2008-03-20

Parallel Processing of Reactive Transport Models Using OpenMP

Jared D. McLaughlin
Brigham Young University - Provo

PARALLEL PROCESSING OF REACTIVE TRANSPORT MODELS USING OPENMP

by

Jared D. McLaughlin

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Civil and Environmental Engineering

Brigham Young University

April 2008

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jared D. McLaughlin

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Norman L. Jones, Chair
Date	T. Prabhakar Clement
Date	E. James Nelson
Date	Gustavious P. Williams
Date	Alan K. Zundel

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jared D. McLaughlin in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Norman L. Jones Chair, Graduate Committee

Accepted for the Department

E. James Nelson
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

PARALLEL PROCESSING OF REACTIVE TRANSPORT MODELS USING OPENMP

Jared D. McLaughlin

Department of Civil and Environmental Engineering

Master of Science

Transport codes are beginning to be parallelized in order to allow more complex add-ons, such as geochemical packages, to utilize finer, more accurate grids, and to reduce solution times making stochastic and Monte Carlo simulations more feasible. Most codes parallelized via MPI (message passing interface) offer good results, but require the development of a new parallel code. OpenMP, the shared-memory standard, offers incremental parallelization, allowing sequential codes to remain relatively intact with minimal changes or additions. OpenMP allows speedup to be seen on personal computers with dual processors or greater, unlike some other parallelization approaches that require a supercomputer. An operator-split strategy creates an environment for easy parallelization by decoupling the transport and reactions of species. The transport, when decoupled from the reactions, is dependent on surrounding nodes and not on species. Therefore, each species transport can be solved on a different processor. The reactions,

when decoupled from the transport, are dependant on the other species concentrations and not on the surrounding nodes, allowing the concentrations for all species to be solve for at a given node as if in a batch reactor. This allows a parallelization of the nodes. Two codes are parallelized in this work. The first is a 100-species 1D theoretical problem. The second is RT3D, a modular computer code for simulating reactive multi-species transport in 3-dimensional groundwater systems written and developed by Dr. T. Prabhakar Clement. RT3D is a sub-component of a parent code, MT3DMS, which utilizes RT3D to solve reaction terms. A speedup factor of 3.91 is seen on four processors, accomplishing a processor efficiency of approximately 98% while spent in RT3D itself.

ACKNOWLEDGMENTS

This research was an effort funded by the Office of Science (BER), U.S. Department of Energy Grant DE-FGO2-06ER64213. A special thanks goes to those responsible for this funding, and for providing an opportunity for further expansion and development in the world of parallel processing in reactive transport modeling.

I would like to thank Dr. T. Prabhakar Clement for his guidance and mentoring as I developed this thesis. He has been a great support and provided me with the opportunity and tools necessary to create parallelized reactive transport code. The time spent at Auburn University enhanced my education greatly.

I would also like to thank my graduate advisor, Dr. Norman L. Jones, for his support and counsel throughout my graduate education and for giving me the idea and opportunity to work on this thesis. His suggestions have been a tremendous help as I completed this paper. Thank you to Dr. E. James Nelson, Dr. Gustavious P. Williams, and Dr. Alan K. Zundel for being a part of my graduate committee and for the knowledge that I have gained from their courses at Brigham Young University.

Finally, I would like to thank my wife, Jeannie, for the support and encouragement she has provided me throughout my schooling and the development of this thesis. She and my son, Jace, always help me remember what is most important in life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction and Background

Supercomputing offers substantial new opportunities and capabilities for reactive transport modeling. Parallelized reactive transport codes can now include geochemical reactions, finer more accurate grids, and faster solution times. The speedup of reactive transport parallel code can be tremendous, cutting down on computational runtime, for those that are willing to pay the price of developing parallel code. Multiprocessing computers, such as dual- and quad-core personal computers are also being placed on the market, offering the speedup of reactive transport code to those without access to a supercomputer. Shared-memory standards in OpenMP (Open Multi-Processing) have simplified the process of developing these parallel codes. RT3D, a modular computer code for simulating reactive multi-species transport in three-dimensional groundwater systems written and developed by Dr. Clement, is a powerful reactive transport model that demonstrates significant speedup through parallelization via OpenMP.

1.1 Objective

The objective of this research is to demonstrate the speedup achievable on shared memory systems implementing OpenMP in reactive transport models. OpenMP allows a parallelization that does not require large supercomputing machines or distributed-memory clusters to achieve speedup, allowing complex reactive transport models to be used on personal computers with multiple processors. This thesis also analyzes the advantages and disadvantages of parallel computing and how OpenMP circumvents some of the disadvantages traditionally faced by message-passing parallel code. Another objective of this thesis is a functional parallel version of the sequential RT3D code developed by Dr. T. Prabhakar Clement. The operator-split (OS) method is also described as a strategy to uncouple transport and reactions in a way to achieve parallelized code. TVD schemes are also outlined as a method to produce a higher-order accurate scheme that reverts back to a lower-order scheme when oscillations would be present, especially as part of the advection equation solved. The speedup and efficiency of processors are calculated from code run time and allow the effectiveness of the parallelized code to be analyzed.

1.2 Computer Architecture

Along with today's technological advances, parallel code has to be developed to take advantage of parallel processors. The manner in which a programmer develops a parallel code depends on the targeted computer architecture. The classes of parallel computer architecture include shared memory, distributed memory, and hybrid systems. Shared memory systems benefit by allowing all processors access to the same memory, eliminating otherwise necessary communication between processors. A Distributed memory system has a high-speed interconnect which transfers data between nodes. Hybrids have also been introduced as well sharing memory at nodes and connecting nodes with high-speed interconnect. Figure 1-1 shows a shared memory system, and Figure 1-2 shows a distributed memory system (Hammond, 2003).

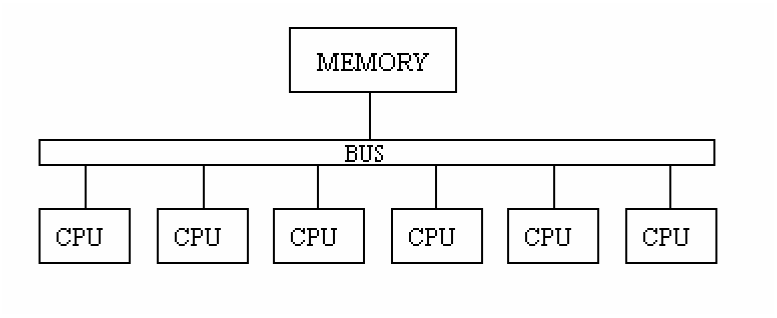

Figure 1-1 Shared Memory System

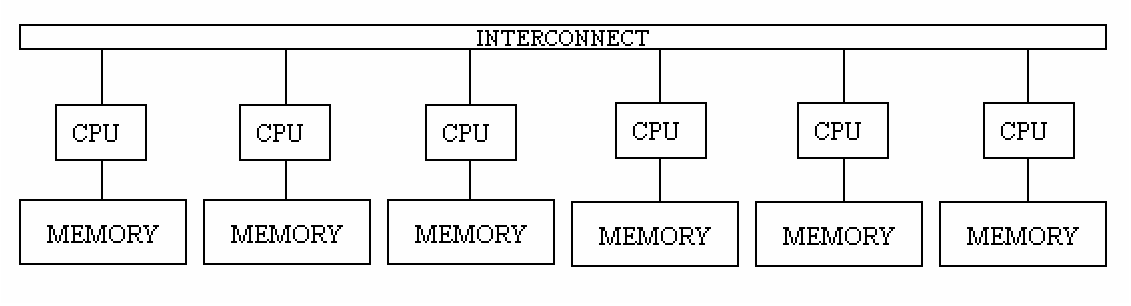

Figure 1-2 Distributed Memory System

Parallel computing is the way of the future. Distributed memory systems keep getting larger and larger. Advancing technology continues to add more and more processors to chips using shared memory. The sequential codes need to be parallelized to take advantage of these advances. Since much has been done using distributed-memory, this paper attempts to demonstrate the alternative, shared-memory approach.

1.3 OpenMP

The shared memory standard is the OpenMP language. The most widely-used distributed-memory language is MPI (Message Passing Interface) (Quinn, 2004). Each standard has advantages and disadvantages. MPI scales well and can be easily placed on a cluster of hundreds of processors, each with its own memory, but message passing requires a unique style of parallel code. Sometimes software developers will go to the trouble of developing two separate codes, one parallel code for distributed-memory systems and another sequential code that can be run on personal computers with shared-memory. The parallel language used depends largely on the target architecture. RT3D is a sequential code designed to be used on personal computers with shared memory systems. OpenMP was a perfect fit for this project in order to keep the RT3D code mostly intact and achieve near linear speedup in the parallel regions.

Some possible disadvantages to parallel computing include parallel hardware availability and affordability, program complexity, code portability, and debugging of parallel code. OpenMP eliminates some of these parallel computing disadvantages. It allows shared memory multiprocessing personal computers to be used as a platform for the parallel code. As a result, parallel hardware becomes available and affordable. One major advantage of OpenMP over MPI is program complexity. OpenMP allows the parallel constructs to be placed right around the sequential loops or regions that are to be parallelized with only minor changes to the code. This allows programs to be parallelized incrementally when more speedup is desired. MPI requires a complete overhaul of the code, by splitting it up, packing the messages to be passed, and structuring the code to receive messages sent between processors. OpenMP is much simpler, leaving sequential legacy code intact (Quinn, 2004). This is one of the most convincing reasons to adopt the shared-memory, or OpenMP, strategy.

Code portability is another issue. OpenMP compilation requires an OpenMP compliant compiler. There are quite a few compliant compilers available. OpenMP has been developed to work on an OpenMP-compliant compiler as well as a normal compiler by hiding the directives in such a way that a normal compiler would see them as comments and neglect their content causing the code to be run in sequential form on one processor. This is a great feature, since it increases code portability with little or no effort (Hermanns, 2002).

Finally, parallel code, whether it be message passing or the shared-memory standard, is difficult to debug. With the simplicity of OpenMP, pinpointing and fixing a problem is typically easier relative to MPI. OpenMP can be turned off or commented out allowing the code to be debugged in sequential form. There are also programs that offer thread debugging, that were not used in this research.

1.4 Scheduling

OpenMP implements multithreading, where a master thread forks and tasks are given to each of the slave threads. At the end of the parallel region, the threads are joined and any synchronization that needs to be done is completed (Van der Pas, 2005). Figure 1-3 below shows how the master thread forks into multiple slave threads that can run across separate processors. The work load is distributing among each.

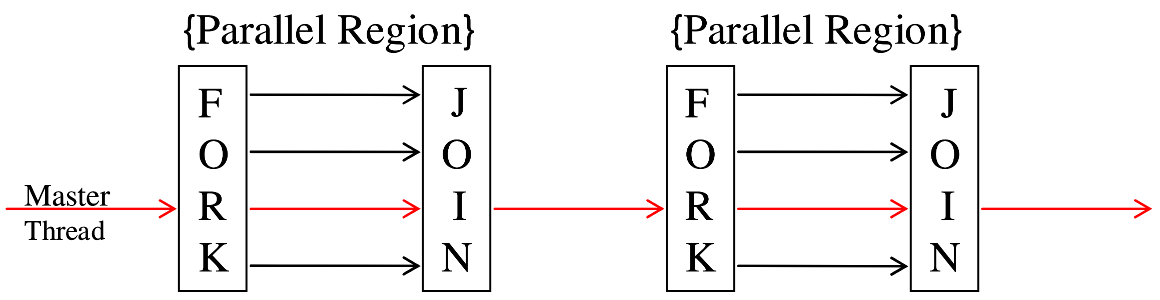

Figure 1-3 Forking and Joining of Threads for Parallel Regions

OpenMP offers fine-grained as well as course grained parallelism. Fine-grained parallelism is defined as each thread doing the same work or the same lines of code as all the other threads but on a different iteration. *Do Loops* in FORTRAN or *For Loops* in C++ are the best examples of fine-grain parallelism. OpenMP parallelizes these loops very efficiently. Course-grained parallelism is defined as each thread doing different work or different lines of code group in sections. Only fine-grained parallelism was exploited in this work.

The scheduling of work loads becomes a major part of the parallelization process. OpenMP offers several types of scheduling options. The first is called *static*. Static scheduling offers the best performance if all the iterations require the same amount of computational time. The iterations are divided equally in the beginning between the threads. For this project, static scheduling was the best choice for solving the advection-dispersion equation in a tri-diagonal solver for each species because the solver takes approximately the same amount of time to compute a solution regardless of the species being solved. The problem arises when iterations are performed to converge on a solution, because one thread might take longer to converge on a solution relative to another thread. This is where *dynamic* scheduling becomes important. Using dynamic scheduling, each thread is given a small amount of work and when it is done it is given more work. This obviously increases the communication overhead. A third option gives the programmer a middle road. This option is called *guided* scheduling. The larger the work loads handed out, the less communication overhead. Guided scheduling hands out large work loads in the beginning, and gives exponentially smaller work loads as the program comes to an end. Guided scheduling was the best choice for the reaction node parallelization since the solution had to be converged upon using an iterative process.

All threads will not finish there work loads at the same time. By default, OpenMP is set to have threads that arrive before other threads wait until all threads of a forked parallel region reach the end of a loop, so that the threads can synchronize and continue. This is important when the iterations of a loop are dependent on the previous iteration of the same loop. If no synchronization is needed a *nowait* clause may be added to the parallel constructs sending the threads that arrive first to go ahead and start on the next iteration. This is the case with RT3D. After applying the nowait clause speedup was increased as well as processor efficiency.

1.5 Performance Analysis

Scalability is how well the parallel code will scale to added processors. Poor scalability is typically due to overhead. Parallel overhead is a function of computer architecture as well as programming algorithms. It is mostly due to the fact that multiple processors have to communicate calculated answers, whereas a single processor does not. Another name for this is called *communication overhead*. OpenMP is considered to have poor scalability, but Brown and Sharapov (2007), in their examples, show that sometimes

OpenMP outperforms its counterpart, MPI (Brown and Sharapov, 2007). Poor scalability in OpenMP could also be due to only portions of the code being parallelized, while in fact inside those parallel regions, the scalability could be significant.

Since it takes time to parallelize a code, it is often good to determine if parallelizing a program will be worthwhile (Quinn, 2004). Amdahl's law predicts speedup for a fixed problem size on the desired number of processors. Amdahl's Law is as follows:

$$\psi \le \frac{1}{f + \frac{(1-f)}{p}} \qquad \textbf{(1-1)}$$

where ψ is the maximum achievable speedup, f is the time to execute the sequential portion of computations, and p is the number of processors. If f were equal to 0, meaning the entire code was parallelized, then the maximum theoretical speedup would be equal to the number of processors. This is called *linear speedup*. Amdahl's Law ignores the overhead associated with parallelism. There comes a point when adding more processors will no longer give a desired increase in speedup. Amdahl's Law was used in the 100-species and large-grid RT3D examples in this project to predict maximum theoretical speedup and validate the actual speedup seen.

Speedup and *efficiency* are used to evaluate the actual performance of the parallelized code (Quinn, 2004). Speedup measures the ratio between sequential and parallel execution time while efficiency measures the processor utilization. Speedup is expressed as

$$Speedup = \frac{sequential\ execution\ time}{parallel\ execution\ time} \tag{1-2}$$

Sequential execution time is defined by the simulation time on one processor. Parallel execution time is defined by the simulation time on *n* processors. The goal is to be as close to linear speedup as possible. Linear speedup in terms of this equation would be achieved when the parallel execution time is half the sequential execution time on 2 processors, a third the sequential execution on 3 processors, and so forth. Linear speedup is considered to be ideal scalability. Poor speedup is primarily due to communication overhead. Efficiency is expressed as

$$Efficiency = \frac{sequential\ execution\ time}{\#\ of\ processors\ used\ *\ parallel\ execution\ time} \tag{1-3}$$

An efficiency of one means all processors are being utilized to full capacity. If the efficiency is less than one, then some of the processors are sitting idle waiting on the other processors to finish work before all processors can move on. Poor efficiency could be due to the type of scheduling chosen, the time required to communicate and synchronize results, or the amount of sequential code inside the parallel region. In order to calculate speedup and efficiency in the example problems, a timer was placed around the entire reactive transport code and a second timer was placed around only the parallel region inside the reactive transport code. From these two timers the sequential execution time and parallel execution times could be measured.

Although linear speedup and efficiencies greater than or equal to 1.0 are the desired goal, sublinear speedup does not necessarily mean failure. If by adding more processors the runtime is decreased sufficiently, the parallelization has still served a purpose. Hammond refers to this as "practical scalability". Efficiencies of 80% to 90% are often considered to be good practical scalability by most parallel programmers (Hammond, 2003).

There are many formulas for analyzing a program's speedup. Amdahl's Law is a widely-used equation to predict speedup for a certain number of processors. Speedup and efficiency equations give an actual evaluation of a program's performance. Other formulas include Gustafson-Barsis's Law and the Karp-Flatt Metric. Gustafson-Barsis's Law is a fixed-time comparison that evaluates performance of an already parallelized program. The Karp-Flatt Metric is used to decide whether a speedup barrier is due to code that cannot be parallelized or parallel overhead (Quinn, 2004). RT3D is a model wrapped by other programs that call it. RT3D was the only code in the system that was parallelized. Amdahl's law worked well in predicting the speedup of the portion of code that was parallelized. The speedup and efficiency equations were used to calculate actual parallel code performance. The other equations were not applied to the examples in this research.

1.6 Existing Parallel Models

Most attempts at parallelizing reactive transport models have involved the use of MPI, or message-passing, to take advantage of large distributed-memory computational power. There are numerous parallel codes, but few are documented and tested thoroughly. Some examples of parallel code in reactive transport modeling are discussed in this section.

1.6.1 HydroBioGeoChem123D or HBGC123D

HBGC123D simulates coupled non-isothermal hydrologic transport and biogeochemical kinetic and/or equilibrium reactions in variably saturated media. It uses a Lagrangian-Eulerian finite element method to solve the transport equations. It uses the Newton-Raphson method to solve the biogeochemical system of equations. It works on multiprocessing shared memory machines using OpenMP directives, while a distributed-memory code is currently under development. HBGC123D was tested on an SGI Origin 2000, SGI multiprocessor Unix workstations, and Cray PVPs. According to Gwo et al. (2001), HBGC123D should also work on similar machines on which OpenMP directives are available. Documented results claim speedup of ~20 on 64 processors for a sample three-dimensional bioremediation problem. This is not a particularly good speedup, but because of the nature of OpenMP, only portions of the code are parallelized leaving the rest of the code in sequential form, greatly simplifying the parallelization process. The biogeochemistry parallelization of this same problem showed speedup of ~49 on 64 processors. To put it in perspective, a problem that took 20 minutes to solve was subsequently solved in under a minute on 64 processors (Gwo et al, 2001).

1.6.2 IPARS-TRCHEM

IPARS, or Integrated Parallel Accurate Reservoir Simulator, is a multiphase flow simulator that was developed at the University of Texas at Austin. It is coupled with TRCHEM which solves the geochemistry. It uses an operator split method to couple the transport and reactions, which offers easy parallelization opportunities. This code was developed for distributed memory. The distributed memory depends on the speed of the interconnect. Wheeler and Peszynska (2002) present results for a black-oil model demonstrating near linear speedup using a fast Myrinet connect at ~14.5 on 16 processors. Speedup of ~12 on 16 processors is seen using an Ethernet interconnect. IPARS demonstrates tremendous scalability. TRCHEM should demonstrate similar speedup due to the nature of being uncoupled with the transport. Those results are not published (Wheeler and Peszynska, 2002).

1.6.3 NUFT

NUFT stands for Nonisothermal Unsaturated-Saturated Flow and Transport. It solves multiphase and multicomponent numerical solutions of non-isothermal flow and transport in porous media. NUFT uses a finite-difference, spatial discretization method to solve the governing equations. The nonlinear equation is solved by the Newton-Raphson method. NUFT does not us an operator-split method. A globally implicit approach is taken, which produces more accurate solutions at the cost of increased computational time. NUFT author Nitao suggests speedup to be somewhere between 200 and 500 on 1000 processors (Nitao, 1998) (Hammond, 2003).

1.6.4 OS3D/GIMRT

OS3D (Operator Splitting 3-Dimensional Reactive Transport) and GIMRT (Global Implicit Multicomponent Reactive Transport) are combined into one software package utilizing shared input files. This allows the user to take advantage of either method. OS3D uses a third-order accurate TVD, or total variation diminishing scheme to produce more accurate results. OS3D was parallelized to work on a distributed memory system (Steefel and Yabusake, 1996).

1.6.5 PARTRAN

PARTRAN is a finite volume flow and biogeochemical transport code. PARTRAN, developed by Glen Hammond for thesis research, uses global implicit and sequential non-iterative approaches. It uses the Newton-Raphson method to solve the reaction and transport. PARTRAN utilizes Portable, Extensible Toolkit for Scientific Computation (PETSc) to implement the parallel algorithms. The library was developed at Argonne National Laboratory. PETSc is a library of parallelized numerical methods. By using this library, sequential code could possibly stay in tact similar to the way OpenMP allows sequential codes to remain in tact. The scalability is excellent for this code. Hammond claims a speedup of ~58 on 64 processors (Hammond, 2003).

1.6.6 PFLOTRAN

PFLOTRAN is a massively parallel reservoir simulator that is also based on the PETSc library. PETSc provides the parallel solvers used to solve the system of nonlinear equations. Domain decomposition is accomplished using these PETSc constructs. When running on the Cray XT3/4 system at Oak Ridge National Laboratory, the code performs linear speedup on up to 2048 processors. This code was designed and developed from the beginning as a parallel simulator (Mills et al, 2007).

2 Numerical Strategy

There are several numerical strategies to solving the transport and reactions in a reactive transport code. The *sequential non-iterative* approach, or "operator splitting", is the simplest and most widely-used approach to couple transport and reaction calculations. Another option is the *global implicit* approach (Hammond, 2003). The global implicit approach is considered to be the most accurate method. It solves a fully-coupled nonlinear system of equations including both transport and reactions in the Jacobian matrix derived for the Newton-Raphson method, requiring extensive memory and computational effort. Since the development of supercomputing, the global implicit approach has become a more feasible solution. However, such an approach requires access to a supercomputer. The operator split strategy is presented as a method to uncouple the transport and reactions allowing for simple parallelization with quick results in any reactive transport code (Hammond, 2003).

2.1 Operator-Split

The operator-split numerical strategy is what makes the simple parallelization developed and demonstrated in the 100-species and large-grid RT3D examples possible. The governing equation is as follows:

$$\frac{\partial C}{\partial t} = -v\frac{\partial C}{\partial x} + D\frac{\partial^2 C}{\partial x^2} + SS + r \qquad \textbf{(2-1)}$$

where C is the concentration of the species [ML^{-3}], v is the pore velocity [LT^{-1}], D is the hydrodynamic dispersion coefficient [L^{2}T^{-1}], SS represents source/sink mixing, and r represents all possible reaction terms. This governing equation is solved for every species at every node of the finite difference grid. The reactions are coupled on a species level and can be solved on a node-by-node basis independent of all other surrounding nodes. The transport portion of the general equation is dependent on the surrounding nodes, but not on the other species. The OS approach offers a way of splitting the reaction terms. This involves dividing the governing equation into four distinct equations:

$$\frac{\partial C}{\partial t} = -v\frac{\partial C}{\partial x} \qquad \textbf{(2-2)}$$

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \qquad \textbf{(2-3)}$$

$$\frac{\partial C}{\partial t} = SS \qquad \textbf{(2-4)}$$

$$\frac{\partial C}{\partial t} = r \qquad \textbf{(2-5)}$$

Equation 2-2 shows the advection equation, equation 2-3 shows the dispersion equation, equation 2-4 shows the source/sink-mixing equation, and equation 2-5 shows the reactions as if they were immobile in a batch reactor. The reactive transport code RT3D applies this OS strategy using transport routines from the EPA code MT3D to solve the advection, dispersion, and source/sink-mixing equations, and then solves the reactions (Clement, 1998).

At larger time steps sizes and faster reaction rates, more error is introduced into the model. This condition, known as "operator splitting error", is the major disadvantage of using the sequential non-iterative approach, but is accepted as a sacrifice worth making to be able to keep the memory and computational effort costs low with the help of TVD schemes. When using this method, attempts should be made to apply small time step sizes (Hammond, 2003).

2.2 TVD Schemes

Since splitting the equations as described above may result in numerical error more extensively presented by Valocchi and Malmstead (1992) and Kaluarachchi and Morshed (1995), small time steps must be taken to minimize the error. The advection equation by itself is a hyperbolic PDE that exhibits numerical oscillation at the advective front when using higher-order accurate schemes (Coray and Koebbe, 1994).

Each of the terms in the governing equation has a unique effect on the actual numerical solution. Dispersion appears to flatten out the solution and is more numerically stable. With low dispersion, the transport becomes dominated by the advection which can cause instability of the solution. The advection term for explicit schemes is stable for $Cr \leq 1$ where Cr is the Courant number defined as $Cr = v\Delta t/\Delta x$. Stability is also affected by the scheme and direction of solving the PDE. The Courant number does apply for implicit methods. Furthermore, higher-order schemes, which are more accurate, introduce more numerical dispersion into the solution. Therefore a higher-order explicit scheme could have oscillations at very low Courant numbers due to dissipation and dispersion errors (Vinh et al, 1992). The Courant number should always be monitored. Ideally the most stable and efficient solution will be produced at a Courant number equal to one. The Courant number should usually stay at one or below. The examples in this section are all produced at a Courant number of 0.2.

The goal is always to achieve more accurate approximations to real solutions. Higher-order schemes produce more accurate approximated solutions. While this is true, Godunov showed that going to second or higher-order schemes necessarily introduces oscillations or instabilities. (Farthing and Miller, 2000). This can be seen in Figure 2-1.

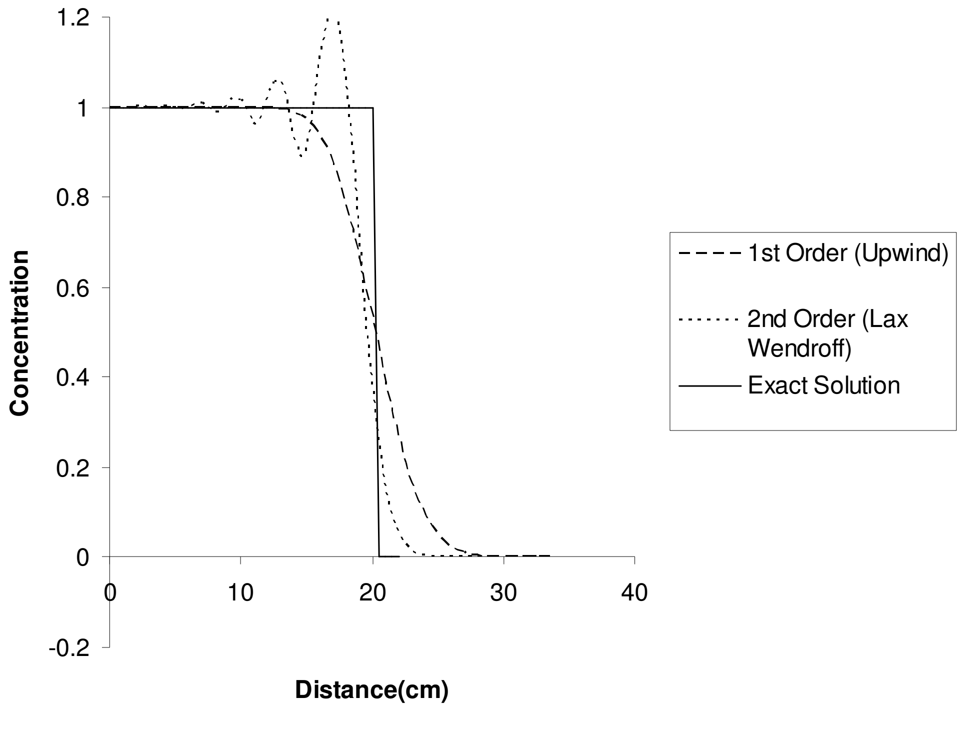

Figure 2-1 Comparison of the 1st Order and 2nd Order Schemes

The slope of the Lax Wendroff scheme better approximates the slope of the exact solution at the advective front with the cost of the oscillations introduced. These oscillations can be prevented using techniques called total-variation-diminishing (TVD) methods. These methods combine the 1st and 2nd order schemes to get a more accurate solution.

There are many higher-order schemes using different stencils. The Lax Wendroff method is second-order accurate in both space and time (Leveque, 2002). This method can be developed using Taylor's Series expansions in the following manner for our advection equation:

Forward Difference Taylor Series Expansion including 2nd Derivative term

$$\frac{\partial C}{\partial t} = \frac{C_i^{n+1} - C_i^n}{\Delta t} - \frac{\partial^2 C}{\partial t^2}\frac{\Delta t^2}{2!} \tag{2-6}$$

2nd Order Central Difference on the Spatial Term of the advection equation

$$\frac{\partial C}{\partial x} = \frac{C_{i+1}^n - C_{i-1}^n}{2\Delta x} \tag{2-7}$$

Plugging equations 2-6 and 2-7 into the advection equation 2-2

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} = -v\frac{C_{i+1}^n - C_{i-1}^n}{2\Delta x} + \frac{\partial^2 C}{\partial t^2}\frac{\Delta t^2}{2!} \tag{2-8}$$

Differentiating the advection equation with respect to t and applying the advection equation for the $\frac{\partial C}{\partial t}$ term:

$$\frac{\partial^2 C}{\partial t^2} = -v\frac{\partial}{\partial t}(\frac{\partial C}{\partial x}) = -v\frac{\partial}{\partial x}(\frac{\partial C}{\partial t}) = -v\frac{\partial}{\partial x}(-v\frac{\partial C}{\partial x}) = v^2\frac{\partial^2 C}{\partial x^2} \tag{2-9}$$

Plugging 2-9 into equation 2-8

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} = -v \frac{C_{i+1}^n - C_{i-1}^n}{2\Delta x} + \frac{v^2 \Delta t^2}{2} \frac{\partial^2 C}{\partial x^2} \tag{2-10}$$

Centered 2nd Difference Term

$$\frac{\partial^2 C}{\partial x^2} = \frac{C_{i+1}^n - 2C_i^n + C_{i-1}^n}{\Delta x^2} \tag{2-11}$$

Plugging 2-11 into equation 2-10 to get the Lax Wendroff Equation

$$C_i^{n+1} = C_i^n - \frac{Cr}{2}(C_{i+1}^n - C_{i-1}^n) + \frac{Cr^2}{2}(C_{i+1}^n - 2C_i^n + C_{i-1}^n) \tag{2-12}$$

Other methods such as Beam & Warming follow the same approach with different stencils.

TVD schemes combine low- and high-order fluxes to get smooth solutions. The higher-order flux is used to provide better resolution. The low-order flux is used when needed to prevent oscillation by summing the flux with an "anti-diffusive" correction term. (Farthing and Miller, 2000). Since the second-order Lax Wendroff method was presented previously, in this example, it will be used as the high-order scheme, while when low-order is needed to limit the oscillations the scheme will revert to an upwind scheme. Rearranging Lax Wendroff as follows:

$$\frac{C_i^{n+1} - C_i^n}{\Delta t} = -\frac{v[C_i^n + \frac{1}{2}(1-Cr)(C_{i+1}^n - C_i^n)] - v[C_{i-1}^n + \frac{1}{2}(1-Cr)(C_i^n - C_{i-1}^n)]}{\Delta x} \tag{2-13}$$

Low-Order Flux

$$F_{i+1/2}^n = vC_i^n \tag{2-14}$$

High-Order Flux

$$F^n_{i+1/2} = v[C^n_i + \frac{1}{2}(1-Cr)(C^n_{i+1} - C^n_i)] \tag{2-15}$$

Equation 2-16 is the advection equation in terms of fluxes

$$\frac{\partial C}{\partial t} = -\frac{F^n_{i+1/2} - F^n_{i-1/2}}{\Delta x} \tag{2-16}$$

The flux limiter, Φ, is added onto the high-order term of the flux

$$F^n_{i+1/2} = v[C^n_i + \frac{1}{2}(1-Cr)(C^n_{i+1} - C^n_i) * \Phi] \tag{2-17}$$

Flux limiters are functions that check for oscillations in order to switch the scheme between high and low resolution. Looking at equation 2-17, if the limiter, Φ, is equal to zero then the flux will revert to low-order and result in a backward difference solution. If the limiter equals unity then the result is a high-order Lax Wendroff Solution. In this manner the scheme can be higher order in the smoother sections of the solution and lower order where needed to prevent oscillations. There are many different types of flux limiters that have been developed (Sweby, 1984). The following are some easily implemented flux-limiters:

Minmod

$$\Phi = \max[0, \min(1, \frac{C^n_i - C^n_{i-1}}{C^n_{i+1} - C^n_i})] \tag{2-18}$$

Superbee

$$\Phi = \max[0, \min(2\frac{C^n_i - C^n_{i-1}}{C^n_{i+1} - C^n_i}, 1), \min(\frac{C^n_i - C^n_{i-1}}{C^n_{i+1} - C^n_i}, 2)] \tag{2-19}$$

Van Leer

$$\Phi = \frac{[4*(C_i^n - C_{i-1}^n)*(C_{i+1}^n - C_i^n) + \Delta x^4]}{[(C_i^n - C_{i-1}^n)^2 + (C_{i+1}^n - C_i^n)^2 + \Delta x^2]^2} \tag{2-20}$$

UMIST

$$\Phi = \max[0, \min(2\frac{C_i^n - C_{i-1}^n}{C_{i+1}^n - C_i^n}, (0.25 + 0.75\frac{C_i^n - C_{i-1}^n}{C_{i+1}^n - C_i^n}), (0.75 + 0.25\frac{C_i^n - C_{i-1}^n}{C_{i+1}^n - C_i^n}), 2)] \tag{2-21}$$

Some limiters are more dissipative and tend to smear discontinuities. An example of this is the Minmod flux limiter. Other limiters are more compressive, such as the Superbee limiter, which sometimes compresses a smooth solution into discontinuity (Wang et al, 2000).

2.3 Examples using TVD Schemes and Flux Limiters

The following examples are for the advection equation with a Courant number of 0.2 solved using the explicit Lax Wendroff scheme. These graphs show the use of different TVD methods. The concentrations have been normalized. Table 2-1 shows each of the parameters for the example simulations in this section. Figure 2-1 to Figure 2-6 show the use of high-order accurate schemes with the implementation of flux limiters.

Table 2-1 Transport Variables for 100-Species Problem

Length (cm)	100
Velocity (cm/day)	1
Δx (cm)	0.5
Δt (days)	0.1
Courant	0.2

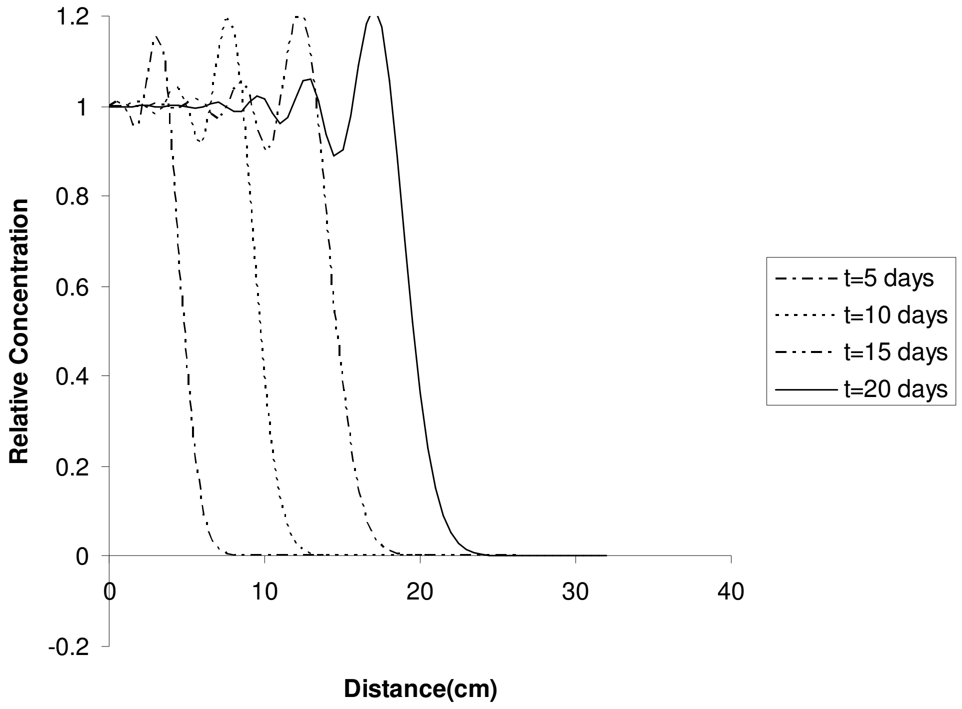

Figure 2-2 Lax Wendroff Method with no flux limiters added

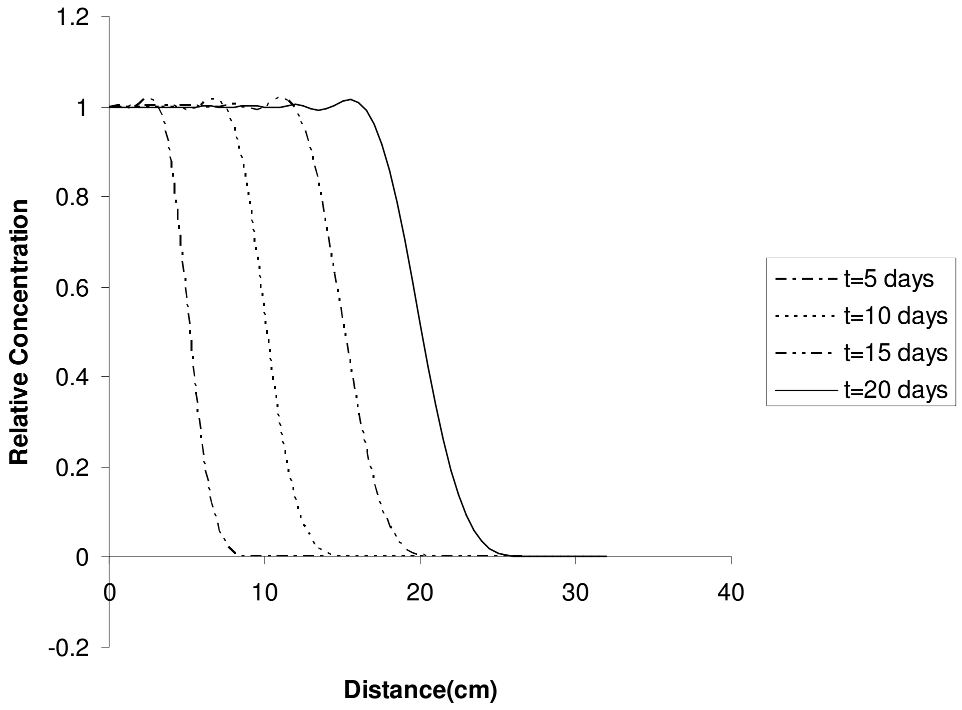

Figure 2-3 Lax Wendroff Method with Van Leer flux limiter

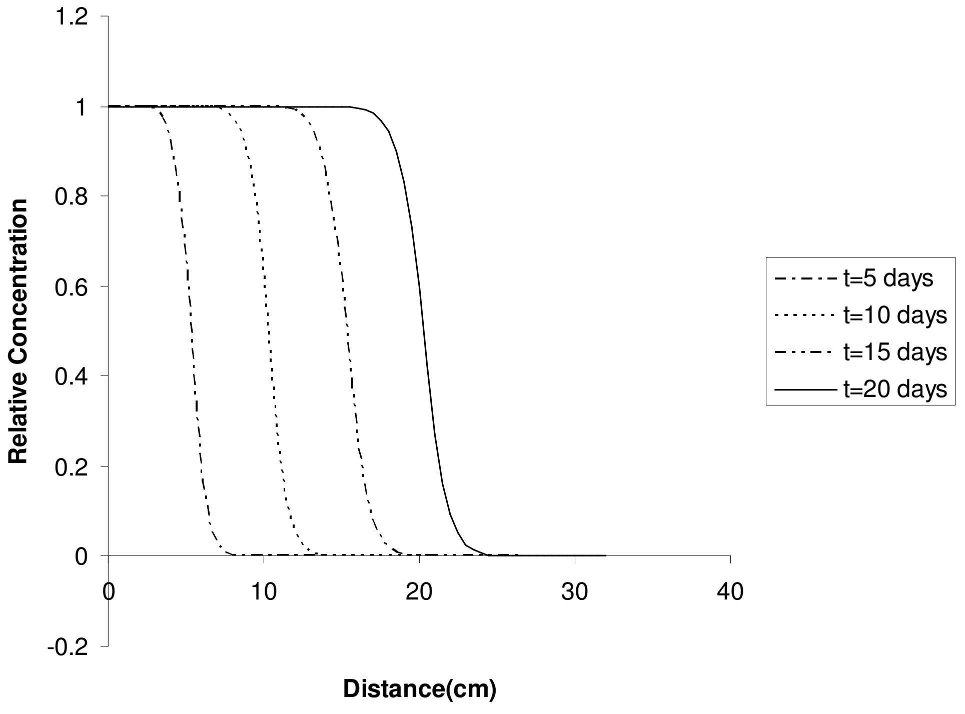

Figure 2-4 Lax Wendroff Method with Minmod flux limiter

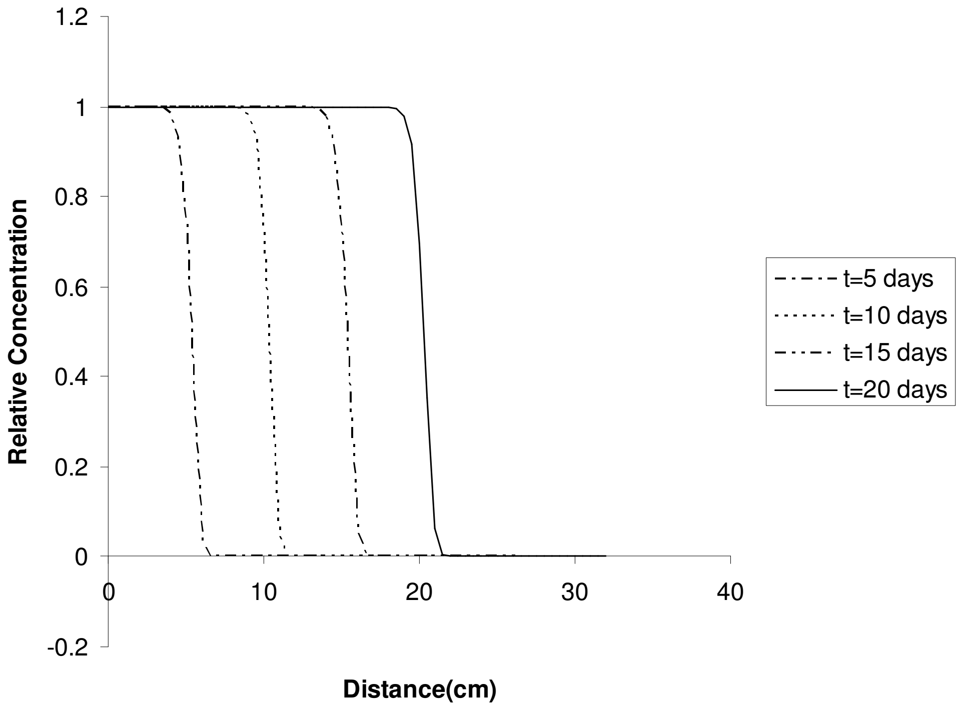

Figure 2-5 Lax Wendroff with Superbee flux limiter

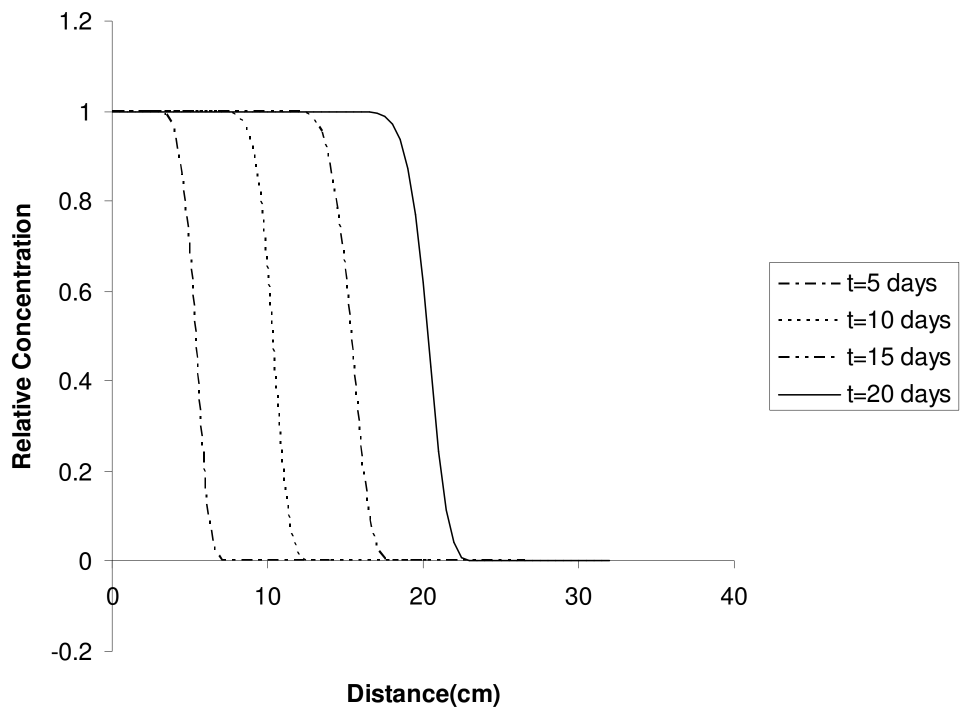

Figure 2-6 Lax Wendroff with UMIST flux limiter

Stability issues arise when using high-order schemes for the advection-dominated solute transport problem. The high-order schemes are desired for the improved accuracy purposes, but at a cost of increased oscillations in the solution principally at the advection front. Adding dispersion to the solution will flatten out these oscillations. The Lax Wendroff scheme is second-order accurate in both space and time. It oscillates as predicted for high-order schemes. TVD methods use flux limiters to check for the oscillations. If an oscillation is found then the TVD method switches the equation back to a first-order backward-difference or upwind scheme. TVD methods should be used to switch between these high and low-resolution schemes, ultimately giving more accurate, stable solutions. These TVD schemes offer improved accuracy to the operator-split strategy make it a feasible alternative to a global implicit strategy.

2.4 OS Offers Simple Uncoupled Parallelization

This operator split strategy allows easy parallel opportunities to be exploited. The first is a simple idea. Since the advection, dispersion, and source/sink-mixing have to be calculated for each species separately, these equations can be solved simultaneously on different processors. An implicit method is used in the 100-species example, using a tridiagonal solver. On a four-processor shared-memory machine the work is split up so each processor solves 25 species in the tri-diagonal solver before moving on to the next time_step and solving them again.

The second parallelization opportunity provided because of the OS strategy is solving the reactions. The time it takes to solve the reactions is large compared to the time required to solve the rest of the equations for each species. The reaction equations are coupled to each other. Since the nodes are not coupled, each processor can iteratively calculate the concentrations of all species at a node simultaneously. The large-grid RT3D example is sized at 31x51x10 giving a total of 15810 cells. On a four-processor shared-memory machine the work could be divided up giving each processor 3952 nodes to calculate the species concentrations at the same time. The decreased time in this case is tremendous and worth the parallelization effort.

3 Multi-Species Means Inherently Parallel (Case Studies)

Two different codes were developed to demonstrate the use of OpenMP in reactive transport modeling. The first parallel code developed was a 100-species problem that uses the operator split method to decouple the transport and reactions. A full OS was not performed on this problem. The advection and dispersion are still solved together. This leads to a more accurate solution than a full OS method. TVD schemes were not used in this problem. A parallelization of the transport on a species by species basis is shown followed by a parallelization of the reactions on a node-by-node basis. The second parallel code developed was a full version of RT3D. The RT3D code is a completely separate code from the 100-species parallel code. The OS strategy was utilized here along with TVD schemes to provide higher resolution and more accurate solutions. Only a parallelization of the reactions on a node-by-node basis is performed in RT3D. Solving the reactions is where most of the time is spent performing calculations. The example shown in this paper is a four-species sequential decay problem. The parallelizations apply to all problems that can be currently solved by RT3D, and are designed to allow further implementation of a geochemical package in the future.

3.1 Advection-Dispersion Species Parallelization

In order to demonstrate speedup of solving the advection-dispersion equation, a 100-species problem was developed using similar parameters to the 10-species example problem developed by Srinivasan and Clement (2008). This 1-D problem was solved using a central-implicit finite difference method. The retardation factors, first-order decay coefficients, source-decay coefficients, yield coefficients, and boundary condition constants are different for each species. Figure 3-1 shows the solution for each of 100 species after being run out to 40 years, demonstrating the complexity of the problem. The transport variables for this problem are seen in Table 3-1.

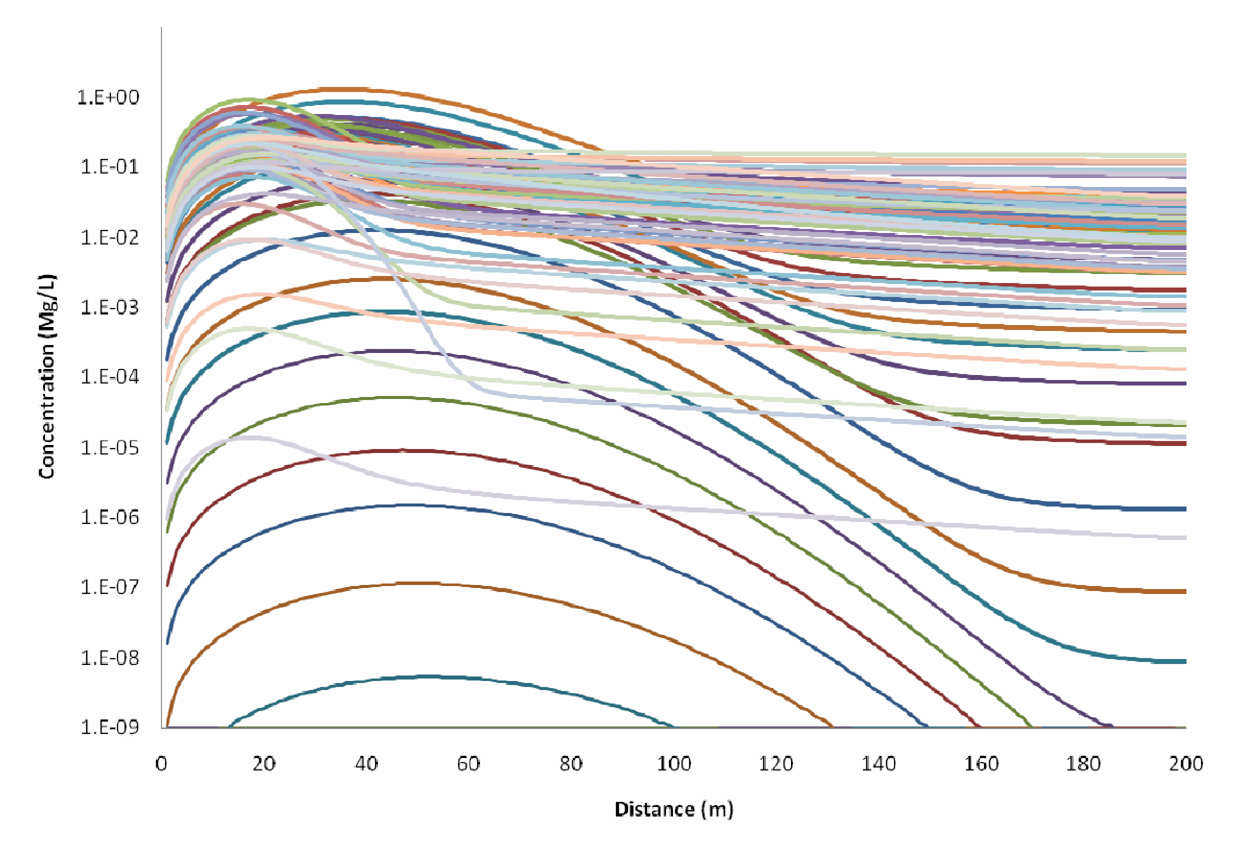

Figure 3-1 100-Species Problem at 40 years

Table 3-1 Transport Variables for 100-Species Problem

Simulation Time (yr)	40
Length (m)	2000
Velocity (m/yr)	5
Δx	1
Δt	0.1
Dispersion; Dx (m^2/yr)	50
Courant	0.5
Peclet	0.1

The general equation for this example excluding the reactions is as follows:

$$\frac{\partial C}{\partial t} = -v\frac{\partial C}{\partial x} + D\frac{\partial^2 C}{\partial x^2} \tag{3-1}$$

A fully-implicit approach involving the application of a truncated Taylor Series transforms this equation to:

$$\frac{C_i^{l+1} - C_i^l}{\Delta t} = D\left(\frac{C_{i-1}^l - 2C_i^l + C_{i+1}^l}{\Delta x^2}\right) - v\left(\frac{C_{i+1}^l - C_{i-1}^l}{2 * \Delta x}\right) \tag{3-2}$$

where the superscript is the time_step and the subscript is the node location. This equation is second-order accurate in space. Implementation of a high-order accurate scheme would also work as the OpenMP parallel constructions only wrap around the sequential code for this governing equation. In a matrix and using a tri-diagonal solver this equation is solved at every node for one species at a given time_step. The parallelization comes into play here as at each time_step we solve this same matrix for four different species on four different processors at the same time. It does not take very long to solve these matrices in the solver, but if four species could be solved at once the time required to solve this section of the code is cut by almost a factor of four.

The speedup and processor efficiency for this section of the code can be seen in Table 3-2. The results are from a desktop machine with two Dual Core processors (a total of four processors), 2.0 GHz clockspeed, and 4GB of Ram. *Program Run Time* was the time it took to run the entire program, whereas *Adv-Disp Run Time* is the time spent only in the parallel region of the code that calculates the advection-dispersion equation for each species. This is a static scheduling example, and guided scheduling did not yield similar results showing a speedup of only ~2.4 on 4 processors for the Adv-Disp portion of the code. The bigger the load distributed, the better the speedup in regions of the code where the processors spend about the same amount of time calculating the solution.

Table 3-2 Speedup and Efficiency for Transport of 100-Species

Number of Processors	**1**	**2**	**3**	**4**
Program Run Time	172.78	153.45	148.95	145.24
Program Speedup		1.1259	1.16	1.1896
Efficiency		0.563	0.3867	0.2974
Adv-Disp Run Time	35.341	18.915	14.415	10.702
Adv-Disp Speedup		1.8684	2.4517	3.3024
Efficiency		0.9342	0.8172	0.8256
Time Spent in Adv-Disp	20.45%	12.33%	9.68%	7.37%

From this table, the first thing to notice is the percentage of the program that is being parallelized which corresponds to about 35 seconds of the total 173 second run time, or

20.45% of the program. Looking at the speedup and efficiency, this is an example of practical speedup, since this problem will not scale well to a large number of processors. On two processors however, the parallelization cuts 16.5 seconds off the processing run time, and on four processors the parallelization cut almost 25 seconds off. Looking at the efficiency for all of the processors, it appears to be diminishing rapidly from two to three processors. Although this particular problem appears to fit better on four processors rather than on three, the efficiency is expected to continue to diminish at the previous rate. The lower efficiency represents processors that are sitting idle while others are working. However, this is about the most one can expect for this portion of the code though.

3.2 Reaction Node Parallelization

Most of the solution time is spent on the reactions as it is an iterative process. The equations for the coupled reactions are as follows:

$$\frac{\partial C_1}{\partial t} = -k_1 C_1 \tag{3-3}$$

$$\frac{\partial C_2}{\partial t} = k_1 C_1 - k_2 C_2 \tag{3-4}$$

$$\frac{\partial C_3}{\partial t} = k_2 C_2 - k_3 C_3 \tag{3-5}$$

…to 100 species

Since they are coupled, they are all solved at the same time at a given node. In this problem there are 2001 nodes. The nodes can be divided and solved for individually because they are independent of the surrounding nodes. Since it is an iterative process it does not make sense to use a static schedule. A guided schedule gives much more desirable results, given the processors larger work loads in the beginning and smaller work loads towards the end. Amdahl's law can be used to determine a maximum theoretical speedup as shown in Table 3-3.

Table 3-3 Max Theoretical Speedup using Amdahl's Law (100-Species)

Number of Processors	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Program Run Time	172.78	87.84	59.53	45.37	36.88	31.21	27.17	24.13	21.77	19.89	18.34	17.06	15.97	15.03	14.22	13.52
Program Speedup	1	1.967	2.903	3.808	4.685	5.536	6.36	7.159	7.935	8.688	9.419	10.13	10.82	11.49	12.15	12.78
Reaction Run Time	134.54	67.27	44.85	33.63	26.91	22.42	19.22	16.82	14.95	13.45	12.23	11.21	10.35	9.61	8.969	8.409
Reaction Speedup	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Adv-Disp Runtime	35.341	17.67	11.78	8.835	7.068	5.89	5.049	4.418	3.927	3.534	3.213	2.945	2.719	2.524	2.356	2.209
Adv-Disp Speedup	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

It can be seen that on 16 processors a speedup of 12.78 is the theoretical maximum which would cut our runtime from 172.78 seconds to 13.5 seconds. Again this does not take into account overhead associated with running in parallel. With eight processors the runtime is down it 24 seconds.

Amdahl's law did not take into account the overhead associated with running in parallel. The actual results are not quite as good as the theoretical. The actual results can be seen in Table 3-4. The reaction runtime in this table is the time spent in the parallel region of the code calculating the reactions for all species at all nodes in parallel.

Table 3-4 Actual Speedup with Parallelized Transport and Reactions

Number of Processors	**1**	**2**	**3**	**4**
Program Run Time	172.78	88.079	61.758	48.336
Program Speedup		1.9616	2.7977	3.5745
Efficiency		0.9808	0.9326	0.8936
Reaction Run Time	134.54	66.125	44.338	34.313
Reaction Speedup		2.0346	3.0343	3.9208
Efficiency		1.0173	1.0114	0.9802
Adv-Disp Run time	35.341	18.849	14.266	10.735
Adv-Disp Speedup		1.8749	2.4774	3.292
Efficiency		0.9375	0.8258	0.823
Time Spent in Reactions	77.87%	75.07%	71.79%	70.99%
Time Spent in Adv-Disp	20.45%	21.40%	23.10%	22.21%

The time spent in the two parallel sections is 170 of the total 173 seconds, which is about 98% of the program. One thing to focus on in Table 3-4 is the efficiency of the reaction speedup. On two and three processors it is better than linear. How is this possible? Two explanations exist: The first is that the experiment was not done correctly and the data are wrong. The second is a super-linear speedup cache effect. This means that when the program ran on one processor, the cache did not contain the entire problem. When another processor is added, the problem fits in the cache available to the two processors. The time to retrieve the rest of the problem is cut out (Gustafson, 2007).

The actual program efficiency includes the communication overhead and as a result is much lower than the theoretical efficiency for all processors. The efficiency is very close to 1.0 on all four processors. This leads to very scalable results on a higher number of processors. The parallelization of the reactions region of the code is warranted by the great results. Figure 1-1 shows the speedup, while Figure 3-3 shows the run time.

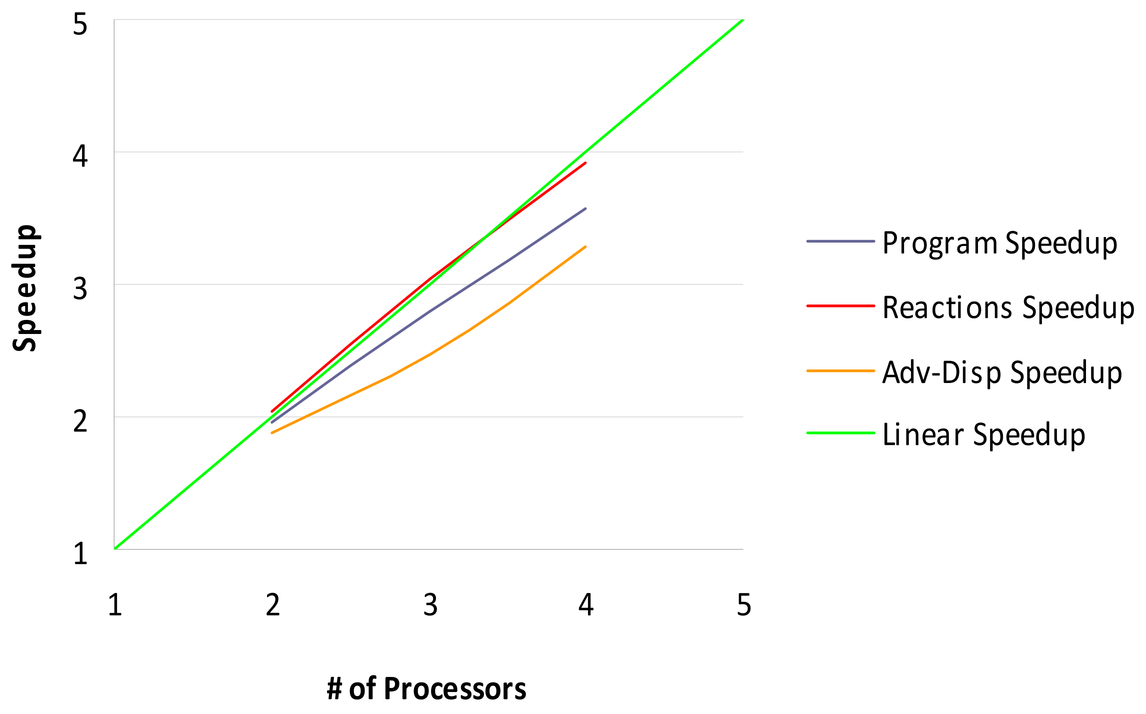

Figure 3-2 Speedup of 100-Species Problem

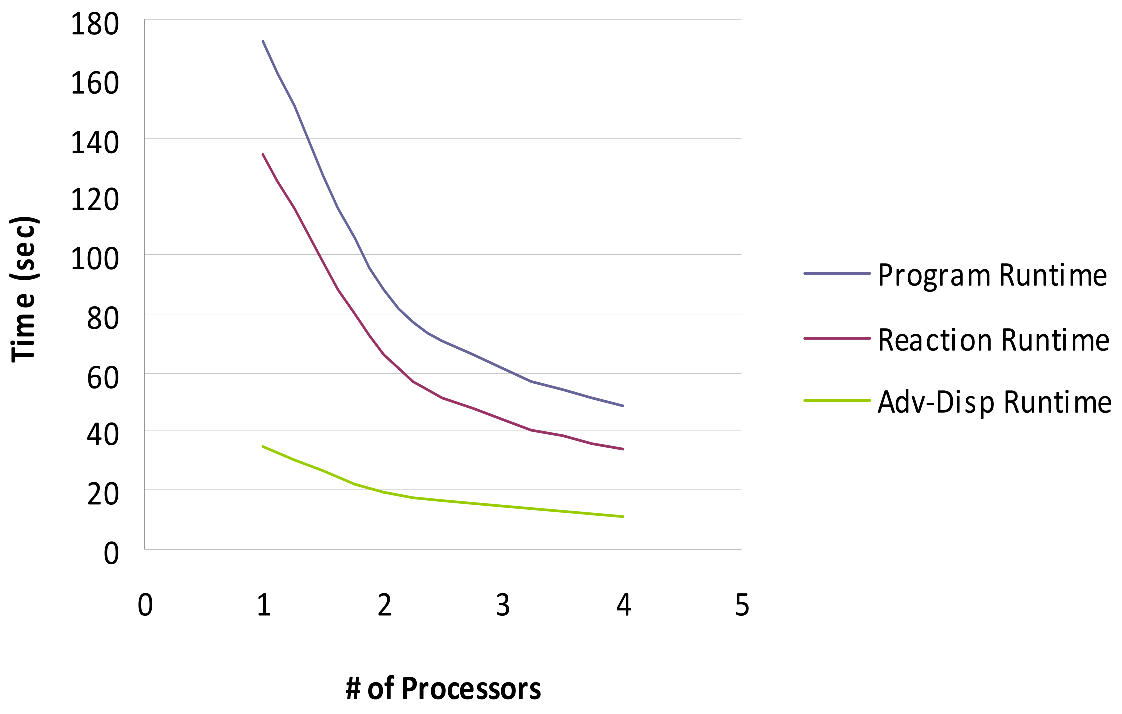

Figure 3-3 Run Times of 100-Species Problem

The program runtime matches the reaction runtime since most of the time is spent solving the reactions anyway.

3.3 RT3D Node Parallelization

It turns out that the species parallelization of the advection-dispersion equation is not always warranted and 100-species is truly hypothetical. A parallelization of the nodes is much more practical and gives the desired computational time decreases sought after in reactive transport numerical simulations. An RT3D tutorial solving sequential decay reactions was selected as a case study example to demonstrate the speedup of parallelizing the nodes (EMRL, 2006). The problem models de-chlorination of PCE and its daughter products under anaerobic conditions. The partial differential equations including advection, dispersion, and source/sink-mixing are as follows:

$$R_A \frac{\partial [A]}{\partial t} = \frac{\partial}{\partial x_i}\left(D_{ij} \frac{\partial [A]}{\partial x_j} \right) \frac{\partial (v_i [A])}{\partial x_i} + \frac{q_s}{\phi}[A]_s - K_A [A] \tag{3-6}$$

$$R_B \frac{\partial [B]}{\partial t} = \frac{\partial}{\partial x_i}\left(D_{ij} \frac{\partial [B]}{\partial x_j} \right) \frac{\partial (v_i [B])}{\partial x_i} + \frac{q_s}{\phi}[B]_s - K_B [B] + Y_{B/A} K_A [A] \tag{3-7}$$

$$R_C \frac{\partial [C]}{\partial t} = \frac{\partial}{\partial x_i}\left(D_{ij} \frac{\partial [C]}{\partial x_j} \right) \frac{\partial (v_i [C])}{\partial x_i} + \frac{q_s}{\phi}[C]_s - K_C [C] + Y_{C/B} K_B [B] \tag{3-8}$$

$$R_D \frac{\partial [D]}{\partial t} = \frac{\partial}{\partial x_i}\left(D_{ij} \frac{\partial [D]}{\partial x_j} \right) \frac{\partial (v_i [D])}{\partial x_i} + \frac{q_s}{\phi}[D]_s - K_D [D] + Y_{D/C} K_C [C] \tag{3-9}$$

where [A], [B], [C], and [D] represent specie concentrations, Y represents stoichiometric yield coefficients, K represents decay coefficients, R represents retardation, D represents the hydrodynamic dispersion coefficient, v represents pore velocity, and $\frac{q_s}{\phi}$ represents source/sink-mixing. Using the operator-split method and allowing MT3D to solve the advection, dispersion, and source/sink-mixing leaves the reactions to be solved by RT3D. These reaction equations are:

$$\frac{d[A]}{dt} = -\frac{K_A [A]}{R_A} \tag{3-10}$$

$$\frac{d[B]}{dt} = \frac{Y_{B/A} K_A [A] - K_B [B]}{R_B} \tag{3-11}$$

$$\frac{d[C]}{dt} = \frac{Y_{C/B} K_B [B] - K_C [C]}{R_C} \tag{3-12}$$

$$\frac{d[D]}{dt} = \frac{Y_{D/C} K_C [C] - K_D [D]}{R_D} \tag{3-13}$$

These equations are coupled and are solved simultaneously at any given node.

The original tutorial is a grid sized at 31x51x1, giving a total of 1582 nodes. The problem size was increased to a grid size of 31x51x10, increasing the number of nodes to 15810. Since the reaction equations are solved iteratively at each of these nodes, but no two nodes are dependent on one another at a given time step, the nodes can be solved simultaneously. The results are from the same desktop machine as was used with the 100-species example with two Dual Core processors (a total of four processors), 2.0 GHz clockspeed, and 4GB of Ram. Guided scheduling is clearly the appropriate strategy for this parallel region as the time for solution convergence at each node is unknown beforehand. Table 3-5 below shows the resulting speedup and processor efficiency:

Table 3-5 RT3D Speedup and Efficiency for Sequential Decay Problem

Number of Processors	**1**	**2**	**3**	**4**
Program Run Time	394.36	278.55	241.37	223.85
Program Speedup	1	1.4157	1.6338	1.7617
Efficiency		0.7079	0.5446	0.4404
RT3D Run Time	231.29	116.19	77.77	59.083
RT3D Speedup	1	1.9907	2.974	3.9147
Efficiency		0.9953	0.9913	0.9787
Time Spent in RT3D	58.65%	41.71%	32.22%	26.39%

The processor efficiency is very high for the parallel region for each of the four cases representing different numbers of processors. The speedup for the parallel region is nearly linear. The communication overhead appears to be quite small. It is true that only about 60% of the entire program was parallelizable, but the results for that section are noteworthy. Figure 3-4 shows the runtime for this problem, while Figure 3-5 shows the corresponding speedup.

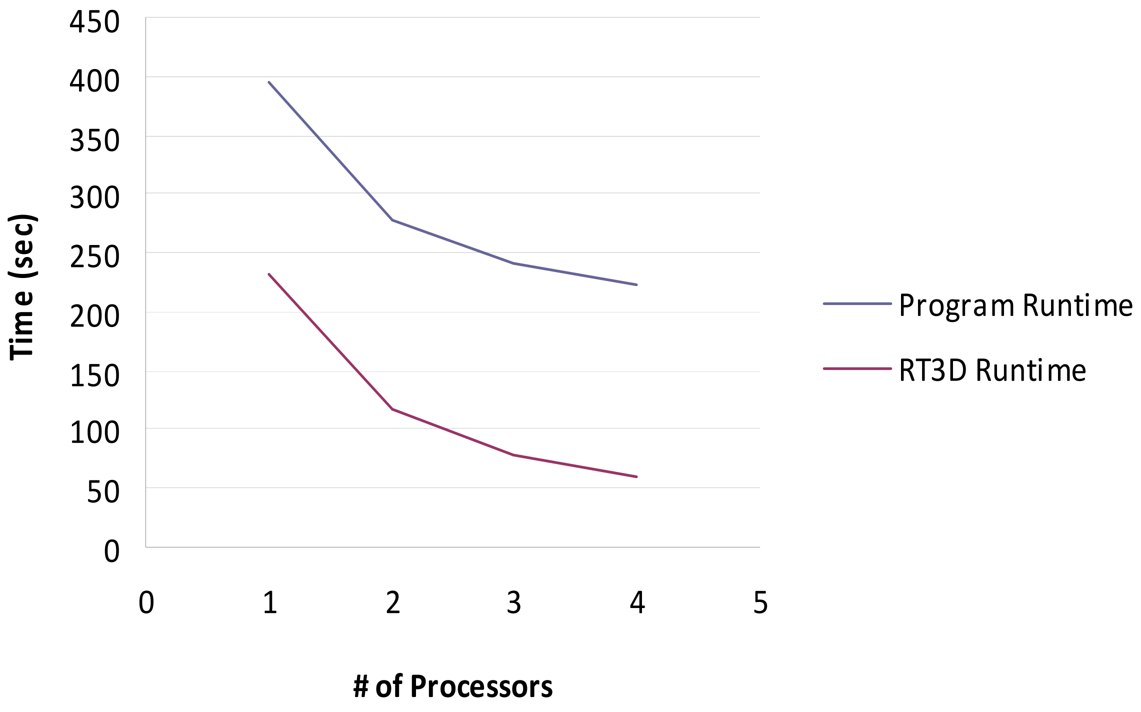

Figure 3-4 Runtimes for Sequential Decay Large-Grid Problem

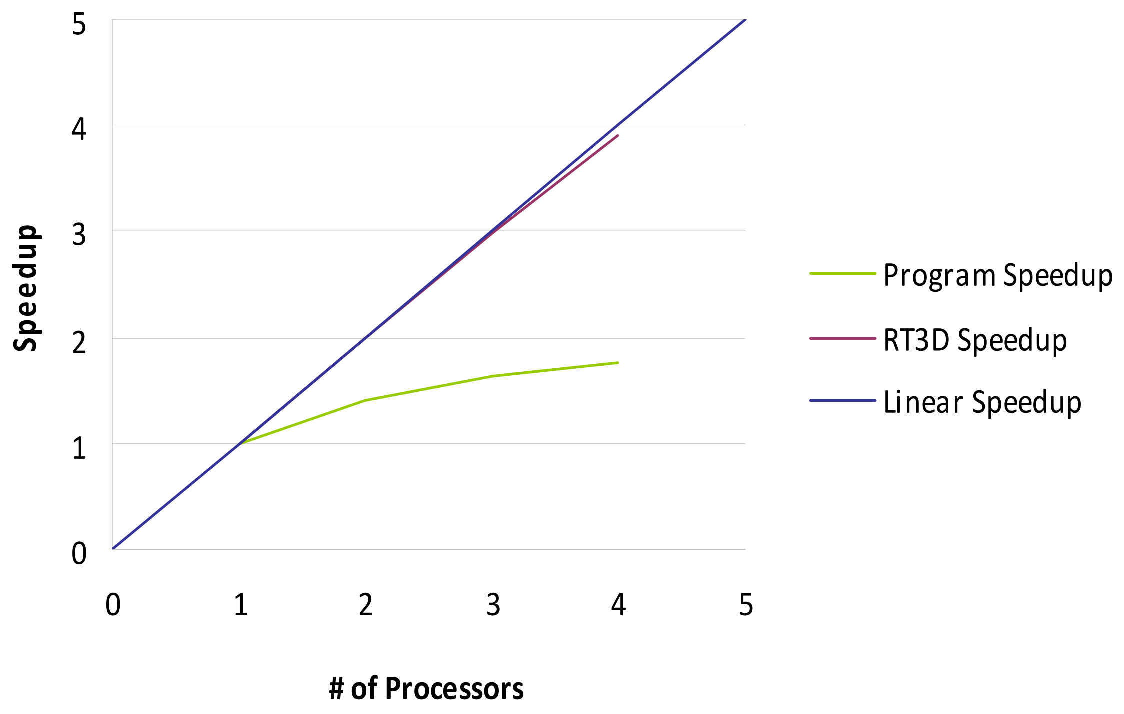

Figure 3-5 Speedup for Sequential Decay Large-Grid Problem

Using Amdahl's Law to predict the speedup on any number of processors, the maximum theoretical speedup for up to 16 processors is shown in Table 3-6 below. There comes a point when adding another processor does not reduce run time enough to make it worth the cost. However, this is a function of problem size. The parallelization of RT3D allows a substantially increased problem size to become feasible. The table shows inside of the RT3D parallel region a theoretical run time of 14.46 seconds on 16 processors which is reduced from 231.3 seconds on 1 processor. These run times are not what would actually be seen on *n* processors, again due to the overhead of parallel processing, but with high efficiencies and near linear speedup the times should be close.

Table 3-6 Max Theoretical Speedup using Amdahl's Law (Sequential Decay)

Number of Processors	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Program Run Time	394.4	278.7	240.2	220.9	209.3	201.6	196.1	192
Program Speedup	1	1.415	1.642	1.785	1.884	1.956	2.011	2.054
RT3D Run Time	231.3	115.6	77.1	57.82	46.26	38.55	33.04	28.91
RT3D Speedup	1	2	3	4	5	6	7	8

Number of Processors	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Program Run Time	188.8	186.2	184.1	182.3	180.9	179.6	178.5	177.5
Program Speedup	2.089	2.118	2.142	2.163	2.18	2.196	2.209	2.221
RT3D Run Time	25.7	23.13	21.03	19.27	17.79	16.52	15.42	14.46
RT3D Speedup	9	10	11	12	13	14	15	16

4 Conclusion

This thesis outlines the development of a new parallelized version of the reactive transport code, RT3D. The new code offers a shared-memory system parallelization using OpenMP which allows users to run RT3D from a personal computer producing faster simulation run times on two or more processors. A supercomputer with a message-passing parallelized code is not needed to achieve speedup. The thesis also shows the results from a separate reactive transport code, developed specifically for this research, demonstrating parallelization of 100-species in transport as well as a parallelization of 100-species sequential decay coupled reactions.

Several approaches have been taken to uncouple transport and reactions when solving the governing equation. The approach taken in RT3D is the operator-split method. This method allows the advection, dispersion, source/sink mixing, and reactions to be solved separately from each other, and then combined in the end. Since the transport and reactions are no longer coupled, they can be treated separately. The transport can be parallelized on a species-by-species basis because each species is independent of other species in these equations. The species concentrations solved in the reaction equations are independent of the species concentrations at all surrounding nodes and therefore a node-by-node parallelization can be achieved. At each node, all species can be solved for as if they were in a batch reactor. The parallelization of the species and the parallelization of the nodes were demonstrated in two codes in this research.

The first case was the 100-species 1D example code developed solely for this thesis. OpenMP constructs were placed around the advection-dispersion equation demonstrating a speedup in this section alone of 3.3 on four processors which equals about 83% efficiency. This amounts to a decrease in runtime of approximately 25 seconds for solving the advection-dispersion for these species. Most of the program time is spent iterating in a solver for the reactions. Including a parallelization of the reactions for this same problem demonstrates a speedup of 3.9 on four processors or around 98% efficiency. The entire program runtime was decreased by parallelizing the transport and reactions from 173 seconds to 48 seconds.

The second parallel code that was developed was the reactive transport code RT3D. The advection, dispersion, and source/sink mixing are solved by MT3D code and then RT3D is called to solve the reactions as if in a batch reactor. Only RT3D was parallelized. A lot of time is spent solving the reactions, causing a parallelization to be very beneficial. A speedup of 3.91 was seen in this example on four processors producing an efficiency of 98% for the reactions portion of the code alone for the sample problem chosen in this thesis. This cuts the computational time spent inside RT3D from 231 seconds down to 59 seconds on four processors. Further speedup of the entire code could possibly be achieved if portions of MT3D where also parallelized. The parallelization works for all chemical reaction packages currently supported by RT3D, all of which produce similar results.

Improvements to the RT3D will continue to be made. The parallelized code was developed in such a way that new package implementation is simple to add to the RT3D code. This paves the way for a geochemical package to be added in the future. As technology continues to grow, parallelized reactive transport codes will continue to be developed. OpenMP is the shared-memory standard. More processors are being placed around shared-memory. This offers tremendous speedup capabilities without message-passing. Reactive transport codes can now begin to add more complexity such as geochemical packages, finer more accurate grids, and decreased simulation run time with parallel computing power.

References

Arbogast, T. and Wheeler, M.F. “A Parallel Numerical Model for Subsurface Contaminant Transport with Biodegradation” (1994).

Brown, R. and Sharapov, I. “High-Scalability Parallelization of a Molecular Modeling Application: Performance and Productivity Comparison Between OpenMP and MPI Implementations.” *Int J Parallel Prog* no. 35 (2007):441-458.

Clement, T.P. “A Modular Computer Code for Stimulating Reactive Multispecies Transport in 3-Dimensional Groundwater Systems.” *Pacific Northwest National Laboratory* (1998): 7-10.

Clement, T.P., Sun, Y., Hooker, B.S. and Petersen, J.N. “Modeling Multispecies Reactive Transport in Ground Water.” *GWMR* (1998): 79-92

Coray, C. and Koebbe, J. “High Order Accuracy Optimized Methods for Constrained Numerical Solutions of Hyperbolic Conservation Laws.” *Society for Industrial and Applied Mathematics* 15, no. 4 (1994): 846-865.

Dere, Y. and Sotelino, E.D. “Domain-by-Domain Algorithm for Nonlinear Finite-Element Analysis of Structures.” *Journal of Computing in Civil Engineering* 22, no. 1 (2008): 58-67.

EMRL, Environmental Modeling Research Laboratory. *Groundwater Modeling System: Tutorials*. Brigham Young University, (2006): 4-7.

Falcone, M. and Ferretti, R. “Convergence Analysis for a Class of High-Order Semi-Lagrangian Advections Schemes.” *Society for Industrial and Applied Mathematics* 35, no. 3 (1998): 909-940).

Farthing, M.W. and Miller, C.T. “A Comparison of High-Resolution, Finite-Volume, Adaptive-Stencil Schemes for Simulation Advective-Dispersive Transport.” *Advances in Water Resources* no. 24 (2000): 29-48.

Gustafson, J.L. "Fixed Time, Tiered Memory, and Superlinear Speedup." *Iowa State University: Ames Laboratory, Department of Energy.* Database on-line. Accessed 15 May 2007: 1-9.

Gwo, J.P., D'Azevedo, E.F., Frenzel, H., Mayes, M., Yeh, G.T., Jardine, P.M., Salvage, K.M., Hoffman, F.M. "HBGC123D: a high-performance computer model of coupled hydrogeological and biogeochemical processes." *Computers & Geosciences* no. 27 (2001): 1231-1242.

Hammond, G.E. "Innovative Methods for Solving Multicomponent Biogeochemical Groundwater Transport on Supercomputers." (2003): 10-61.

Hammond, G.E., Lichtner, P. and Lu, C. "Subsurface Multiphase Flow and Multicomponent Reactive Transport Modeling using High-Performance Computing." *Journal of Physics: Conference Series* no.78 (2007)

Hammond, G.E., Valocchi, A.J. and Lichtner, P.C. "Modeling Multicomponent Reactive Transport on Parallel Computers Using Jacobian-Free Newton Krylov with Operator-Split Preconditioning."

Hermanns, M. "Parallel Programming in Fortran 95 using OpenMP." *Universidad Politècnica de Madrid: School of Aeronautical Engineering* (2002): 4-52.

Leveque, R.J. "Finite Volume Methods for Hyperbolic Problems." New York: Cambridge University Press, 2002: 1682-1685.

Mey, D., Sarholz, S. and Terboven C. "Nested Parallelization with OpenMP." *Int J Parallel Prog* no.35 (2007): 459-476.

Mills, R.T., Lu, C., Lichtner, P.C., and Hammond, G.E. "Stimulating Subsurface Flow and Transport on Ultrascale Computers using PFLOTRAN." *Journal of Physics: Conference Series* no. 78 (2007): 1-7.

Nitao, J.J. "Reference Manual for the NUFT Flow and Transport Code, Version 2.0." *Lawrence Livermore National Laboratory* (1998): 1.

OpenMP. "OpenMP." Available from http://www.openmp.org/drupal/. Internet; accessed 17 July 2007.

Ortega, J.M. and Voigt, R.G. "Solution of Partial Differential Equations on Vector and Parallel Computers." *SLAM Review* 27, no. 2 (1985): 149-240.

Quinn, M.J. *Parallel Programming in C with MPI and OpenMP.* New York: McGraw Hill (2004): 159-170, 404-435.

Shu, C. (1988, November). Total-Variation-Diminishing Time Discretizations. *Society for Industrial and Applied Mathematics*, 9(6), 1073-1084.

Srinivasan, V. and Clement, T.P. “Analytical solutions for sequentially couple one-dimensional reactive transport problems – Part I: Mathematical derivations.” *Advances in Water Resources* no. 31 (2008): 203-218.

Srinivasan, V. and Clement, T.P. “Analytical solutions for sequentially couple one-dimensional reactive transport problems – Part II: Special cases, implementation and testing.” *Advances in Water Resources* no. 31 (2008): 219-232.

Steefel, C.I. and Yabusake, S.B. “Software for Modeling Multicomponent-Multidimensional Reactive Transport: User Manual & Programmer’s Guide.” *Pacific Northwest National Laboratory,* version 1.0 (1996).

Sweby, P.K. “High Resolution Schemes Using Flux Limiters for Hyperbolic Conservations Laws.” *Society for Industrial and Applied Mathematics* 21, no. 5 (1984): 995-1011.

Van der Pas, R. “An Introduction into OpenMP.” *Presented at the University of Oregon* (2005): 37-40.

Vinh, H., Dwyer, H.A., and van Dam, C.P. *Finite-Difference Methods for Computational Electromagnetics* (CEM), (1992): 1682-1685.

Wang, J.S., Ni, H.G., and He, Y.S. “Finite-Difference TVD Scheme for Computation of Dam-Break Problems.” *Journal of Hydraulic Engineering* 126, no. 4 (2000): 253-262.

Watson, I.A., Crouch, R.S., Bastian, P. and Oswald, S.E. “Advantages of using adaptive remeshing and parallel processing for modeling biodegradation in groundwater.” *Advances in Water Resources* no. 28 (2005): 1143-1158.

Wheeler, M.R. and Peszynska, M. “Computational engineering and science methodologies for modeling and simulation of subsurface applications.” *Advances in Water Resources* no. 25 (2002): 1147-1173.

Zheng, C. and Wang, P.P. “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.” *US Army Corps of Engineers: Engineer Research and Development Center* (1999).

Appendix A. RT3D Version 3.0 Format and Functionality

The structure of the new RT3D version 3.0 is completely different from the structure of the old RT3D. The reasoning behind this is to make the program easier to update when newer versions of MT3D come out, as well as the implementation of OpenMP and parallel code. The new RT3D uses all the same files as MT3D, and adds four more files that solve the desired reactions. In the current version of MT3D, the only changes that exist are in the mt3dms5 file where the RT3D code is added in a number of sections, and no code is deleted. The input file for MT3D is a .rct file and is read by MT3D itself, while a .rtr file is read and used by the RT3D subroutines. This appendix attempts to document all changes compared to the previous version of RT3D.
shows the flow from MT3D to the four added RT3D files and each of there subroutines.

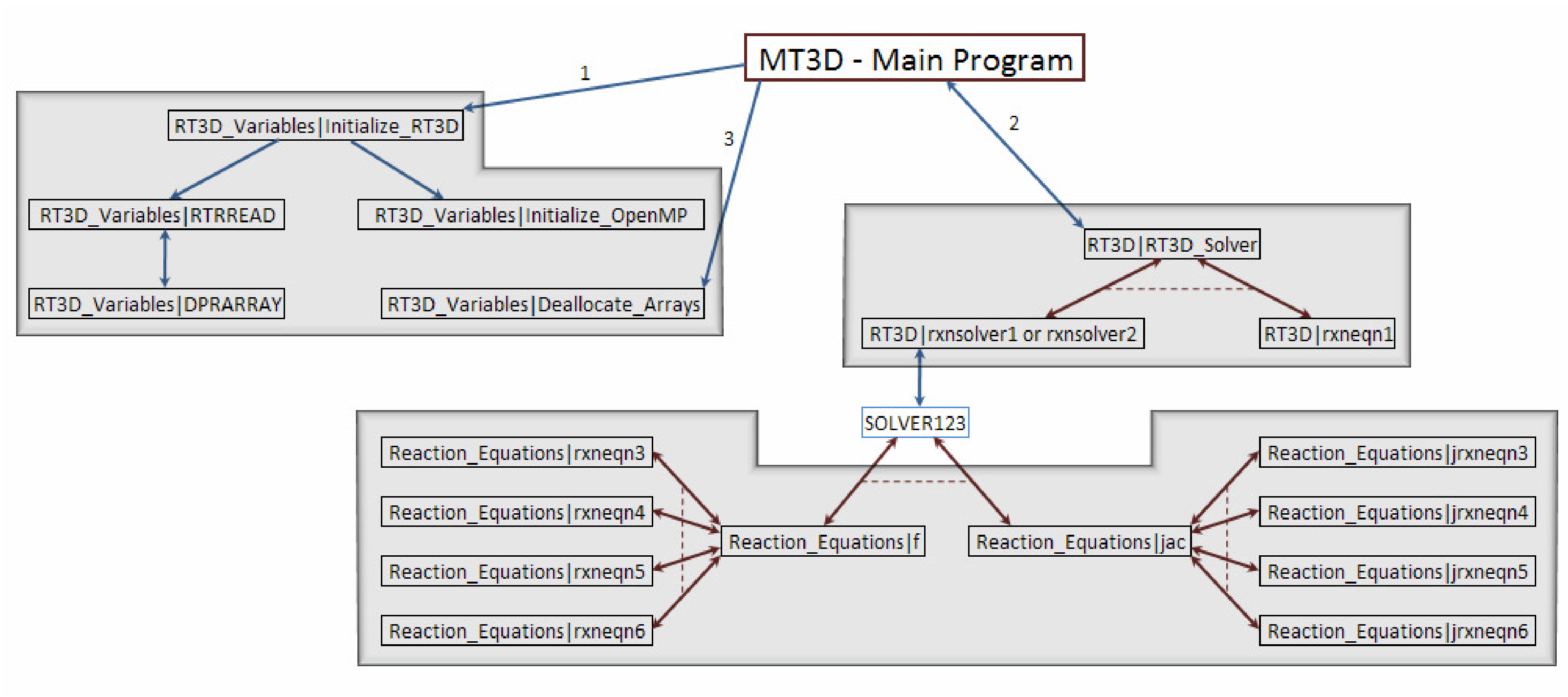
MT3D - Main Program
1
2
3
RT3D_Variables|Initialize_RT3D
RT3D_Variables|RTRREAD
RT3D_Variables|Initialize_OpenMP
RT3D_Variables|DPRARRAY
RT3D_Variables|Deallocate_Arrays
RT3D|RT3D_Solver
RT3D|rxnsolver1 or rxnsolver2
RT3D|rxneqn1
SOLVER123
Reaction_Equations|f
Reaction_Equations|jac
Reaction_Equations|rxneqn3
Reaction_Equations|rxneqn4
Reaction_Equations|rxneqn5
Reaction_Equations|rxneqn6
Reaction_Equations|jrxneqn3
Reaction_Equations|jrxneqn4
Reaction_Equations|jrxneqn5
Reaction_Equations|jrxneqn6

Figure A-1 RT3D Version 3.0 Structure

A.1 Input File Structure

The .RCT file remains will be read in by MT3D. It will no longer look like the old RT3D format. This file can be viewed in the mt3dms manual (Zheng and Wang, 1999). Isotherm can be 0 to 3 and is completely taken care of by MT3D, while IREACT will always be 0, IRCTOP will always be 2, and IGETSC will always be 0. A new .RTR file will be created for RT3D. This input file will look similar to the old input file following the directions of the following 4 variables only:

E1 Record: **RTREACT, NCRXNDATA, NVRXNDATA, ISOLVER**

RTREACT = Reaction module number

= 0, no reaction is simulated (i.e., tracer transport)

= 1, Two-Species Instantaneous Reactions (BIOPLUME-II type reactions)

= 2, {module reserved for future implementation}

= 3, Six Species, First-Order, Rate-Limited, BTEX Degradation using Sequential Electron Acceptors

= 4, Rate-Limited Sorption

= 5, Double Monod Model

= 6, Sequential First-Order Decay (up to 4 species, e.g., PCE/TCE/DCE/VC)

= 7, {module reserved for future implementation}

= 8, {module reserved for future implementation}

= 9, {module reserved for future implementation}

NCRXNDATA = number of constant reaction parameter values

NVRXNDATA = number of variable reaction parameter arrays

ISOLVER

= 0, for the instantaneous reaction modules 1 and 2.

= 1, Automatic switching Gear-stiff/non-stiff solver. For stiff systems, this option will automatically compute the Jacobian matrix using finite-difference approximations.

= 2, Automatic switching Gear-stiff/ non-stiff solver. For stiff systems, this option will require an external routine to compute analytical Jacobian. Need to provide an external subroutine .jacrxns.f. that specifies the Jacobian matrix for the differential reaction equations.

= 3, Fehlberg fourth-fifth order Runge-Kutta method RT3D v2.5 Update Document 6

= 4, Stiff solver based on a semi-implicit extrapolation method. This option requires an external routine .jacrxns.f. to compute the analytical Jacobian matrix for the differential reaction equations.

= 5, Non-stiff Runge-Kutta solver

A.2 General Changes

All files have been updated to work in free format instead of the old fixed format FORTRAN 77. This means the file extensions have changed to a .F90 extension. Many new FORTRAN 95 commands are used more extensively in this version such as MODULES.

A.3 RT3D Code Files

Since RT3D was restructured to simplify the process of a version change. Certain files need never be changed again. When a new version of MT3DMS is created, only a few lines of code need to be added to call the RT3D subroutines. These files can be seen in Table A-1.

Table A-1 Equivalent RT3D and MT3D files that do not need to be changed

RT3D Version 3.0	MT3D Version 5.0
adv30d	mt_adv5
btnrtv25	mt_btn5
dsp30d	mt_dsp5
fmi30d	mt_fmi5
gcg30d	mt_gcg5
ssmrtv25	mt_ssm5
utlrtv25	mt_utl5
-	mt_tob5

The MT3DMS5 file only has a few lines that need to be inserted. Finally, the four RT3D files need to be added to the project.

MT3DMS5 (like the old rt3dv25)

Code is added in 7 different locations. The sections are as follows:

(RT3D – 1) – is to be added above the variable declaration statements and IMPLICIT NONE statement; near the very top of the file

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   RT3D - 1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    !links this file the the MODULES RT3D & RT3D_Variables
    USE RT3D, ONLY: RT3D_Solver
    USE RT3D_Variables, ONLY: Initialize_RT3D, Deallocate_Arrays
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   END RT3D - 1
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(RT3D -2) – to be added somewhere in the variable declarations section

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   RT3D - 2
!     RT3D specific data is read from .RTR file in unit number 41
!     iUnitTRNOP(41) is also the storage space for flag for !
      indicating the presence of RTR package
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    INTEGER INRTR
    DATA INRTR/41/
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   END RT3D - 2
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(RT3D -3) – needs to be somewhere shortly after this line

```
603   READ(ISUP,602,END=604) FLTYPE,FLNAME
```

in the if statement checking the file extensions and opening those files

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   RT3D - 3
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    ELSEIF(FLTYPE(1:3).EQ.'RTR') THEN
      CALL SETPATH(PATH,FLNAME)
      CALL OPENFL(INRTR,1,FLNAME,1,FINDEX)
      iUnitTRNOP(41)=INRTR
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   END RT3D - 3
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(RT3D -4) – needs to be in the section saying

```
!--READ AND PREPARE INPUT DATA RELEVANT TO
!--THE ENTIRE SIMULATION
```

Preferentially in order of iUnitTRNOP, but before the stress period loop

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   RT3D - 4
!     Since we are Using RT3D we reset these values
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    IF(iUnitTRNOP(41).GT.0) THEN
      CALL Initialize_RT3D(iUnitTRNOP(41),IOUT,NCOL,NROW,NLAY, &
                              NCOMP,DTRANS,X(LCPR),X(LCRETA))
    END IF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   END RT3D - 4
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(RT3D -5) – This is a bug in MT3D that could or could not be fixed. Until then this must add it in the section solving implicit schemes formulating matrix coefficients. Just the highlighted part

```
!!!!!RT3D 5 - (.AND. ICOMP.LE.MCOMP) was a bug and needed to be
added here!
          IF(iUnitTRNOP(4).GT.0 .AND. ICOMP.LE.MCOMP) THEN
            .........
          END IF

!!!!!RT3D 5 - (.AND. ICOMP.LE.MCOMP) was a bug needed and to be
added here!
          IF(iUnitTRNOP(5).GT.0 .AND. ICOMP.LE.MCOMP) THEN
            .........
          END IF
```

(RT3D -6) – This is the main call for the RT3D. It goes in the section calculate mass budgets for implicit scheme after the (iUnitTRNOP(4).GT.0) if statement and before the calculate global mass budgets and check mass balance calls

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   RT3D - 6 - put your stuff here
!           Remember to cut MT3D mass balance loop into 2 loops
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          END DO
          !Solver for the Reactions
          CALL RT3D_Solver(IX(LCIB),X(LCCNEW),X(LCDELR), &
                        X(LCDELC),X(LCDH),RMASIO(13,:,:))
          DO ICOMP=1,NCOMP
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   END RT3D - 6
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

(RT3D -7) – this last section is a call to deallocate our arrays and goes somewhere near the end of the main program after the end of the stress period loop

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   RT3D - 7
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    IF(iUnitTRNOP(41).GT.0) THEN
      CALL Deallocate_Arrays()
    END IF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!   END RT3D - 7
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

RT3D_Variables.F90 – this is a new addition

This file is divided into 3 sections. Section 1 contains the following variables: NCOMP, RTREACT, ncrxndata, nvrxndata, ISOLVER, atoll(:), rtol(:), vrc(:,:,:,:), rc(:), NTREADS, RunOpenMP. Section 1 also holds pointers to the following variables held by MT3D: delt_ptr, prsity_ptr(:,:,:,:). All variables and pointers can be used inside any subroutine containing the USE RT3D_Variables statement if they are included after the ONLY: command. Section 2 contains the RT3D initialization subroutines. This section initializes all variables. It also contains the RTREAD subroutine and DPRARRAY subroutine to read the .rtr input file. Section 3 contains a subroutine that deallocates all our arrays at the end of the program.

RT3D.F90

This is the main driver for the RT3D reactions. All module rules apply. To access any subroutine in the module, outside subroutines must include the USE RT3D statement. The equivalent file in the old RT3D is rtrtcv25 file. The reading of the input file no longer resides in this file, but rather in the RT3D_Variables.F90 file. The driver of the

RT3D (previously called RCTRTSV) is now called RT3D_Solver. The number of variables passed to it is considerably smaller, since most of the variables or pointers to the variables are now stored in RT3D_Variables.F90. The rest of the subroutine is pretty self explanatory. The RTREACT variable holds the kinetics number or equation number that is to be solved. The OpenMP parallel code is also include, though a discussion of OpenMP will not be done here.

The second subroutine in this file is rxneqn1 or the Reaction model #1 subroutine. This is to prevent a connection to the RXNEQNS file except through the integrator SOLVER123. The rxnsolver1 and rxnsolver2 follow rxneqn1 subroutine.

RXNEQNS.F90

This file is similar to rteqnv25, except for how the variables are accessed through a USE statement connecting the RT3D_Variables module. The first two subroutines f and jac contain calls to the actually equations using a SELECT CASE command which goes to the case RTREACT. There is only 3 blocks in each model now. Anyone can still go in and insert there own variables if they want to. There are no SAVE statements attached to any variables nor COMMON statements to insure the OpenMP works correctly. Also bulk density is hardwired into the models for now.

SOLVER123.F90

This contains the solver. All variables it uses are passed to it. It calls the reaction equations to solve the differential equations the user wants to use but then those values get sent right back to it. This file should not have had to change for the new version.

Some changes were necessary to insure that OpenMP would run correctly. Therefore the following changes were made as shown in the Table.

Table A-2 Changes to the solver123 file

	OLD RT3D	NEW RT3D
(Lines 423,433,447,3154)	Call stopfile	Call USTOP(‘ ‘)
(solver, prja, stoda, rkfs, & fehl)	-	Added in Subroutines USE Reaction_Equations
(solver, prja, stoda, rkf45, rkfs, & fehl)	-	Pass node location j,i,k through subroutines
Deleted	External f,jac,pjac,slvs	These functions link directly to a function named f,jac,prja,solsy respectively
Deleted	COMMON /ls0001/ & COMMON /lsa001/	The variables are stored locally in solver and passed to the subroutines that require them as in the following.
All variables are store locally to the SOLVER subroutine and passed into others when needed except these: Local to rxnsolver1(in the RT3D file) - illin, init, lyh, lewt, lacor, lsavf, lwm, liwm, mxstep, mxhnil, nhnil, ntrep, nslast, nyh – the reason for this is that they have to be saved either in a SAVE variable statement inside of solver or kept one tier higher to avoid that statement and allow the code to be parallelized Local to SOLVER – conit, crate, el, elco,hold, rmax, tesco, ccmax, el0, h, hmin, hmxi, hu, rc, tn, uround, ialth, ipup, lmax, nqnyh, nslp, icf, ierpj, iersl, jcur, jstart, kflag, l, meth, miter, maxord, maxcor, msbp, mxncf, n, nq, nst, nfe, nje, nqu, tsw, pdnorm, pdest, pdlast, ratio, cm1, cm2, insufr, insufi, ixpr, icount, irflag, jtyp, mused, mxordn, mxords		

A.4 Benchmark Times for New Parallel RT3D Version 3.0

While creating a more versatile and parallel version of RT3D, a cost was incurred of opening and reading a new file as well as parallel code implementation. Table A-3 shows the time comparisons between the old version of RT3D and the new. This table shows that indeed there was a small increase in the time it takes to run a file in the new version. The fact is that the parallel code allows for the times of all the tests to be cut almost in half on a two processor machine. If in fact, the way of the future is multiple processors, the potential of the new RT3D Version 3.0 over the old RT3D Version 2.5 far outweighs the small time costs on one processor.

Table A-3 Time Comparison of RT3D v2.5 and RT3D v3.0 on 1 processor

	Old_RT3D	**New_RT3D**
Tracer	0.1988699	2.19779462
Package 1	0.3631664	0.402091904
Package 3	3.23772	3.395094872
Package 4	4.71075	5.080254021
Package 5	3.849252	4.026111774
Package 6	34.84665	36.27360922
1D Problem	2.226375	2.247753997
10 Layer	352.8463	367.8341506

Appendix B. OpenMP Commands

OpenMP has been developed for shared memory systems. With OpenMP, the sequential code does not need to be changed much, if at all. The parallel constructs are placed right around the section of code that is to run on multiple processors. This allows programs to be parallelized incrementally as needed. Each variable needs to be evaluated to determine if it should be a private or shared variable to the parallel region. (Quinn, 2004). OpenMP has been developed to work on an OpenMP-compliant compiler as well as a normal compiler. This is achieved by hiding the OpenMP directives in such a way that a normal compiler would see them as a comment and neglect their content. An OpenMP-compliant compiler would recognize and run the line. The following commands were used in the reactive transport examples: (Hermanns, 2002)

```
!$OMP PARALLEL clause1 clause2 …
        …parallel code is placed here…
!$OMP END PARALLEL
```

A parallel region must be created/opened and destroyed/closed. Each thread in a parallel region has a specific thread ID. The master thread with ID 0 forks at the beginning of the parallel region and rejoins at the end. (Van der Pas, 2005). Clauses are amended to the end of the construct specifying how the parallel region is to treat each variable and how the threads will divide up the work load. Optional clauses include: (Hermanns, 2002)

- PRIVATE (*list*): each thread has its own copy of this variable; a private variable must be initialized inside the parallel region constructs and does not exist before or after this region
- SHARED (*list*): each thread has access to this variables location and can change it, erase it, etc.; be careful of race conditions with shared variables
- DEFAULT (PRIVATE | SHARED | NONE): using this clause allows the programmer to implicitly declare all undeclared variables to be PRIVATE or SHARED, or cause all variables to be declared explicitly; leaving this clause out leaves the default to SHARED
- FIRSTPRIVATE (*list*): this clause allows the listed variables to be private to each thread giving them an initial value of what the variable existed as before the parallel region was entered
- REDUCTION (*operator:list*): when multiple threads need to write to a memory location of a shared variable, one at a time, this clause solves this problem by keeping track of what is to be written until the variable can be synchronized at the end of the parallel region; the operator states what operation the synchronization is to perform; operators include +, *, -, .AND., .OR., .EQV., .NEQV., MAX, MIN, IAND, IOR or IEOR

- IF (*scalar logical expression*): allows the programmer to specify if the code should be executed in parallel or serially; often running in parallel would require more overhead than actually running the code serially
- NUM THREADS (*scalar integer expression*): allows the programmer to specify how many threads the parallel region will run on

```
!$OMP DO clause1 clause2 …
        DO i=1, N
                …parallel code is placed here…
        END DO
!$OMP END DO
```

OpenMP is very good at parallelizing Do Loops. Each thread computes part of the iterations. The index counter is automatically assumed to be PRIVATE, but it is divided up between the threads according to the specified SCHEDULE. Optional clauses include: (Hermanns, 2002)

- PRIVATE (*list*): same as above
- FIRSTPRIVATE (*list*): same as above
- LASTPRIVATE (*list*): since private variables do not exist after the parallel region is ended, a LASTPRIVATE command causes the variable to get a copy of what the last iteration has so that it exists afterwards
- REDUCTION (*operator:list*): same as above

• SCHEDULE (*type, chunk*): this allows the threads to receive work according to a static, dynamic, or guided schedule; each has its own benefits; efficiency of the processor is what should be looked at to decide which to use

```
!$OMP ATOMIC
!$OMP CRITICAL SECTION
```

These statements cause the enclosed block to be executed by all threads but only one thread at a time. This is important to protect a shared variable from a race condition.

This is not an all inclusive list of OpenMP commands, but rather commands that were used in the reactive transport modeling examples in this paper. Some other important commands include !$OMP SECTIONS, !$OMP SINGLE, and !$OMP MASTER. Also the PARALLEL constructs can be combined with the DO and SECTIONS commands on one line. (Hermanns, 2002)