LBNL-63827

TMVOC-MP: A Parallel Numerical Simulator for Three-Phase Non-isothermal Flows of Multicomponent Hydrocarbon Mixtures in Porous/Fractured Media

Keni Zhang[1], Hajime Yamamoto[2], and Karsten Pruess[1]

[1] Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720
[2] Civil Engineering Research Institute, Taisei Corporation, Yokohama 245-0051, Japan

September 2007

TABLE OF CONTENTS

1. Introduction 4

2. Requirements and Code Installation 6
   2.1 Hardware and Software Requirements 6
   2.2 Code Compilation and Installation 6

3. Methodology and Code Architecture 9
   3.1 Grid Domain Partitioning and Gridblock Reordering 10
   3.2 Organization of Input and Output Data 12
   3.3 Assembly and Solution of Linearized Equation Systems 13
   3.4 Communication between Processors 16
   3.5 Program Structure and Flow Chart 16

4. Description of Input Files 18
   4.1 Preparation of Input Data 18
   4.2 Special Input Requirements for TMVOC-MP V1.0 18
   4.3 Output from TMVOC-MP V1.0 26

5. Sample Problem 27
   5.1 Model Setup 27
   5.2 Running Simulations 28
   5.3 Simulation Results and Code Performance 29

6. Concluding Remarks 34

Acknowledgements 34

References 35

1. INTRODUCTION

TMVOC-MP is a massively parallel version of the TMVOC code (Pruess and Battistelli, 2002), a numerical simulator for three-phase non-isothermal flow of water, gas, and a multicomponent mixture of volatile organic chemicals (VOCs) in multidimensional heterogeneous porous/fractured media. TMVOC-MP was developed by introducing massively parallel computing techniques into TMVOC. It retains the physical process model of TMVOC, designed for applications to contamination problems that involve hydrocarbon fuels or organic solvents in saturated and unsaturated zones. TMVOC-MP can model contaminant behavior under "natural" environmental conditions, as well as for engineered systems, such as soil vapor extraction, groundwater pumping, or steam-assisted source remediation. With its sophisticated parallel computing techniques, TMVOC-MP can handle much larger problems than TMVOC, and can be much more computationally efficient.

TMVOC-MP models multiphase fluid systems containing variable proportions of water, non-condensible gases (NCGs), and water-soluble volatile organic chemicals (VOCs). The user can specify the number and nature of NCGs and VOCs. There are no intrinsic limitations to the number of NCGs or VOCs, although the arrays for fluid components are currently dimensioned as 20, accommodating water plus 19 components that may be either NCGs or VOCs. Among them, NCG arrays are dimensioned as 10. The user may select NCGs from a data bank provided in the software. The currently available choices include O_2, N_2, CO_2, CH_4, ethane, ethylene, acetylene, and air (a pseudo-component treated with properties averaged from N_2 and O_2). Thermophysical property data of VOCs can be selected from a chemical data bank, included with TMVOC-MP, that provides parameters for 26 commonly encountered chemicals. Users also can input their own data for other fluids. The fluid components may partition (volatilize and/or dissolve) among gas, aqueous, and NAPL phases. Any combination of the three phases may present, and phases may appear and disappear in the course of a simulation. In addition,

VOCs may be adsorbed by the porous medium, and may biodegrade according to a simple half-life model. Detailed discussion of physical processes, assumptions, and fluid properties used in TMVOC-MP can be found in the TMVOC user's guide (Pruess and Battistelli, 2002).

TMVOC-MP was developed based on the parallel framework of the TOUGH2-MP code (Zhang et al. 2001, Wu et al. 2002). It uses the MPI (Message Passing Forum, 1994) for parallel implementation. A domain decomposition approach is adopted for the parallelization. The code partitions a simulation domain, defined by an unstructured grid, using partitioning algorithm from the METIS software package (Karypsis and Kumar, 1998). In parallel simulation, each processor is in charge of one part of the simulation domain for assembling mass and energy balance equations, solving linear equation systems, updating thermophysical properties, and performing other local computations. The local linear-equation systems are solved in parallel by multiple processors with the Aztec linear solver package (Tuminaro et al., 1999). Although each processor solves the linearized equations of subdomains independently, the entire linear equation system is solved together by all processors collaboratively via communication between neighboring processors during each iteration. Detailed discussion of the prototype of the data-exchange scheme can be found in Elmroth et al. (2001). In addition, FORTRAN 90 features are introduced to TMVOC-MP, such as dynamic memory allocation, array operation, matrix manipulation, and replacing "common blocks" (used in the original TMVOC) with modules. All new subroutines are written in FORTRAN 90. Program units imported from the original TMVOC remain in standard FORTRAN 77.

This report provides a quick starting guide for using the TMVOC-MP program. We suppose that the users have basic knowledge of using the original TMVOC code. The users can find the detailed technical description of the physical processes modeled, and the mathematical and numerical methods in the user's guide for TMVOC (Pruess and Battistelli, 2002).

2. REQUIREMENTS AND CODE INSTALLATION

2.1 Hardware and Software Requirements

TMVOC-MP has been tested on IBM and CRAY supercomputers, Linux clusters, Macs, and multi-core PCs under different operating systems. It has been successfully compiled using g95, and Fortran compilers from Intel, IBM, and the Portland Group. The code requires 64-bit arithmetic (8 byte word length for floating point numbers) for successful execution. TMVOC-MP can be run on any shared- or distributed-memory multiple CPU computer system on which MPI is installed. The code has been tested on LAM/MPI, OPEN MPI, and MPICH2.

The total computer memory required by TMVOC-MP depends on the problem size. For a given problem, memory requirement is split among the processors used for the simulation. The code automatically distributes memory requirements to all processors based on the partitioning of the domain. All major arrays are dynamically allocated according to the numbers of local gridblocks and connections assigned by domain partitioning to each processor. As a result, larger problems can be solved using more processors on a distributed memory computer system. For example, by far the largest array used in TMVOC-MP is "*PAR*". Its size in bytes (using 8-byte real data) is

$$M=(NPH*(NB+NK)+2)*(NEQ+1)*NEL*8 \qquad (2.1)$$

Here the parameters are the total phase number *NPH*, secondary parameter number *NB*, component number *NK*, and gridblock number *NEL*. If *NPH=3, NB=8, NK=3, NEQ=4,* $NEL=10^6$, the total memory requirement for this array is about 1400 MB. If 64 processors are used to solve this problem, each processor requires about 22 MB of memory for this array.

2.2 CODE COMPILATION AND INSTALLATION

The TMVOC-MP code was developed through successive modifications from the original serial (or sequential) TMVOC code. The source code consists of 10 FORTRAN

files (Compu_Eos.f, Data_DD.f, Input_Output.f, Main_Comp.f , Mem_Alloc.f, Mesh_Maker.f, MULTI.f, Paral_Subs.f, TOUGH2.f, Utility_F.f) and two library files (*libmetis.a and libaztec.a)*. The two library files are generated by compiling the METIS and AZTEC software packages.

Compilation and installation can be done through the following steps:

1. Download METIS at:

http://www-users.cs.umn.edu/~karypis/metis/metis/download.html

2. Compile METIS in the computer system where TMVOC-MP will be installed.

3. Download AZTEC at:

http://www.cs.sandia.gov/CRF/aztec1.html

4. Compile AZTEC in the computer system where TMVOC-MP will be installed.

(Guides for compiling METIS and AZTEC are provided with the downloaded packages.)

5. Transfer the file "tmvoc-mp_1.0.tar.gz" from the installation CD-ROM to your working directory.

6. Use gunzip to unzip the file and then use the tar command to untar the archived files and directories as followings:

gunzip tmvoc-mp_1.0.tar.gz

tar –xvf tmvoc-mp_1.0.tar

A directory named tmvoc-mp_1.0 will be created under the current working directory. Source files, make scripts, and installation test input files will be located in the subdirectories. Two additional subdirectories are created under the directory tmvoc-mp_1.0: ~/tmvoc-mp_1.0/partition/ and ~/tmvoc-mp_1.0/utility/.

7. Copy az_aztecf.h and libaztec.a from ~/aztec/lib and libmetis.a from ~/metis-4.0 to the subdirectory where source codes are located (~/tmvoc-mp_1.0/). libaztec.a and libmetis.a were created by compiling Metis and Aztec in steps 2 and 4, respectively.

8. "makefile" for three different compilers are provided: IBM, INTEL and PORTLAND GROUP. You can choose the one most close to your complier. In the "makefile", a wrapper compiler, mpif90, was specified for compiling the source codes. The user may need to change the compiler name to the one appropriate for his/her system by editing the file "makefile" at the line with "FC=mpif90". The user may also need to specify the path

for MPI "include" and "library" files. Figure 2.1 shows the makefile using PORTLAND GROUP Fortran 90.

```
# for clusters
FC = mpif90
FFLAGS = -O -r8 -i4

# The following specifies the files used for the "standard
version"
OBJS = Data_DD.o Mem_Alloc.o MULTI.o Main_Comp.o TOUGH2.o \
       Compu_Eos.o Input_Output.o Mesh_Maker.o \
       Paral_Subs.o Utility_F.o \

LIBS =  libmetis.a libaztec.a
tough2: $(OBJS)
     $(FC) -o tmvoc-mp $(FFLAGS) $(OBJS) $(LIBS)
clean:
     rm -f *.o *.mod
```

Figure 2.1 A makefile for TMVOC-MP compilation

9. Type "make" under the ~/tmvoc-mp_1.0/ subdirectory to compile the code. The executable file "tmvoc-mp" will be created. After compilation, you may type "make clean" to clean all intermediate files.

In order to successfully build TMVOC-MP V1.0, the *c* and Fortran compilers used for compiling the MPI system, AZTEC, METIS and TMVOC-MP source codes must be compatible. A Fortran 90 or higher version must be used for TMVOC-MP Fortran source code compilation.

If you encounter "invalid communicator" or other communication problems during running the executable, you may try following:

1. Copy ~/tmvoc-mp_1.0/utility/md_wrap_mpi_c.c to ~/aztec/lib to replace the original one.

2. Recompile AZTEC and then use the new library libaztec.a to recompile the TMVOC-MP executable.

If you have difficulty to use linear solver "AZ_gmres", you may try the following:

1. Copy ~/tmvoc-mp_1.0/utility/la_dlaic1.f to ~/aztec/lib to replace the original one.

2. Recompile AZTEC and then use the new library libaztec.a to recompile the TMVOC-MP executable.

You may get additional speedup by using non-blocking communication version AZTEC through:

1. Copy ~/tmvoc-mp_1.0/utility/az_comm.c and

~/tmvoc-mp_1.0/utility/az_matvec_mult.c to ~/aztec/lib to replace the original files.

2. Recompile AZTEC and then use the new library libaztec.a to recompile the TMVOC-MP executable.

The library file "libmetis.a" contains subroutines of the METIS package for partitioning irregular graphs and meshes. For reducing the requirement of computer memory, we use 4-byte integer for all large integer arrays in TMVOC-MP. The corresponding arrays in METIS must also be in 4-byte integer. This can be implemented by simply removing the line of "#define IDXTYPE_INT" in head file "struct.h" of the METIS source code.

3. METHODOLOGY AND CODE ARCHITECTURE

TMVOC-MP V1.0 is based on a fully implicit formulation with Newton iteration and offers a choice of Krylov iterative methods such as conjugate gradient (CG), generalized minimum residual (GMRES), and stabilized biconjugate gradient (BiCGSTAB). The fully implicit scheme has proven to be the most robust numerical approach in modeling multiphase flow and heat transfer in reservoirs in the past several decades. For a typical simulation with fully implicit scheme and Newton iteration, the most time-consuming parts of the execution are (1) assembling the Jacobian matrix, (2) solving the linearized system of equations, and (3) updating thermophysical parameters. The basic strategy of a parallel code is to distribute the computational work of these three parts as evenly as possible among the processors, while minimizing communication cost. To reach this goal, a number of computing strategies and methods are implemented in TMVOC-MP, such as grid node/element domain partitioning, grid node/element reordering, data input and output optimizing, and efficient message exchange between processors. These will be further described below.

3.1 Grid Domain Partitioning and Gridblock Reordering

In a TMVOC-MP V1.0 simulation, a model domain is represented by a set of three-dimensional gridblocks (elements), and interfaces between ordered pairs of gridblocks, called "connections." The entire connection system of the gridblocks is treated as an unstructured grid. From the connection information, subdomain partitioning can be generated and stored in an adjacency matrix. The adjacency or connection structure of the model meshes is stored in a compressed storage format (CSR).

The adjacency structure of the model grids can be described as follows: In the CSR format, the adjacency structure of a global-mesh domain with n gridblocks and m connections is represented by two arrays, *xadj* and *adj*. The *xadj* array has a size of $n+1$, whereas the *adj* array has a size of $2m$. Assuming that element numbering starts from *1*, the adjacency list of element i is stored in an array *adj,* starting at index *xadj(i)* and ending at index *xadj(i+1)-1*. That is, for each element i, its adjacency list is stored in the consecutive locations in the array *adj*, and the array *xadj* is used to point to where it begins and where it ends. Figure 3.1a shows the connection of a 12-element domain; Figure 3.1b illustrates its corresponding CSR-format arrays.

TMVOC-MP utilizes three partitioning algorithms of the METIS software package (version 4.0) (Karypsis and Kumar, 1998) for grid domain partitioning. The three algorithms are denoted as the *K-way*, the *VK-way*, and the *Recursive* partitioning algorithm. *K-way* is used for partitioning a global mesh (graph) into a large number of partitions (more than 8). The objective of this algorithm is to minimize the number of edges that straddle different partitions. If a small number of partitions is desired, the *Recursive* partitioning method, a recursive bisection algorithm, should be used. *VK-way* is a modification of *K-way,* and its objective is to minimize the total communication volume. Both *K-way* and *VK-way* are multilevel partitioning algorithms.

Figure 3.1a shows a scheme for partitioning a sample domain into three parts. Gridblocks are assigned to different processors through partitioning methods and reordered by each processor to a local index ordering. Elements corresponding to these blocks are explicitly stored in local memory of the processor and are defined by a set of indices referred to as the processor's *update* set. The *update* set is further divided into two subsets: *internal* and *border*. Elements of the *internal* set are updated using only the information on the current processor. The *border* set consists of blocks with at least one edge of a block belonging to another processor. As a result, updating values of the *border* elements requires values of those blocks from the other processors. These blocks are called as *external* blocks whose values are not calculated in the current processor, but are obtained from other processers via communication to update the components in the *border* set. Table 3.1 shows the partitioning results. One of the local numbering schemes for the sample problem is presented in Figure 3.1a.

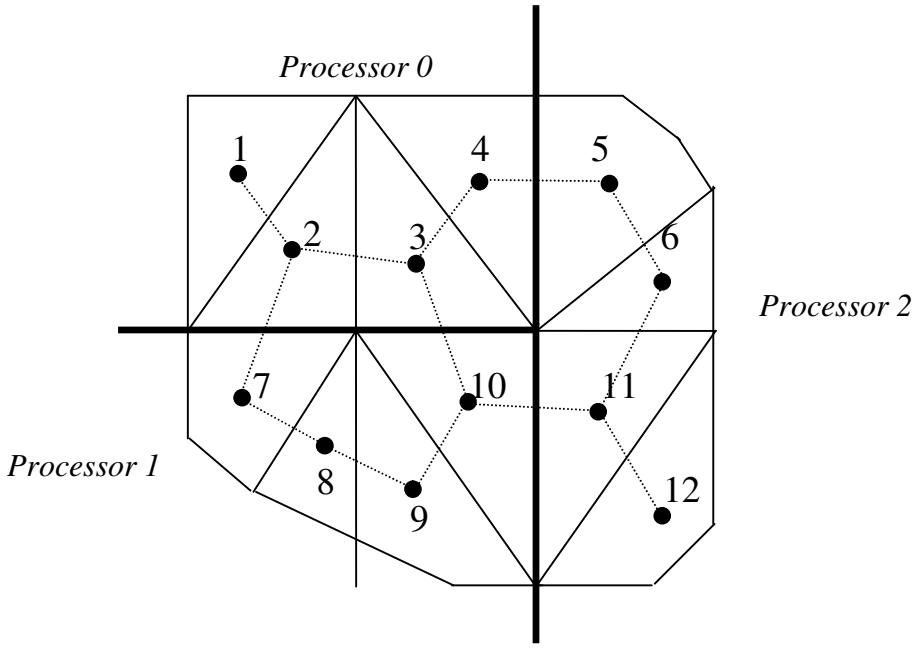
Table 3.1. Example of Domain Partitioning and Local Numbering

		Update		**External**
		Internal	Border	
Processor 0	Gridblocks	1	2 3 4	5 7 10
	Local Numbering	1	2 3 4	5 6 7
Processor 1	Gridblocks	8 9	7 10	2 3 11
	Local Numbering	1 2	3 4	5 6 7
Processor 2	Gridblocks	6 12	5 11	4 10
	Local Numbering	1 2	3 4	5 6

The local numbering of gridblocks is carried out to facilitate the communication between processors. The numbering sequence is *internal* block set followed by *border* block set and finally by the *external* block set. In addition, all *external* blocks on the same processor are in a consecutive order.

For the Jacobian matrix, only nonzero entries of a submatrix for a partitioned mesh domain are stored on each processor. Each processor stores only the rows that correspond

to its *update* set (including internal and border blocks, see Table 3.1). These rows form a submatrix whose entries correspond to the variables of both the *update* set and the *external* set defined on this processor.



(a) A 12-elements domain partitioning on 3 processors

Elements		1	2	3	4	5	6	7	8	9	10	11	12
xadj	1	2	5	8	10	12	14	16	18	20	23	26	27
adj		2	1,3,7	2,4,10	3,5	4,6	5,11	2,8	7,9	8,10	3,9,11	6,10,12	11

(b) CSR format

Figure 3.1. An example of domain partitioning and CSR format for storing connections

3.2 Organization of Input and Output Data

The input data include hydrogeologic parameters and constitutive relations of porous media, such as absolute and relative permeability, porosity, capillary pressure, thermophysical properties of fluids and rocks, and initial and boundary conditions of the system. Other processing requirements include the specification of space-discretized geometric information (grid) and various program options (computational parameters and

time-stepping information). For large-scale three-dimensional models, computer memory of several gigabytes may be required, and the distribution of the memory to all processors is necessary.

To efficiently use the memory of each processor (considering the distributed-memory computer), the input data files for the TMVOC-MP V1.0 simulation are organized in sequential format. TMVOC-MP V1.0 uses two large groups of data blocks: one with dimensions equal to the number of gridblocks; another with dimensions equal to the number of connections (interfaces). Large data blocks are read one by one through a temporary full-sized array and then distributed to different processors. This method avoids storing all input data in the memory of a single processor (which may be too small) and enhances the code scalability. Other small-volume data, such as simulation control parameters, are duplicated on all processors. Note that the formats of the input data mentioned above are used internally by TMVOC-MP (i.e., generated from the regular TMVOC input file when needed). As an interface to the user, the original input file formats of TMVOC are fully compatible with TMVOC-MP.

3.3 Assembly and Solution of Linearized Equation Systems

In the TMVOC-MP V1.0 formulation, the discretization in space using the integral finite difference method (IFD; Narasimhan and Witherspoon, 1976) leads to a set of strongly coupled nonlinear algebraic equations, which are linearized by the Newton method. Within each Newton iteration, a Jacobian matrix is first constructed by numerical differentiation. The resulting system of linear equations is then solved using an iterative linear solver with chosen preconditioning procedures. The following gives a brief discussion of assembling and solving the linearized equation systems using parallel computing.

The discrete mass and energy balance equations solved by the TMVOC-MP code can be written in residual form as follows (Pruess, 1991; Pruess et al., 1999):

$$R_n^{\kappa}(x^{t+1}) = M_n^{\kappa}(x^{t+1}) - M_n^{\kappa}(x^t) - \frac{\Delta t}{V_n}\{\sum_m A_{nm} F_{nm}^{\kappa}(x^{t+1}) + V_n q_n^{\kappa,t+1}\} = 0 \qquad (3.1)$$

where the vector x^t consists of primary variables at time t, R_n^{κ} is the residual of component κ (heat is regarded as a "component") for block n, M denotes mass or thermal energy per unit volume for a component, V_n is the volume of the block n, Δt denotes the current time step size, $t+1$ denotes the current time, A_{nm} is the interface area between blocks n and m, F_{nm} is the flow between them, and q denotes sinks and sources of mass or energy. Equation (3.1) is solved using the Newton method, leading to

$$-\sum_i \left.\frac{\partial R_n^{\kappa,t+1}}{\partial x_i}\right|_p (x_{i,p+1} - x_{i,p}) = R_n^{\kappa,t+1}(x_{i,p}) \qquad (3.2)$$

where $x_{i,p}$ represents the value of ith primary variable at the pth iteration step.

The Jacobian matrix as well as the right-hand side of (3.2) needs to be recalculated for each Newton iteration, leading to a large computational effort for large-scale simulations. In the parallel code, the assembly of the linear equation system (3.2) is shared by all the processors, and each processor is responsible for computing the rows of the Jacobian matrix that correspond specifically to the blocks in the processor's own *update* set. Computation of the elements in the Jacobian matrix is performed in two parts. The first part consists of the computations related to the individual blocks (accumulation and source/sink terms). Such calculations are carried out using the information stored on the current processor, without communications with other processors. The second part includes the computations related to the connections or flow terms. Elements in the *border* set need information from the *external* set, which requires communication with neighboring processors. Before performing these computations, an exchange of relevant primary variables is required. For the elements corresponding to *border* set blocks, each processor sends the values of these elements to the corresponding processors, which receive them for their *external* blocks.

The Jacobian matrix for local gridblocks in each processor is stored in the distributed variable block row (DVBR) format, a generalization of the VBR format. All matrix blocks are stored row-wise, with the diagonal blocks stored first in each block row. Scalar elements of each matrix block are stored in column major order. The data structure consists of a real vector and five integer vectors, forming the Jacobian matrix. Detailed explanation of the DVBR data format can be found in Tuminaro et al. (1999).

The linearized equation system arising at each Newton step is solved using a parallel iterative linear solver from the AZTEC package V2.1 (Tuminaro et al., 1999). Several different solvers and preconditioners from the package can be selected, including conjugate gradient, restarted generalized minimal residual, conjugate gradient squared, transpose-free quasi-minimal residual, and bi-conjugate gradient with stabilization methods. The work solving the global linearized equation is shared by all processors, with each processor responsible for computing its own portion of the partitioned domain equations. To accomplish this parallel solution, communication among processors is required to exchange data between the neighboring grid partitions at each iteration. Moreover, global communication is also required to compute the norms of vectors for checking the convergence.

During a simulation, the time step usually is automatically adjusted (increased or reduced), depending on the convergence rate of the iterations. In the TMVOC-MP V1.0 code, time-step size is calculated at the master processor, after collecting necessary data from all processors. The convergence rates may be different in different processors. Only when all processors reach stopping criteria will the calculation proceed to the next time step.

At the end of a time step or a simulation, the solutions obtained from all processors are transferred to the master processor for output. Results related to the connections or flow terms that cross the boundary of two different processors are obtained by arithmetic averaging of the solutions from the two processors involved. Output of time series for

selected gridblock or connection is done by the processor that the gridblock or connection belong to.

3.4 Communication between Processors

Communication or exchange between processors is an essential component of the parallel-computing algorithm. In addition to the communication that takes place inside the AZTEC routine to solve the linear system, the communication between neighbor processors is necessary to update the primary and secondary variables during Newton iterations. These tasks are implemented in a subroutine, which performs the exchanges of vector elements corresponding to the *external* set of the gridblocks between neighboring processors. To minimize the communication cost, only the primary variables of the *external* blocks are passed between the neighboring processors. The secondary variables for the *external* set are computed on the appropriate distributed processors after communications. Detailed discussion of the scheme used for data exchange is given in Elmroth et al. (2001), and Zhang and Wu (2006).

3.5 Program Structure and Flow Chart

TMVOC-MP V1.0 has a program structure very similar to the original TMVOC, except that the parallel version solves a problem using multiple processors. We implement dynamic memory management, modules, array operations, matrix manipulation, and other FORTRAN 90 features in the parallel code. In particular, the message-passing interface (MPI) library is used for message passing. Another important modification to the original code is in the time-step looping subroutine CYCIT. This subroutine now provides the general control of problem initialization, grid partitioning, data distribution, memory requirement balancing among all processors, time stepping, and output options. In summary, all data input and output is carried out through the master processor. The most time-consuming tasks (assembling the Jacobian matrix, updating thermophysical parameters, and solving the linear equation systems) are distributed over all processors. In addition, memory requirements are also distributed to all processors. Figure 3.2 gives an abbreviated overview of the program flow chart.

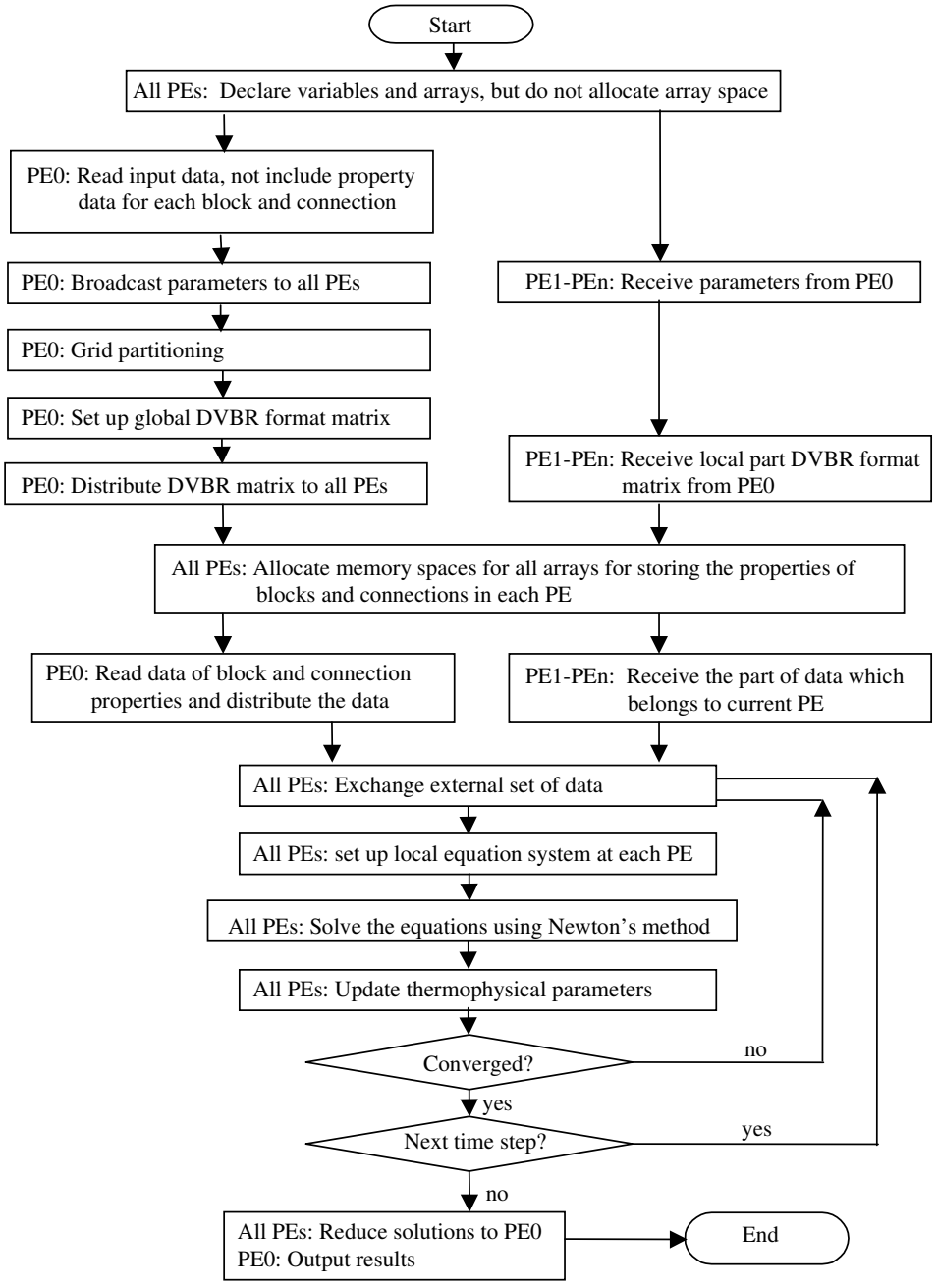


Figure 3.2 Flow Chart of TMVOC-MP V1.0 (PE0: master processor; PE1-PEn: slave processors)

4. DESCRIPTION OF INPUT FILES

4.1 Preparation of Input Data

Input of TMVOC-MP V1.0 is provided through a fixed-name file (INFILE), organized into data blocks, labeled by five-character keywords. The format of "INFILE" is the same as that of the input file for TMVOC. TMVOC-MP may also receive input data through additional optional input files.

4.2 Special Input Requirements for TMVOC-MP V1.0

In some cases, TMVOC-MP needs to be run in batch mode. To run a job in batch mode, the user submits a job to a computer and the computer schedules the job in a queue. When the requested number of processors is available, the job will be run. In batch running mode, all data are provided in input files, since run-time communication with computer is not feasible. The input files include:

INFILE

This file is in standard TOUGH2 (TMVOC) input file format. In the file, data are organized in blocks that are defined by five-character keywords typed in Columns 1-5. The first record must be a problem title of up to 80 characters. The last record usually is ENDCY. Data records beyond ENDCY will be ignored. The most important data blocks include ROCKS, MULTI, PARAM, ELEME, CONNE, INCON, and GENER. All input data in INFILE are in fixed format and standard metric (SI) units. Detailed information about this file format can be found in Pruess and Battistelli (2002).

The data blocks of ELEME, CONNE, GENER and INCON can be extremely large. It is good practice to provide these blocks through separate data files. Input for the ELEME and CONNE blocks can be provided through an ASCII-formatted MESH file, or alternatively through two binary files: MESHA and MESHB. The two binary files are intermediate files which are created by TMVOC-MP at its first run for a model. If MESHA and MESHB exist in the working folder, the code will ignore MESH file and

read information directly from these two files. Once the mesh file is changed, MESHA and MESHB must be deleted from the working folder in order for the changes to take effect. The two files have completely different data formats from the ELEME and CONNE blocks. The detailed formats are given in the following.

MESHA, MESHB

The purpose of replacing file MESH (or blocks ELEME and CONNE in input file) with MESHA and MESHB is to reduce the memory requirements for the master processor and to enhance I/O efficiency. Both MESHA and MESHB are binary files. These two files contain all information provided by file MESH.

The file MESHA is written (to file unit 20 which was opened as an unformatted file) in the following sequence:

write(20) NEL
write(20) (EVOL(iI),iI=1,NEL)
write(20) (AHT(iI),iI=1,NEL)
write(20) (pmx(iI),iI=1,NEL)
write(20) (gcoord(iI,1),iI=1,NEL)
write(20) (gcoord(iI,2),iI=1,NEL)
write(20) (gcoord(iI,3),iI=1,NEL)
write(20) (DEL1(iI), iI=1,NCON)
write(20) (DEL2(iI), iI=1,NCON)
write(20) (AREA(iI), iI=1,NCON)
write(20) (BETA(iI), iI=1,NCON)
write(20) (sig(iI), iI=1,NCON)
write(20) (ISOX(iI),iI=1,NCON)
write(20)(ELEM1(iI), iI=1,NCON)
write(20)(ELEM2(iI), iI=1,NCON)

where

NEL	Total gridblock number, in 4-byte integer.
NCON	Total connection number, in 4-byte integer.
EVOL	Element volume (m^3), in 8-byte real.
AHT	Interface area (m^2) for heat exchange with semi-infinite confining beds, in 8-byte real.
pmx	Permeability modifier, in 8-byte real.
gcoord(*,1-3)	Cartesian coordinates (X,Y,X) of gridblock center, in 8-byte real.
DEL1, DEL2	Distance (m) from first and second element, respectively, to their common interface, in 8-byte real.
AREA	Interface area (m^2), in 8-byte real.
BETA	Cosine of the angle between the gravitational acceleration vector and the line between two elements, in 8-byte real.
sig	Radiant emittance factor for radiative heat transfer, in 8-byte real.
ISOX	Specify absolute permeability for the connection, in 4-byte integer.
ELEM1	Code name for the first element of a connection, in 8 characters.
ELEM2	Code name for the second element of a connection, in 8 characters.

The file MESHB is written (to file unit 30, unformatted) in the following sequence:

write(30) NCON,NEL
write(30) (ELEM(iI),iI=1,NEL)
write(30) (MA12(iI),iI=1,NEL)
write(30) (NEX1(iI),iI=1,NCON)
write(30) (NEX2(iI),iI=1,NCON)

where

ELEM	Code name of the element, in 8 characters.
MA12	Material identifier of the element, in 5 characters.

NEX1, NEX2 First and second element number of the connection, in 4-byte integer.

For a detailed explanation of these parameters, the reader may refer to the *TOUGH2 User's Guide, Version 2.0* (Pruess et al., 1999).

After the first run of a simulation, the material name array MA12 will be replaced by the material index (array MATX, in 4-byte integer). In addition, NEL will be replaced by –NEL to inform the program of the replacement. Through this replacement, the material index search is avoided for subsequent runs with the same computational grid.

MESHA and MESHB can also be created directly from the MESH file through a preprocessing program. For extremely large problems, generation of MESHA and MESHB is the bottleneck of memory requirement for TMVOC-MP. By using a preprocessing program, the bottleneck for memory requirement can be avoided.

PARAM.prm

PARAM.prm is an optional file providing computational parameters for TMVOC-MP. If this file does not exist in the working folder, the code will take default parameters. These parameters are needed if the user wants to try different options for the parallel linear solver, partitioning algorithms, and other program options. Following is an example of the file.

1008680, 4000000, 0
AZ_solver AZ_bicgstab
AZ_scaling AZ_BJacobi
AZ_precond AZ_dom_decomp
AZ_tol 1e-6
AZ_overlap 0
AZ_max_iter 250
AZ_conv AZ_rhs

AZ_subdomain_solve AZ_ilut

AZ_output AZ_none

EE_partitioner METIS_Kway

EE_output 100

END OF INPUTS

The three numbers in the first line are:

MNEL : Estimated total gridblocks, should be larger than the number of the actual model gridblocks.

MCON: Estimated total connections, should be larger than the number of the actual model connections.

PartReady: A parameter to inform the program that domain partitioning was done by a preprocessing program or will be done inside the TMVOC-MP. If PartReady=0, TMVOC-MP will perform domain partitioning as part of the execution. If PartReady>0, the code will not perform domain partitioning and partition data will be read directly from file "part.dat". Default PartReady=0.

The default values of MNEL and NCON are 500,000 and 2,300,000 respectively. The two parameters are required only in generating MESHA and MESHB and when a model has more than 500,000 gridblocks or 2,300,000 connections.

From the second line onward, each line provides a single parameter. These parameters give options for running the Aztec and METIS packages, and SAVE file output frequency control. The parameters can be in any order. If one parameter is not present, its default value will be used. Each line in the file consists of two terms. The first term is the parameter name and the second term is its value. Detailed content of the parameters is discussed below.

AZ_solver Specifies solution algorithm, available solvers:

AZ_cg conjugate gradient (only applicable to symmetric positive definite matrices).

AZ_gmres restarted generalized minimal residual.

AZ_cgs conjugate gradient squared.

AZ_tfqmr transpose-free quasi-minimal residual.

AZ_bicgstab bi-conjugate gradient with stabilization.

AZ_lu sparse direct solver (single processor only).

AZ_scaling Specifies scaling algorithm, user can select from:

AZ_none no scaling.

AZ_Jacobi point Jacobi scaling.

AZ_BJacobi Block Jacobi scaling where the block size corresponds to the VBR blocks.

Az_row_sum scale each row so the magnitude of its elements sum to 1.

AZ_sym_diag symmetric scaling so diagonal elements are 1.

AZ_sym_row_sum symmetric scaling using the matrix row sums.

AZ_precond Specifies preconditioner. Available selections include:

AZ_none no preconditioning.

AZ_Jacobi k step Jacobi (or block Jacobi for DVBR matrices)

AZ_Neumann Neumann series polynomial.

AZ_ls least-squares polynomial.

AZ_sym_GS non-overlapping domain decomposition (additive Schwarz) k step symmetric Gauss-Seidel.

AZ_dom_decomp domain decomposition preconditioner (additive Schwarz).

AZ_tol Specifies tolerance value used in conjunction with convergence tests.

AZ_type_overlap Determines how overlapping subdomain results are combined when different processors have computed different values for the same unknown.

AZ_standard the resulting value of an unknown is determined by the processor owning that unknown.

AZ_symmetric average the results obtained from different processors corresponding to the same unknown.

AZ_overlap Determines the submatrices factored with the domain decomposition algorithms.

AZ_max_iter Maximum number of iterations.

AZ_conv Determines the residual expression used in convergence check and printing. Available selections include: AZ_r0, AZ_rhs, AZ_Anorm, AZ_noscaled, AZ_sol, AZ_weighted.

AZ_subdomain_solve Specifies the solver to use on each subdomain when AZ_precond is set to AZ_dom_decomp, available selections include: AZ_lu, AZ_ilut, AZ_ilu, AZ_rilu, AZ_bilu, and AZ_icc.

AZ_reorder Determines whether RCM reordering will be done in conjunction with domain decomposition incomplete factorizations, 1 yes; 0 no.

AZ_pre_calc Indicates whether to use factorization information from previous calls to AZ_solve, three selections: AZ_calc, AZ_recalc, and AZ_reuse.

AZ_output Specifies information to be printed, available selections: AZ_all, AZ_none, AZ_warnings, AZ_last, and >0.

EE_partitioner Specifies the partitioner to be used, user can select partitioners from:

- METIS_Kway uses the multilevel *k*-way partitioning algorithm. The objective of this partitioning method is to minimize the edgecut. It should be used to partition a graph into a large number of partitions (greater than 8).
- METIS_Vkway uses the multilevel *k*-way partitioning algorithm. The objective of this partitioning method is to minimize the total communication volume.
- METIS_Recursive uses multilevel recursive bisection. The objective of this partitioning method is to minimize the edgecut, this function should be used to partition a graph into a small number of partitions (less than 8).

EE_output Output control for solution results. The SAVE file will be written

every EE_output time steps. If EE_output=0, no SAVE file will be written out until last time step.

Note that there are additional options or parameters for Aztec parallel linear equation solver. For further discussion, readers may refer to Tuminaro et al. (1999). Table 3 presents the default values used in TMVOC-MP.

Table 3. Default values of the options and parameters

Parameters or options	Values
AZ_solver	AZ_bicgstab
AZ_scaling	AZ_Bjacobi
AZ_pecond	AZ_dom_decomp
AZ_tol	$1x10^{-6}$
AZ_type_overlap	AZ_standard
AZ_max_iter	500
AZ_conv	AZ_r0
AZ_subdomain_solve	AZ_ilut
AZ_reorder	1
AZ_pre_calc	AZ_calc
AZ_output	AZ_none
EE_partitioner	METIS_Kway
EE_output	200

part.dat

If parameter *PartReady* in "PARAL.prm" has a value larger than 0, TMVOC-MP will read file "part.dat" during run-time. The file contains domain-partitioning results. It is read by following code:

```
open (unit=50,file='part.dat',form='formatted',status='old')
read(50,133) nparts,edgecut,NEL
```

```
      read(50,144) (part(iI),iI=1,NEL)
133   format(3I10)
144   format(10I8)
```

where

nparts	Number of parts that the domain has been partitioned to. It should equal to the number of processors used for solving the problem.
edgecut	Number of cut edges.
nel	Total number of elements or gridblocks in the domain.
part	Partitioning result of each gridblock. The integer value indicates which processor the gridblock belongs to.

File "part.dat" may be created through a preprocessing program based on the user's special requirements, e.g. based on physical boundaries of the modeling domain for grid partitioning.

4.3 Output from TMVOC-MP V1.0

TMVOC-MP V1.0 produces a variety of output, most of which can be controlled by the user. The output file for TMVOC is replaced by two files in TMVOC-MP V1.0, naming OUTPUT and OUTPUT_DATA, and the file format is maintained. The first file provides problem initialization and time-stepping information, and the second file gives a complete element-by-element and/or connection-by-connection report of thermodynamic state variables, fluxes and other important parameters. Information written in the initialization phase to the OUTPUT file includes parameter settings in the main program for dimensioning of problem-size dependent arrays, and disk files in use. This is followed by documentation on settings of the MOP-parameters for choosing program options, and on the EOS-module. At the end of the OUTPUT file, parallel computing information is documented. During execution, TMVOC-MP can optionally generate a brief message for Newtonian iterations and time steps based on the specifications defined in the INFILE.

5. SAMPLE PROBLEM

5.1 Model Setup

This is sample problem 7 from the TMVOC user's guide (Pruess and Battistelli, 2002). It demonstrates the application of TMVOC-MP to a flow problem with realistic features as would be encountered in actual field situations. We consider a multi-component NAPL spill in the unsaturated zone that is followed by redistribution of the NAPL plume within the unsaturated zone and along the water table. Subsequent remediation operations involve VOC recovery from the unsaturated zone by soil vapor extraction close to the spill area and NAPL extraction by pumping from the saturated zone. In this simulation, the model domain was simplified as a two-dimensional vertical section. Figure 5.1 shows the cross-section. Detailed discussion of the problem setup can be found in Pruess and Battistelli (2002).

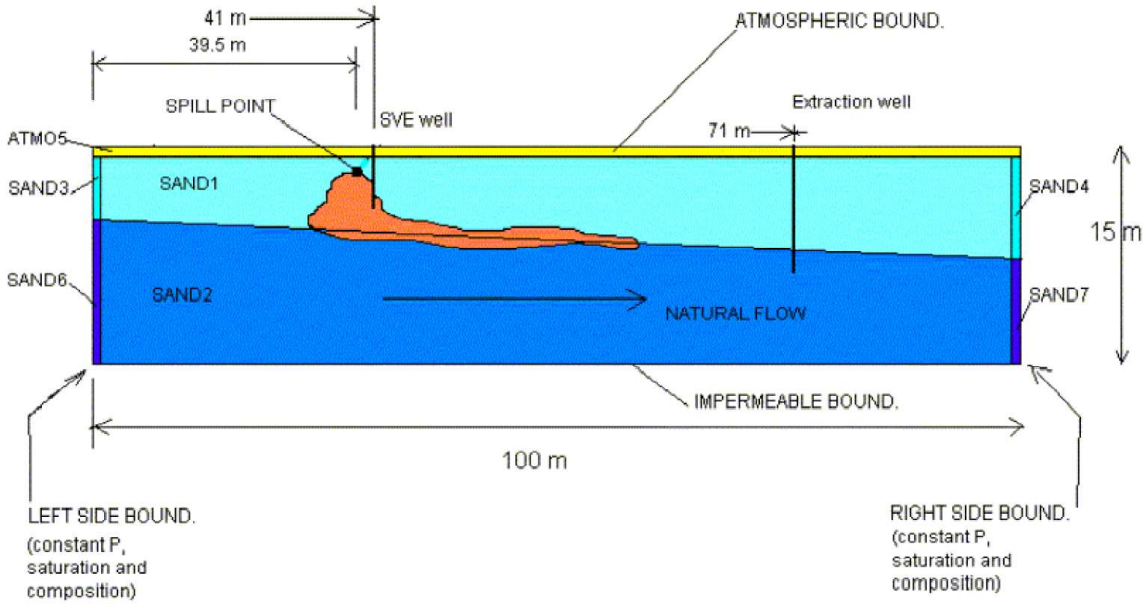


Figure 5.1 Conceptual model of 2-D vertical section problem for simulating a multicomponent NAPL spill in the unsaturated zone. Rock domain assignments and boundary conditions are shown together with the position of the spill point, a soil vapor extraction well, and a downstream extraction well (From Pruess and Battistelli [2002]).

The 100 m×15 m simulation domain was discretized into 40 columns and 17 layers with a total of 680 gridblocks. Vertical grid spacing is 0.5 m in the unsaturated zone and in the upper part of the unconfined aquifer. Horizontal grid spacing is 1 m in the region affected by the NAPL spill and the subsequent soil vapor extraction, and gradually increases to 10 m near the left and right boundaries. Depth to the water table is 5 meter on the left side of the section and 5.5 m on the right side.

The problem is run in four segments, (1) generation of steady flow prior to introduction of NAPL, (2) NAPL spill in the unsaturated zone, (3) redistribution of NAPL, and (4) extraction simultaneously in the saturated and unsaturated zones. The MESH and 4 input files (r2dl1, r2dl2, r2dl3, and r2dl4) for these simulations are directly copied from TMVOC sample problem 7, because TMVOC-MP accepts the same input file formats as TMVOC.

5.2 Running Simulations

A parallel simulation can be run on multi-core PCs, Linux clusters, supercomputers, or other multi-CPU computers. On different platforms or MPI installations, simulations may be run in slightly different ways. Following is the procedure for running a simulation on a typical Linux cluster with OPEN-MPI installed.

(1) Create a working directory. Copy the executable file to the working directory (or run the code by specifying the path where the executable is located).
(2) Copy input file(s) (MESH, r2dl1, r2dl2, r2dl3, and r2dl4) to the working directory. Rename the input file to "INFILE" (case sensitive); e.g., for the initial simulation of generating steady flow, we need to rename file "r2dl1" to "INFILE". Furthermore, the parallel version: (a) does not support simplified mesh format (NSEQ is not 0); (b) does not support inactive elements (inactive elements are automatically replaced by large-volume gridblocks internally); (c) requires removal of previously saved MESHA and MESHB files if MESH was changed.
(3) A host file may be needed. The host file contains a list of computer nodes that can be used for computations.

(4) For a cluster the computational tasks can not be automatically scheduled, and we need to monitor the usage of CPUs on the cluster. Rearrange the sequence of nodes listed in the host file by putting the idle nodes at the top of the list, and busy nodes at the bottom. MPI assigns jobs to CPUs from the top to the bottom of the list. If several programs share a CPU computing resource, execution of TMVOC-MP could be extremely slow.

(5) An optional input file "PARAL.prm" may be needed for extremely large models (more than 0.5 million grid blocks), using a specific domain partitioning algorithm, needing different Aztec solver options, or needing controlling the SAVE file output frequency. A template for this file is included in the installation package.

(6) Under the working directory, type:

"mpirun --hostfile ~/*host_file* –np *x* ./tmvoc_mp" to run the simulation, where *x* is the number of CPUs being used.

5.3 Simulation Results and Code Performance

The first segment of the simulation is to generate ambient steady-state flow under gravity-capillary equilibrium conditions prior to the contamination event. File "r2dl1" is the input file for this simulation segment. The input file includes soil properties and initial conditions of different portions of the flow domain (see Figure 5.1). The INCON data for the left and right boundary columns were generated from 1D column models and atmospheric pressure was assigned at the top boundary as fixed boundary condition. A net infiltration of 10 cm/year of water with a density of 998.32 kg/m^3 (at T = 20 °C and ambient pressure) is applied at the top of the model domain.

Simulations were run on a Linux cluster equipped with AMD Athlon(tm) MP 2100+ processors. Both TMVOC and TMVOC-MP run 120 time steps to reach steady-state solutions, using 42 and 17 seconds (with 2 CPUs), respectively. Simulation results from the two codes are identical.

The second run segment simulates the process of NAPL spill in the unsaturated zone. In the input file "r2dl2", data for six VOCs are introduced through data block CHEMP. Altogether 8 components (water, air, and 6 VOCs) are considered. The thermodynamic conditions for the two-component water-air system obtained in the first run segment are used as the initial conditions for this run. The SAVE file from the first run segment is renamed INCON, and simulation time and time step counter in the file are reset to zero. Water sources for infiltration at the land surface are kept unchanged. The six VOCs with different injection rates are applied to a gridblock at x=39.5 m and z=4.25 m for a period of 1 year.

Figure 5.2 shows distribution of total VOC mass fractions in the aqueous phase at the end of the NAPL spill period simulated by the parallel simulator. By comparing the TMVOC-MP predicted pattern of VOC mass fraction distribution with the contour plot of the Figure 10.7.8 in Pruess and Battistelli (2002), we can find both codes produced almost identical results.

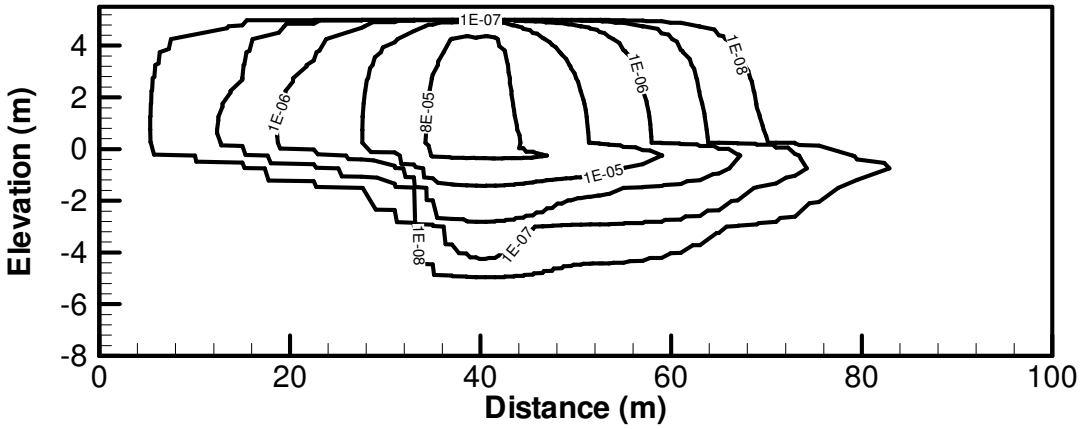


Figure 5.2. Contour plot of total VOC mass fraction in the aqueous phase at the end of the NAPL spill period simulated by TMVOC-MP.

The performance of the parallel code for different numbers of processors was investigated (see Table 5.1). For a different number of processors, the simulation may require a different number of time steps. This is due to changes in the convergence rate

when using different domain partitioning, which in turn will impact the automatic time step selection. Because this problem has only 680 gridblocks, it only shows modest speedups from parallel computing.

Table 5.1. Comparison of performance for the second segment run using different numbers of processors

Number of processors	1*	2	4	6
Time for updating thermophysical parameters and assembling Jacobian matrix (second)		166	56	49
Time for solving linear equations(second)		445	156	133
Total execution time (second)	779	612	212	182
Total time steps	120	167	117	125
Average iterations for solving linear equations		20	23.7	26.5

** run with the original TMVOC code, with exactly the same input files and on the same computer.*

The third run segment simulates redistribution of the VOCs for a period of 1 year after the end of the NAPL spill, using the SAVE file written at the end of the spill period as INCON. We again replace the second-last line of file SAVE with a blank line, to reset time and iteration counters. The input file for this run is "r2dl3", which needs to be renamed "INFILE". Simulation for this segment again was done by both sequential and parallel codes. The simulation results presented in Pruess and Battistelli (2002) can be reproduced by both codes. Figure 5.3 shows TMVOC-MP simulated distribution of total VOC mass fractions in the aqueous phase at 1 year after the end of the NAPL spill, and Figure 5.4 shows the simulated NAPL saturations for the same time. Comparing these two figures to the Figures 10.7.11 and 10.7.12 in Pruess and Battistelli (2002), it is seen that the simulation results from both codes agree well.

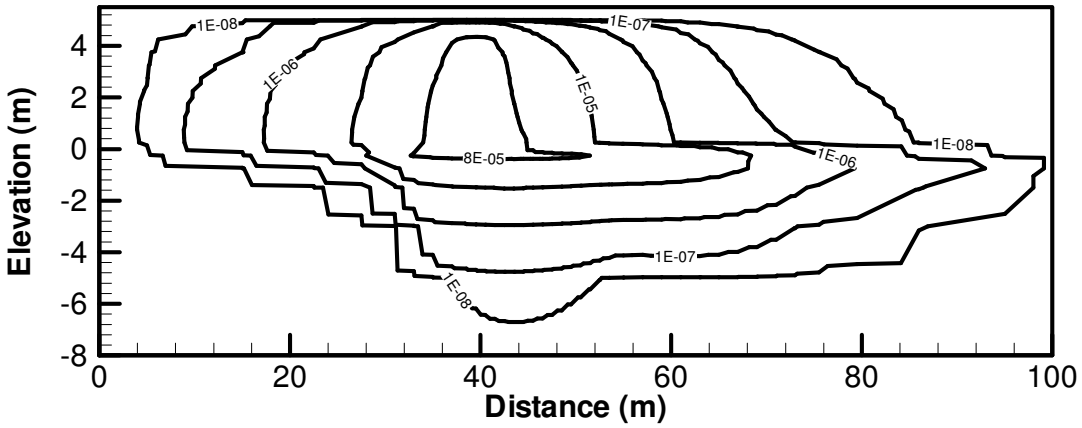


Figure 5.3 Contour plot of total VOC mass fraction in the aqueous phase at the end of the NAPL redistribution period.

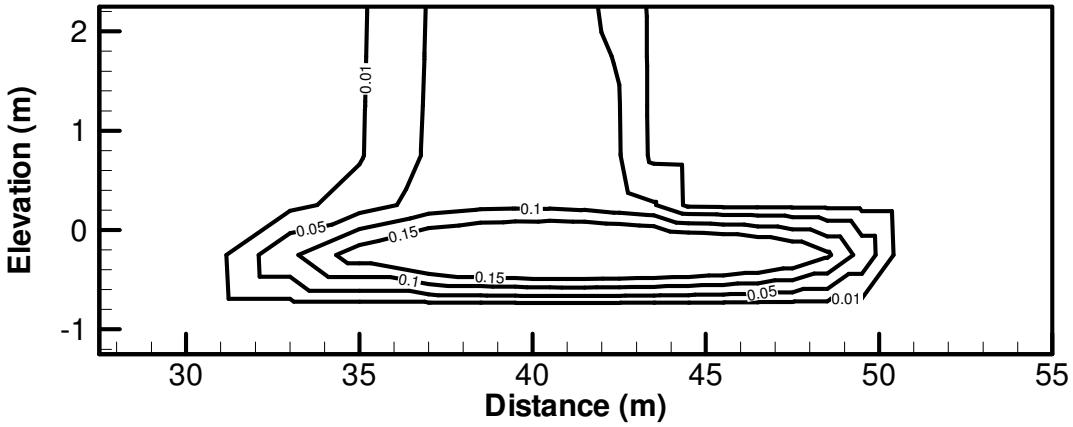


Figure 5.4 NAPL saturations at the end of the redistribution period simulated by TOUGH2-MP.

The performance of the parallel code was also investigated through using different number of processors to solve this problem. Table 5.2 shows the performance of the parallel code by comparing to performance of the original sequential code. Speedup for parallel computing is quite significant.

Table 5.2. Comparison of performance for the third segment run using different numbers of processors

Number of processors	1*	2	4	6
Time for updating thermophysical parameters and assembling Jacobian matrix (second)		109	63	48
Time for solving linear equations(second)		242	136	99
Total execution time (second)	831	351	199	148
Total time steps	202	176	193	188
Average iterations for solving linear equations		17.9	19	20.7

** run with the original TMVOC code, with exactly the same input files and on the same computer.*

The final segment of this problem models extraction of VOCs. The extraction was simulated by specifying a soil vapor extraction well, which produces on deliverability against a sandface pressure of 0.9×10^5 Pa from grid layers 4 through 8. Another extraction well produces from the saturated zone. The final SAVE file written by the redistribution simulation is used as initial conditions for the current run, again renaming it "INCON" and resetting time and iteration counters. The input file "r2dl4" is renamed "INFILE" for the simulation. At the end of the simulation, 470.48 kg of VOCs, corresponding to 54.3 % of the original inventory, are remaining in the flow system. Approximately 99.8 % of this inventory is present as a free NAPL phase, 0.18 % is dissolved in water, and 0.02 % is in the gas phase. The spatial extent of VOC contamination has been much reduced. All these results agree closely with the simulation using the sequential code. The parallel code demonstrates similar speedup as for previous run segments.

6. CONCLUDING REMARKS

A parallel simulator (TMVOC-MP) for three-phase non-isothermal flow of water, gas, and a multicomponent mixture of volatile organic chemicals (VOCs) in multidimensional heterogeneous porous/fractured media has been developed. This parallel simulator uses fully implicit time differencing and solves large, sparse linear systems arising from discretization of the partial differential equations for mass and energy balance. TMVOC-MP was developed based on the sequential TMVOC code. It is written in Fortran 90 with MPI for parallel implementation.

The parallel simulator retains all the process modeling capabilities of TMVOC. Thus it is a versatile numerical simulator for multi-phase flows involving aqueous, nonaqueous, and gas phases, which may include several volatile organic compounds, along with water and a variety of non-condensable gases. All possible phase combinations in a water-air-NAPL system are treated, including single-phase gas, aqueous, NAPL; two-phase gas-aqueous, gas-NAPL, and aqueous-NAPL; and three-phase gas-aqueous-NAPL. Like TMVOC, TMVOC-MP has capabilities for modeling flow and transport processes in variably saturated media at ambient conditions, as well as under non-isothermal conditions such as electric resistance heating or steam flooding.

TMVOC-MP is more efficient than its sequential counterpart, especially for larger problems. It provides a powerful tool for tackling larger-scale and more complex problems than can be solved currently by sequential codes. The new simulator will enhance modeling capacity in terms of model size and simulation time by 1-3 orders of the magnitude. The code is designed to be easy to use and little learning is necessary for experienced TMVOC users.

ACKNOWLEDGEMENTS

The authors would like to thank Lehua Pan for his review of this report. Development of TMVOC-MP code was partially supported by the Taisei Corporation, Japan, under Contract No. DE-AC02-05CH11231 with the U.S. Department of Energy.

REFERENCES

Elmroth E, Ding C, and Wu YS. High Performance computations for Large-Scale Simulations of Subsurface Multiphase Fluid and Heat Flow, *The Journal of Supercomputing,* **18**(3), pp. 233-258, 2001.

Karypsis, G. and Kumar, V.: "A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, V4.0," Technical Report, Department of Computer Science, University of Minnesota, 1998.

Message Passing Formum: "A message-passing interface standard," International Journal of Supercomputing Applications and High performance Computing, **8**(3-4), 1994.

Narasimhan, T. N. and Witherspoon P. A.: "An integrated finite difference method for analyzing fluid flow in porous media," *Water Resour. Res*. (1976), **12**, 1, 57-64.

Pruess, K., TOUGH2, *a general-purpose numerical simulator for multiphase fluid and heat flow, Report LBL-29400,* Lawrence Berkeley Laboratory, Berkeley, CA, May 1991.

Pruess, K.; Oldenburg, C.; Moridis, G., *TOUGH2 User's Guide, Version 2.0,* Report LBL-43134*,* Lawrence Berkeley Laboratory, Berkeley, CA, November 1999. MOL.20020829.0120

Pruess, K., and Battistelli, A., TMVOC-A numerical simulator for three-phase non-isothermal flows of multicomponent hydrocarbon mixtures in saturated-unsaturated heterogeneous media, LBNL-49375, Lawrence Berkeley Laboratory, Berkeley, CA, 2002.

Tuminaro, R. S., Heroux, M., Hutchinson, S. A., and Shadid J. N., Official Aztec user's guide, Ver 2.1, Massively Parallel Computing Research Laboratory, Sandia National Laboratories, Albuquerque, NM, 1999.

Wu, Y. S., K. Zhang, C. Ding, K. Pruess, E. Elmroth, and G. S. Bodvarsson, 2002, An

efficient parallel-computing scheme for modeling nonisothermal multiphase flow and multicomponent transport in porous and fractured media, ADVANCES IN WATER RESOURCES, 25, 243-261.

Zhang K, Wu YS, Ding C, Pruess K, and Elmroth E. Parallel Computing Techniques for Large-Scale Reservoir Simulation of Multi-Component and Multiphase Fluid Flow, Paper SPE 66343, *Proceedings of the 2001 SPE Reservoir Simulation Symposium*, Houston, Texas, 2001.

Zhang K. and Wu YS, 2006, Enhancing Scalability and Efficiency of the TOUGH2_MP for Linux Clusters, LBNL-60953, proceedings of TOUGH symposium 2006, Berkeley, CA.