactive transport on parallel computers using Jacobian-Free Newton Krylov with operator-split preconditioning." *Developments in Water Resources*, 47, 727-734.

Killingstad, M.W., Widdowson, M.A., Smith, R.L. (2002). "Modeling Enhanced In Situ Denitrification in Groundwater." *Journal of Environmental Engineering*, 128(6), 491-504.

Lingen, F.J. (2000). "A versatile load balancing framework for parallel applications based on domain decomposition." *International Journal of Numerical Methods in Engineering*, 49, 1431-1454.

Madabhushi, S.P., Belloti, D., Clematis, A., Fernandes, P. (1998). "Parallelization of a finite difference code for modeling groundwater flow." *Proc. XII International Conference on Computational Methods in Water Resources,* Crete, Greece, 1, 371-378.

Nolan, B.T., Ruddy, B.C., Hitt, K.J., Helsel, D.R. (1997). "Risk of nitrate in groundwaters in the United States – a national perspective." *Environmental Science and Technology*, 31(8), 2229-2236.

Pacheco, P.S., Ming, W.C. (1997). "Introduction to Message Passing Programming: MPI Users' Guide in FORTRAN." (pdf)

Schwartz, F.W., Zhang, H. (2003). *Fundamentals of Groundwater*, John Wiley & Sons, New York.

Smith, R.L., Miller, D.N., Brooks, M.H., Widdowson, M.A., Killingstad, M.W. (2001). "In situ stimulation of groundwater denitrification with formate to remediate nitrate contamination." *Environmental Science and Technology*, 35, 196-203.

Stefanovic, D.L., Stefan, H.G. (2001). "Accurate two-dimensional simulation of advective-diffusive-reactive transport." *Journal of Hydraulic Engineering,* 127, 728-737.

Tsai, W.F., Shen, C.Y., Fu, H.H, Kou, C.C. (1999). "Study of Parallel Computation for Ground-Water Solute Transport." *Journal of Hydrologic Engineering*, 4, 49-56.

Widdowson, M.A. (1996). N2D-H2: Fortran program for the simulation of two-dimensional solute transport in porous media.

Wiedemeier, T.H., Rifai, H.S., Newell, C.J., Wilson, J.T. (1999). *Natural Attenuation of Fuels and Chlorinated Solvents in the Subsurface*, John Wiley & Sons, New York.

Wu, Y., Zhang, K. (2002). "An efficient parallel-computing method for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media." *Advances in Water Resources*, 25, 243-261.

Yang, D. (1997). "A parallel grid modification and domain decomposition algorithm for local phenomena capturing and load balancing." *Journal of Scientific Computing*, 12(1), 99-117.

Zhang, K., Wu, Y., Bodvarsson, G.S. (2003). "Parallel computing simulation of fluid flow in the unsaturated zone of Yucca Mountain, Nevada." *Journal of Contaminant Hydrology*, 62-33, 381-399.

Zheng, C., Bennett, G.D. (1995). *Applied Contaminant Transport Modeling: Theory and Practice,* John Wiley & Sons, New York.

Zheng, C., Wang, P.P. (1999). "MT3DMS: A modular three-dimensional multispecies transport model for simulation of advection, dispersion, and chemical reactions of contaminants in groundwater systems." *Documentation and User's Guide*, U.S. Army Corps of Engineers, Washington, D.C.

9 Appendix

9.1 Running a parallel N2D simulation

Compile the source code

Save the source code given in Appendix 9.2 to a text file named **N2D_decomp1D.f**. In the directory where you have the FORTRAN file **N2D_decomp1D.f** saved, compile the code using the command:

```
mpif77 –o N2D N2D_decomp1D.f
```

This creates an executable file called **N2D.exe**.

Run a simulation

a. Make sure you are in the directory where you have created the executable **N2D.exe**.

b. The input file corresponding to the test problem you wish to run must be in the same directory as the executable (see Appendix 9.3 for the five input files corresponding to the test problems discussed in this report). Detailed information about the input data can be found in Chapter 3.7. The input file must have the name **INPUT.DAT**.

c. Enter the command:

```
mpirun –np 2 N2D
```

This command will run the simulation corresponding to the input file **INPUT.DAT** on 2 processors. The option **-np** means that you are defining the number of processors at run time. In this case you are running on 2 processors. If you want to run on a different number of processors, change the 2 to the desired number of processors (it is possible to run the parallel code on 1 processor).

d. Output to the screen gives you the CPU time and the exchange time for the simulation.

9.2 Parallel N2D code

```
!     Last change:  JW   4 May 2006    11:13 am
!
!
!       PROGRAM:  N2D-H2
!       VERSION:  1 (Parallel Version)
!
!       TWO-DIMENSIONAL TRANSPORT OF HYDROGEN, DO, NO3, NO2 AND
!       DENITRIFICATION END PRODUCTS.
!       PROGRAM TO SOLVE TWO-DIMENSIONAL ADVECTION-DISPERSION EQ.
!
!
      include 'mpif.h'
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
!      integer mpi_comm_world
      integer numprocs,comm1d,ierr
      integer myid,first,last,nbrtop,nbrbottom
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
      COMMON /TAL1/ A(999,100,10), B(999,100,10), C(999,100,10)
      COMMON /TAL2/ D(999,100,10), E(999,100,10), F(999,100,10)
      COMMON /TAL3/ GB(999,100,10), GS(999,100,10)
      COMMON /CON1/ RL(999,100,10), NIT, TOL
      COMMON /CON2/ RO(10),RN1(10),RN2(10),KHO(10),KHN1(10),KHN2(10)
      COMMON /CON3/ GAMO, KO, KC(10,10), GAMN1, KN1, GAMN2, KN2
      COMMON /CON4/ YD(9), DK(9), ZETA1, ZETA2, RF(10)
      COMMON /CON5/ SNAME(10), BNAME(10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      COMMON /CON7/ UMIN(10), UBMIN(10)
      COMMON /CON8/ XNODE(999,100), YNODE(999,100)
      COMMON RT(999,100,10)
      COMMON DXX(999,100,10), DYY(999,100,10)
      COMMON UOBS(15,2000,10), IOBS(15), JOBS(15)
      COMMON /MASS1/ OSTOR(10), OFLUXIN(10), OFLUXOUT(10), OBIO(10)
      COMMON /MASS2/ STOR(10), FLUXIN(10), FLUXOUT(10), BIO(10)
      COMMON /MASS3/ CM1(99,10), CM2(99,10), CM3(99,10), IMB(10), DM
      COMMON /MASS4/ FP(99), IP(99), JP(99), OFLXH(10)
      COMMON /MASS5/ AMEM(99), DMEM(99), PMEM(99,9), HMTC(999), FM, FMB
      common /mpi/ numprocs,comm1d,myid,ierr,nbrtop,nbrbottom,first,last
      COMMON IC(10), BCT(10), BIC(10), DF(10)
      COMMON IBCT(10), IIC(10), IBIC(10)
      COMMON TIMOUT(15), BT(10)
      DOUBLE PRECISION KHO,KHN1,KHN2,KO,KN1,KN2,KC,IC
      CHARACTER TITL1*80, TITL2*80, BNAME*40, SNAME*40
      common /dat1/ TITL1,TITL2,TIMSIM,AX,AY,ATOT,VTHK,AL,VELY,AT,BDEN
      common /dat2/ DELDT,NPS,HLC,HMW,WKV,PORW,ACF,BCF,CCF,NFP
      common /dat3/ REY,SC,DTAOPT,ISOL,IRS,NTSTP,DT,TM,NOBS,NTIM
      integer debugFile
      integer source(1:100)
      double precision exchng_time
!
      call mpi_init( ierr )
      call mpi_comm_rank( mpi_comm_world, myid, ierr )
      call mpi_comm_size( mpi_comm_world, numprocs, ierr )
```

```
!
!All processes open main input file
      OPEN(15, FILE='INPUT.DAT', status = 'old')
!
!Process 0 only opens the output files
      if(myid.eq.0)then
      !Open main output file
         OPEN(16, FILE='P_OUTPUT.DAT', status = 'unknown')
      !Open main output file
         OPEN(17, FILE='P_CvsX.DAT', status = 'unknown')
      !Open mass balance output file
         OPEN(18, FILE='P_BALANCE.DAT', status = 'unknown')
      !Open observation node output file
         OPEN(19, FILE='P_OBS.DAT', status = 'unknown')
      !Open concentration output file
         OPEN(30, FILE='P_CONC.DAT', status = 'unknown')
      !Open concentration output file
         OPEN(31, FILE='P_CvsXY.DAT', status = 'unknown')
      endif
!
!******Debugging files, one for each proecss to print to*************
!      if(myid.eq.0)then
!         debugFile=60
!         OPEN(debugFile, FILE='proc0out.dat',status='unknown')
!      elseif(myid.eq.1)then
!         debugFile=61
!         OPEN(debugFile, FILE='proc1out.dat',status='unknown')
!      elseif(myid.eq.2)then
!         debugFile=62
!         OPEN(debugFile, FILE='proc2out.dat',status='unknown')
!      else
!         debugFile=63
!         OPEN(debugFile, FILE='proc3out.dat',status='unknown')
!      endif
!
! Call subroutine to read in all data
      call read_data
!
! Call subroutine to write data to main output file
      if(myid.eq.0)call write_data
!
      call mpi_barrier(mpi_comm_world,ierr)
      close(15)
!
1111  FORMAT(/,10X,'RESTART COMPLETED',/)
1112  FORMAT(10X,'BOUNDARY CONDITIONS COMPLETED',/)
1113  FORMAT(10X,'MASS BAL COEFFS AND FLUX SET',/)
1114  FORMAT(/,10X,'INITIAL CONDITIONS COMPLETED',/)
1115  FORMAT(10X,'TOTAL MASS SET',/)
1116  FORMAT(10X,'TRIDIA COEFF SET',/)
1117  FORMAT(10X,'ADVECTION COEFF SET',/)
1118  FORMAT(10X,'ADVECTION SUBROUTINE COMPLETE',/)
1119  FORMAT(10X,'TRIDIA SUBROUTINE COMPLETE:  M=',I3,/)
!
!*******INITIALIZE CONCENTRATION ARRAY*******************************
!
!      IF(IRS.EQ.1)GO TO 17
```

```
!
      DO 11 L=1,NTS
        IF(IIC(L).GT.0)GO TO 11
!
        DO J=1,NYN
          DO I=2,NXN
            W(I,J,L) = IC(L)
            V(I,J,L) = IC(L)
            U(I,J,L) = IC(L)
          END DO
        END DO
11    CONTINUE
!
      DO 12 L=1,NBIO
        IF(IBIC(L).GT.0)GO TO 12
        DO J=1,NYN
          DO I=1,NXN
            WB(I,J,L) = BIC(L)
            UB(I,J,L) = BIC(L)
          END DO
        END DO
12    CONTINUE
!
      if(myid.eq.0)WRITE(16,1114)
!      WRITE(*,1114)
!
!*******INITIALIZE BOUNDARY CONDITIONS*********************************
!
      DO 13 L=1,NTS
        IF(IBCT(L).GT.0)GO TO 13
!
        DO J=1,NYN
          W(1,J,L)=BCT(L)
          V(1,J,L)=BCT(L)
          U(1,J,L)=BCT(L)
        END DO
!
13    CONTINUE
!
      if(myid.eq.0)WRITE(16,1112)
!      WRITE(*,1112)
!
      TIME = 0.D0
      if(myid.eq.0) CALL TDOUT
!
      GO TO 19
!
17    CONTINUE
!
!
!*******CALCULATE MASS FLUX AT BOUNDARIES******************************
!
19    CM = POR*DY*VTHK*DT
      DO 21 L=1,NTS
        OFLUXIN(L) = 0.D0
        OFLUXOUT(L) = 0.D0
        OBIO(L) = 0.D0
```

```
        DO 23 J=1,NYN
          YN=1.D0
          IF(J.EQ.1)YN=0.5D0
        IF(J.EQ.NYN)YN=0.5D0
        CM1(J,L) = VELX*CM/RF(L)
          CM2(J,L) = CM*DXX(1,J,L)/(DX*RF(L))
          CM3(J,L) = CM*DYY(NXN,J,L)/(DX*RF(L))
          OFLUXIN(L) = OFLUXIN(L) + YN*CM1(J,L)*W(1,J,L)
          OFLUXIN(L) = OFLUXIN(L) + YN*(W(2,J,L)-W(1,J,L))*CM2(J,L)
          OFLUXOUT(L) = OFLUXOUT(L) + YN*CM1(J,L)*W(NXN,J,L)
          OFLUXOUT(L)=OFLUXOUT(L)+YN*(W(NXN,J,L)-W(NXN-1,J,L))*CM2(J,L)
23      CONTINUE
21    CONTINUE
!
!      WRITE(*,1113)
      if(myid.eq.0)WRITE(16,1113)
!
!*******CALCULATION OF INITIAL MASS IN SYSTEM************************
!
      DM = DX*DY*VTHK*POR
      DO 27 L=1,NTS
        OSTOR(L) = 0.D0
        DO 28 I=1,NXN
          XN = 1.D0
          IF(I.EQ.1)XN=0.5D0
          IF(I.EQ.NXN)XN=0.5D0
          DO 29 J=1,NYN
              YN = 1.0D0
              IF(J.EQ.NYN)YN=0.5D0
              IF(J.EQ.1)YN=0.5D0
              OSTOR(L) = OSTOR(L) + XN*YN*W(I,J,L)*DM
29          CONTINUE
28        CONTINUE
27    CONTINUE
!
!      WRITE(*,1115)
      if(myid.eq.0)WRITE(16,1115)
!
!*******INITIALIZE COEFFICIENTS**************************************
!
      FM = HMW/HLC
      FMB = DT/(DX*DY*VTHK*POR)
      TH = 1.D0
      DO 41 J=1,NYN
        DO 42 I=1,NXN
          DO 43 L=1,NTS
            RL(I,J,L) = (DXX(I,J,L)/RF(L))*DT/DX**2
            RT(I,J,L) = (DYY(I,J,L)/RF(L))*DT/DY**2
!
            A(I,J,L) = -1.D0*RL(I,J,L)*TH
            B(I,J,L) = 1.D0+(2.D0*RL(I,J,L)+2.D0*RT(I,J,L))*TH
            C(I,J,L) = -1.D0*RL(I,J,L)*TH
            D(I,J,L) = RT(I,J,L)*TH
            E(I,J,L) = 0.D0
!
43        CONTINUE
42      CONTINUE
```

```
41    CONTINUE
!
      DO K=1,NPS
        I = IP(K)
        J = JP(K)
        BHF = HMTC(K)*AMEM(K)*FMB*TH/RF(1)
        B(I,J,1) = B(I,J,1) + BHF
        E(I,J,1) = PMEM(K,1)*FM*BHF
      END DO
      TIMFP = FP(1)
      NNFP = 1
!
!      WRITE(*,1116)
      if(myid.eq.0)WRITE(16,1116)
!
!******* BEGIN TIME STEPS ************************************************
!************************************************************************
      call mpi_cart_create(mpi_comm_world,1,numprocs,.false.,.true.,
     $     comm1d,ierr)
      call mpi_comm_rank(comm1d,myid,ierr)
      call mpe_decomp1d(nyn,numprocs,myid,first,last,source)
      call mpi_cart_shift(comm1d,0,1,nbrbottom,nbrtop,ierr)
!
!
      KOUT = 1
      KOBS = 1
      ISOL = 1
!
!      t1_timeloop = mpi_wtime()
      call CPU_TIME(t1_timeloop)
      exchng_time=0.0
       do 50 K=1,NTSTP
!
        TIME = DTA*K + TM
!        if(myid.eq.0)WRITE(16,804)
!
        IF(ISOL.EQ.1)THEN
!          if (myid.eq.0)print *,'call advect'
          CALL ADVECT(ISOL)
!          if (myid.eq.0)print *,'call biotrans'
          CALL BIOTRANS(ISOL,source)
!          if (myid.eq.0)print *,'call tridia'
          CALL TRIDIA(ISOL,source,exchng_time)
        ENDIF
!      if(myid.eq.0.and.K.eq.NTSTP)then
!         print 333,((U(i,j,5),j=1,nyn),i=nxn,1,-1)
! 333     FORMAT(///49(13F7.2//))
!         print 333,((U(i,j,3),j=1,nyn),i=15,1,-1)
! 333     FORMAT(///15(13F7.2//))
!      endif
!
        if (myid.eq.0)then
!!          WRITE(*,111)TIME,NIT
           WRITE(16,111)TIME,NIT
           WRITE(18,111)TIME,NIT
        endif
!!
```

```
111     FORMAT(/,5X,'TIME =',F10.4,5X,'NUMBER OF ITERATIONS = ',I5,/)
!
!
!******* CALL MASS BALANCE SUBROUTINE ******************************
!
        CALL MASBAL(KOBS,NNFP,NPS)
!
!******* CALL OUTPUT SUBROUTINE ************************************
!        call mpi_barrier(comm1d,ierr)
        if(myid.eq.0)then
           IF(KOUT.EQ.NTIM)THEN
              CALL TDOUT
              KOUT = 1
!
!     RECORD CENTERLINE CONCENTRATION PROFILE
              JJ = 0.5*(NYN-1) + 1
              WRITE(17,997)TIME
 997          FORMAT(/,1X,'TIME = ',F10.3,/,4X,' X')
              DO I=1,NXN
                 WRITE(17,998)XNODE(I,JJ),(U(I,JJ,L),L=1,NTS),UB(I,JJ,1)
 998             FORMAT(7(3X,F7.3))
              END DO
           ELSE
             KOUT = KOUT + 1
           ENDIF
!        endif
!
!     RECORD OBSERVATION NODES
!this is used to calculate UOBS for print out in balance.dat
!
           DO KB=1,NOBS
              IB = IOBS(KB)
              JB = JOBS(KB)
              DO L=1,NTS+1
                 IF(L.LE.NTS)UOBS(KB,K,L) = U(IB,JB,L)
                 IF(L.GT.NTS)UOBS(KB,K,L) = UB(IB,JB,1)
              END DO
           END DO
           KOBS = KOBS + 1
        endif  !end proc 0 prints out data
!
!******* UPDATE CONCENTRATION ARRAY FOR NEXT TIME STEP **********
!
        DO J=first,last
          DO I=2,NXN
            DO L=1,NTS
              W(I,J,L) = U(I,J,L)
            END DO
          END DO
        END DO
!
!******* CHECK FOR CHANGES IN POINT SOURCE MASS FLUX **********
!
        IF(TIME.LT.TIMFP)GO TO 50
        NNFP = NNFP +1
        DO KS=1,NPS
          IS = IP(KS)
```

```
          JS = JP(KS)
          BHF = HMTC(KS)*AMEM(KS)*FMB*TH/RF(1)
          B(IS,JS,1) = 1.D0+(2.D0*RL(IS,JS,1)+2.D0*RT(IS,JS,1))*TH + BHF
          E(IS,JS,1) = PMEM(KS,NNFP)*FM*BHF
        END DO
        TIMFP = TIMFP + FP(NNFP)
!
 50   CONTINUE        !end time loop
!      t2_timeloop = mpi_wtime()
      call CPU_TIME(t2_timeloop)
!
!******* PRINT OBSERVATION NODE DATA **************************
!
      if(myid.eq.0)then
         DO KB=1,NOBS
            I = IOBS(KB)
            J = JOBS(KB)
           XP = XNODE(I,J)
            YP = YNODE(I,J)
            WRITE(19,778)KB
            WRITE(19,779)XP,YP
            DO K=1,NTSTP
               TOBS = K*DTA
               WRITE(19,777)TOBS,(UOBS(KB,K,L),L=1,NTS+1)
            END DO
         END DO
      endif     !end proc 0 prints out observation node data
777   FORMAT(7(1X,E11.4))
778   FORMAT(/,'Observation Node #',I2)
779   FORMAT(1X,'X =',F10.3,1X,'Y =',F10.3)
!
      if(myid.eq.0)then
         print *,'numprocs = ',numprocs
         print *,'time of simulation = ',t2_timeloop-t1_timeloop,' secs'
         print *,'proc',myid,' exchange time=',exchng_time
      endif
      if (myid.eq.1) print *,'proc',myid,' exchange time=',exchng_time
      call mpi_finalize( ierr )
      STOP
      END
!
!***************************END OF MAIN PROGRAM ****************************
!**************************************************************************
!
      SUBROUTINE TRIDIA(ISOL,source,exchng_time)
!     SOLVES ARRAYS, LINE-BY-LINE
!
      include 'mpif.h'
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      integer numprocs,comm1d,myid,ierr,nbrtop,nbrbottom
!      integer mpi_comm_world
      integer first,last
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
      COMMON /TAL1/ A(999,100,10), B(999,100,10), C(999,100,10)
      COMMON /TAL2/ D(999,100,10), E(999,100,10), F(999,100,10)
      COMMON /TAL3/ GB(999,100,10), GS(999,100,10)
```

```
      COMMON /CON1/ RL(999,100,10), NIT, TOL
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      common /mpi/ numprocs,comm1d,myid,ierr,nbrtop,nbrbottom,first,last
!
      DIMENSION EE(999,1000,10), DD(999,100,10)
      DIMENSION ALF(999), BETA(999), Y(999)
      integer status(mpi_status_size)
      double precision TS,myTSCK,TSCK
      integer source(1:nyn)
      integer debugFile
      double precision exchng_time
!
!       BEGIN CALCULATION OF FINAL ARRAY COEFFS
!
      NIT = 1
299   CONTINUE
!      write(debugFile,*) 'iteration ',NIT
!      write (debugFile,333)((U(i,j,2),j=1,nyn),i=8,1,-1)
! 333  FORMAT(///8(13F7.2//))
! (debug)
!      WRITE(16,806)
!806   FORMAT(10X,'OK-t')
!
      DO 250 L=1,NTS
!         t1=mpi_wtime()
         CALL CPU_TIME(t1)
         call exchng1(L,NXN,first,last,comm1d,myid,nbrbottom,nbrtop)
!         t2=mpi_wtime()
         CALL CPU_TIME(t2)
!         if(myid.eq.0.and.nit.eq.1)print *,'exchange time=',exchng_time
         exchng_time=exchng_time+(t2-t1)
!
!       LOOP TO CALCULATE TIME-DEPENDENT COEFFICIENTS
!
        DO J=first,last
          DO I=2,NXN
            IF(J.EQ.1)THEN
              DD(I,J,L) = 2.D0*D(I,J,L)*V(I,J+1,L)
              GO TO 291
            ENDIF
            IF(J.EQ.NYN)THEN
              DD(I,J,L) = 2.D0*D(I,J,L)*V(I,J-1,L)
              GO TO 291
            ENDIF
            DD(I,J,L) = D(I,J,L)*(V(I,J-1,L)+V(I,J+1,L))
!            if(J.eq.8.and.L.eq.1)then
!               write(61,806)V(i,j-1,L)
! 806           format(F7.4)
!            endif
291         CONTINUE
            F(I,J,L) = W(I,J,L) +DD(I,J,L) +E(I,J,L)+GS(I,J,L)-GB(I,J,L)
          END DO
        END DO
!
!       LOOP TO MODIFY F AT INFLOW BOUNDARIES
        DO J=first,last
          F(2,J,L) = F(2,J,L) - A(2,J,L)*U(1,J,L)
```

```
        END DO
!
!       LOOP TO THROUGH GRID BY ROW
!
!       BEGIN SOLUTION OF DIFFUSION ARRAY
!Thomas algorithm/ADI to solve tridiagonal matrices
        DO J=first,last
          ALF(1) = B(2,J,L)
          BETA(1) = C(2,J,L)/ALF(1)
          Y(1) = F(2,J,L)/ALF(1)
          DO 201 I=3,NXN
            AA = A(I,J,L)
            IF(I.EQ.NXN)AA = AA - C(I,J,L)
            BBB = B(I,J,L)
            IF(I.EQ.NXN)BBB = BBB + 2.D0*C(NXN,J,L)
!
            ALF(I-1) = BBB - AA*BETA(I-2)
            BETA(I-1) = C(I,J,L)/ALF(I-1)
            Y(I-1) = (F(I,J,L) - AA*Y(I-2))/ALF(I-1)
201       CONTINUE
          U(NXN,J,L) = Y(NXN-1)
          DO I=NXN-1,2,-1
            U(I,J,L) = Y(I-1) - BETA(I-1)*U(I+1,J,L)
          END DO
        END DO
!
250     CONTINUE
!*******************************************************
!(if v&u still not converged then V=U and iterations+=1)
!     CHECK CONVERGENCE OF CONCENTRATION ARRAYS
!
      myTSCK = 0.D0
      DO J=first,last
        DO I=2,NXN
          DO L=1,NTS
            TS = ABS(U(I,J,L)-V(I,J,L))
            IF(TS.GT.myTSCK)myTSCK=TS
          END DO
        END DO
!        write (debugFile,*)'process ',myid,'    j=',j,' myTSCK= ',myTSCK
      END DO
      call mpi_allreduce(myTSCK,TSCK,1,mpi_double_precision,mpi_max,
     $                   comm1d,ierr)
!     write (debugFile,*) 'process ',myid, '    TSCK=',TSCK
      IF(TSCK.GT.TOL)GO TO 281
      GO TO 298
!
!     UPDATE NEW ITERATION VALUES
281   DO J=first,last
        DO I=2,NXN
          DO L=1,NTS
            V(I,J,L) = U(I,J,L)
          END DO
        END DO
      END DO
      NIT = NIT + 1
      GO TO 299
```

```
!
298   CONTINUE
!
! Every proc sends their section of U back to proc 0 to write to output
      do L=1,NTS
      if(myid.gt.0)then
         do j=first,last
            call mpi_send(U(1,j,L),nxn,mpi_double_precision,0,j,
     $           comm1d,ierr)
!!            write(debugFile,*)'myid = ',myid,'   row ',j,' sent',' L=',L
         enddo
      endif
      if (myid.eq.0)then
         do j=last+1,nyn
            call mpi_recv(U(1,j,L),nxn,mpi_double_precision,source(j),j,
     $           comm1d,status,ierr)
!!            write (debugFile,*) 'row ',j,' L=',L,' received'
         enddo
      endif
      enddo
!
      RETURN
      END subroutine TRIDIA
!
!*************************************************************
      subroutine mpe_decomp1d( n, numprocs, myid, s, e, source )
      integer n, numprocs, myid, s, e
      integer nlocal
      integer deficit
        integer source(1:n)
!The first do loop has process 0 get the process id of each row--to be
!used when messages are sent at the end of the program
        if(myid.eq.0)then
           do i=1,numprocs
                 nlocal = n/numprocs
                 s = (i-1) * nlocal + 1
                 deficit = mod(n,numprocs)
                 s = s + min(i-1, deficit)
                 if (i-1.lt.deficit) then
                    nlocal = nlocal +1
                 endif
                 e = s + nlocal - 1
                 if (e.gt.n.or.i-1.eq.numprocs-1) e=n
                 do j=s,e
                    source(j)=i-1
                 enddo
           enddo
        endif
!This part is the actual decomposition
      nlocal = n/numprocs
      s = myid * nlocal + 1
      deficit = mod(n,numprocs)
      s = s + min(myid, deficit)
      if (myid.lt.deficit) then
         nlocal = nlocal +1
      endif
      e = s + nlocal - 1
```

```
      if (e.gt.n.or.myid.eq.numprocs-1) e=n

      return
      end subroutine mpe_decomp1d
!
!
      subroutine exchng1( l, n, s, e, comm1d, myid, nbrbottom, nbrtop )
      include 'mpif.h'
      integer n, s, e, k, l
!     double precision v(1:n,s-1:e+1,1)
!        double precision v(999,100,10)
        COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      integer comm1d, myid, nbrbottom, nbrtop
      integer status(MPI_STATUS_SIZE), ierr
!
!       MPI_SENDRECV(sendbuf,sendcount,sendtype,dest,sendtag,
!                    recvbuf,recvcount,recvtype,source,recvtag,
!                  comm,status,ierror)
!
!       Exchange Neighbors
      call MPI_SENDRECV(
     &            v(1,e,l), n, MPI_DOUBLE_PRECISION, nbrtop, 0,
     &            v(1,s-1,l), n, MPI_DOUBLE_PRECISION, nbrbottom, 0,
     &            comm1d, status, ierr )
      call MPI_SENDRECV(
     &            v(1,s,l), n, MPI_DOUBLE_PRECISION, nbrbottom, 1,
     &            v(1,e+1,l), n, MPI_DOUBLE_PRECISION, nbrtop, 1,
     &            comm1d, status, ierr )
!
      return
      end subroutine exchng1
!
!************************************************************
      SUBROUTINE ADVECT(ISOL)
!     CALCULATE ADVECTION CONCENTRATIONS ARRAYS V(I,J,L)
!
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      include 'mpif.h'
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      DIMENSION ADX(10), IADX(10)
      common /mpi/ numprocs,comm1d,myid,ierr,nbrtop,nbrbottom,first,last
!
! (debug)
!      WRITE(16,306)
!306   FORMAT(10X,'OK-A')
!
      DO 300 L=1,NTS
!
        DO 305 J=first,last
          DO 310 I=2,NXN
            V(I,J,L) = W(I-1,J,L)
!
!            I1 = I - IADX(L)
!            I2 = I1 - 1
!            DELX = ADX(L) - DX*(I1-I)
!            DELC = W(I2,J,L) - W(I1,J,L)
```

```
!            IF(I1.LE.1)THEN
!              V(I,J,L) = W(1,J,L)
!              GO TO 310
!            ENDIF
!            V(I,J,L) = W(I1,J,L) + DELC*DELX/DX
310       CONTINUE
305     CONTINUE
!
        DO 315 J=first,last
          DO 320 I=2,NXN
            W(I,J,L) = V(I,J,L)
320       CONTINUE
315     CONTINUE
!
300   CONTINUE
      RETURN
      END subroutine ADVECT
!
!
      SUBROUTINE MASBAL(KOBS,NNFP,NPS)
!     CALCULATES MASS BALANCE FOR EACH CONSTITUENT
!
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /CON5/ SNAME(10), BNAME(10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      COMMON /MASS1/ OSTOR(10), OFLUXIN(10), OFLUXOUT(10), OBIO(10)
      COMMON /MASS2/ STOR(10), FLUXIN(10), FLUXOUT(10), BIO(10)
      COMMON /MASS3/ CM1(99,10), CM2(99,10), CM3(99,10), IMB(10), DM
      COMMON /MASS4/ FP(99), IP(99), JP(99), OFLXH(10)
      COMMON /MASS5/ AMEM(99), DMEM(99), PMEM(99,9), HMTC(999), FM, FMB
      common /mpi/ numprocs,comm1d,myid,ierr,nbrtop,nbrbottom,first,last
      CHARACTER BNAME*40, SNAME*40
!
      DIMENSION BAL(10), ER(10), PDIF(10), DSTOR(10)
      DIMENSION AFLO(10), AFLIN(10), AFLXH(10), FLXH(10), ABIO(10)
!
      if(myid.eq.0)WRITE(18,451)
!
      DO 400 L=1,NTS
!
        IF(IMB(L).EQ.0)GO TO 400
!
        FLUXIN(L) = 0.D0
        FLUXOUT(L) = 0.D0
        FLXH(L) = 0.D0
        STOR(L) = 0.D0
!
!*******CALCULATE MASS FLUX AT BOUNDARIES
!
        DO 423 J=first,last
          YN=1.D0
          IF(J.EQ.1)YN=0.5D0
        IF(J.EQ.NYN)YN=0.5D0
        FLUXIN(L) = FLUXIN(L) + YN*CM1(J,L)*U(1,J,L)
          FLUXIN(L) = FLUXIN(L) + YN*(U(2,J,L)-U(1,J,L))*CM2(J,L)
          FLUXOUT(L) = FLUXOUT(L) + YN*CM1(J,L)*U(NXN,J,L)
```

```
          FLUXOUT(L)= FLUXOUT(L)+YN*(U(NXN,J,L)-U(NXN-1,J,L))*CM2(J,L)
423     CONTINUE
!
!*******CALCULATE HYDROGEN MASS FLUX AT POINT SOURCES
!
        IF(L.GT.1)GO TO 422
        FLXH(1) = 0.D0
        IF(KOBS.EQ.1)OFLXH(L) = 0.D0
        IF(NPS.EQ.0)GO TO 424
        DO KP=1,NPS
          IS = IP(KP)
          JS = JP(KP)
          FXH = HMTC(KP)*AMEM(KP)*(PMEM(KP,NNFP)*FM-U(IS,JS,1))
          FLXH(L)=FLXH(L)+FXH*FMB
        END DO
424     AFLXH(L) = (OFLXH(L) + FLXH(L))*0.5D0
!
!*******CALCULATION OF NEW MASS IN SYSTEM************************
!
422     DO 428 I=1,NXN
         XN = 1.D0
         IF(I.EQ.1)XN=0.5D0
         IF(I.EQ.NXN)XN=0.5D0
          DO 429 J=first,last
              YN = 1.0D0
              IF(J.EQ.NYN)YN=0.5D0
              IF(J.EQ.1)YN=0.5D0
              STOR(L) = STOR(L) + XN*YN*U(I,J,L)*DM
429         CONTINUE
428     CONTINUE
!
        AFLIN(L) = (FLUXIN(L) + OFLUXIN(L))*0.5D0
        AFLO(L) = (FLUXOUT(L) + OFLUXOUT(L))*0.5D0
        ABIO(L) = (BIO(L) + OBIO(L))*0.5D0
        DSTOR(L) = OSTOR(L) + AFLIN(L) - AFLO(L) - ABIO(L)
        IF(L.EQ.1)DSTOR(L) = DSTOR(L) + AFLXH(L)
!
        BAL(L) = STOR(L) - DSTOR(L)
        IF(STOR(L).EQ.0)GO TO 416
        ER(L) = BAL(L)/STOR(L)
        PDIF(L) = 100.D0*ER(L)
!

416    if(myid.eq.0) WRITE(18,452)L,PDIF(L),OSTOR(L),STOR(L),FLUXIN(L),
     +       FLUXOUT(L),BIO(L),FLXH(L)
!
        OSTOR(L) = STOR(L)
        OFLUXIN(L) = FLUXIN(L)
        OFLUXOUT(L) = FLUXOUT(L)
        OBIO(L) = BIO(L)
        OFLXH(L) = FLXH(L)
!
400   CONTINUE
!
451   FORMAT(3X,'SOLUTE#',5X,'% ERROR',4X,'OLD MASS',4X,'NEW MASS',5X,
     + 'MASS IN',4X,'MASS OUT',4X,'BIO LOSS',5X,'H2 FLUX')
452   FORMAT(5X,I2,5X,F10.2,6(2X,E10.3))
```

```
!
      RETURN
      END subroutine MASBAL
!
!
      SUBROUTINE RESTART(IRS)
!     USED FOR READING ARRAYS FROM FILE STARTUP
!
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
      COMMON /CON5/ SNAME(10), BNAME(10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      COMMON /MASS1/ OSTOR(10), OFLUXIN(10), OFLUXOUT(10), OBIO(10)
      COMMON /MASS4/ FP(99), IP(99), JP(99), OFLXH(10)
      CHARACTER BNAME*40, SNAME*40
!
      IF(IRS.EQ.2)GO TO 1001
!
      DO 1000 I=1,NXN
        DO 1002 J=1,NYN
1002      READ(28,1005)(W(I,J,L),L=1,NTS),(WB(I,J,L),L=1,NBIO)
1000  CONTINUE
!
      DO 1004 L=1,NTS
1004    READ(28,1007)OSTOR(L),OFLUXIN(L),OFLUXOUT(L),OBIO(L),OFLXH(L)
!
      GO TO 1010
!
1001  DO 1006 I=1,NXN
        DO 1008 J=1,NYN
1008      WRITE(29,1005)(W(I,J,L),L=1,NTS),(WB(I,J,L),L=1,NBIO)
1006  CONTINUE
!
      DO 1009 L=1,NTS
1009    WRITE(29,1007)OSTOR(L),OFLUXIN(L),OFLUXOUT(L),OBIO(L),OFLXH(L)
!
1010  CONTINUE
1005  FORMAT(8(1X,E11.5))
1007  FORMAT(5(2X,E11.5))
      RETURN
      END subroutine RESTART
!
!
      SUBROUTINE TDOUT
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
      COMMON /CON5/ SNAME(10), BNAME(10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      COMMON /CON8/ XNODE(999,100), YNODE(999,100)
      CHARACTER BNAME*40, SNAME*40
!
      WRITE(30,1101)TIME
      DO L=1,NTS
        WRITE(30,1102)SNAME(L)
        DO J=NYN,1,-1
```

```
          WRITE(30,1103)(U(I,J,L),I=1,NXN)
        END DO
      END DO
!
      DO L=1,NBIO
        WRITE(30,1104)
        DO J=NYN,1,-1
          WRITE(30,1103)(UB(I,J,L),I=1,NXN)
        END DO
      END DO
!
1101  FORMAT(/,5X,'TIME = ',F10.4)
1102  FORMAT(1X,'SOLUTE:',A40)
!1103  FORMAT(8E10.3)
1103  FORMAT(8F10.2)
1104  FORMAT(1X,'MICROBIAL POPULATION')
!
      WRITE(31,1101)TIME
      DO J=1,NYN
       DO I=1,NXN
        WRITE(31,1105)XNODE(I,J),YNODE(I,J),(U(I,J,L),L=1,NTS),UB(I,J,1)
!          write(31,1105)(U(i,j,L),L=1,nts),UB(i,j,L)
       END DO
      END DO
!
1105  FORMAT(2F10.2,6E10.3)
! 1105 format(6E10.3)
!
      RETURN
      END
!
!
      SUBROUTINE BIOTRANS(ISOL,source)
!     SOLVES TERMS FOR BIOTRANSFOMATION AND GENERATION
!
!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      include 'mpif.h'
      integer numprocs,comm1d,myid,ierr,nbrtop,nbrbottom
      integer first,last
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
      COMMON /TAL3/ GB(999,100,10), GS(999,100,10)
      COMMON /CON2/ RO(10),RN1(10),RN2(10),KHO(10),KHN1(10),KHN2(10)
      COMMON /CON3/ GAMO, KO, KC(10,10), GAMN1, KN1, GAMN2, KN2
      COMMON /CON4/ YD(9), DK(9), ZETA1, ZETA2, RF(10)
      COMMON /CON5/ SNAME(10), BNAME(10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      COMMON /CON7/ UMIN(10), UBMIN(10)
      COMMON /MASS2/ STOR(10), FLUXIN(10), FLUXOUT(10), BIO(10)
      COMMON /MASS3/ CM1(99,10), CM2(99,10), CM3(99,10), IMB(10), DM
      common /mpi/ numprocs,comm1d,myid,ierr,nbrtop,nbrbottom,first,last
      DIMENSION SMTO(9), SMTN1(9), SMTN2(9), EFC(999,100,10)
      DOUBLE PRECISION KHO,KHN1,KHN2,KO,KN1,KN2,KC,IC,INON1,INON2,INN1N2
      CHARACTER BNAME*40, SNAME*40
      double precision EX
      integer source(1:nyn)
      integer status(mpi_status_size)
```

```
!
! (debug)
!      WRITE(16,406)
!406   FORMAT(10X,'OK-B')
!
      DO L =1,NTS
        BIO(L) = 0.D0
      END DO
!
      DO 300 I=2,NXN
        DO 301 J=first,last
!
        GB(I,J,5) = 0.D0
        GS(I,J,1) = 0.D0
        GS(I,J,2) = 0.D0
        GS(I,J,3) = 0.D0
!
!       CALCULATE HYDROGEN MONOD TERMS
!
          EFC(I,J,1) = W(I,J,1) - UMIN(1)
          EFKHO = KHO(1)-UMIN(1)
          IF(EFKHO.LT.0)EFKHO = 0.D0
          EFKHN1 = KHN1(1)-UMIN(1)
          IF(EFKHN1.LT.0)EFKHN1 = 0.D0
          EFKHN2 = KHN2(1)-UMIN(1)
          IF(EFKHN2.LT.0)EFKHN2 = 0.D0
!
          IF(EFC(I,J,1).GT.0)THEN
            SMTO(1) = EFC(I,J,1)/(EFKHO+EFC(I,J,1))
            SMTN1(1) = EFC(I,J,1)/(EFKHN1+EFC(I,J,1))
            SMTN2(1) = EFC(I,J,1)/(EFKHN2+EFC(I,J,1))
          ELSE
            SMTO(1) = 0.D0
            SMTN1(1) = 0.D0
            SMTN2(1) = 0.D0
          ENDIF
          SO = SMTO(1)*RO(1)
          SN1 = SMTN1(1)*RN1(1)
          SN2 = SMTN2(1)*RN2(1)
!
!       CALCULATE EA MONOD TERMS
!
            !oxygen
            EFC(I,J,2) = W(I,J,2) - UMIN(2)
            EFKO = KO-UMIN(2)
            IF(EFKO.LT.0)EFKO = 0.D0
            IF(EFC(I,J,2).GT.0)THEN
              EAO = EFC(I,J,2)/(EFKO+EFC(I,J,2))
              INON1 = KC(2,3)/(KC(2,3)+W(I,J,2))
              INON2 = KC(2,4)/(KC(2,4)+W(I,J,2))
            ELSE
              EAO = 0.D0
              INON1 = 1.D0
              INON2 = 1.D0
            ENDIF
!
            !nitrate
```

```
          EFC(I,J,3) = W(I,J,3) - UMIN(3)
          EFKN1 = KN1-UMIN(3)
          IF(EFKN1.LT.0)EFKN1 = 0.D0
          IF(EFC(I,J,3).GT.0)THEN
            EAN1 = INON1*EFC(I,J,3)/(EFKN1+EFC(I,J,3))
            INN1N2 = KC(3,4)/(KC(3,4)+W(I,J,3))
          ELSE
            EAN1 = 0.D0
            INN1N2 = 1.D0
          ENDIF
!
          !nitrite
          EFC(I,J,4) = W(I,J,4) - UMIN(4)
          EFKN2 = KN2-UMIN(4)
          IF(EFKN2.LT.0)EFKN2 = 0.D0
          IF(EFC(I,J,4).GT.0)THEN
            EAN2 = INON1*INN1N2*EFC(I,J,4)/(EFKN2+EFC(I,J,4))
          ELSE
            EAN2 = 0.D0
          ENDIF
!
!      CALCULATE TERMS FOR SOLUTION AND TERMS FOR MASS BALANCE CHECK
!
          WBB = WB(I,J,1)/POR
          GB1 = WBB*(SO*EAO + SN1*EAN1 + SN2*EAN2)
          GB2 = GAMO*WBB*SO*EAO
          GB3 = GAMN1*WBB*SN1*EAN1
          GB4 = GAMN2*WBB*SN2*EAN2
!
          GB(I,J,1) = GB1*DTA/RF(1)
          GB(I,J,2) = GB2*DTA/RF(2)
          GB(I,J,3) = GB3*DTA/RF(3)
          GB(I,J,4) = GB4*DTA/RF(4)
!
          DO L=1,NTS
            GBCHK = GB(I,J,L)
            IF(GBCHK.GT.W(I,J,L))GB(I,J,L) = W(I,J,L)
          END DO
!
          GS(I,J,4) = GB(I,J,3)*ZETA1
          GS(I,J,5) = GB(I,J,4)*ZETA2
!
          DO L=1,NTS
            BIO(L) = BIO(L) + (GB(I,J,L)-GS(I,J,L))*DM
          END DO
!
!      CALCULATE BIOMASS CONCENTRATION
!
          EX = (YD(1)*(SN1*EAN1+SN2*EAN2)-DK(1))*DTA
          UB(I,J,1) = WB(I,J,1)*DEXP(EX)
          WB(I,J,1) = UB(I,J,1)
!
301     CONTINUE
300   CONTINUE
!
!      do L=1,NTS
      if(myid.gt.0)then
```

```
          do j=first,last
            call mpi_send(UB(1,j,1),nxn,mpi_double_precision,0,j,
     $           comm1d,ierr)
          enddo
        endif
        if (myid.eq.0)then
          do j=last+1,nyn
            call mpi_recv(UB(1,j,1),nxn,mpi_double_precision,source(j),j,
     $           comm1d,status,ierr)
          enddo
        endif
!       enddo
!
        RETURN
        END subroutine BIOTRANS
!
!
!
!
        subroutine read_data
!
        COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
        COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
        COMMON /TAL1/ A(999,100,10), B(999,100,10), C(999,100,10)
        COMMON /TAL2/ D(999,100,10), E(999,100,10), F(999,100,10)
        COMMON /TAL3/ GB(999,100,10), GS(999,100,10)
        COMMON /CON1/ RL(999,100,10), NIT, TOL
        COMMON /CON2/ RO(10),RN1(10),RN2(10),KHO(10),KHN1(10),KHN2(10)
        COMMON /CON3/ GAMO, KO, KC(10,10), GAMN1, KN1, GAMN2, KN2
        COMMON /CON4/ YD(9), DK(9), ZETA1, ZETA2, RF(10)
        COMMON /CON5/ SNAME(10), BNAME(10)
        COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
        COMMON /CON7/ UMIN(10), UBMIN(10)
        COMMON /CON8/ XNODE(999,100), YNODE(999,100)
        COMMON RT(999,100,10)
        COMMON DXX(999,100,10), DYY(999,100,10)
        COMMON UOBS(15,2000,10), IOBS(15), JOBS(15)
        COMMON /MASS1/ OSTOR(10), OFLUXIN(10), OFLUXOUT(10), OBIO(10)
        COMMON /MASS2/ STOR(10), FLUXIN(10), FLUXOUT(10), BIO(10)
        COMMON /MASS3/ CM1(99,10), CM2(99,10), CM3(99,10), IMB(10), DM
        COMMON /MASS4/ FP(99), IP(99), JP(99), OFLXH(10)
        COMMON /MASS5/ AMEM(99), DMEM(99), PMEM(99,9), HMTC(999), FM, FMB
        common /mpi/ numprocs,comm1d,myid,ierr,nbrtop,nbrbottom,first,last
        COMMON IC(10), BCT(10), BIC(10), DF(10)
        COMMON IBCT(10), IIC(10), IBIC(10)
        COMMON TIMOUT(15), BT(10)
        DOUBLE PRECISION KHO,KHN1,KHN2,KO,KN1,KN2,KC,IC
        CHARACTER TITL1*80, TITL2*80, BNAME*40, SNAME*40
        common /dat1/ TITL1,TITL2,TIMSIM,AX,AY,ATOT,VTHK,AL,VELY,AT,BDEN
        common /dat2/ DELDT,NPS,HLC,HMW,WKV,PORW,ACF,BCF,CCF,NFP
        common /dat3/ REY,SC,DTAOPT,ISOL,IRS,NTSTP,DT,TM,NOBS,NTIM
!
! Read Title
        READ(15,612)TITL1
        READ(15,612)TITL2
! Read Model Grid Parameters
        READ(15,645)NXN,NYN,NTS,NBIO
```

```
      READ(15,630)DX,DY,VTHK
!change number of nodes
!      NXN = 250
!      NYN = 95
! Calculate spatial dimensions and coordinates
      AX = DX*(NXN-1)
      AY = DY*(NYN-1)
      ATOT = AX*AY
      N = NXN - 1
      DO 901 I=1,NXN
        DO 902 J=1,NYN
          XNODE(I,J) = DX*(I-1)
902       YNODE(I,J) = DY*(J-1)
901   CONTINUE
!
! Read Solution & Restart Option
      READ(15,625)ISOL,IRS
!
! Read Iteration Error Tolerance
      READ(15,610)TOL
!
! Read Time Step Data
      READ(15,634)NTSTP,DTA
      TIMSIM = DTA*NTSTP
!
! Read Velocity & Dispersivity & Porosity & Bulk Density
      READ(15,650)VELX,AL,AT,POR,BDEN
      VELY = 0.D0
!
! Check Time Step Size
      DTAOPT = DX/VELX
      DELDT = DTAOPT-DTA
      DT = DTA
      TM = 0.D0
! Read Initial & Boundary Conditions
!
      DO 622 L=1,NTS
         READ(15,611)SNAME(L)
         READ(15,630)IC(L),BCT(L),UMIN(L)
         READ(15,635)IBCT(L),IIC(L),IMB(L)
         IF(IBCT(L).EQ.1)READ(15,*)(U(1,J,L),I=1,NYN)
         IF(IIC(L).EQ.1)THEN
            DO I=1,NXN
                  READ(15,*)(W(I,J,L),J=1,NYN)
            END DO
         ENDIF
!
! Read Diffusion and Distribution Coefficients
         READ(15,620)DF(L),BT(L)
622   CONTINUE
!
! Calculate Retardation Factors & Dispersion Coefficients
!
      DO 903 L=1,NTS
         RF(L) = 1.D0 + BT(L)*BDEN/POR
         DO I=1,NXN
           DO J=1,NYN
```

```
            DXX(I,J,L) = VELX*AL + DF(L)
            DYY(I,J,L) = VELX*AT + DF(L)
          END DO
        END DO
903   CONTINUE
!
! Read Initial Microbial Concentrations
!
! (NBIO = 1 for Autotrophic Denitrifiers)
      NBIO = 1
      DO 632 L=1,NBIO
         READ(15,611)BNAME(L)
         READ(15,620)BIC(L),UBMIN(L)
         READ(15,620)YD(L),DK(L)
         READ(15,615)IBIC(L)
         IF(IBIC(L).EQ.1)THEN
           DO I=1,NXN
                 READ(15,*)(WB(I,J,L),J=1,NYN)
           END DO
         ENDIF
632   CONTINUE
!
! Read Bio Parameters (Hydrogen)
      READ(15,630)RO(1),RN1(1),RN2(1)
      READ(15,630)KHO(1),KHN1(1),KHN2(1)
!
! Read Bio Parameters (Oxygen)
      READ(15,620)GAMO,KO
! Read Oxygen-induced Inhibition Coefficients
      READ(15,620)KC(2,3),KC(2,4)
!
! Read Bio Parameters (Nitrate)
      READ(15,630)GAMN1,KN1,ZETA1
! Read Nitrate-induced Inhibition Coefficients
      READ(15,610)KC(3,4)
!
! Read Bio Parameters (Nitrite)
      READ(15,630)GAMN2,KN2,ZETA2
!
! Read Hydrogen Flux Parameters
      READ(15,625)NPS,NFP
      IF(NPS.EQ.0)GO TO 642
!
      READ(15,630)HLC,HMW,WKV
      READ(15,640)ACF,BCF,CCF,PORW
      SC = (WKV/DF(1))**CCF
!
      DO K=1,NPS
        READ(15,620)AMEM(K),DMEM(K)
        REY = (POR*VELX*DMEM(K)/(PORW*WKV))**BCF
        HMTC(K) = DF(1)*ACF*SC*REY/DMEM(K)
      END DO

      IF(NFP.GT.1)READ(15,680)(FP(M),M=1,NFP)
      IF(NFP.EQ.1)FP(1)=TIMSIM
      DO 644 M=1,NFP
         DO 646 K=1,NPS
```

```
646      READ(15,633)IP(K),JP(K),PMEM(K,M)
644   CONTINUE
!
! Make the hydrogen sources extend to every node in the width
! (Used when the number of nodes is increased)
!!      NPS = NYN
!!      AMEM(NYN) = AMEM(13)
!!      DO K=13,NPS-1
!!         AMEM(K)=AMEM(K-1)
!!      END DO
!!      DO K=13,NPS
!!          DMEM(K)=DMEM(K-1)
!!          REY = (POR*VELX*DMEM(K)/(PORW*WKV))**BCF
!!        HMTC(K) = DF(1)*ACF*SC*REY/DMEM(K)
!!      END DO
!!
!!      DO M=1,NFP
!!      DO K=2,NPS
!!          IP(K) = IP(1)
!!          JP(K) = K
!!          PMEM(K,M) = 1.0
!!      END DO
!!      END DO
!
! Output Options
!
! Observation Nodes
642   READ(15,625)NOBS
      IF(NOBS.EQ.0)GO TO 648
      DO 652 L=1,NOBS
652      READ(15,625)IOBS(L),JOBS(L)
!
! Frequency of Output
648   READ(15,615)NTIM
!
610   FORMAT(D10.5)
615   FORMAT(I5)
611   FORMAT(A40)
612   FORMAT(A80)
620   FORMAT(2D10.5)
625   FORMAT(2I5)
630   FORMAT(3D10.5)
635   FORMAT(3I5)
640   FORMAT(4D10.5)
645   FORMAT(4I5)
650   FORMAT(5D10.5)
655   FORMAT(5I5)
660   FORMAT(6D10.5)
665   FORMAT(6I5)
670   FORMAT(7D10.5)
675   FORMAT(7I5)
680   FORMAT(8D10.5)
685   FORMAT(8I5)
633   FORMAT(2I5,D10.3)
634   FORMAT(I5,D12.3)
!
!      close(15)
```

```
!
      end subroutine read_data
!
!
!
      subroutine write_data
!
      COMMON /CONC/ U(999,100,10), V(999,100,10), W(999,100,10)
      COMMON /BIO/ UB(999,100,10), VB(999,100,10), WB(999,100,10)
      COMMON /TAL1/ A(999,100,10), B(999,100,10), C(999,100,10)
      COMMON /TAL2/ D(999,100,10), E(999,100,10), F(999,100,10)
      COMMON /TAL3/ GB(999,100,10), GS(999,100,10)
      COMMON /CON1/ RL(999,100,10), NIT, TOL
      COMMON /CON2/ RO(10),RN1(10),RN2(10),KHO(10),KHN1(10),KHN2(10)
      COMMON /CON3/ GAMO, KO, KC(10,10), GAMN1, KN1, GAMN2, KN2
      COMMON /CON4/ YD(9), DK(9), ZETA1, ZETA2, RF(10)
      COMMON /CON5/ SNAME(10), BNAME(10)
      COMMON /CON6/ NTS,NXN,NYN,NBIO,DX,DY,DTA,TIME,POR,TH,VELX
      COMMON /CON7/ UMIN(10), UBMIN(10)
      COMMON /CON8/ XNODE(999,100), YNODE(999,100)
      COMMON RT(999,100,10)
      COMMON DXX(999,100,10), DYY(999,100,10)
      COMMON UOBS(15,2000,10), IOBS(15), JOBS(15)
      COMMON /MASS1/ OSTOR(10), OFLUXIN(10), OFLUXOUT(10), OBIO(10)
      COMMON /MASS2/ STOR(10), FLUXIN(10), FLUXOUT(10), BIO(10)
      COMMON /MASS3/ CM1(99,10), CM2(99,10), CM3(99,10), IMB(10), DM
      COMMON /MASS4/ FP(99), IP(99), JP(99), OFLXH(10)
      COMMON /MASS5/ AMEM(99), DMEM(99), PMEM(99,9), HMTC(999), FM, FMB
      COMMON IC(10), BCT(10), BIC(10), DF(10)
      COMMON IBCT(10), IIC(10), IBIC(10)
      COMMON TIMOUT(15), BT(10)
      DOUBLE PRECISION KHO,KHN1,KHN2,KO,KN1,KN2,KC,IC
      CHARACTER TITL1*80, TITL2*80, BNAME*40, SNAME*40
      common /dat1/ TITL1,TITL2,TIMSIM,AX,AY,ATOT,VTHK,AL,VELY,AT,BDEN
      common /dat2/ DELDT,NPS,HLC,HMW,WKV,PORW,ACF,BCF,CCF,NFP
      common /dat3/ REY,SC,DTAOPT,ISOL,IRS,NTSTP,DT,TM,NOBS,NTIM
!
      WRITE(16,700)
      WRITE(16,701)TITL1
      WRITE(16,701)TITL2
      WRITE(16,705)NXN,NYN
      WRITE(16,706)DX,DY
      WRITE(16,707)AX,AY,VTHK
      WRITE(16,708)VELX,AL,POR
      WRITE(16,709)VELY,AT,BDEN
      WRITE(16,702)DTA,TIMSIM
      if(DELDT.ge.TOL)then
         WRITE(16,703)
         WRITE(*,703)
         WRITE(16,704)DTAOPT
         WRITE(*,704)DTAOPT
      endif
      DO L=1,NTS
         WRITE(16,710)SNAME(L)
         WRITE(16,775)IC(L)
         WRITE(16,780)BCT(L)
         WRITE(16,785)UMIN(L)
```

```
      WRITE(16,715)DF(L)
      WRITE(16,720)DXX(2,2,L)
      WRITE(16,725)DYY(2,2,L)
      WRITE(16,730)RF(L)
      WRITE(16,735)BT(L)
   END DO
   DO L=1,NBIO
      WRITE(16,770)BNAME(L)
      WRITE(16,775)BIC(L)
      WRITE(16,785)UBMIN(L)
      WRITE(16,747)YD(L)
      WRITE(16,748)DK(L)
   END DO
   DO L=2,4
      IF(L.EQ.2)THEN
         WRITE(16,750)SNAME(L)
         WRITE(16,749)RO(1)
         WRITE(16,751)GAMO
         WRITE(16,752)KHO(1)
         WRITE(16,753)KO
         WRITE(16,754)KC(L,L+1)
         WRITE(16,755)KC(L,L+2)
      ENDIF
      IF(L.EQ.3)THEN
         WRITE(16,750)SNAME(L)
         WRITE(16,749)RN1(1)
         WRITE(16,751)GAMN1
         WRITE(16,752)KHN1(1)
         WRITE(16,753)KN1
         WRITE(16,756)KC(L,L+1)
         WRITE(16,757)ZETA1
      ENDIF
      IF(L.EQ.4)THEN
         WRITE(16,750)SNAME(L)
         WRITE(16,749)RN2(1)
         WRITE(16,751)GAMN2
         WRITE(16,752)KHN2(1)
         WRITE(16,753)KN2
         WRITE(16,757)ZETA2
      ENDIF
   END DO
   if(NPS.gt.0)then
      WRITE(16,761)NPS
      WRITE(16,764)HLC
      WRITE(16,765)HMW
      WRITE(16,768)WKV
      WRITE(16,769)PORW
      WRITE(16,771)ACF
      WRITE(16,772)BCF
      WRITE(16,773)CCF
      WRITE(16,762)NFP
      DO M=1,NFP
         WRITE(16,763)FP(M)
         WRITE(16,766)
         DO K=1,NPS
            WRITE(16,767)IP(K),JP(K),PMEM(K,M),HMTC(K),
     +           DMEM(K),AMEM(K)
```

```
           END DO
         END DO
       endif
!
701    FORMAT(A80,/)
700    FORMAT(///,25X,'COMPUTER PROGRAM FOR THE SIMULATION OF',/,15X,
     +  'TWO-DIMENSIONAL SOLUTE TRANSPORT IN POROUS MEDIA',///,
     +  26X,'**********************************',/,26X,'*',34X,'*',/,
     +  26X,'*      N2D-H2, parallel Ver 1      *',/,26X,'*',34X,'*',/,
!     +  26X,'*           N2D-H2, VER 2          *',/,26X,'*',34X,'*',/,
     +  26X,'*  DEVELOPED BY MARK A. WIDDOWSON  *',/,26X,'*',34X,'*',/,
     +  26X,'*       and JENNY WRIGHT        *',/,26X,'*',34X,'*',/,
     +  26X,'*           VIRGINIA TECH          *',/,26X,'*',34X,'*',/,
     +  26X,'* DEPARTMENT OF CIVIL ENGINEERING  *',/,26X,'*',34X,'*',/,
     +  26X,'**********************************',///)
702    FORMAT(10X,'ADVECTION TIME STEP = ',F9.5,/,
     +  10X,'TOTAL TIME OF SIMULATION = ',F9.2,/)
804    FORMAT(10X,'OK')
705    FORMAT(10X,'NUMBER OF X NODES = ',I3,/,
     +  10X,'NUMBER OF Y NODES = ',I3)
706    FORMAT(10X,'NODE SPACING IN X-DIR = ',F8.4,/,10X,
     +   'NODE SPACING IN Y-DIR = ',F8.4)
707    FORMAT(10X,'X-DIM OF DOMAIN =  ',F9.3,/,10X,'Y-DIM OF DOMAIN =  ',
     +   F9.3,/,10X,'VERTICAL THICKNESS =',F9.4,/)
708    FORMAT(10X,'LONGT VELOC = ',E11.4,/,10X,'DISP(L) = ',E11.4,/,10X,
     +          'POROSITY = ',E11.4)
709    FORMAT(10X,'TRANSV VELO = ',E11.4,/,10X,'DISP(T) = ',E11.4,/,10X,
     +  'BULK DENSITY = ',E11.4,/)
!
710    FORMAT(//,5X,'SOLUTE = ',A40)
775    FORMAT(5X,' INITIAL CONC=',9X,E11.4)
780    FORMAT(5X,' BND SOURCE CONC=',6X,E11.4)
785    FORMAT(5X,' MINIMUM CONC=',9X,E11.4)
715    FORMAT(5X,' MOLC DIFF COEFF=',6X,E11.4)
720    FORMAT(5X,' LONG DISP COEFF=',6X,E11.4)
725    FORMAT(5X,' TRANS DISP COEFF=',5X,E11.4)
730    FORMAT(5X,' RETARDATION FACT=',5X,E11.4)
735    FORMAT(5X,' DISTRIBUTN COEFF=',5X,E11.4)
!
740    FORMAT(5X,' H2-O2 SATUR CONST=',4X,E11.4)
745    FORMAT(5X,' H2-NO3 SATUR CONST=',3X,E11.4)
746    FORMAT(5X,' H2-NO2 SATUR CONST=',3X,E11.4)
747    FORMAT(5X,' BIOMASS YIELD COEFF=',2X,E11.4)
748    FORMAT(5X,' BIOMASS DECAY COEFF=',2X,E11.4)
!
750    FORMAT(//,5X,'ELECTRON ACCEPTOR = ',A40)
749    FORMAT(5X,' H2 RATE OF UTIL=',6X,E11.4)
751    FORMAT(5X,' USE COEF FOR EA=',6X,E11.4)
752    FORMAT(5X,' H2 SAT CONST=',9X,E11.4)
753    FORMAT(5X,' EA SAT CONST=',9X,E11.4)
754    FORMAT(5X,' O2-NO3 INHIB COEF=',4X,E11.4)
755    FORMAT(5X,' O2-NO2 INHIB COEF=',4X,E11.4)
756    FORMAT(5X,' NO3-NO2 INHIB COEF=',3X,E11.4)
757    FORMAT(5X,' PRODUCTION COEF=',6X,E11.4)
!
761    FORMAT(//,5X,' NO OF PT SOURCE=',I5)
764    FORMAT(5X,' HENRYS LAW COEFF=',F10.5)
```

```
765   FORMAT(5X,' H2 MOLECULAR WT=',F10.5)
768   FORMAT(5X,' WATER KINEMATIC VISCOSITY=',E11.4)
769   FORMAT(5X,' H2 FLUX ZONE POROSITY=',F10.5)
!
771   FORMAT(5X,' H2 MASS TRANSFER CONST (a) =',F10.5)
772   FORMAT(5X,' H2 MASS TRANSFER CONST (b) =',F10.5)
773   FORMAT(5X,' H2 MASS TRANSFER CONST (c) =',F10.5)
!
762   FORMAT(5X,' NO OF FLUX PERIODS=',I5)
763   FORMAT(/,5X,' DURATION OF FLUX PERIOD=',F10.4,/)
766   FORMAT(/,5X,'COLUMN NO',5X,'ROW NO',5X,'MEM PRESSURE',4X,
     + 'MEM DIAMETER',4X,'MEM SURFAREA',4X,'H2 MASS TRANS COEFT',/)
767   FORMAT(8X,I2,11X,I2,8X,F10.4,5X,E12.4,4X,E12.4,7X,E12.4)
!
770   FORMAT(//,5X,'MICROBIAL POPULATION = ',A40)
!
703   FORMAT(/,10X,'********** WARNING !!! **********',//,10X,
     + '***** COURANT NUMBER DOES NOT EQUAL ONE *****',/,
     + '***** SET ADVECTION TIME STEP TO DTAOPT *****',/,
     + '************ AND RESTART PROGRAM ************',/)
704   FORMAT(10X,'DTA OPTIMUM = ',E11.4,/)
!
      end subroutine write_data
```

9.3 Input files

```
Test 1 – 1D (no H2)
Reactive Transport – Lab Experiment
   49   25    5    1
  2.55E-02  2.55E-02  2.50E-02
    1    0
  1.00E-02
  600  8.3333E-02
  3.06E-01  1.00E-02  1.00E-02  3.50E-01  1.90E-06
 Hydrogen
  1.00E-04  1.00E-04  1.00E-20
    0    0    1
 3.456E-04  0.00E+00
 Oxygen
  1.00E-05  2.00E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrate
  1.00E-08 10.20E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrite
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 N2
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Autotrophic Denitrifiers
  2.50E-00  2.50E-03
  5.00E-01  0.00E-03
    0
  8.00E+00  5.00E-01  2.50E-01
  1.00E-02  1.00E-01  1.00E+00
  8.00E+00  1.10E+01
  9.00E+09  9.00E+09
  2.30E+01  2.00E+01 7.419E-01
  9.00E+09
 1.533E+00  5.00E+00 3.043E-01
    0    1
    3
    5    7
   13    7
   49    7
   60
```

```
Test 2 - 1D (1 H2 source)
Reactive Transport - Lab Experiment
   49   25    5    1
  2.55E-02  2.55E-02  2.50E-02
    1    0
  1.00E-02
  600  8.3333E-02
  3.06E-01  1.00E-02  1.00E-02  3.50E-01  1.90E-06
 Hydrogen
  1.00E-04  1.00E-04  1.00E-20
    0    0    1
 3.456E-04  0.00E+00
 Oxygen
  1.00E-05  2.00E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrate
  1.00E-08 10.20E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrite
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 N2
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Autotrophic Denitrifiers
  2.50E-00  2.50E-03
  5.00E-01  0.00E-03
    0
  8.00E+00  5.00E-01  2.50E-01
  1.00E-02  1.00E-01  1.00E+00
  8.00E+00  1.10E+01
  9.00E+09  9.00E+09
  2.30E+01  2.00E+01 7.419E-01
  9.00E+09
 1.533E+00  5.00E+00 3.043E-01
   25    1
 1.230E+00 2.016E+00 8.667E-02
  8.24E-01  3.90E-01 3.300E-01 3.500E-01
 3.965E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
```

```
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 3.965E-04  1.65E-03
    8    1  1.00E+00
    8    2  1.00E+00
    8    3  1.00E+00
    8    4  1.00E+00
    8    5  1.00E+00
    8    6  1.00E+00
    8    7  1.00E+00
    8    8  1.00E+00
    8    9  1.00E+00
    8   10  1.00E+00
    8   11  1.00E+00
    8   12  1.00E+00
    8   13  1.00E+00
    8   14  1.00E+00
    8   15  1.00E+00
    8   16  1.00E+00
    8   17  1.00E+00
    8   18  1.00E+00
    8   19  1.00E+00
    8   20  1.00E+00
    8   21  1.00E+00
    8   22  1.00E+00
    8   23  1.00E+00
    8   24  1.00E+00
    8   25  1.00E+00
    3
    5    7
   13    7
   49    7
   60
```

```
Test 3 - 2D (no H2)
Reactive Transport - Lab Experiment
   49   25    5    1
  2.55E-02  2.55E-02  2.50E-02
    1    0
  1.00E-02
  600  8.3333E-02
  3.06E-01  1.00E-02  1.00E-02  3.50E-01  1.90E-06
 Hydrogen
  1.00E-04  1.00E-04  1.00E-20
    0    0    1
 3.456E-04  0.00E+00
 Oxygen
  1.00E-05  2.00E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrate
  1.00E-08  10.2E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrite
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 N2
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Autotrophic Denitrifiers
  2.50E-00  2.50E-03
  5.00E-01  0.00E-03
    0
  8.00E+00  5.00E-01  2.50E-01
  1.00E-02  1.00E-01  1.00E+00
  8.00E+00  1.10E+01
  9.00E+09  9.00E+09
  2.30E+01  2.00E+01 7.419E-01
  9.00E+09
 1.533E+00  5.00E+00 3.043E-01
    0    1
    3
    5    7
   13    7
   49    7
   60
```

```
Test 4 - 2D (1 H2 source)
Reactive Transport - Lab Experiment
   49   25    5    1
  2.55E-02  2.55E-02  2.50E-02
    1    0
  1.00E-02
  600  8.3333E-02
  3.06E-01  1.00E-02  1.00E-02  3.50E-01  1.90E-06
 Hydrogen
  1.00E-04  1.00E-04  1.00E-20
    0    0    1
 3.456E-04  0.00E+00
 Oxygen
  1.00E-05  2.00E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrate
  1.00E-08  10.2E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrite
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 N2
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Autotrophic Denitrifiers
  2.50E-00  2.50E-03
  5.00E-01  0.00E-03
    0
  8.00E+00  5.00E-01  2.50E-01
  1.00E-02  1.00E-01  1.00E+00
  8.00E+00  1.10E+01
  9.00E+09  9.00E+09
  2.30E+01  2.00E+01 7.419E-01
  9.00E+09
 1.533E+00  5.00E+00 3.043E-01
   11    1
 1.230E+00 2.016E+00 8.667E-02
  8.24E-01  3.90E-01 3.300E-01 3.500E-01
 3.965E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 3.965E-04  1.65E-03
    8    8  1.00E+00
    8    9  1.00E+00
```

```
    8   10  1.00E+00
    8   11  1.00E+00
    8   12  1.00E+00
    8   13  1.00E+00
    8   14  1.00E+00
    8   15  1.00E+00
    8   16  1.00E+00
    8   17  1.00E+00
    8   18  1.00E+00
    3
    5    7
   13    7
   49    7
   60
```

```
Test 5 - 2D (3 H2 sources)
Reactive Transport - Lab Experiment
   49   25    5    1
  2.55E-02  2.55E-02  2.50E-02
    1    0
  1.00E-02
  600  8.3333E-02
  3.06E-01  1.00E-02  1.00E-02  3.50E-01  1.90E-06
 Hydrogen
  1.00E-04  1.00E-04  1.00E-20
    0    0    1
 3.456E-04  0.00E+00
 Oxygen
  1.00E-05  2.00E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrate
  1.00E-08   10.2E+00  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Nitrite
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 N2
  1.00E-08  1.00E-08  1.00E-20
    0    0    1
  0.00E+00  0.00E+00
 Autotrophic Denitrifiers
  2.50E-00  2.50E-03
  5.00E-01  0.00E-03
    0
  8.00E+00  5.00E-01  2.50E-01
  1.00E-02  1.00E-01  1.00E+00
  8.00E+00  1.10E+01
  9.00E+09  9.00E+09
  2.30E+01  2.00E+01 7.419E-01
  9.00E+09
 1.533E+00  5.00E+00 3.043E-01
   33    1
 1.230E+00 2.016E+00 8.667E-02
  8.24E-01  3.90E-01 3.300E-01 3.500E-01
 3.965E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 3.965E-04  1.65E-03
 3.965E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
```

```
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 3.965E-04  1.65E-03
 3.965E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 7.931E-04  1.65E-03
 3.965E-04  1.65E-03
    8    8  1.00E+00
    8    9  1.00E+00
    8   10  1.00E+00
    8   11  1.00E+00
    8   12  1.00E+00
    8   13  1.00E+00
    8   14  1.00E+00
    8   15  1.00E+00
    8   16  1.00E+00
    8   17  1.00E+00
    8   18  1.00E+00
   16    8  1.00E+00
   16    9  1.00E+00
   16   10  1.00E+00
   16   11  1.00E+00
   16   12  1.00E+00
   16   13  1.00E+00
   16   14  1.00E+00
   16   15  1.00E+00
   16   16  1.00E+00
   16   17  1.00E+00
   16   18  1.00E+00
   24    8  1.00E+00
   24    9  1.00E+00
   24   10  1.00E+00
   24   11  1.00E+00
   24   12  1.00E+00
   24   13  1.00E+00
   24   14  1.00E+00
   24   15  1.00E+00
   24   16  1.00E+00
   24   17  1.00E+00
   24   18  1.00E+00
    3
    5    7
   13    7
   49    7
   60
```

VITA

Jennifer L. Wright was born in Moorestown, NJ on October 29, 1980. She received her Bachelor of Science degree from Washington and Lee University with a double major in Mathematics and Physics-Engineering in June, 2003. She worked for one year at Lockheed Martin Maritime Systems and Sensors in Moorestown, NJ and participated in the Engineering Leadership Development Program with that company. She then completed her Master of Science degree in Environmental Engineering at the Virginia Polytechnic Institute and State University in Blacksburg, VA in May, 2006. While she was working on her degree, she worked for the Environmental and Water Resources Department as a graduate teaching assistant and research assistant. She is going to work as a groundwater modeler for the Virginia Department of Environmental Quality in Richmond, VA in June, 2006.