Parallelization of the Reactive Transport Code MIN3P-THCm

NWMO-TR-2015-23 October 2015

Danyang Su[1], K. Ulrich Mayer[1] and Kerry T. B. MacQuarrie[2]

[1]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia
[2]Department of Civil Engineering, University of New Brunswick

nwmo
NUCLEAR WASTE MANAGEMENT ORGANIZATION
SOCIÉTÉ DE GESTION DES DÉCHETS NUCLÉAIRES

Nuclear Waste Management Organization
22 St. Clair Avenue East, 6th Floor
Toronto, Ontario
M4T 2S3
Canada

Tel: 416-934-9814
Web: www.nwmo.ca

Parallelization of the Reactive Transport Code MIN3P-THCm

NWMO-TR-2015-23

October 2015

Danyang Su[1], K. Ulrich Mayer[1], Kerry T. B. MacQuarrie[2]
[1]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia
[2]Department of Civil Engineering, University of New Brunswick

Document History

Title:	Parallelization of the Reactive Transport Code MIN3P-THCm		
Report Number:	NWMO-TR-2015-23		
Revision:	R000	Date:	October 2015
[1]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia [2]Department of Civil Engineering, University of New Brunswick			
Authored by:	Danyang Su[1]		
Verified by:	Kerry T.B. MacQuarrie[2]		
Approved by:	K. Ulrich Mayer[1]		
Nuclear Waste Management Organization			
Reviewed by:	Tammy Yang		
Accepted by:	Mark Jensen		

ABSTRACT

Title: **Parallelization of the Reactive Transport Code MIN3P-THCm**
Report No.: **NWMO-TR-2015-23**
Author(s): Danyang Su[1], K. Ulrich Mayer[1] and Kerry T.B. MacQuarrie[2]
Company: [1]Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia
[2]Department of Civil Engineering, University of New Brunswick
Date: October 2015

Abstract
Reactive transport modelling can be time consuming and memory-intensive, especially for large-scale, long-term simulations with a large number of chemical components and interactions. The objective of this research was to develop a parallel version of MIN3P-THCm, a general purpose multicomponent reactive transport code for variably saturated porous media. The resulting program, entitled ParMIN3P-THCm, is able to deal with the significant computational burden of reactive transport simulations involving large spatial scales and long time frames and can be run efficiently on machines ranging from desktop PCs, shared-memory workstations, to distributed-memory supercomputers.

Parallelization of MIN3P-THCm (ParMIN3P-THCm) was achieved through the domain decomposition method based on PETSc (Portable Extensible Toolkit for Scientific Computation) libraries. PETSc is also used as the parallel solver package, and for data structure and message communication. A hybrid MPI and OpenMP parallel programming approach is implemented in the code to take advantage of leadership-class supercomputers that combine both shared memory and distributed memory architectures. Features of the code include a modular input file, parallel configuration file, and parallel I/O, with potential expansibility to incorporate additional features in the near feature such as high-performance I/O using parallel HDF5, as well as parallel multigrid and unstructured grid methods. ParMIN3P-THCm has been developed from the ground up for parallel scalability and has been run using up to 768 processors with problem sizes up to 100 million unknowns. The code has demonstrated excellent speedup for reactive transport simulation problems using 8 processors on a local shared-memory workstation, 128 processors on the WestGrid supercomputer using MPI parallelization and 768 processors on the WestGrid supercomputer using hybrid MPI-OpenMP parallelization. The code has shown strong scalability in modelling large-scale reactive transport problems.

TABLE OF CONTENTS

Page

ABSTRACT iii

1. INTRODUCTION 1

1.1 PREVIOUS RESEARCH OF REACTIVE TRANSPORT CODE PARALLELIZATION 2
1.2 PROJECT OBJECTIVE 3
1.3 REPORT ORGANIZATION 4

2. PARALLEL FRAMEWORK 4

2.1 CODE ARCHITECUTRE 4
2.1.1 Target System 5
2.1.2 Development Tools and Libraries 5
2.2 PARALLEL ARCHITECTURES 6
2.2.1 Shared-memory Multiprocessor Architecture 6
2.2.2 Distributed-memory Multiprocessor Architecture 6
2.2.3 Hybrid Distributed-shared-memory Multiprocessor Architecture 7
2.2.4 Parallel Levels 7
2.2.5 Parallel Levels in ParMIN3P-THCm 8
2.3 PARALLEL IMPLEMENTATION 8
2.3.1 Domain Decomposition 8
2.3.2 Computational Workflow 9
2.3.3 Shared-Memory Parallel Implementation 11
2.3.4 Distributed-Memory Parallel Implementation 13
2.3.5 Hybrid Parallel Implementation 15
2.4 PARALLEL MODULES 16
2.4.1 Global and Local Numbering 16
2.4.2 Parallel Matrix and Right Hand Side Assembly 18
2.4.3 Parallel Linear Solver 21
2.4.4 Parallel Input and Output 22
2.4.5 Summary of Parallel Modules 24

3. PARALLEL PERFORMANCE 24

3.1 RUNTIME PROFILING TOOL 24
3.2 COMPUTER ARCHITECTURE FOR PERFORMANCE TESTING 24
3.2.1 Shared-memory Architecture 24
3.2.2 Distributed-memory Architecture 25
3.2.3 Hybrid Distributed-shared-memory Architecture 25
3.3 CASES FOR PARALLEL PERFORMANCE TESTING 25
3.3.1 Case I: Complex cement/clay interactions 25
3.3.1.1 Case Introduction 25
3.3.1.2 Model Discretization 26
3.3.1.3 Sample Results 27
3.3.1.4 Runtime Profiling 27
3.3.2 Case II: Uranium Remediation by Lactate Injection 28
3.3.2.1 Case Introduction 28

3.3.2.2 Model discretization 29
3.3.2.3 Sample Results 31
3.3.2.4 Runtime Profiling 31
3.3.3 Case III: Flow and Reactive Transport in a Hypothetical Sedimentary Basin 32
3.3.3.1 Case Introduction 32
3.3.3.2 Model Discretization 33
3.3.3.3 Sample Results 34
3.3.3.4 Runtime Profiling 35
3.4 SHARED-MEMORY PARALLEL PERFORMANCE 36
3.4.1 Case I: Complex Cement/Clay Interactions 36
3.4.1.1 Solver Statistics 36
3.4.1.2 Parallel Speedup 37
3.4.2 Case II: Uranium Remediation by Lactate Injection 37
3.4.2.1 Solver Statistics 37
3.4.2.2 Parallel Speedup 38
3.4.3 Case III: Flow and Reactive Transport in a Hypothetical Sedimentary Basin 39
3.4.3.1 Solver Statistics 39
3.4.3.2 Parallel Speedup 39
3.4.4 Summary of Shared-memory Parallel Performance 40
3.5 DISTRIBUTED-MEMORY PARALLEL PERFORMANCE 40
3.5.1 Case I: Complex Cement/clay Interactions 41
3.5.1.1 Solver Statistics 41
3.5.1.2 Parallel Speedup 41
3.5.2 Case II: Uranium Remediation by Lactate Injection 42
3.5.2.1 Solver Statistics 42
3.5.2.2 Parallel Speedup 43
3.5.3 Case III: Flow and Reactive Transport in a Hypothetical Sedimentary Basin 43
3.5.3.1 Solver Statistics 43
3.5.3.2 Parallel Speedup 44
3.5.4 Summary of Distributed-Memory Parallel Performance 45
3.6 HYBRID PARALLEL PERFORMANCE 45
3.6.1 Case I: Complex Cement/Clay Interactions 45
3.6.1.1 Solver Statistics 45
3.6.1.2 Parallel Speedup 46
3.6.2 Case II: Uranium remediation by lactate injection 48
3.6.2.1 Solver Statistics 48
3.6.2.2 Parallel Speedup 49
3.6.3 Case III: Flow and reactive transport in a hypothetical sedimentary basin 51
3.6.3.1 Solver Statistics 51
3.6.3.2 Parallel Speedup 51
3.6.4 Summary of Hybrid Distributed-Shared-Memory Parallel Performance 53
3.7 SYSTEM SCALABILITY 53

4. SUMMARY AND CONCLUSIONS 56

ACKNOWLEDGEMENTS 58

REFERENCES 59

LIST OF TABLES

Page

Table 1: Linear Solvers in PETSc 22
Table 2: Parallel Modules Included in ParMIN3P-THCm 24
Table 3: Numerical Parameters Used in Case I 26
Table 4: Runtime Percentage Distribution for Case I 28
Table 5 Numerical Parameters Used in Case II 30
Table 6: Runtime Percentage Distribution for Case II 32
Table 7: Numerical Parameters Used in Case III 34
Table 8: Runtime Percentage Distribution for Case III 36
Table 9: Solver and Runtime Statistics of Shared-memory Parallelization for Case I* 36
Table 10: Solver and Runtime Statistics of Shared-memory Parallelization for Case II* 38
Table 11: Solver and Runtime Statistics of Shared-memory Parallelization for Case III* 39
Table 12: Solver and Runtime Statistics of Distributed-memory Parallelization for Case I* 41
Table 13: Solver and Runtime Statistics of Distributed-memory Parallelization for Case II* 42
Table 14: Solver and Runtime Statistics of Distributed-memory Parallelization for Case III* 44
Table 15: Solver and Runtime Statistics for Hybrid Parallelization for Case I* 46
Table 16: Solver and Runtime Statistics for Hybrid Parallelization for Case II 48
Table 17: Solver and Runtime Statistics for Hybrid Parallelization for Case III 51

LIST OF FIGURES

Page

Figure 1: Shared-memory Multiprocessor Architecture 6
Figure 2: Distributed-memory Multiprocessor Architecture 7
Figure 3: Hybrid distributed-shared-memory Multiprocessor Architecture 7
Figure 4: Domain Decomposition and Subdomain Representation 9
Figure 5: Computational Workflow Implemented in ParMIN3P-THCm 10
Figure 6: Diagram of Domain Decomposition for Shared-memory Parallel Implementation. (a) Domain Decomposition with Chunk Number 4 for 4 Threads, (b) Domain Decomposition with Chunk Number 8 for 4 Threads 11
Figure 7: Diagram of Basic OpenMP Pseudo Code as Implemented in ParMIN3P-THCm 12
Figure 8: Flow Control for a PETSc Application (Balay et al. 1997, 2014a, 2014b) 13
Figure 9: Diagram of Domain Decomposition for Distributed-memory Parallel Implementation. (a) Domain Decomposition with Stencil Width Equals 1 for 4 Processors, (b) Domain Decomposition with Stencil Width Equals 2 for 4 Processors 14
Figure 10: Diagram of Domain Decomposition for Hybrid Parallel Implementation. (a) Domain Decomposition with Stencil Width 1 for 4 Processors and 8 Threads, (b) Domain Decomposition with Stencil Width 2 for 4 Processors and 8 Threads 16
Figure 11: Domain Decomposition and Node Numbering in ParMIN3P-THCm 17
Figure 12: Sample Code of Domain Decomposition and Mapping 18
Figure 13: Matrix and Right Hand Side Assembly of a Subdomain 19
Figure 14: Matrix and Right Hand Side Assembly of Subdomain 20
Figure 15: Sample Code for Jacobi Matrix Assembly 20
Figure 16: Sample Code for PARDISO Solver 21
Figure 17: Sample Code for PETSc Solver 22

Figure 18: Parallel Output of Non-contiguous Data 23
Figure 19: Three Level Parallel Input and Output 23
Figure 20: Sample Code of Data Write Using Collective and Contiguous I/O 23
Figure 21: Model Domain for Case I, Discretized into a Heterogeneous Mesh with a Refined Spatial Resolution of 0.05 m at the Concrete/Clay Interface (from Marty et al. 2015) 26
Figure 22: Mineralogical Alterations and pH Changes after 10000 Years of Concrete/Clay Interactions (from Marty et al. 2015). The Concrete-Claystone Interface Is Located at a Distance of 3.0 M 27
Figure 23: Simplified Conceptual Model Illustrating the Biotic and Abiotic Reaction Network of Case II (from Şengör et al. 2015) 29
Figure 24: Model Grid and Boundary Conditions for Case II (from Şengör et al. 2015) 30
Figure 25: Concentration Distributions for Selected Aqueous Components, Minerals and Biomass at 60 Days for Case II 31
Figure 26: Location and Main Geological Features for the Intracratonic Sedimentary Basins in North America (Illinois Michigan and Appalachian Basins, Taken from McIntosh and Walter 2005) 32
Figure 27: Geometry and Main Hydrogeological Units Considered for Case III (Bea et al. 2010) 33
Figure 28: Distribution of Total Ca Concentration at Different Output Times for Case III 35
Figure 29: Speedup of OpenMP Parallel Version for Case I, Executed on a Shared-memory Workstation 37
Figure 30: Speedup of OpenMP Parallel Version for Case II, Executed on a Shared-memory Workstation 38
Figure 31: Speedup of OpenMP Parallel Version for Case III, Executed on a Shared-memory Workstation 40
Figure 32: Speedup of MPI Parallel Version for Case I, Executed on the WestGrid Orcinus Cluster 42
Figure 33: Speedup of MPI Parallel Version for Case II, Executed on the WestGrid Orcinus Cluster 43
Figure 34: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Orcinus Cluster 44
Figure 35: Speedup of Hybrid Parallel Version for Case I, Executed on the WestGrid Jasper Cluster 47
Figure 36: Speedup of MPI Parallel Version for Case I, Executed on the WestGrid Jasper Cluster 47
Figure 37: Comparison of Total Speedup of Hybrid Parallel Version and MPI Parallel Version for Case I, Executed on the WestGrid Jasper Cluster 48
Figure 38: Speedup of Hybrid Parallel Version for Case II, Executed on the WestGrid Jasper Cluster 49
Figure 39: Speedup of MPI Parallel Version for Case II, Executed on the WestGrid Jasper Cluster 50
Figure 40: Comparison of Total Speedup of the Hybrid Parallel Version and the MPI Parallel Version for Case II, Executed on the WestGrid Jasper Cluster 50
Figure 41: Speedup of Hybrid Parallel Version for Case III, Executed on WestGrid Jasper Cluster 51
Figure 42: Speedup of the MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster 52
Figure 43: Comparison of Total Speedup between the Hybrid Parallel Version and the MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster 53

Figure 44: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster, Total Degrees of Freedom is 405,000 54
Figure 45: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster, Total Degrees of Freedom is 6,480,000 55
Figure 46: Speedup of MPI Parallel Version for Case III, Executed on WestGrid Jasper Cluster, Total Degrees of Freedom is 25,920,000 55
Figure 47: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster, Total Degrees of Freedom is 103,680,000 56

1. INTRODUCTION

This technical report is part of the NWMO project GS60 “Development and application of reactive transport models for assessing the long-term geochemical stability of geological formations”. The main goal of this project was to develop a parallel version of MIN3P-THCm (Mayer et al. 2002; Mayer and MacQuarrie 2010), named ParMIN3P-THCm. The supported operating systems include, but are not limited to, Windows, Mac and Linux/Unix.

MIN3P-THCm is a general purpose multicomponent reactive transport code that is designed to simulate coupled hydrogeological, thermal, and biogeochemical processes in the subsurface. The code solves Richards’ equation for 3D saturated/unsaturated subsurface flows; for reactive mass transport, the code uses the direct substitution approach (DSA) and employs the global implicit method (GIM) for solution of the multicomponent advection-dispersion equations and the geochemical reactions. Spatial discretization is performed based on the finite volume method and allows conducting simulations in one, two, and three spatial dimensions. Features of the code include 3D saturated/unsaturated fluid flow, biogeochemical reactions, heat transport, reactive transport and 1D hydromechanical coupling. The code has been developed over the past 15 years and has steadily grown in capabilities and complexity. The code has been used at the University of British Columbia and numerous other institutions worldwide.

Multicomponent reactive transport modelling has become a powerful tool in earth and environmental sciences; however, more widespread use is continuously challenged by high computational demands. The numerical methods implemented in MIN3P-THCm include the global implicit approach with adaptive time-stepping, efficient ILU preconditioning with BICGS acceleration and an efficient Newton-Raphson linearization. Even though MIN3P-THCm employs robust numerical methods, it still cannot meet the requirement for large-scale long-term simulations with numerous chemical components and interactions. With increased complexity and simulation scale, reactive transport modelling can be very time consuming and memory intensive, which greatly hinders the application of reactive transport models. For example, the simulation of geochemical conditions in deep sedimentary basins that might be considered for nuclear waste repositories may involve spatial scales of 100’s of kilometres and time scales of 1000’s of years, and is computationally intensive, even if restricted to two spatial dimensions. The computational time for such problems often exceeds a week or more when using a single processor workstation, which significantly hampers the progress and analysis capabilities.

With the rapid development of computing technology and numerical algorithms, approaches that make use of high performance computing (HPC) have become more popular in many fields, including reactive transport modelling. The objective of this work was to develop a cutting-edge reactive transport code that can run efficiently on machines ranging from desktop PCs, shared-memory workstations, to distributed-memory supercomputers. The resulting program is able to deal with the significant computational burden of far-field simulations involving large spatial scales and long time frames.

Parallelization of MIN3P-THCm (i.e. ParMIN3P-THCm) was achieved by using the domain decomposition method using PETSc (Balay et al. 1997; Balay et al. 2014a; Balay et al. 2014b), a Portable Extensible Toolkit for Scientific Computation. PETSc is also used to manage the parallel solvers, data structures and message communications. A hybrid MPI and OpenMP parallel programming approach was implemented in the code to take advantage of leadership-class supercomputers that combine both shared memory and distributed memory architectures.

ParMIN3P-THCm has been developed from the ground up for parallel scalability and has been run on up to 768 processors with problem sizes up to 100 million unknowns.

1.1 PREVIOUS RESEARCH OF REACTIVE TRANSPORT CODE PARALLELIZATION

Over the past few decades, subsurface flow and reactive transport models have become essential tools in earth and environmental sciences. These models help researchers to gain a better understanding of the physical, chemical and biological processes that affect geochemical stability, contaminant transport and remediation. Several state-of-the-art reactive transport models have been developed in the past few decades to assess and quantify contaminant migration affected by a suite of biogeochemical reactions in subsurface media. These models are widely used in waste disposal, groundwater remediation and carbon sequestration. Such models include CORE^{2D}V4 (Samper et al. 2012), CRUNCHFLOW (Steefel 2009), eSTOMP (White and Oostrom 2006), HYDROGEOCHEM (Yeh and Tripathi 1990; Yeh et al. 2012), HYTEC (van der Lee et al. 2003), HPx (Simunek et al. 2012), IPARS (Wheeler et al. 2012), OpenGeoSys (Kolditz et al. 2012), ORCHESTRA (Meeussen 2003), PFLOTRAN (Hammond et al. 2012), PHREEQC3 (Parkhurst and Appelo 2013), iPHREEQC3 (Charlton and Parkhurst 2011), PHT3D (Prommer and Post 2010), RT3D (Clement and Johnson 2012), TOUGHREACT (Xu et al. 2012), NUFT (Hao et al. 2012), and MIN3P (Mayer et al. 2002).

Advances in computer technology and numerical algorithms have led to remarkable increases in the spatiotemporal scales and process complexity that can be represented in simulations. The most widely used method is to transition sequential numerical models to parallel numerical models that can use multiprocessor structures, large memory and storage to increase the simulation scale, while reducing runtime. To the authors' knowledge, only some of the abovementioned reactive transport codes have been parallelized or partially parallelized, including CRUNCHFLOW, HYTEC, iPHREEQC, NUFT, OpenGeoSys, ORCHESTRA, PFLOTRAN, PHT3D, eSTOMP and TOUGHREACT.

The parallel version of CRUNCHFLOW is called CHOMBO-CRUNCH, which was developed based on the open-source adaptive mesh refinement (AMR) software framework CHOMBO (Adams et al. 2014). The code is designed to perform simulations of reactive transport in complex micro-scale geometries. The approach has been tested using 48K processors with up to 1 billion grid points. Another parallel version of CRUNCHFLOW is ParCrunchFlow (Beisman et al. 2015), which was created by coupling CRUNCHFLOW with a parallel hydrologic model PARFLOW (Kollet and Maxwell 2006). HYTEC (Lagneau and Lee 2010) has been parallelized to run on massively parallelized supercomputers by launching hydrodynamics and chemistry on different processors. The solution of chemical reactions at the computational grid cells can be distributed to an arbitrary number of processors. iPHREEQC3 (Charlton and Parkhurst 2011) uses the domain decomposition parallelization method with nodes/cells predefined for each processor. The parallel implementation of NUFT (Hao et al. 2012) is intended for a distributed memory parallel system, and the inter-processor communication and data exchange are achieved with MPI. PETSc is employed to solve the large sets of linear equations obtained from the Newton-Raphson linearization. OpenGeoSys (Kolditz et al. 2012) is based on an object-oriented concept, but the parallelization of the code still lacks efficiency and is the subject of future research. ORCHESTRA (Meeussen 2003) was coded in Java and was parallelized through Java threads to make use of multiprocessor hardware. PHT3D (Prommer and Post 2010) is an MPI-based parallel reactive transport model and the parallel version is currently being tested. PFLOTRAN (Hammond et al. 2012) is an open source, state-of-the-art massively parallel subsurface flow and reactive transport code. The code is designed to run on massively parallel computing architectures as well as workstations and laptops. Parallelization was

achieved through domain decomposition using the PETSc software. PFLOTRAN has been developed from the ground up for parallel scalability and has been run on up to 2^{18} (>260,000) processors with problem sizes up to 2 billion degrees of freedom. eSTOMP (White and Oostrom 2006) is a highly scalable version of the STOMP code for subsurface characterization and modelling, allowing for high resolution of the model parameters and processes. It was built using the GA Toolkit (Nieplocha et al. 2006) and PETSc and has been run on up to 2^{17}(>130,000) processors. TOUGHREACT (Xu et al. 2012) was developed using a hybrid MPI-OpenMP parallelization of the reactive transport routines. Recently, a new parallel version of TOUGHREACT, THC-MP (Wei et al. 2015), was developed using the domain decomposition method and has been applied using up to 120 processors.

Generally, there are two types of parallel architectures, the shared-memory architecture and the distributed-memory architecture. Parallelization on the shared-memory architecture is straight-forward while parallelization on the distributed-memory architecture is more difficult to achieve. The advantage of shared-memory parallelization is that it has better load balancing and less overhead as it does not include explicit communication, while for the distributed-memory parallelization the advantage is that it can distribute the workload over different machines, making it more scalable for large supercomputers. Shared-memory parallelization is preferable for small scale simulations while distributed-memory parallelization is preferable for large scale simulations.

Modern supercomputers are usually built with a hybrid distributed-shared memory architecture. That is to say, every computing node is a shared-memory system. To take advantage of this hybrid computer architecture, recently developed parallel codes also consider hybrid parallel implementation.

1.2 PROJECT OBJECTIVE

The main objective of this research was to develop a parallel version of MIN3P-THCm (ParMIN3P-THCm) that can be executed on desktop PCs, shared-memory workstations and distributed-memory supercomputers. The resulting program aims at dealing with the significant computational burden of simulations involving large spatial scales and long time frames.

During the development of ParMIN3P-THCm, the following tasks were undertaken:

- Identify computational “hot spots” within MIN3P-THCm including the analysis of computational bottlenecks to determine parallelization strategies and priorities.
- Design a coherent parallel framework that can work on desktop PCs, workstations, PC clusters and supercomputers for Windows, Unix/Linux and Mac operating systems.
- Develop a shared-memory parallel version of MIN3P-THCm using OpenMP multithreading techniques for shared-memory architecture computers.
- Develop a distributed-memory parallel version of MIN3P-THCm using MPI and PETSc for distributed-memory architecture computers.
- Develop a hybrid distributed-shared-memory parallel version of MIN3P-THCm using OpenMP, MPI and PETSc for supercomputers with hybrid distributed-shared-memory architecture.
- Test the parallel performance of ParMIN3P-THCm on computers with different architectures for the three parallel versions of MIN3P-THCm.
- Perform real-world simulations with higher resolution discretization on supercomputers.

1.3 REPORT ORGANIZATION

Chapter 2 of this report describes the ParMIN3P-THCm parallelization framework including code architecture, the theory behind the parallelization method, parallel implementation, and a description of the parallel modules; Chapter 3 describes and analyzes the parallel performance for a series of test examples for varying degrees of freedom which were executed on different computer platforms; and Chapter 4 presents concluding remarks.

2. PARALLEL FRAMEWORK

Over the history of computing hardware development, the number of transistors in a dense integrated circuit has doubled every 18 months, as described by Moore's law[1]. However, the speed of the processors has not improved much after the year 2000. Instead, the number of processors per chip has increased significantly. With the rapid development of multi-processor multi-core computing technology, even personal computers are now equipped with multiple cores and multi-thread processors. If only one processor can be used for a numerical simulation task, then multi-threaded computer hardware cannot help to increase computing efficiency because the computing time for a single processor mainly depends on the processor's speed. For parallel code development, the main challenge is to take advantage of all available processors and memory, independent of the computer architecture used.

Today's high-end computers are characterized by complex architectures for both the individual processors and the entire system. Achieving good performance on these systems can be quite difficult. In-depth knowledge of programming techniques and numerical algorithms are the key factors in high performance computing (HPC) code development. It is not realistic for a small development team to develop tools or methodologies that allow gaining optimal parallel performance. Recently, several state-of-the-art and open-source software tools have become available, and the best strategy for developing parallelized reactive transport codes is to make use of these software tools. These tools can provide performance portability over the next decade or more, and one can greatly reduce the effort to transition legacy codes to existing and future parallel architectures, while simultaneously achieving performance portability. These open-source software tools have been used for the parallelization of MIN3P-THCm.

Specifically, parallelization of MIN3P-THCm is achieved through the domain decomposition method as implemented in the PETSc library (Balay et al. 1997; Balay et al. 2014a; Balay et al. 2014b). PETSc is also used to manage the parallel solvers, data structures and message communications. A hybrid MPI and OpenMP parallel programming technology is implemented in the code to take advantage of leadership-class supercomputers that combine both shared memory and distributed memory architectures.

2.1 CODE ARCHITECUTRE

ParMIN3P-THCm was mainly written in Fortran 90/95 with some of the code written in Fortran 2003. To maximize code compatibility and manageability, several code development tools were used and specific syntax was embedded in the code.

[1] Moore's law, https://en.wikipedia.org/wiki/Moore%27s_law

2.1.1 Target System

The target operating systems for ParMIN3P-THCm include Windows, Unix/Linux and Mac OS. The code is compatible with most of the popular Fortran compilers including Intel Fortran, GFortran and IBM XL compilers. For the Windows platform, Visual Studio solutions and Intel Fortran project files are provided. For the Unix/Linux based system, a makefile is provided to compile the code.

In addition to desktop PCs and shared-memory workstations, the target platforms for ParMIN3P-THCM include distributed-memory supercomputers such as IBM Blue/Gene, Cray and Unix/Linux clusters.

2.1.2 Development Tools and Libraries

Several state-of-the-art software tools and libraries were used for the development of ParMIN3P-THCm. These packages play different roles in the code and they can be used alone or together, depending on the target system and parallel version to be compiled. Fortran preprocessing syntax is embedded throughout the code to ensure the code can be managed in a single coherent framework, which means that all developers are working on the same code and all executable files are compiled from the same code but with different configurations. The main software tools include:

- OpenMP: A flexible interface for developing parallel applications for shared-memory multiprocessing platforms. OpenMP is used as the interface for the shared-memory version and it is also used together with MPI for the hybrid distributed-shared-memory version.
- MPI: A language-independent communications protocol that provides essential virtual topology, synchronization and communication functionality between a set of processors. MPI is used as the interface for the distributed-memory version and is also used together with OpenMP for the hybrid distributed-shared-memory version.
- PETSc: A suite of data structures and routines for the scalable solution of scientific applications modelled by partial differential equations. PETSc is used to manage the parallel solvers, data structures and message communications.
- VisualSVN Server/VisualSVN: A professional grade subversion server and client integration plug-in for Visual Studio. VisualSVN is used as the source code version control tool.
- Doxygen: A standard tool for generating documentation from source code. It is used to generate the programmer's manual for ParMIN3P-THCm.
- Pardiso: A thread-safe, high-performance software for solving sparse linear systems of equations on shared-memory systems. It is an optional solver for the shared-memory parallel version of ParMIN3P-THCm and can also be used as a third-party solver through the PETSc interface.
- HYPRE: A library for solving large, sparse linear systems of equations on massively parallel computers. HYPRE is used together with PETSc providing a suite of optional preconditioners in ParMIN3P-THCm.
- Other packages (e.g., SuperLU) that have an interface to PETSc are also supported in ParMIN3P-THCm.
- Other tools/scripts to run benchmarking tests and verification tests in an automated fashion.

2.2 PARALLEL ARCHITECTURES

2.2.1 Shared-memory Multiprocessor Architecture

In a shared-memory multiprocessor computer, all processors share the same memory or address space. The shared space or address is used for communication between the processors. All processors can access the same address space of global memory through the interconnection network. Typically, that interconnection network is called the system bus, as shown in Figure 1. For the shared-memory multiprocessor architecture, the memory bandwidth becomes the system's bottleneck due to the interconnection network limitation and memory collision when many processors try to access the memory simultaneously.

The advantage of the shared-memory architecture is that each processor sees only one memory address space which means that the developer does not have to create explicit communications between processors, making the program development more straightforward. Another advantage is that data sharing between threads is both fast and uniform due to the proximity of memory to processor. The primary disadvantage is the lack of scalability between memory and CPUs. The difficulty in shared-memory parallelization programming is that the programmer is responsible for synchronization constructs to ensure correct access of the global memory. For shared-memory parallelization, OpenMP is used as the programming library where synchronization is achieved through barriers, locks, mutex or semaphores. OpenMP does not introduce message communication as MPI does, but it still adds some overhead when threading, such as startup overhead, loop scheduling overhead and lock management overhead. In some cases, typically with small loops, the overhead numbers may be high enough so that it does not make sense to implement OpenMP parallelization for this kind of code.

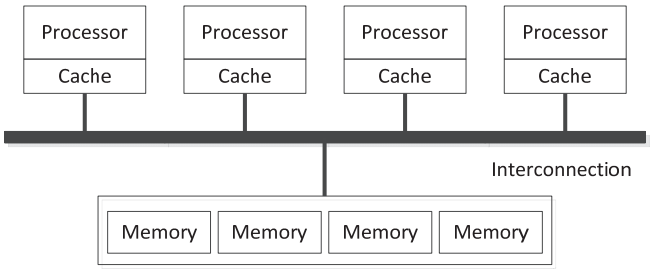

Figure 1: Shared-memory Multiprocessor Architecture

2.2.2 Distributed-memory Multiprocessor Architecture

In a distributed-memory multiprocessor, each processor is associated with its own memory. The processor can directly get access to its own memory, as shown in Figure 2. In order to allow a processor to access the memory owned by the other processor, processor-to-processor communication is needed. For the distributed-memory multiprocessor architecture, the interconnection bandwidth/latency becomes the system's bottleneck due to the large volume of communication simultaneously requested by processors.

The advantage of the distributed-memory architecture is that it is scalable both in terms of memory and number of processors. Each processor can access its own memory rapidly without the overhead incurred by trying to maintain the global cache coherency. But on the other hand, this makes program development more difficult as the developer has to map existing data structure onto the individual processors and create explicit communications between processors. The difficulty in distributed-memory parallelization programming is to deal with the

communication overhead. For distributed-memory parallelization, MPI is used as the programming library through which messages are sent and received between processors.

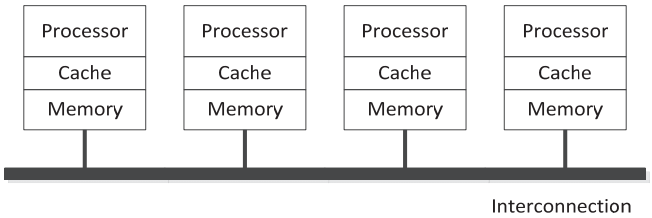

Figure 2: Distributed-memory Multiprocessor Architecture

2.2.3 Hybrid Distributed-shared-memory Multiprocessor Architecture

The largest and fastest supercomputers today employ both shared-memory and distributed-memory architecture. As shown in Figure 3, each computing node is a shared memory multiprocessor. Network communications are required to send data from one computing node to another. Current usage and projections indicate that this type of architecture will continue to prevail or even increase for future high-end supercomputers[2].

The most important advantage of the hybrid architecture is the increased scalability while the knowledge needed in programming is common to both shared and distributed memory architectures. On the other hand, the disadvantage of this approach is that it also increases the programming complexity.

A typical example of hybrid parallelization is the combined implementation of MPI and OpenMP. The threads perform computationally intensive tasks using local on-node data while the communications between processors on different nodes are done by MPI.

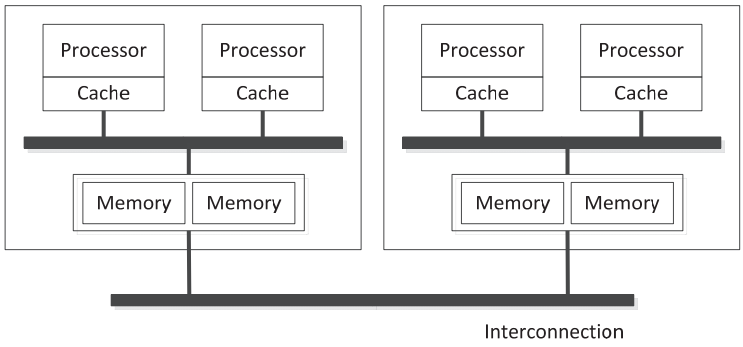

Figure 3: Hybrid distributed-shared-memory Multiprocessor Architecture

2.2.4 Parallel Levels

Parallel algorithms and parallel architectures are closely tied together. It is not possible to design a parallel algorithm without taking into consideration the parallel hardware that will support it. Conversely, it is also not possible to install parallel hardware without taking into

[2] Blaise Barney, Lawrence Livermore National Laboratory. Introduction to parallel computing. https://computing.llnl.gov/tutorials/parallel_comp/

consideration the parallel software that will be used. Parallelization can be implemented at different levels in a computing system using hardware and software techniques. Generally, parallelization can be classified into four levels (Gebali 2011):

- Data-level parallelization, where multiple data are operated on simultaneously. Examples are bit-parallel additions, multiplication and division of binary numbers, vectors and arrays.
- Instruction-level parallelization, where multiple instructions are executed simultaneously. An example is the use of instruction pipelining.
- Thread-level parallelization. A thread is a portion of a program that shares processor resources with other threads. Multiple threads are executed simultaneously on one process or multi-core processors in thread-level parallelization.
- Process-level parallelization. A process is an independent program that is running on a computer. Every process reserves its own computer resources such as cache and memory space. For process-level parallelization, several programs are running simultaneously on a computer with multi-processors or computer clusters with distributed processors and memory.

2.2.5 Parallel Levels in ParMIN3P-THCm

One of the challenges of parallelization is that each processor must be kept as busy as possible with tasks to avoid the processor from being idle. To meet this challenge requires careful program development; however, efficiency is also affected by the compiler and operating system performance. In ParMIN3P-THCm, data-level parallelization and instruction-level parallelization are managed by the compiler and the operating system, while program development focused on thread-level parallelization and process-level parallelization.

Thread-level parallelization in ParMIN3P-THCm is designed for shared-memory computers. This kind of computer is popular as most modern desktop PCs are now equipped with multi-core processors. To meet the requirement of the users who do not have access to supercomputers, thread-level parallelization is developed using OpenMP. On the other hand, thread-level parallelization is more suitable for small scale problems, as it is generally faster compared to process-level parallelization.

The process-level parallelization in ParMIN3P-THCm is designed for distributed-memory computers equipped with a large number of processors and memory availability. Process-level parallelization is primarily used for large scale simulations that require more processors and memory (e.g., > 32 processors).

It should be mentioned that ParMIN3P-THCm is designed at a high level to allow more straightforward adoption to modern supercomputers with hybrid distributed-shared-memory architecture, with potential consideration of the next generation supercomputers.

2.3 PARALLEL IMPLEMENTATION

2.3.1 Domain Decomposition

Domain decomposition generally refers to the splitting of partial differential equations, or an approximation thereof, into coupled problems on smaller subdomains forming a partition of the original domain (Toselli and Widlund 2005). This decomposition may enter at the continuous level, where a large boundary value problem is split into small boundary value problems on

subdomains and iterating to coordinate the solution between adjacent subdomains, or in the solution of the algebraic systems arising from the approximation of the partial differential equations. The domain decomposition method is used in ParMIN3P-THCm in both forms, i.e. at the continuous level and in the solution of algebraic systems.

For the continuous level domain decomposition, the entire simulation domain is split into small subdomains or small “chunks”. All independent tasks that do not require communication with adjacent subdomains are executed simultaneously on different threads/processors. The domain decomposition for the shared-memory parallelization is simplified and does not require consideration of ghost nodes because all threads share the same memory space and no data communication is needed when building the Jacobian matrix. However, for the distributed-memory and hybrid distributed-shared-memory parallelizations, ghost nodes with different stencil widths are considered to provide a copy of the boundary values from the adjacent subdomains. The ghost nodes, located at both sides of the subdomain boundaries, are also treated as the bridge for updating values between adjacent subdomains after the global linear equations are solved, as shown in Figure 4.

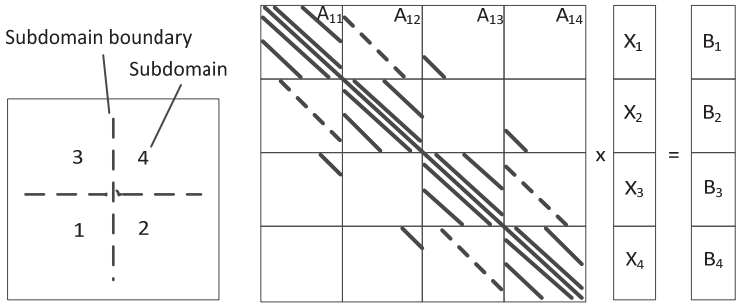

Figure 4: Domain Decomposition and Subdomain Representation

For the domain decomposition of algebraic systems, the basic idea is to decompose the solution space into several subspaces; for each of which there is an efficient solver to generate the result. For the domain decomposition shown in Figure 4, the iteration scheme for subdomain i (i = 1, 2, 3, 4) is expressed in equation (2.1):

$$M_{ii}X_i^m = B_i^{m-1} - (A_{ii} - M_{ii})X_i^{m-1} - \sum_j A_{ij}X_i^{m-1} \qquad (i \neq j) \tag{2.1}$$

A_{ii} is the diagonal block split of the global coefficient matrix *A*, $A_{ij}(i \neq j)$ is the off-diagonal block split of the global coefficient matrix *A*, B_i^{m-1} is the right hand side of subdomain i, M_{ii} is the diagonal block of subdomain i, and X_i^{m-1} is the solution of subdomain i. External parallel linear solvers (e.g., Pardiso, PETSc) are employed as the global solvers in ParMIN3P-THCm.

The detailed procedures of the ParMIN3P-THCm domain decomposition parallelization are described in the following sections.

2.3.2 Computational Workflow

The solution of the entire system of equations in ParMIN3P-THCm consists of the solution of the variably saturated flow and energy balance equations, with the subsequent solution of the

reactive transport problem based on the fluxes and phase saturations obtained from the flow solution. The system of algebraic nonlinear equations for the variably saturated flow, energy balance and reactive transport is linearized using Newton's method and the coupling between fluid density and solute concentrations is resolved using the Picard iterative approach (Mayer et al. 2014). The workflow of ParMIN3P-THCm is depicted in Figure 5.

Compared to the serial version of MIN3P-THCm, the parallel version requires additional processing including domain decomposition, ghost values updating and data conversion/communication between the subdomains. The most significant difference between the serial and parallel versions of MIN3P-THCm is that most of the computationally intensive work such as solving local mass conservation equations for each control volume, and matrix value computing for the global linear equations, is executed for the subdomains simultaneously. Without considering output information, these parts do not require communication between the subdomains because all data are locally available. These parts take most of the computing time in MIN3P-THCm, are relatively straightforward to parallelize, and are highly scalable. Message communications are introduced for assembly of the global matrix, solution of the global linearized equations and updating of the ghost cell values after the global linear equations are solved. Message communications are also employed in the nonlinear solver (e.g., Newton iteration and Picard iteration), time stepping solver and parallel input/output.

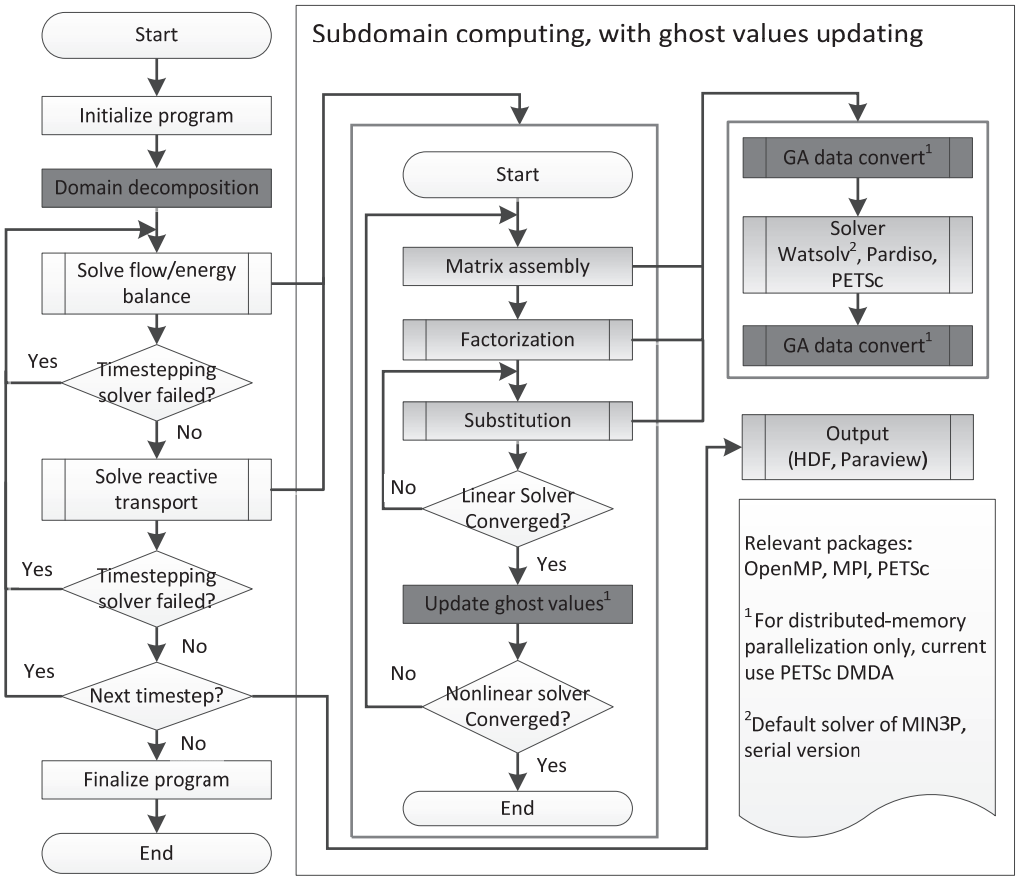

Figure 5: Computational Workflow Implemented in ParMIN3P-THCm

2.3.3 Shared-Memory Parallel Implementation

ParMIN3P-THCm employs the OpenMP 3.1 framework (http://openmp.org/wp/) for the shared-memory parallelization. Development of the ParMIN3P-THCm shared-memory version focuses on the parallelization of chemical reactions and various CPU-intensive routines, e.g., local mass conservation equations for each control volume and matrix assembly, the solution of the linearized equations is parallelized using an external solver such as PARDISO.

Domain decomposition in the shared-memory version provides support for different scheduling methods available in OpenMP. The most commonly used scheduling methods are static scheduling and dynamic scheduling, with different numbers of "chunk sizes", as shown in Figure 6. The static scheduling has less overhead while the dynamic scheduling has better load balancing.

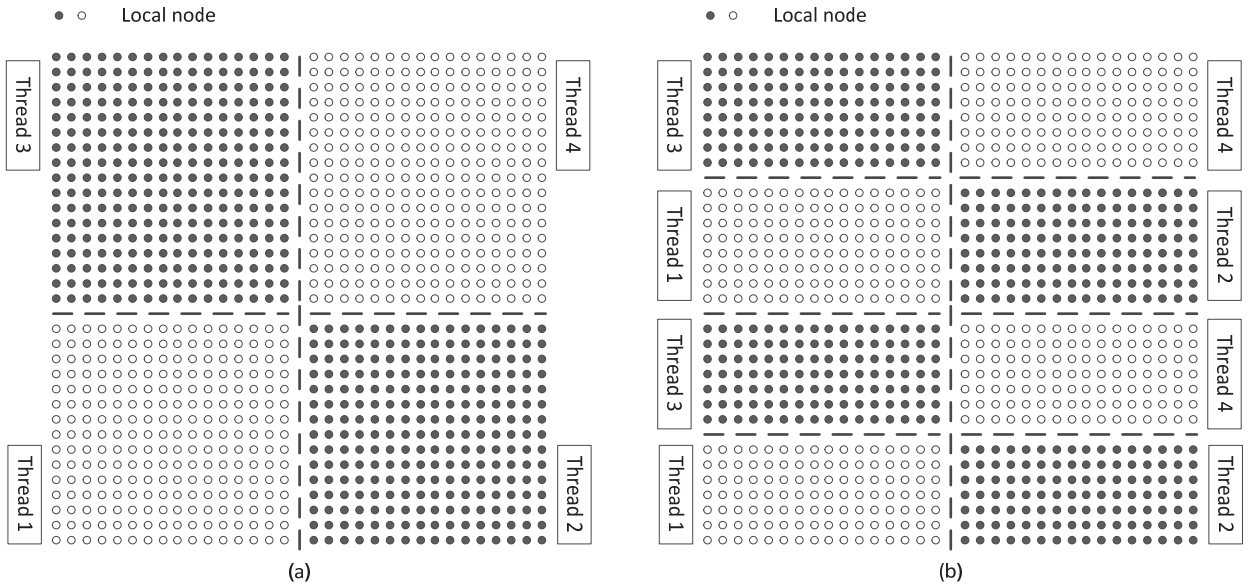

Figure 6: Diagram of Domain Decomposition for Shared-memory Parallel Implementation. (a) Domain Decomposition with Chunk Number 4 for 4 Threads, (b) Domain Decomposition with Chunk Number 8 for 4 Threads

In the ParMIN3P-THCm shared-memory version, most of the CPU-intensive routines are independent of the spatial discretization and perform operations for single control volumes. Accordingly, most of the parallel code for the shared-memory parallel implementation focuses on the parallelization of loops over these control volumes. Figure 7 depicts a diagram of OpenMP pseudo code as implemented in ParMIN3P-THCm. The user has full control on the parallel implementation, e.g., whether parallelization for a particular task is active or not, which scheduling method and chunk size are to be used, and how many threads will be used for the particular parallel task. By default, the variables in the OpenMP framework are shared by all threads. To avoid conflicts or race conditions for some variables (e.g., some global variables in the serial version), these variables are declared as private, first private or last private variables, or modified to be thread private variables. The parallel overhead is an important factor for the development of OpenMP code, in particular for the parallelization of tasks with many private variables, because for private variable memory space needs to be allocated for each of the threads. To reduce parallel overhead, ParMIN3P-THCm is structured with independent parallel

loops wrapped together in a larger outer parallel loop and several outer parallel loops wrapped together in a parallel section. As shown in Figure 7, task A and task B are independent from each other in the parallel loop and are wrapped together in a single do loop I, similarly, task C and task D are wrapped together in parallel loop II. Because parallel loop I and parallel loop II are not independent, i.e. parallel loop II requires the results from parallel loop I, it is impossible to wrap task C and task D immediately after task A and task B without introducing additional overhead. As a result, the code must be structured to execute parallel loops I and II simultaneously to allow for synchronization.

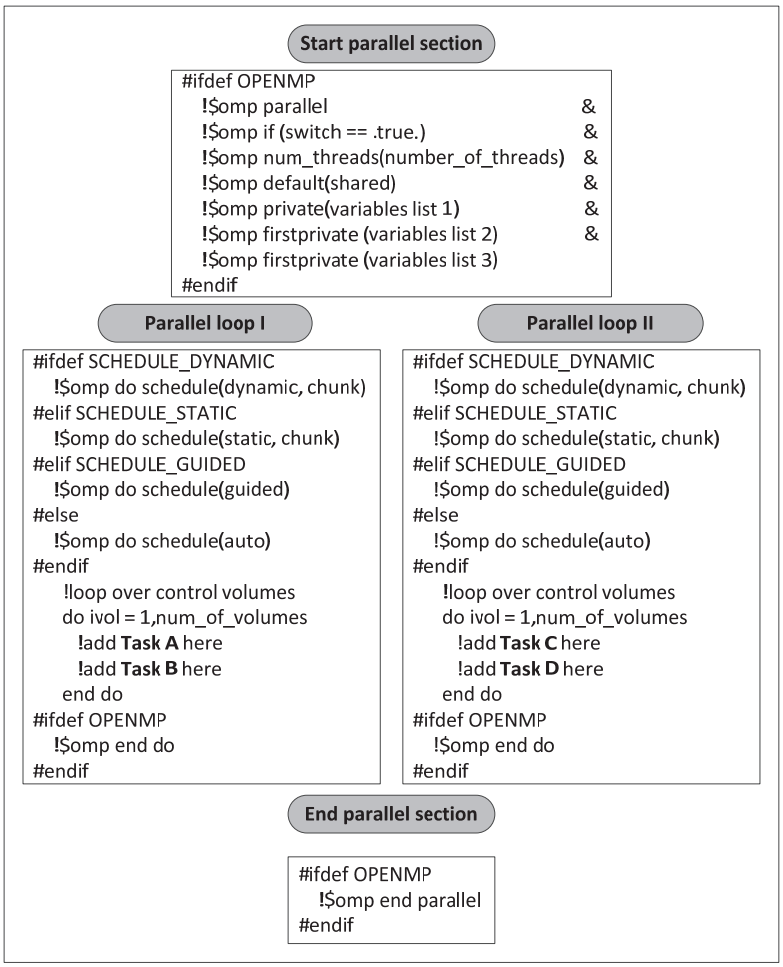

Figure 7: Diagram of Basic OpenMP Pseudo Code as Implemented in ParMIN3P-THCm

The linear equations solver used in the ParMIN3P-THCm shared-memory version is PARDISO, a high performance, memory efficient direct solver (Schenk and Gärtner 2011). PARDISO is used in ParMIN3P-THCm for symbolic factorization, numerical factorization and substitution. It is implemented in the same way as the original MIN3P-THCm solver WatSolv, an iterative solver.

It is difficult to compare the performance of a direct solver with that of an iterative solver. Usually, iterative solvers with effective ILU preconditioners that are capable of generating coefficient matrices with good condition numbers, converge faster than direct solvers. On the other hand, direct solvers return a near exact solution. For a situation with ill-conditioned

matrices, the convergence of iterative solvers becomes poor and direct solvers are capable of producing the solution more quickly. Whether the coefficient matrix will be well-conditioned or ill-conditioned is problem dependent and the code must be able to deal with both situations. For this reason, the WatSolv solver is partially parallelized and is retained in the ParMIN3P-THCm code as an iterative solver option. In MIN3P-THCm, it takes only a relatively small fraction of computing time to solve the linearized equations compared to the time spent on other CPU-intensive tasks. This is especially true for the simulation of problems with complex biogeochemical reaction networks. If parallel overhead for the solution of the linearized equations is significant, or if the WatSolv solver is faster than PARDISO, users still have the choice to use the WatSolv solver, even if it is not fully parallelized.

The input and output (I/O) of the ParMIN3P-THCm shared-memory version is executed by the master thread, implying that these tasks are executed in serial.

2.3.4 Distributed-Memory Parallel Implementation

ParMIN3P-THCm employs MPI and PETSc for the distributed-memory parallelization. PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modelled by partial differential equations. It supports MPI, shared memory pthreads, and GPUs through CUDA or OpenCL, as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism (Balay et al. 1997; 2014a; 2014b). The flow control for a PETSc application is shown in Figure 8. The ParMIN3P-THCm distributed-memory version is built using PETSc with the implementation of linear solvers (KSP) and preconditioners (PC). Linearization of the discretized equations in ParMIN3P-THCm is performed using the Newton and Picard iterative methods, and the time stepping solver consists of a global implicit, adaptive time stepping method.

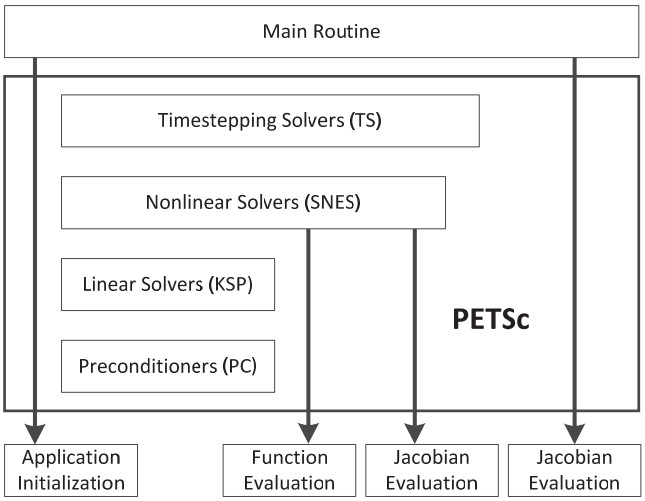

Figure 8: Flow Control for a PETSc Application (Balay et al. 1997; 2014a; 2014b)

The domain decomposition method in the ParMIN3P-THCm distributed-memory version is based on PETSc's DMDA model, an object that is used to manage data for a structured grid in 1, 2 or 3 dimensions. In the global representation, each processor stores a non-overlapping rectangular (or slab in 3D) portion of the global grid points (nodes). In the local representation, these rectangular regions (slabs) are extended in all directions by a stencil width. Figure 9 depicts the domain decomposition for a 2D grid using the distributed-memory parallel implementation. Examples for two alternative stencil widths (equalling 1 and 2, respectively) are shown. Features of the DMDA model used in ParMIN3P-THCm include:

- Interface for topologically structured grids
- Definition of a finite-dimensional function space
- Provision of a parallel layout
- Refinement and coarsening
- Ghost value coherence
- Matrix pre-allocation

The stencil width in domain decomposition represents the number of ghost nodes used in the subdomains. To evaluate a local function $f(x)$, each process requires its local portion (local nodes in a subdomain) of the vector x and the associated ghost node values, originating portions of x owned by neighboring processes (adjacent subdomain). Generally, single stencil width is applied for domain decomposition. For a problem with van Leer spatial weighting (van Leer 1977), double stencil width is applied as the spatial weighting requires a second upstream node.

In the ParMIN3P-THCm distributed-memory parallel version, the entire domain is commonly evenly distributed among different processors. As a result, the parallel efficiency mainly depends on the balancing problem. Generally, the default domain decomposition can meet the efficiency requirement. For highly heterogeneous problems, i.e. the various subdomains take substantially different execution times, domain decomposition with spatial weighting is a better choice. Considering the limited time available for development, this is currently not included in ParMIN3P-THCm.

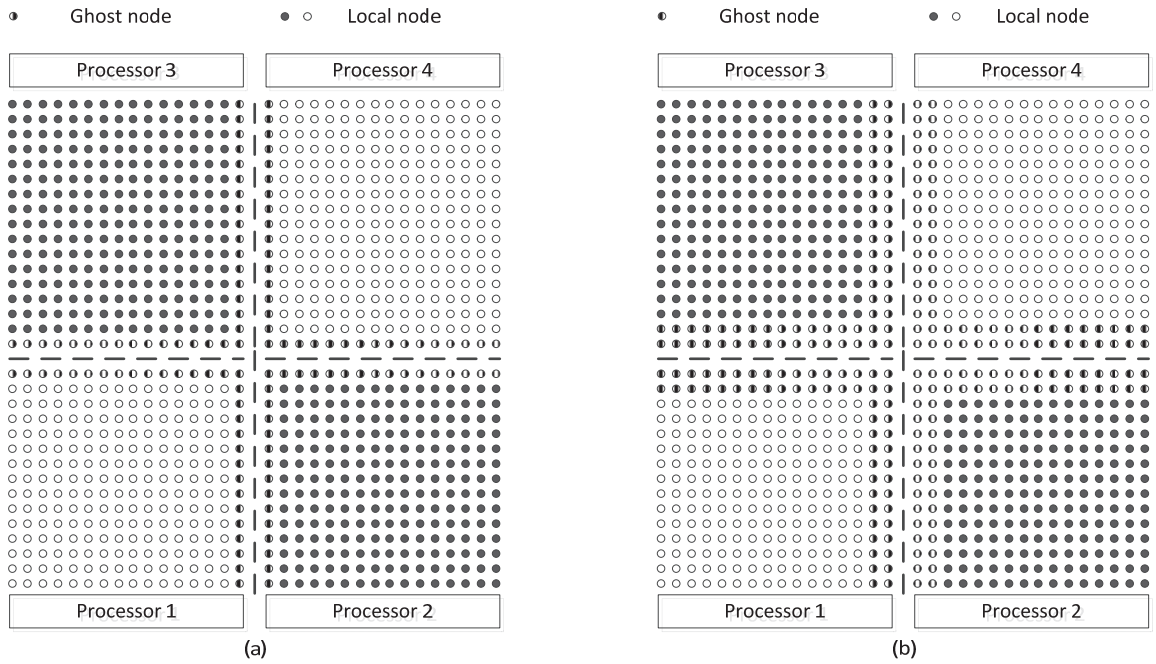

Figure 9: Diagram of Domain Decomposition for Distributed-memory Parallel Implementation. (a) Domain Decomposition with Stencil Width Equals 1 for 4 Processors, (b) Domain Decomposition with Stencil Width Equals 2 for 4 Processors

The linear equations solver used in the ParMIN3P-THCm distributed-memory version is the PETSc linear solver package (KSP), with several iterative methods (e.g., GMRES, BiCGSTAB) and preconditioning methods (e.g., ILU, BLOCK JACOBI) included. PETSc linear solvers are the only choice for the distributed-memory version but users have full privilege to set the iterative method, preconditioning method and convergence criteria.

Inter-processor communications in the ParMIN3P-THCm distributed-memory version are handled by PETSc and MPI together. Communications related to the PETSc DMDA model and linear solvers are controlled by PETSc while all other communications are controlled by native MPI routines.

The input and output in the ParMIN3P-THCm distributed-memory version are executed by all processors and these routines are fully parallelized.

2.3.5 Hybrid Parallel Implementation

The hybrid distributed-shared-memory parallel version (hybrid version) of ParMIN3P-THCm is a combination of the shared-memory version and the distributed-memory version. The hybrid version is not simply a combination of the two versions, but involves a task-specific optimization of both versions. The hybrid version is developed due to the following circumstances:

- Modern or next generation supercomputers are based on the hybrid distributed-shared memory architecture.
- The hybrid implementation increases the scalability and can take advantage of both shared-memory and distributed-memory architectures.
- Using the hybrid approach, the scalability of CPU-intensive tasks (e.g., the solution of local mass conservation equations) is generally better than the scalability for the global linear solver and system I/O. For some special problems, the scalability of the global linear solver may deteriorate when using more processors. Using the hybrid parallel approach for CPU-intensive tasks while using the distributed-memory approach for low-scalability tasks can increase the total parallel performance.

The domain decomposition method in the hybrid version of ParMIN3P-THCm is also based on PETSc's DMDA model, similar to the distributed-memory parallel version. However, the hybrid version is distinct, because each processor stores a non-overlapping rectangular (or slab in 3D) portion of the global grid points (nodes) that is shared by two or more threads, depending on the number of available threads for the processor. For example, as shown in Figure 10, each subdomain is shared by two threads that are managed by the same processor. Compared to the distributed-memory version, the communication does not increase if using the same number of processors, however, the computing in each subdomain can be accelerated. In this way, the ParMIN3P-THCm hybrid version can take advantage of both the distributed-memory and the shared-memory implementations.

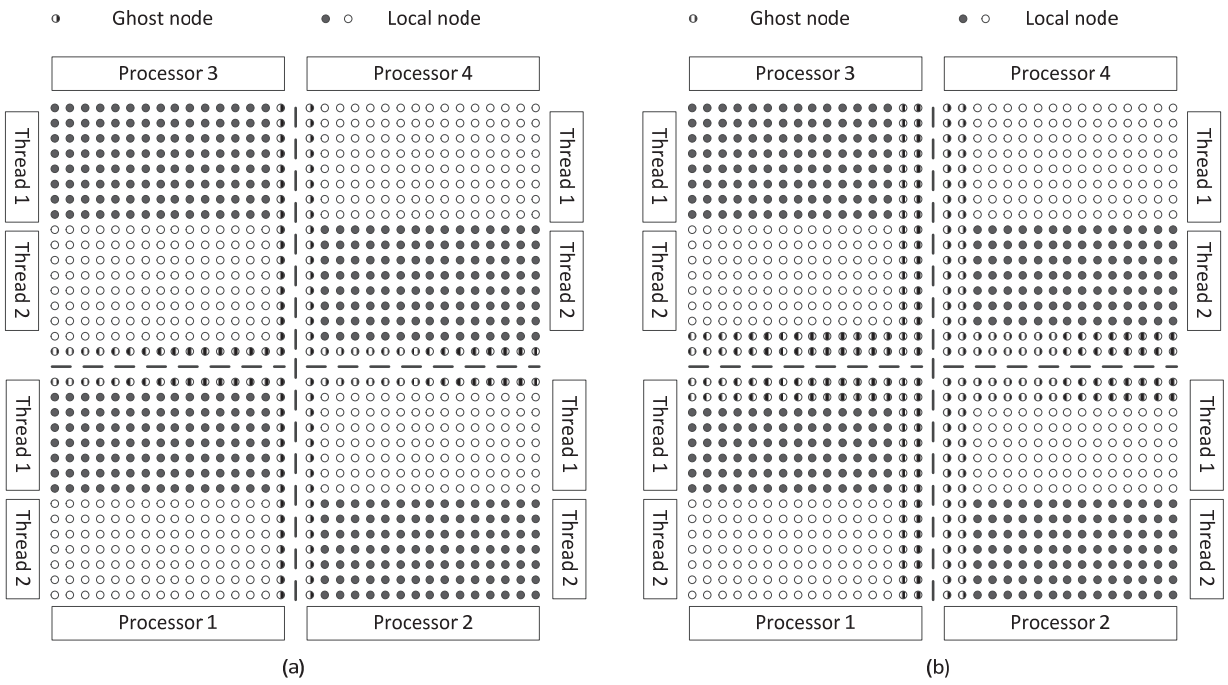

Figure 10: Diagram of Domain Decomposition for Hybrid Parallel Implementation. (a) Domain Decomposition with Stencil Width 1 for 4 Processors and 8 Threads, (b) Domain Decomposition with Stencil Width 2 for 4 Processors and 8 Threads

2.4 PARALLEL MODULES

Most of the functionalities available in ParMIN3P-THCm are modularized. These functionalities have been separated into independent, interchangeable modules such that each module contains information necessary to execute the desired functionality. Modular programming improves the code readability and reduces the cost for code maintenance.

2.4.1 Global and Local Numbering

ParMIN3P-THCm (and previous versions of MIN3P) uses natural ordering when assembling the coefficient matrix for its structured grid. For the natural ordering scheme, the node numbers increase along the X-direction first, then the Y-direction and finally the Z-direction. Domain decomposition for shared-memory parallelization is straightforward because the numbering scheme for the subdomain is exactly the same as that of the serial version. However, for the distributed-memory version and the hybrid implementation, the local numbering scheme is different from the global numbering. The PETSc DMDA module is used as the domain decomposition tool and also manages the local and global node mapping. An example for the domain decomposition of a 2D grid is shown in Figure 11. First, domain decomposition is executed using the natural ordering scheme. Then each subdomain is renumbered to build PETSc global numbering, and finally the subdomains are renumbered using natural ordering with or without ghost nodes. Topology mappings have been built to facilitate straightforward access to the nodes within different subdomains and the global domain.

Sample code illustrating the domain decomposition and the local to global mapping is given in Figure 12. In this example, the grid is a 3D structured grid with node numbers *nvxgbl*, *nvygbl* and *nvzgbl* in X-, Y-, and Z-directions, respectively. The degree of the freedom per node is *dmda_react%dof* and the stencil width is *dmda_react%swidth*. The local to global mapping *ldtog* is created following domain decomposition.

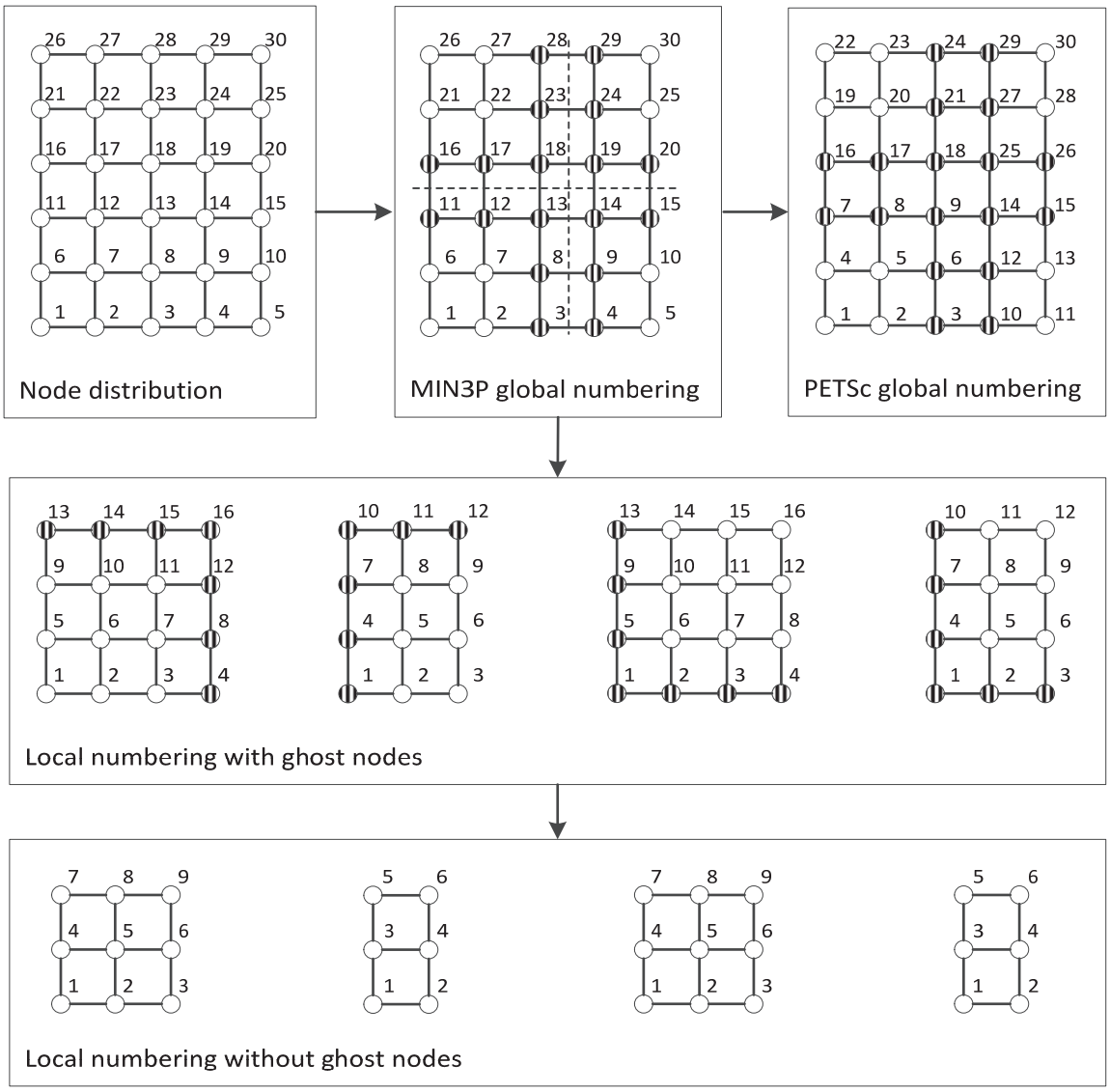

Figure 11: Domain Decomposition and Node Numbering in ParMIN3P-THCm

```
!Domain decomposition for 3D structured grid
call DMDACreate3d(Petsc_Comm_World,DM_BOUNDARY_NONE,      &
                  DM_BOUNDARY_NONE,DM_BOUNDARY_NONE,      &
                  DMDA_STENCIL_BOX,nvxgbl,nvygbl,nvzgbl,  &
                  PETSC_DECIDE,PETSC_DECIDE, PETSC_DECIDE,&
                  dmda_react%dof, dmda_react%swidth,      &
                  PETSC_NULL_INTEGER,PETSC_NULL_INTEGER,  &
                  PETSC_NULL_INTEGER,dmda_react%da,ierr)

!Create local to global mapping
call DMGetLocalToGlobalMapping(dmda_react%da,ltogm,ierr)
call ISLocalToGlobalMappingGetIndices(ltogm,ltog,idltog,ierr)
```

Figure 12: Sample Code of Domain Decomposition and Mapping

2.4.2 Parallel Matrix and Right Hand Side Assembly

For the distributed-memory and hybrid distributed-shared-memory parallel versions, message passing is the key factor that affects the parallel efficiency. To reduce communication, ParMIN3P-THCm computes all matrix entries and the right hand side locally, because all information related to ghost nodes associated with a subdomain is stored locally. Communication is not required for computing matrix and right hand side entries for the ghost nodes.

Figure 13 shows an example of a 2D subdomain with 16 nodes. For each subdomain, only local entries are computed and assembled (entries without a “box”) while the remaining entries (entries within a “box”) are computed and assembled by a processor dealing with the neighboring subdomain. In other words, each processor needs to process only elements that it owns locally, but any non-local elements will be sent to the appropriate processor during matrix assembly.

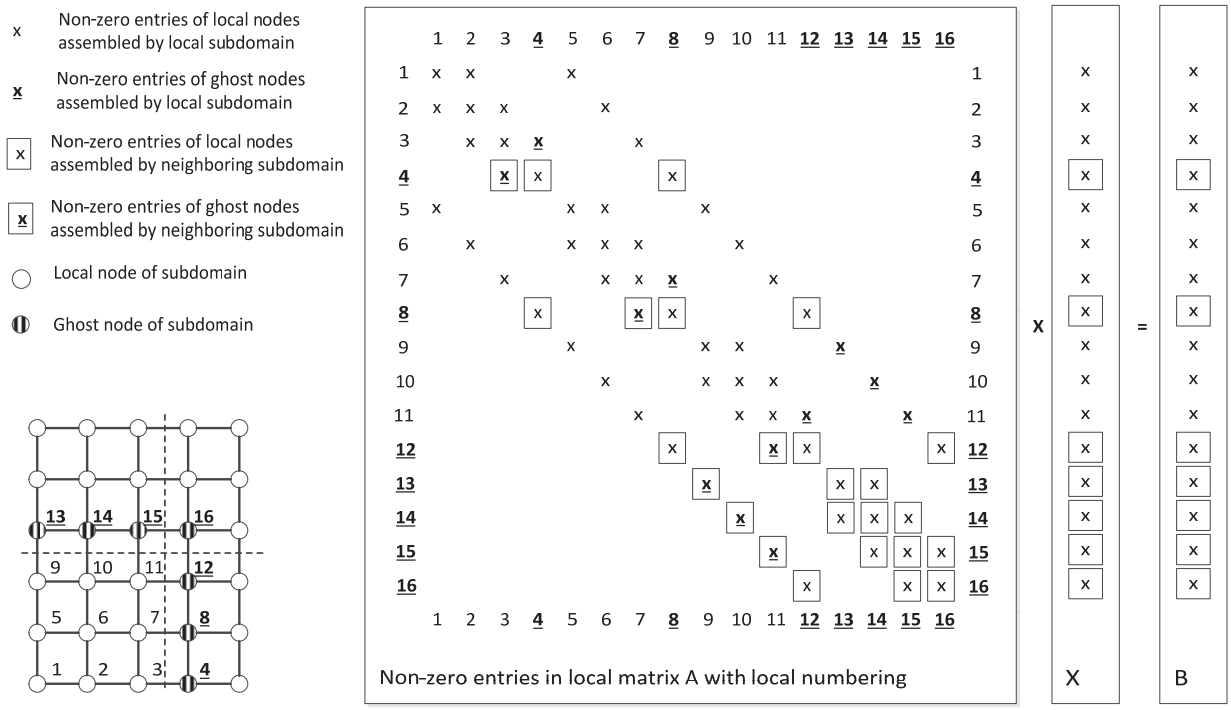

Figure 13: Matrix and Right Hand Side Assembly of a Subdomain

From a mathematical point of view, the local entries of every subdomain can be represented by a matrix block, as shown in Figure 14. The diagonal block includes the entries originating from the local nodes of the subdomain while the off-diagonal blocks contain the entries related to the connections between the subdomain and its neighboring subdomains. The entries within a dashed box or dash-dotted box identify the local entries owned by a specific subdomain. Each subdomain only computes and assembles its local entries towards the global entries.

Sample code for the Jacobi matrix assembly is provided in Figure 15. The matrix assembly in PETSc is a 2-step process: MatAssemblyBegin() and MatAssemblyEnd(). Additional code can be placed between these two functions to further improve parallel processing during communication.

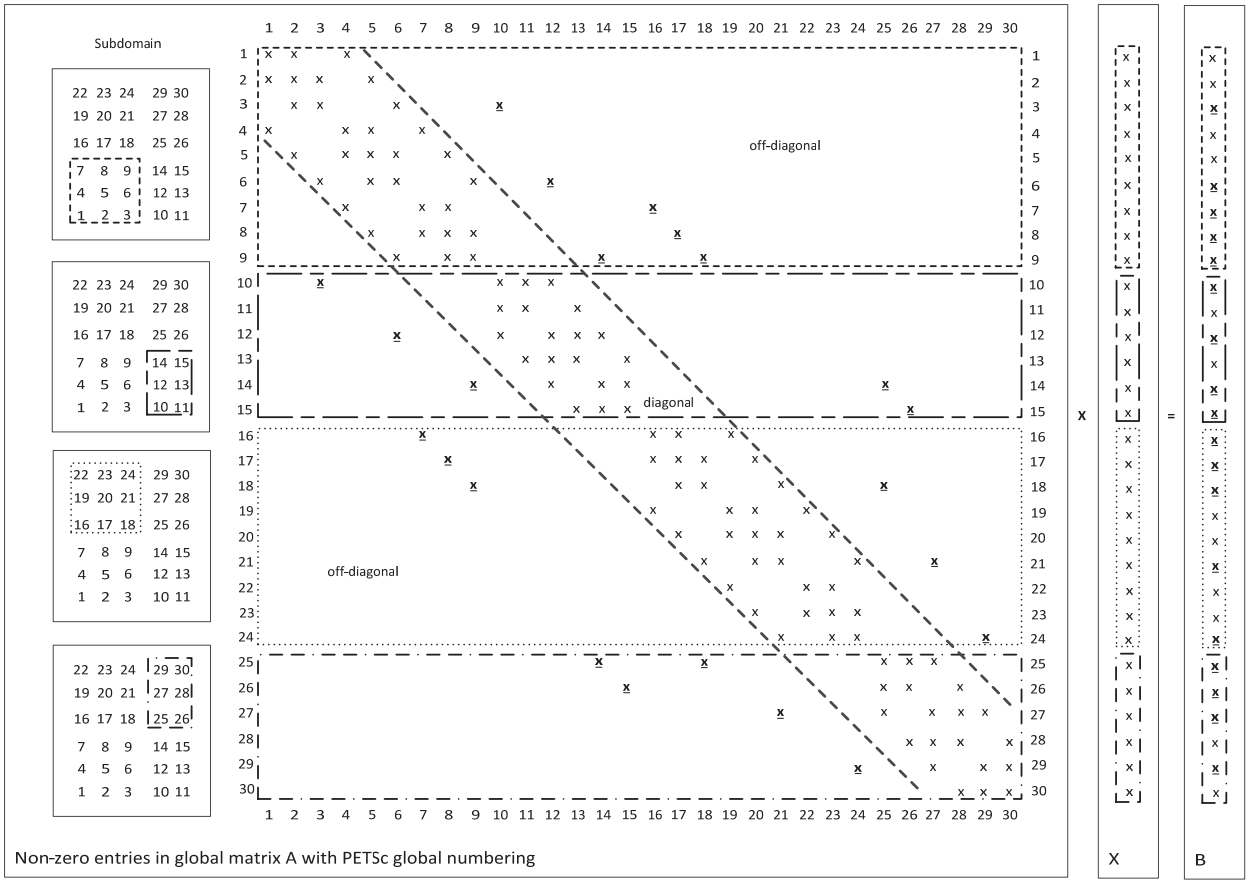

Figure 14: Matrix and Right Hand Side Assembly of Subdomain

```
call MatZeroEntries(mat_gbl, ierr)
!Loop over subdomain control volumes
do i = 1, nngl
  j = row_idx_l2pg(i)
  !Skip ghost nodes, only assemble local part
  if(j < 0) then
    cycle
  end if
  istart = ia_loc(i)
  iend = ia_loc(i+1)-1
  !Set non-zero entries
  call MatSetValues(mat_gbl,1,j-1, iend-istart+1,          &
                    col_idx_l2pg(istart:iend)-1,           &
                    a_loc(istart:iend),Insert_Values, ierr)
end do
!Assemble matrix, using the 2-step process
call MatAssemblyBegin(mat_gbl,MAT_FINAL_ASSEMBLY,ierr)
!Additional computations while communication
call MatAssemblyEnd(mat_gbl,MAT_FINAL_ASSEMBLY,ierr)
```

Figure 15: Sample Code for Jacobi Matrix Assembly

2.4.3 Parallel Linear Solver

The solution of the governing equations in ParMIN3P-THCm is obtained in a three-level process, the first level consists of the solution of the linearized equations, the second level involves Newton's iterative method and a Picard iterative method for the linearization of the equations, and the third level consists of a time-stepping solver employing the global implicit adaptive time stepping method. ParMIN3P-THCm employs the external solver PARDISO as the first level solver for the shared-memory parallel version and PETSc KSP for the distributed-memory and hybrid distributed-shared-memory parallel version.

The PARDISO package (Schenk and Gärtner 2011) is a robust software package for solving large sparse linear systems of equations on computers with shared-memory architecture. In order to improve the numerical factorization performance, the algorithm is based on the Level-3 BLAS update, and pipelining parallelism is exploited with a combination of left- and right-looking Level-3 BLAS supernode techniques. The parallel pivoting methods allow complete supernode pivoting in order to balance numerical stability and scalability during the factorization process (Schenk and Gärtner 2011).

PARDISO calculates the solution of a set of sparse linear equations $A\mathbf{X}=B$ with a parallel *LU*, *LDL* or *LLT* factorization, where *A* is the sparse matrix, *B* is the right hand side vector and **X** is the solution vector. The solver allows a combination of direct and iterative methods in order to accelerate the linear solution process for transient simulations. The solver uses a numerical factorization $A=LU$ for the first system and applies these exact factors *L* and *U* for the next steps in a preconditioned Krylov-subspace iteration. If the iteration does not converge, the solver will automatically switch back to the numerical factorization. This is particularly useful for systems with gradually changing values of nonzero entries in the coefficient matrix, but the same identical sparse patterns. Sample code involving calls to the PARDISO solver is given in Figure 16.

```
phase = 11 ! only reordering and symbolic factorization
call pardiso(pt_in, maxfct, mnum, mtype, phase, n_in, &
             a_in, ia_in, ja_in,idum, nrhs, iparm_in,      &
             msglvl, ddum, ddum, error)
phase = 22 ! only numerical factorization
call pardiso (pt_in, maxfct, mnum, mtype, phase, n_in,&
              a_in, ia_in, ja_in, idum, nrhs, iparm,         &
              msglvl, ddum, ddum, error)
phase = 33 ! only substitution iterative refinement
call pardiso (pt_in, maxfct, mnum, mtype, phase, n_in,&
              a_in, ia_in, ja_in, idum, nrhs, iparm,         &
              msglvl, b_in, x_out, error)
```

Figure 16: Sample Code for PARDISO Solver

PETSc (Balay et al. 2014a; Balay et al. 2014b; Balay et al. 1997) specializes in Krylov-type iterative solvers but offers interfaces for external direct solvers (e.g., MUMPS, SuperLU) and other iterative solvers (e.g., HYPRE, Trilinos/ML). Each linear solver object in PETSc actually contains two parts: the Krylov space methods and preconditioners, as shown in Table 1. Sample code involving calls to the PETSc solver is provided in Figure 17.

Table 1: Linear Solvers in PETSc

Krylov Methods (KSP)	Preconditioners (PC)
Conjugate Gradient	Block Jacobi
GMRES	Overlapping Additive Schwarz
CG-Squared	ICC
BI-CG-stab	ILU
etc.	etc.

```
!Form initial guess, only assemble local part without ghost nodes
call form_initial_guess(rank,dmda_react%da,x_inout,x_react_loc,x_react,        &
                        nngl_in, row_idx_l2pg,col_idx_l2pg,b_non_interlaced)
!Compute function, only assemble local part without ghost nodes
call compute_function(rank,dmda_react%da,b_in,b_react_loc,b_react,nngl_in, &
                        row_idx_l2pg,col_idx_l2pg,b_non_interlaced)
!Compute jacobian matrix, only assemble local part without ghost nodes
call compute_jacobian(rank,dmda_react%da,a_react,a_in,ia_in,ja_in,nngl_in,    &
                        row_idx_l2pg,col_idx_l2pg,b_non_interlaced)
!Solve linear equations AX=B
call KSPSetOperators(ksp_react,a_react,a_react,ierr)
call KSPSetDM(ksp_react,dmda_react%da,ierr)
call KSPSetDMActive(ksp_react,PETSC_FALSE,ierr)
call KSPSolve(ksp_react,b_react,x_react,ierr)
!Get iteration and convergence information
call KSPGetIterationNumber(ksp_react,itsolv,ierr)
call KSPGetErrorIfNotConverged(ksp_react, over_flow,ierr)
!Get residual norm
call KSPGetResidualNorm(ksp_react, rnorm, ierr)
!Scatter ghost points to local vector, using the 2-step process
call
DMGlobalToLocalBegin(dmda_react%da,x_react,INSERT_VALUES,x_react_loc,ierr)
!Additional computations while communication
call DMGlobalToLocalEnd(dmda_react%da,x_react,INSERT_VALUES,x_react_loc,ierr)
```

Figure 17: Sample Code for PETSc Solver

2.4.4 Parallel Input and Output

For the shared-memory parallelization, input and output (file read and write only, excluding other computations) are executed by the master thread and are processed sequentially. For the distributed-memory parallelization and hybrid distributed-shared-memory parallelization, parallel input and output are supported.

Two types of data file output are considered in ParMIN3P-THCm: the distributed data file and the integrated data file. The distributed data file is a single file containing local data from a subdomain without information from the ghost nodes. The integrated data file contains data from the entire domain. The distributed data file output is suitable for all computational platforms with distributed memory including PC clusters with distributed storage for each PC node. The integrated data file output is designed to work for the computers with a shared (parallel) file system.

The spatial data in a parallel system is stored in a non-contiguous space, as shown in Figure 18. For distributed data files, file operation is straightforward as each processor only deals with

its own data; as a result no communication is needed. For the integrated data files, collective and contiguous I/O is employed making use of MPI libraries.

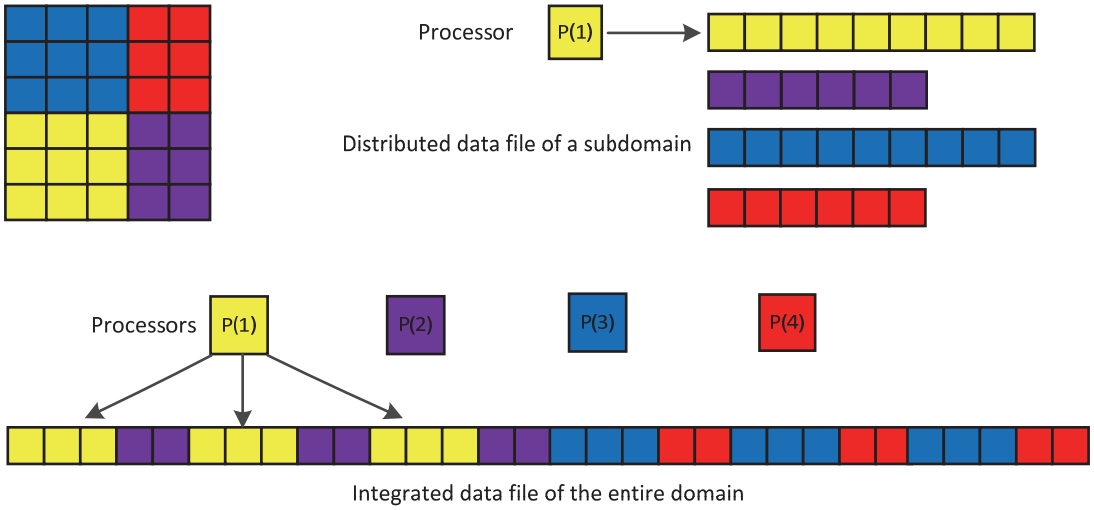

Figure 18: Parallel Output of Non-contiguous Data

Generally, there are three levels of file operation (Figure 19). The first level operation reads and writes a single piece of data for each I/O function, the second level operation uses a block of data and the third level operation reads and writes the entire data. Implementation of I/O following the first level approach is easiest from a coding perspective, but is subject to poor parallel performance. The third-level approach is most efficient from the perspective of parallelization, but is also most complex and difficult to implement. In ParMIN3P-THCm, data read is based on the Level 2 approach and data write is based on the Level 3 approach. Since data read is usually used only once when the program starts, utilizing Level 2 for data read provides a balance between efficiency and coding complexity. Sample code of data write based on Level 3 file operation is given in Figure 20.

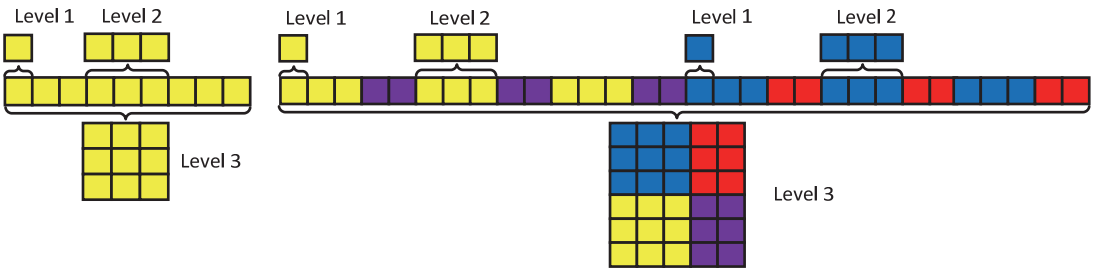

Figure 19: Three Level Parallel Input and Output

```
call MPI_TYPE_CREATE_SUBARRAY(mpiarray_ndim,mpiarray_sizes_gbl,mpiarray_sizes_sub,
          mpiarray_starts_sub, MPI_ORDER_FORTRAN,MPI_REAL4,mpiarray_filetype,ierrcode)
call MPI_TYPE_COMMIT(mpiarray_filetype, ierrcode)
call MPI_FILE_SET_VIEW(iunit,offset,MPI_REAL4,filetype, 'native', MPI_INFO_NULL,ierrcode)
call MPI_FILE_WRITE_ALL(iunit,datavalue,ndatavalue,MPI_REAL4, wstatus, ierrcode)
```

Figure 20: Sample Code of Data Write Using Collective and Contiguous I/O

Parallel read and write of the restart files in ParMIN3P-THCm is also implemented in the same way as mentioned above.

2.4.5 Summary of Parallel Modules

A summary of the major parallel modules included in ParMIN3P-THCm is given in Table 2.

Table 2: Parallel Modules Included in ParMIN3P-THCm

Module/Version	Shared-Memory	Distributed-Memory	Hybrid
Global/local numbering	-	X	X
Matrix Assembly	X	X	X
Linear Solver	X	X	X
Input/Output	-	X	X
others (e.g., mass balance, restart)	X	X	X

Symbols X: Included in parallel version, -: Not Applied

3. PARALLEL PERFORMANCE

Traditionally, speedup is defined as the execution time using one processor divided by the execution time using n processors. ParMIN3P-THCm has been run with up to 768 processors for problem sizes up to 100 million unknowns. Examples demonstrating the code performance are discussed below. Larger problem sizes would be possible; however, access to such computer hardware is currently not available.

3.1 RUNTIME PROFILING TOOL

Runtime profiling is an indispensable tool to identify the hot spots and bottlenecks of the parallel system, and is useful for improving parallel efficiency. Two types of profiling tools are embedded in ParMIN3P-THCm. The first tool is based on the statistics of wall-clock time spent on matrix assembly, in the linear equation solver and on other tasks, such as mass balance calculations, for every time step, Newton iteration and Picard iteration. The second tool is provided by PETSc and is based on the CPU Flops and wall-clock time spent on the functions used in the parallel solver. Both tools are used for parallel performance assessment.

In this report, we focus on using the wall-clock time to assess the parallel performance, considering that this parameter is most relevant for the end-user. The time spent on the following processes has been calculated: matrix assembly in flow equations, matrix assembly in reactive transport equations, linear solver in flow equations and linear solver in reactive transport equations. The speedup is calculated based on these components as well as total runtime.

3.2 COMPUTER ARCHITECTURE FOR PERFORMANCE TESTING

3.2.1 Shared-memory Architecture

The computer employed for the performance testing of the ParMIN3P-THCm shared-memory parallel version was a workstation with 2 Intel Xeon E5 2650 sockets. Each socket had 8 cores

(processors) that ran at 1.8GHz with turbo speed at 2.3 GHz. The available memory was 128 GB shared by the 16 processors.

3.2.2 Distributed-memory Architecture

The computer used for the performance testing of the ParMIN3P-THCm distributed-memory parallel version was the Orcinus cluster (https://www.westgrid.ca/support/systems/Orcinus) operated by WestGrid. Phase One of the Orcinus cluster is comprised of 12 c7000 chassis, each containing 16 dual-density BL2x220 Generation 5 blades. There are 2 compute servers per blade (an A node, and a B node). Every node has 2 sockets, each containing an Intel Xeon E5450 quad-core processor, running at 3.0 GHz. In total, there are 3072 Phase One cores. The 8 cores in a single Phase One node share 16 GB of RAM. Phase Two is comprised of 17 c7000 chassis, each containing 16 dual-density BL2x220 Generation 6 blades. Again, there are 2 compute servers per blade (an A node, and a B node). Every node has 2 sockets, each containing an Intel Xeon X5650 six-core processor, running at 2.66 GHz. In total there are 6528 Phase Two cores. The 12 cores in a single Phase Two node share 24 GB of RAM. The total number of cores available is 9600. For this work, Phase Two was used for the parallel performance testing.

3.2.3 Hybrid Distributed-shared-memory Architecture

The computer used for the performance test of the ParMIN3P-THCm hybrid distributed-shared-memory parallel version was the Jasper cluster (https://www.westgrid.ca/support/systems/Jasper) operated by WestGrid. Jasper is an SGI Altix XE cluster with an aggregate 400 nodes, 4160 cores and 8320 GB of memory. Phase One has 240 nodes and each node is equipped with Xeon X5675 processors, 12 cores (2 x 6) and 24 GB of memory. Of these, 32 have additional memory for a total of 48 GB. Phase Two has 160 nodes, formerly part of the Checkers cluster, and each node is equipped with Xeon L5420 processors, 8 cores (2 x 4) and 16 GB of memory. Phase One was used for the parallel performance testing. The Jasper cluster was also used for additional performance testing of the ParMIN3P-THCm distributed-memory parallel version.

3.3 CASES FOR PARALLEL PERFORMANCE TESTING

3.3.1 Case I: Complex cement/clay interactions

3.3.1.1 Case Introduction

Use of the subsurface for CO_2 storage, geothermal energy and nuclear waste geological disposal will greatly increase the interaction between clay or claystone and concrete. The development of models describing the mineralogical transformations at this interface can be computationally challenging because contrasting geochemical conditions (Eh, pH, solution composition, etc.) induce steep concentration gradients and high mineral reactivity. Due to the geochemical complexity of the problem, analytical solutions are not available to verify code accuracy, and the problem must be solved numerically. Processes considered in the simulations are diffusion-controlled transport in saturated porous media under isothermal conditions, involving both equilibrium and kinetically controlled mineral-dissolution-precipitation reactions and cation exchange. A recent model intercomparison demonstrates that reactive-transport modelling can be used effectively in support of long-term performance assessment related to clay-concrete systems (Marty et al. 2015).

3.3.1.2 Model Discretization

An one-dimensional radial geometry was chosen for modelling the long-term geochemical evolution surrounding a tunnel in a radioactive nuclear waste repository site. For the two interacting materials, the host rock can be considered of infinite extent, whereas the spatial extent of the concrete is limited. A heterogeneous mesh with a refined spatial resolution of 0.05 m focused on the concrete/clay interface was used for the simulations. Details of the spatial discretization are given in Figure 21. The mesh size was selected by Marty et al. (2015) to ensure a satisfactory compromise between computational time and a spatial resolution capable of capturing the expected geochemical processes, especially at the interface. The numerical parameters used for this case are given in Table 3.

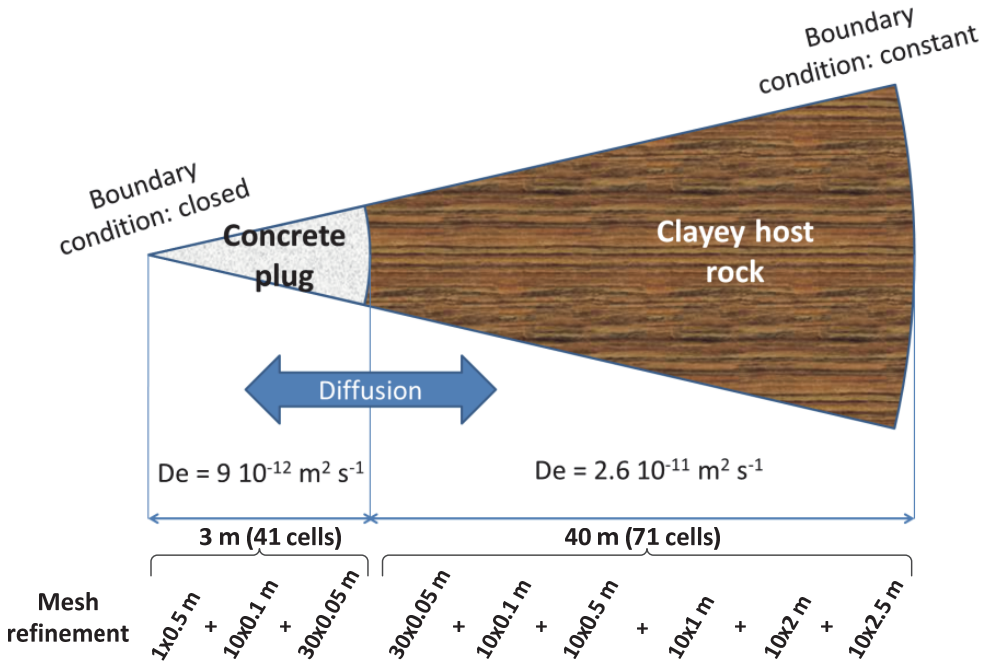

Figure 21: Model Domain for Case I, Discretized into a Heterogeneous Mesh with a Refined Spatial Resolution of 0.05 m at the Concrete/Clay Interface (from Marty et al. 2015)

Table 3: Numerical Parameters Used in Case I

Parameters	Value
Number of nodes in horizontal direction	112
Total number of nodes	112
Degree of freedom per node (number of components)	13
Total degrees of freedom	1456
Non-zero matrix entries for flow equations	334
Non-zero matrix entries for reactive transport equations	56446
Simulation time	10000 years
Maximum time step	0.1 year

3.3.1.3 Sample Results

Sample results showing mineralogical changes and pH evolution after 10000 years are depicted in Figure 22. The following mineralogical transformations were observed: dissolution of smectite (weak), quartz and dolomite from the claystone, and of C_3FH_6, monocarboaluminate, CSH1.6 and portlandite in the concrete; precipitation of calcite, saponite and clinoptilolite in the claystone, and of ettringite, saponite, ferrihydrite, magnetite, CSH1.2 and CSH0.8 in the concrete.

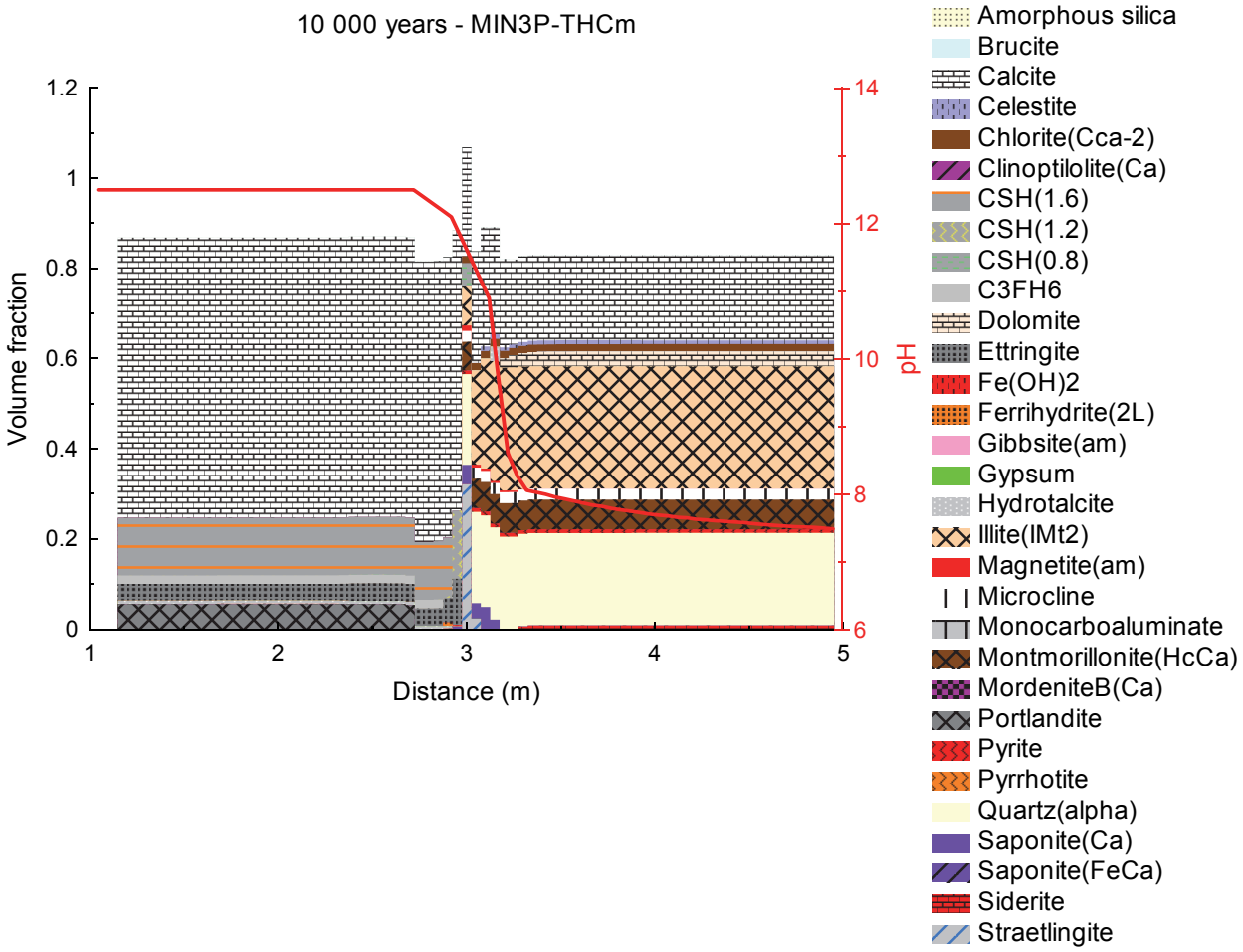

Figure 22: Mineralogical Alterations and pH Changes after 10000 Years of Concrete/Clay Interactions (from Marty et al. 2015). The Concrete-Claystone Interface is Located at a Distance of 3.0 m

3.3.1.4 Runtime Profiling

Case I represents a small-scale simulation with most CPU time spent on matrix assembly and in the reactive transport solver. The runtime percentages for this case are shown in Table 4. It should be mentioned that the CPU time in the matrix assembly of the reactive transport equations includes the time for solving local mass conservation equations for each control volume.

Table 4: Runtime Percentage Distribution for Case I

Flow equations			Reactive transport equations			Other
Assembly	Solver	Other	Assembly	Solver	Other	
0.0%	0.0%	0.0%	90.0%	7.4%	0.2%	2.4%

3.3.2 Case II: Uranium Remediation by Lactate Injection

3.3.2.1 Case Introduction

This case explores the hydrogeochemical patterns that develop under steady-state and transient groundwater flow conditions during uranium bioremediation. A simplified conceptual model illustrating the major biotic and abiotic reactions considered is depicted in Figure 23. The main challenge is the complexity of the biogeochemical reaction network, which includes various parallel, sequential and competing kinetic reactions with a strong interdependency of processes. In addition, some of these reactions are mixing-controlled and the reaction progress as well as the resulting solution chemistry are highly sensitive to physical mixing and therefore potentially compromised by numerical dispersion. This causes a strong coupling between the physical transport and geochemical reaction processes. Inaccurate solutions of the physical transport processes will be more pronounced in multi-dimensional problems, where finer grid resolutions that typically minimize numerical dispersion come at larger computational costs and pragmatic choices have to be made to attain sufficient accuracy at reasonable computational costs (Şengör et al. 2015). This case also includes an injection well, implying that geochemical changes occur locally in the system, posing an additional challenge for parallel execution due to load balancing.

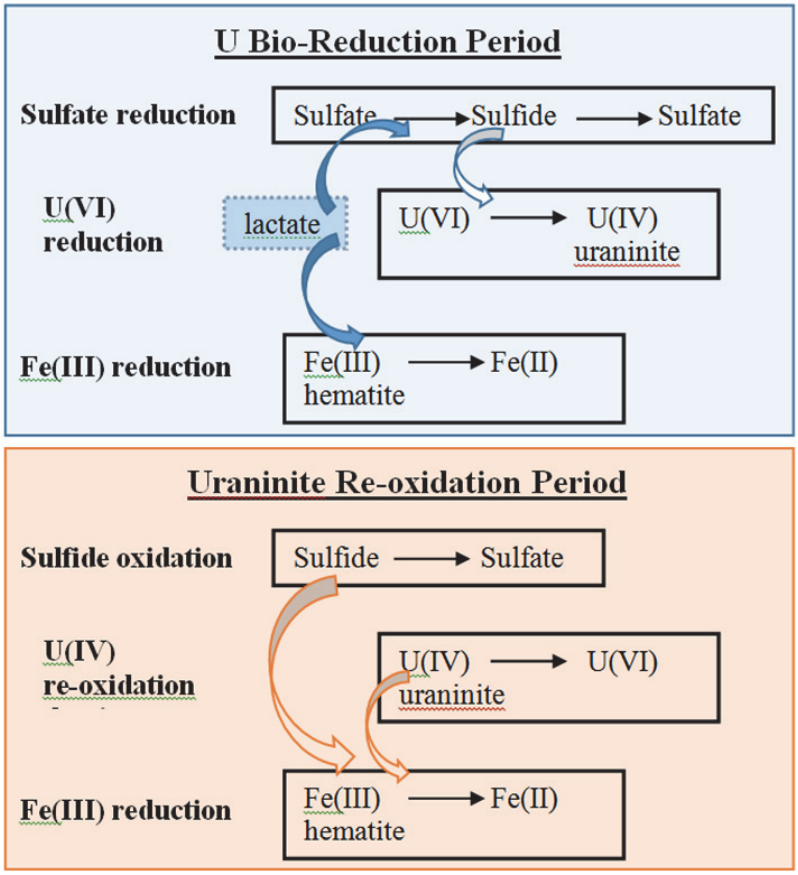

Figure 23: Simplified Conceptual Model Illustrating the Biotic and Abiotic Reaction Network of Case II (from Şengör et al. 2015)

3.3.2.2 Model discretization

The model domain for the 2D simulation was defined to be 18 m in length and 10.5 m in width, with a uniform grid discretization of 0.125 m in both horizontal and vertical directions, as shown in Figure 24. An injection well was defined at a location 7.25 m downstream of the influent boundary. It was assumed that the injection of a lactate-containing solution occurred at this location at a rate of 0.2 m^3/day during the initial 8 days of the simulation. The 2D simulations assumed a longitudinal dispersivity of 1.0 m and a transverse dispersivity of 0.1 m. The total simulation period was defined to be 60 days with a nominal time step size of 0.01 days. Numerical parameters used for this case are summarized in Table 5.

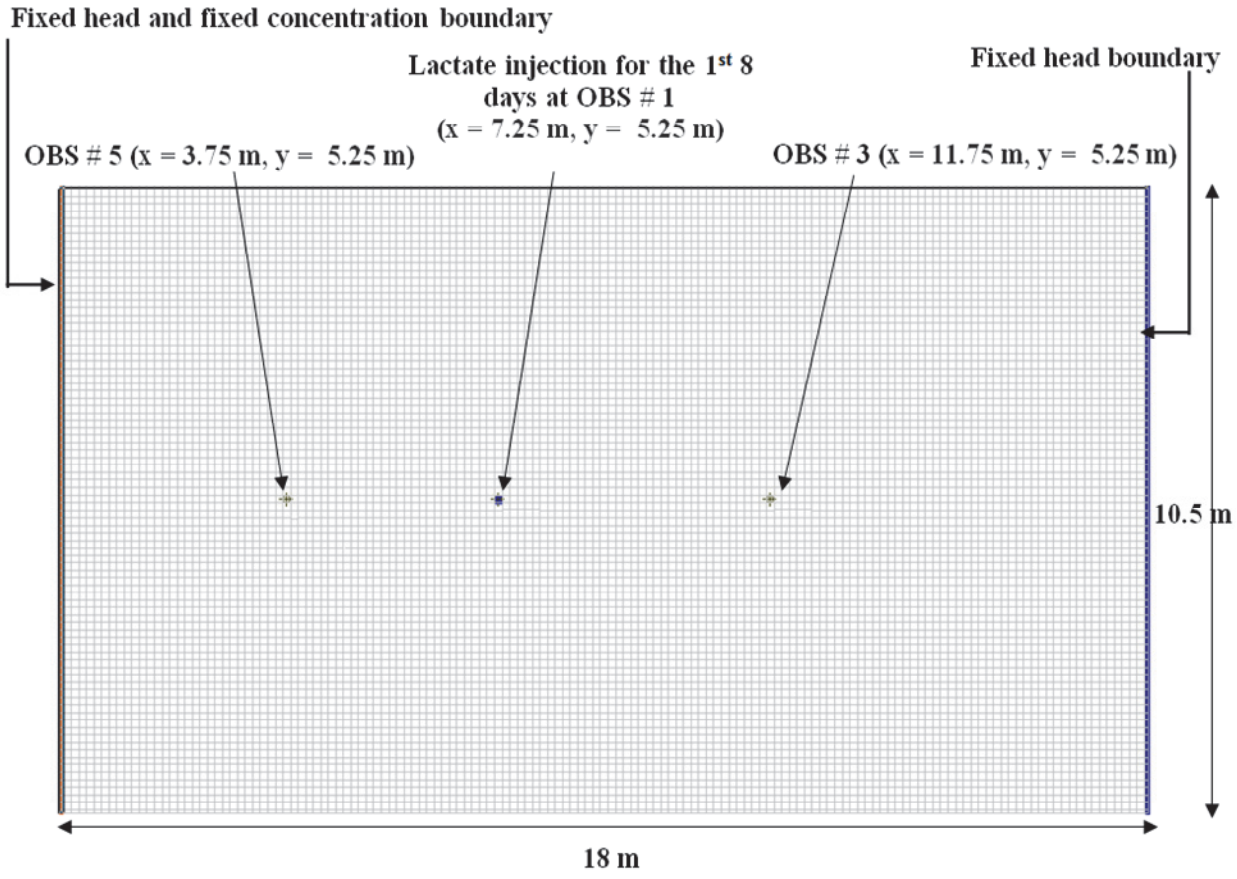

Figure 24: Model Grid and Boundary Conditions for Case II (from Şengör et al. 2015)

Table 5: Numerical Parameters Used in Case II

Parameters	Value
Number of nodes in horizontal direction	145
Number of nodes in vertical direction	85
Total number of nodes	12325
Degree of freedom per node (number of components)	17
Total degrees of freedom	209525
Non-zero matrix entries for flow equations	61165
Non-zero matrix entries for reactive transport equations	6134805
Simulation time	60 days
Maximum time step	0.01 day

3.3.2.3 Sample Results

Concentration contours for selected aqueous components, mineral phases, and biomass are depicted in Figure 25. The figure illustrates the spatial extent and distribution of the uranium immobilization that was induced by the lactate injection and associated geochemical changes surrounding the injection well. A detailed description and interpretation of the simulation results can be found in Şengör et al. (2015).

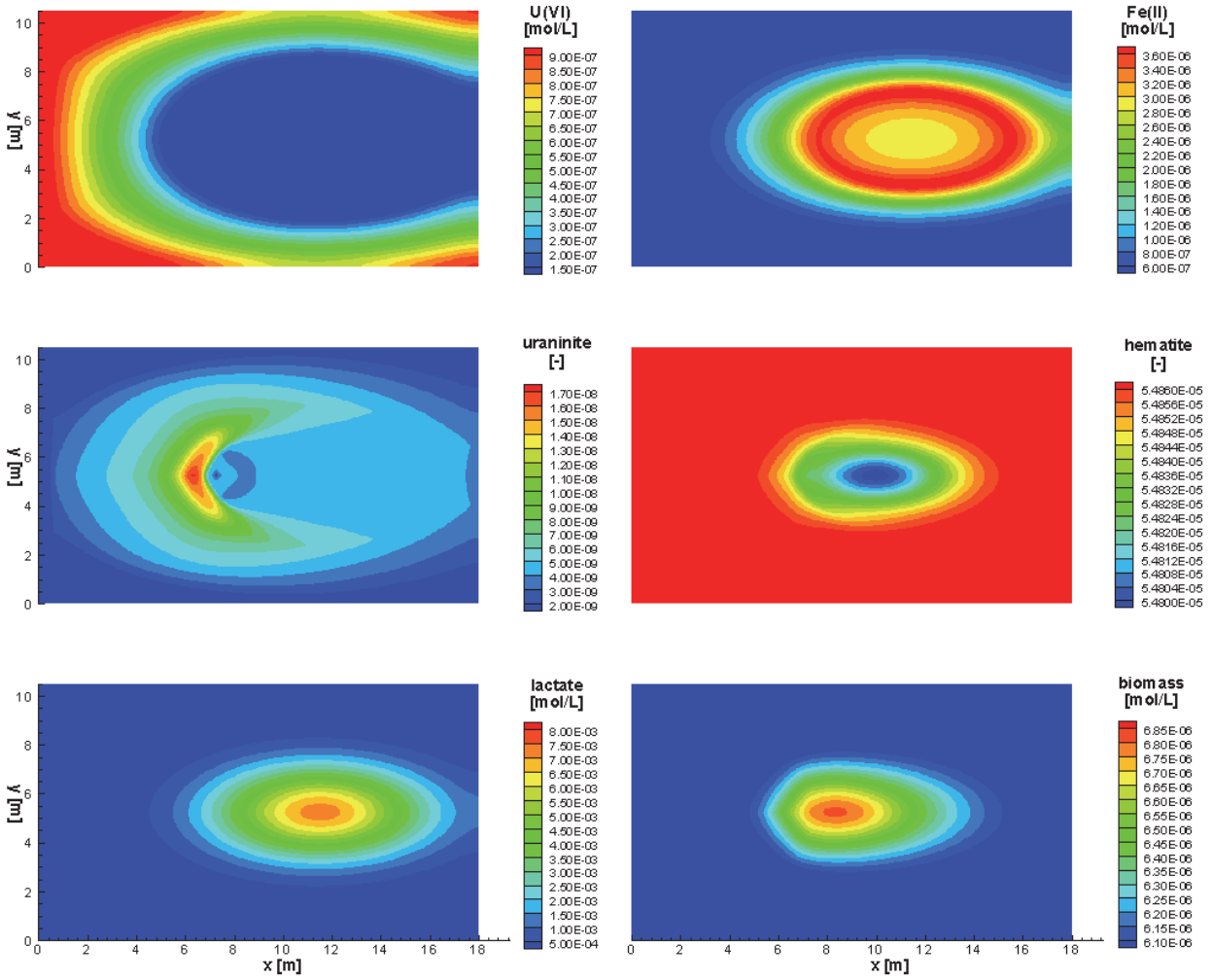

Figure 25: Concentration Distributions for Selected Aqueous Components, Minerals and Biomass at 60 Days for Case II

3.3.2.4 Runtime Profiling

Case II represents a simulation with an intermediate number of unknowns. Most of the CPU time is spent on matrix assembly and in the reactive transport solver. Compared to the CPU time required to solve the reactive transport equations, the CPU requirement for solution of the flow equations is much smaller, amounting to only 0.3% of the total CPU time. CPU time spent on the matrix assembly of the reactive transport equations includes the time for solving local mass conservation equations at each control volume. The runtime percentages for this case are summarized in Table 6.

Table 6: Runtime Percentage Distribution for Case II

Flow equations			Reactive transport equations			Other
Assembly	Solver	Other	Assembly	Solver	Other	
< 0.1%	< 0.1%	<0.1%	92.3%	3.8%	0.3%	3.6%

3.3.3 Case III: Flow and Reactive Transport in a Hypothetical Sedimentary Basin

3.3.3.1 Case Introduction

Sedimentary basins are complex systems affected by numerous interacting processes (i.e. groundwater flow, heat transfer, mass transport, water-mixing, rock-water interactions, mechanical loading, etc.). These processes affect the hydrogeological system in shallow and deep aquifers to various degrees. For instance, climate change events occurring on the time scale of thousands of years, during periods of glaciation and deglaciation can trigger hydrogeological, geochemical and mechanical alterations in sedimentary basins. Understanding the interactions between these processes is of importance for the evaluation of the hydrodynamic and geochemical stability of these sedimentary basins. This case models the complex coupled processes that could occur in a typical sedimentary basin in North America as depicted in Figure 26. Details on the simulation approach, results and interpretation can be found in Bea et al. (2010).

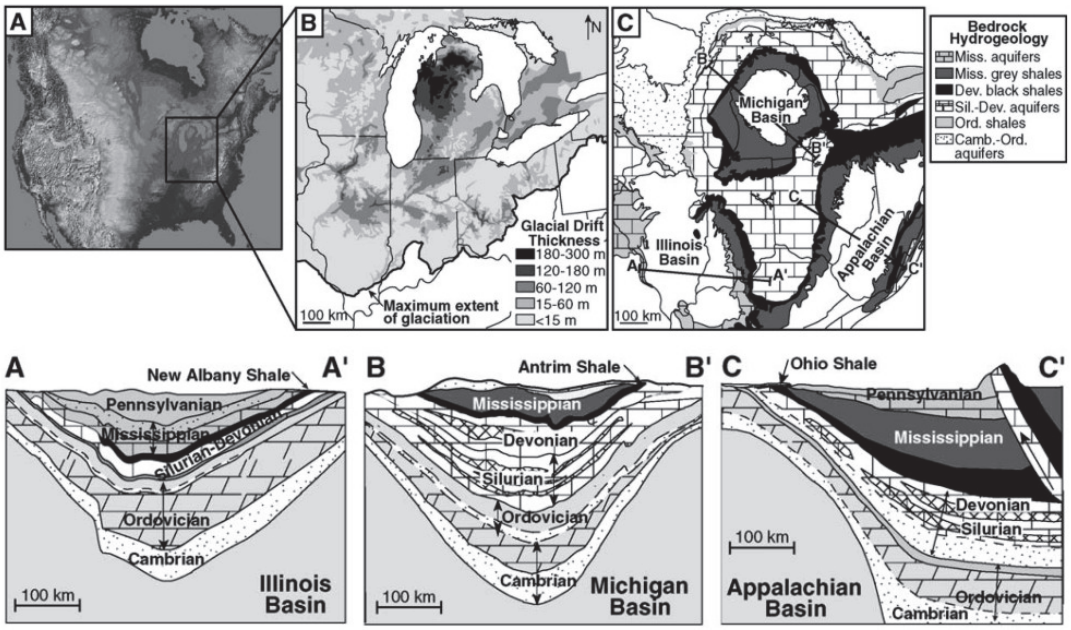

Figure 26: Location and Main Geological Features for the Intracratonic Sedimentary Basins in North America (Illinois Michigan and Appalachian Basins, Taken from McIntosh and Walter 2005)

3.3.3.2 Model Discretization

The geometry and geology of the sedimentary basin used in this case are depicted in Figure 27. A symmetrical sedimentary basin of about 400 km in length and 4 km in depth is characterized by a sequence of carbonates (dolostones and limestones, Dol1, Dol2, Dol3 and Lim1) interbedded by sandstones, which constitute the main aquifers (Sand1, Sand2, Sand3 and Sand4), and shales, which constitute the main confining units (Sh1, Sh2, Sh3). All units overlay the Pre-Cambrian basement (G). Note that a weathered zone in the basement rocks (Gw) is in direct contact with the Sand1 sedimentary unit. Interbedded evaporites (Ev) and dolostones units (Dol1) are also considered. Numerical parameters used in this case are summarized in Table 7.

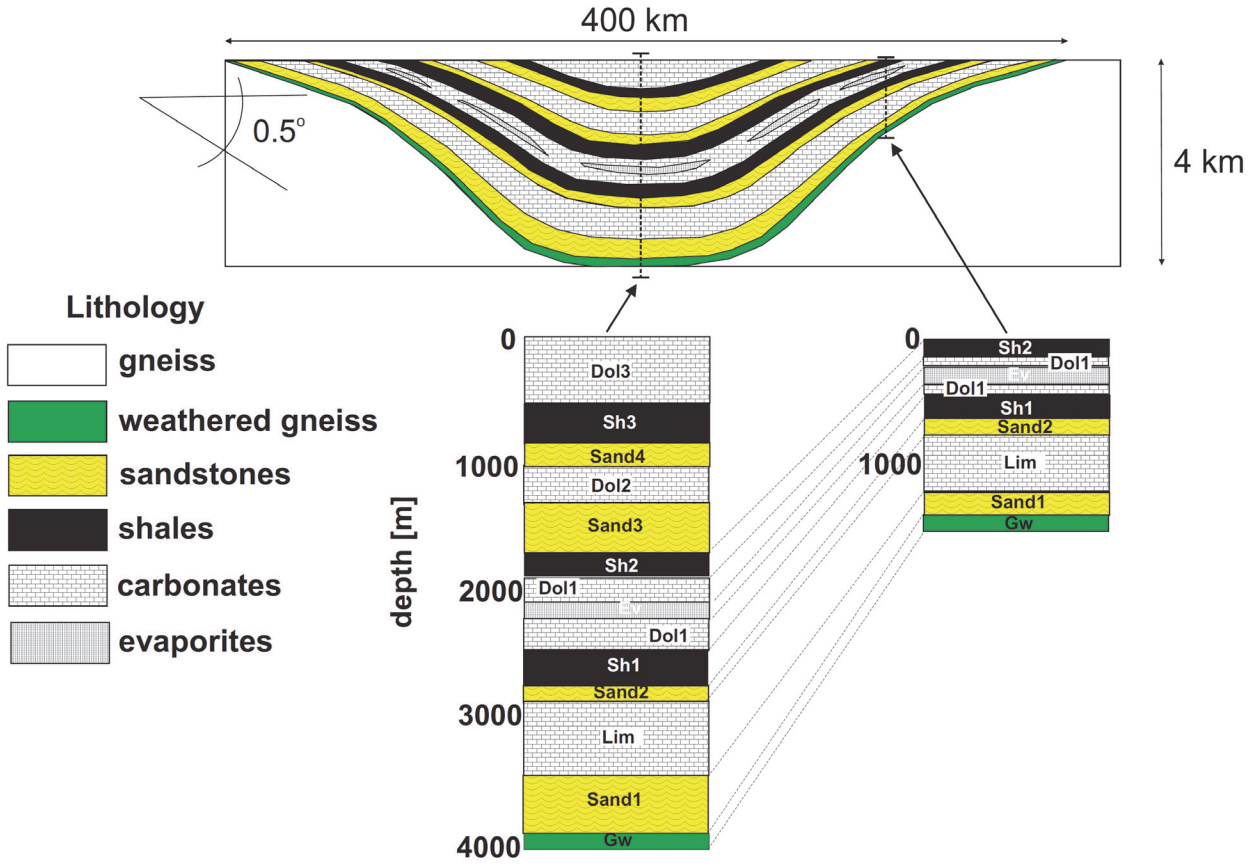

Figure 27: Geometry and Main Hydrogeological Units Considered for Case III (Bea et al. 2010)

Table 7: Numerical Parameters Used in Case III

Parameters	Value
Number of nodes in horizontal direction	450
Number of nodes in vertical direction	100
Total number of nodes	45000
Degrees of freedom per node (number of components)	9
Total degrees of freedom	405000
Non-zero matrix entries for flow equations	1251208
Non-zero matrix entries for reactive transport equations	18135900
Simulation time	32500 years
Maximum time step	5.0 years

3.3.3.3 Sample Results

Although glaciation events significantly affect the fresh water balance in the system (especially during melting and glacial retreat), the modelling results demonstrate that these ancient and deep sedimentary basins are hydrodynamically and geochemically stable. Sample results of total Ca concentrations at different times are depicted in Figure 28 and show that temporal changes are restricted to shallow depths. Detailed simulation results and interpretations are provided in Bea et al. (2010).

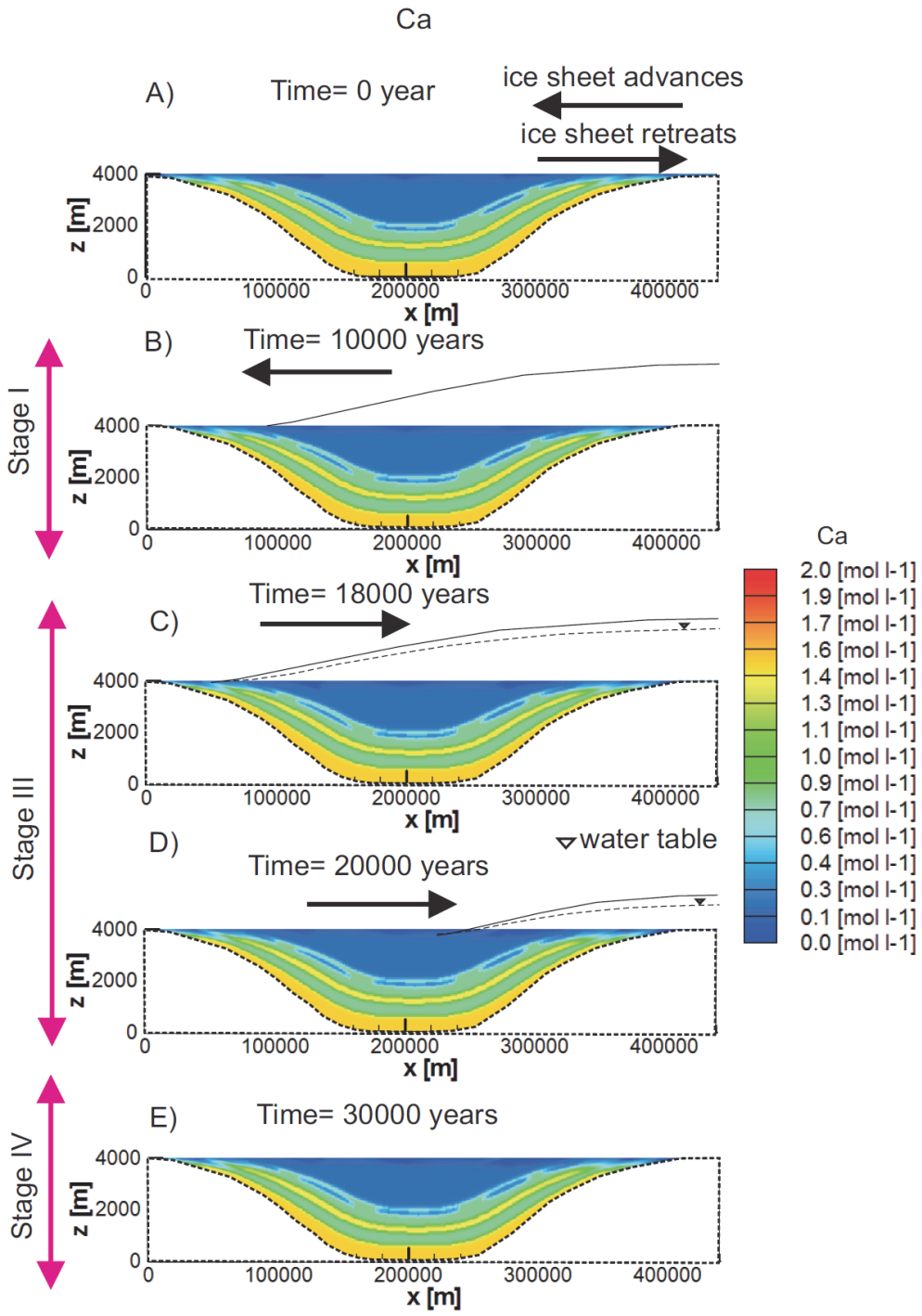

Figure 28: Distribution of Total Ca Concentration at Different Output Times for Case III

3.3.3.4 Runtime Profiling

Similar to Case II, this simulation represents a case with an intermediate number of unknowns. Most of the CPU time is spent on matrix assembly for the reactive transport problem and in the reactive transport solver. In comparison to the CPU-times required for the solution of the reactive transport equations, the computational requirements for solution of the flow equations is much smaller, requiring less than 1.8% of the total computing time. The CPU time spent in the

matrix assembly for the reactive transport equations includes the time for solving local mass conservation equations at each control volume. The runtime percentage distribution for this case is summarized in Table 8. System I/O is significant in this case and comprises around 19.7% of the total runtime.

Table 8: Runtime Percentage Distribution for Case III

Flow equations			Reactive transport equations			Other
Assembly	Solver	Other	Assembly	Solver	Other	
0.2%	1.5%	< 0.1%	72.8%	5.2%	0.6%	19.7%

3.4 SHARED-MEMORY PARALLEL PERFORMANCE

The matrices of the three cases used for the performance testing are well conditioned, implying that the iterative solver WatSolv is faster than the direct solver PARDISO, if the number of processors used is small (i.e., not more than 8 processors). For the shared-memory parallel performance testing, the iterative solver WatSolv was used. The parallel performance therefore mainly depends on the speedup of the matrix assembly for the flow and reactive transport equations.

3.4.1 Case I: Complex Cement/Clay Interactions

3.4.1.1 Solver Statistics

The solver and runtime statistics for Case I are given in Table 9. The number of time steps, Newton iterations and solver iterations change when different numbers of processors are used. This is due to round-off errors and is generally inevitable in parallel codes. The linear solver requires around 25 iterations for each Newton iteration, and around 11 Newton iterations are required for every time step. For this case, despite the small number of unknowns, the total runtime is improved significantly with an increasing number of processors. However, due to the small size of the problem, the performance improvement does not scale ideally with the number of processors due to the overhead in the parallel part of the code and the remaining sequential sections of the code in the shared-memory version.

Table 9: Solver and Runtime Statistics of Shared-memory Parallelization for Case I*

Number of Processors	1	4	8
Number of Newton iterations in flow equations	0	0	0
Number of solver iterations in flow equations	0	0	0
Number of Newton iterations in reactive transport equations	83580	81594	80711
Number of solver iterations in reactive transport equations	2088798	2084596	2084495
Number of time steps	7597	7418	7333
Total runtime (hours)	2.01	0.64	0.43

* Performance testing based on simulation time of 200 years.

3.4.1.2 Parallel Speedup

Parallel speedup of matrix assembly and total speedup are depicted in Figure 29. The matrix assembly shows good speedup for up to 8 processors. However, the total speedup does not increase as much due to the parallel overhead, the sequential solver and system IO. For such a small case, this speedup can be considered satisfactory and indicates good code performance.

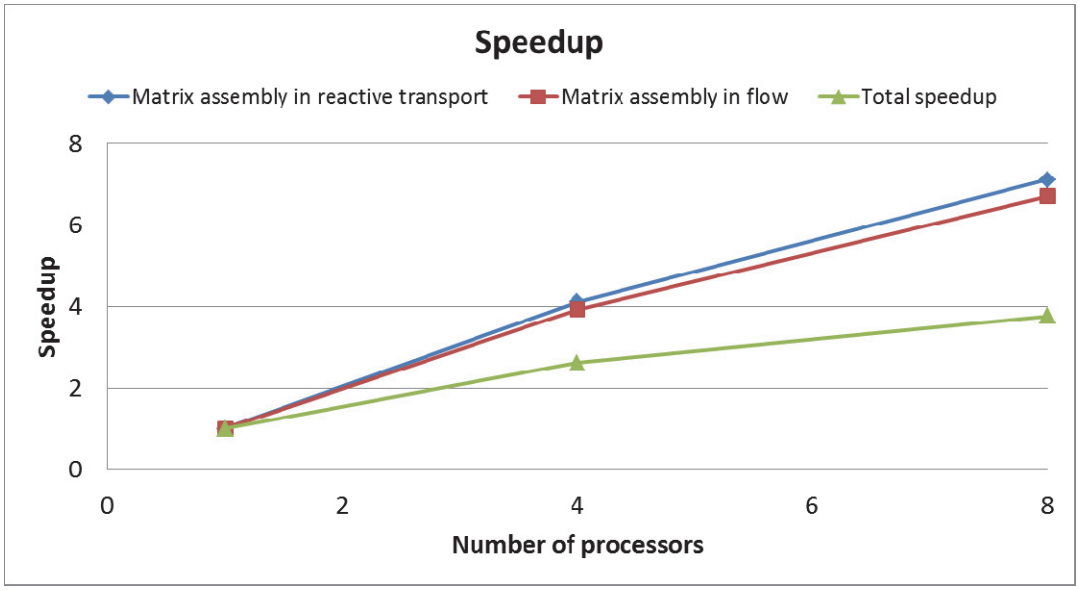

Figure 29: Speedup of OpenMP Parallel Version for Case I, Executed on a Shared-memory Workstation

3.4.2 Case II: Uranium Remediation by Lactate Injection

3.4.2.1 Solver Statistics

The solver and runtime statistics for Case II are summarized in Table 10. For the solution of the flow equations, the linear solver takes less than 4 iterations for each nonlinear Newton iteration. For the solution of the reactive transport equations, the linear solver also only requires a small number of iterations (3 iterations) per nonlinear Newton iteration. For this case, significant reductions of runtime are seen for an increased number of processors; however, as for Case I, runtime reduction does not scale linearly with the number of processors mostly due to the remaining sequential parts in the shared memory version. Memory bandwidth may be also an important factor that affects the parallel performance.

Table 10: Solver and Runtime Statistics of Shared-memory Parallelization for Case II*

Number of Processors	1	4	8
Number of Newton iterations in flow equations	190	190	190
Number of solver iterations in flow equations	661	661	661
Number of Newton iterations in reactive transport equations	1449	1449	1449
Number of solver iterations in reactive transport equations	3274	3274	3274
Number of time steps	190	190	190
Total runtime (hours)	3.19	0.98	0.64

* Performance testing based on simulation time of 1 day.

3.4.2.2 Parallel Speedup

Parallel speedup of the matrix assembly for both flow and reactive transport, and total speedup are presented in Figure 30. The matrix assembly for reactive transport shows good speedup for up to 8 processors. However, the matrix assembly for flow does not achieve good speedup due to the parallel overhead, because the size of the matrix for the flow equations is much smaller than the size of the matrix for the reactive transport equations. Since the solution of the flow equations requires much less CPU time compared to the solution of the reactive transport equations, the performance of the flow part has a negligible impact on overall performance. The total speedup using 8 processors is around 5 for this case.

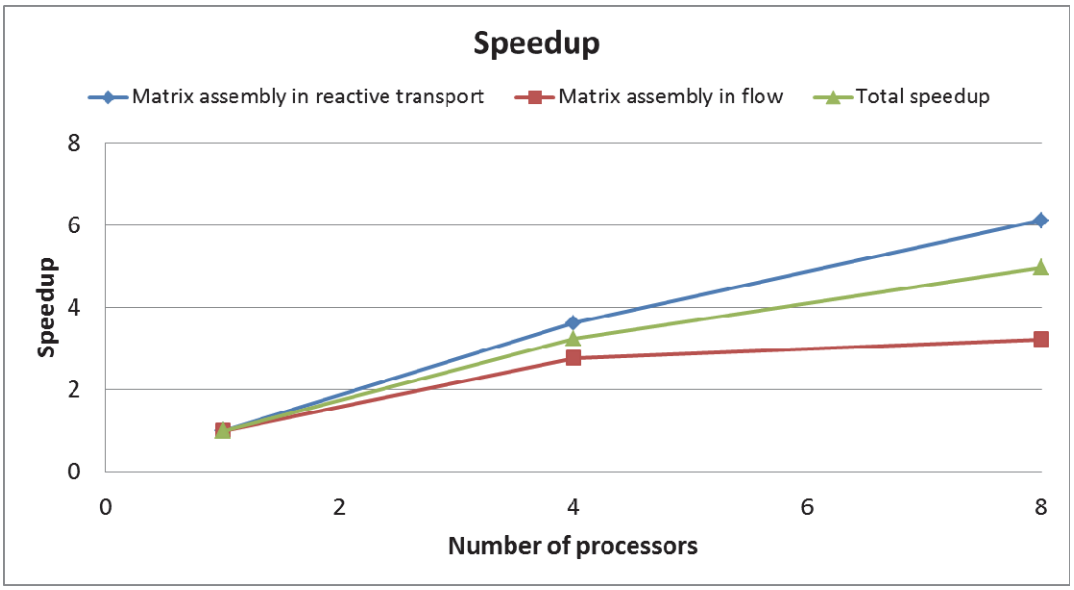

Figure 30: Speedup of OpenMP Parallel Version for Case II, Executed on a Shared-memory Workstation

3.4.3 Case III: Flow and Reactive Transport in a Hypothetical Sedimentary Basin

3.4.3.1 Solver Statistics

The solver and runtime statistics for Case III are summarized in Table 11. For solution of the flow equations, the linear solver requires around 30 iterations for each nonlinear Newton iteration. For solution of the reactive transport equations, the linear solver requires around 3 iterations for each nonlinear Newton iteration. The total runtime is reduced substantially with an increasing number of processors, although the performance is not ideal due to the remaining sequential parts of the code and parallel overhead.

Table 11: Solver and Runtime Statistics of Shared-memory Parallelization for Case III*

Number of Processors	1	4	8
Number of Newton iterations in flow equations	116	101	96
Number of solver iterations in flow equations	3235	2913	2834
Number of Newton iterations in reactive transport equations	131	105	105
Number of solver iterations in reactive transport equations	353	296	295
Number of time steps	24	25	25
Total runtime (minutes)	13.67	5.20	3.63

* Performance testing based on simulation time of 100 years.

3.4.3.2 Parallel Speedup

The parallel speedup for matrix assembly for flow, reactive transport and total computing time are depicted in Figure 31. The matrix assembly for both reactive transport and flow shows good speedup for up to 8 processors. The total speedup using 8 processors is around 4 for this case due to the sequential solver and other sequential parts of the shared-memory version.

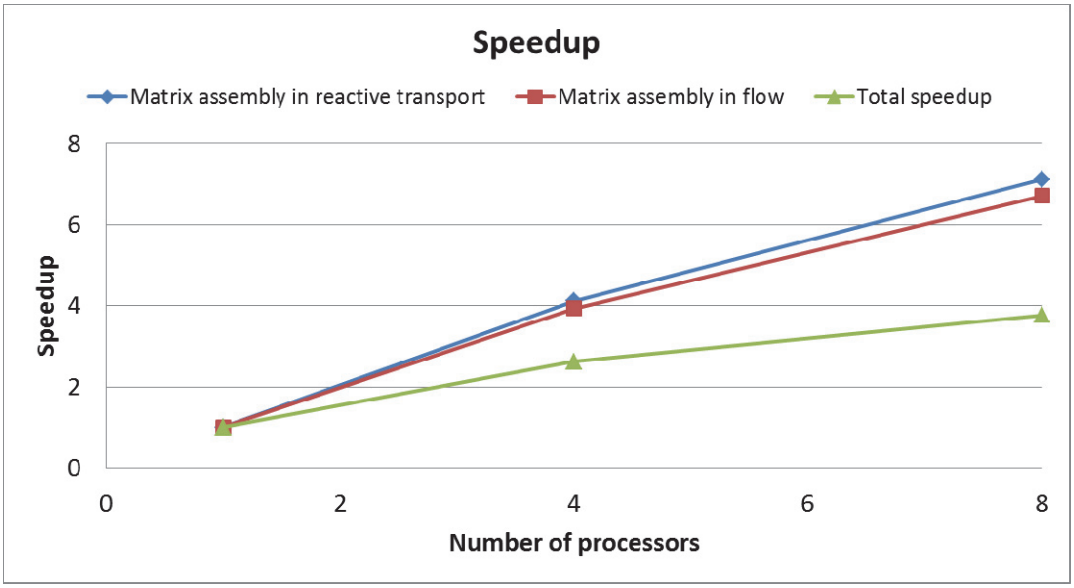

Figure 31: Speedup of OpenMP Parallel Version for Case III, Executed on a Shared-memory Workstation

3.4.4 Summary of Shared-memory Parallel Performance

For the shared-memory version, the code is not fully parallelized. The lack of complete parallelization plays an important role in the parallel performance, especially when larger numbers of processors are used. Generally, good speedup is achieved for the matrix assembly in the reactive transport part of the code, including the computation of local mass conservation equations. Other parts of the code do not show good speedup due to the parallel overhead. For ill-conditioned problems, the PARDISO direct solver provides a suitable alternative to the WATSOLV iterative solver. However, for most reactive transport problems the matrices are well conditioned and the iterative solver is generally much faster than the direct solver - if a good preconditioning method is used. These results suggest that the shared-memory version is useful for cases that do not require much CPU-time in the solver and system input and output.

3.5 DISTRIBUTED-MEMORY PARALLEL PERFORMANCE

The distributed-memory version of ParMIN3P-THCm is fully parallelized and PETSc is used as the default parallel solver. For Case I, because it represents a small scale problem, the parallel performance is only tested for up to 8 processors; however, for Case II and Case III, the parallel performance is tested for up to 128 processors.

3.5.1 Case I: Complex Cement/clay Interactions

3.5.1.1 Solver Statistics

The solver and runtime statistics for Case I are presented in Table 12. The number of Newton iterations and time steps generally increase as the number of processors increases. This is due to the domain decomposition method; the parallel solver usually requires more iterations when more processors are used.

Table 12: Solver and Runtime Statistics of Distributed-memory Parallelization for Case I*

Number of Processors	1	4	8
Number of Newton iterations in flow equations	0	0	0
Number of solver iterations in flow equations	0	0	0
Number of Newton iterations in reactive transport equations	106371	131504	138835
Number of solver iterations in reactive transport equations	2299276	2494335	2555785
Number of time steps	7846	9494	9940
Total runtime (hours)	2.86	1.31	0.88

* Performance testing based on simulation time 200 years.

3.5.1.2 Parallel Speedup

Parallel performance for Case I, including speedup for the matrix assembly of the reactive transport problem, speedup for the reactive transport solver and total speedup, is depicted in Figure 32. For this small-scale problem, the performance of the parallel solver is quite sensitive to the matrix pattern and communication cost. The speedup of the solver does not increase when 4 processors are used, but increases up to 5 when 8 processors are used. Generally, the total speedup for this case is less than 4 due to communication costs and load balancing problems, which both play a significant role due to the small problem size. The load imbalance is mainly caused by the output computational costs since not all processors export transient data during the simulation.

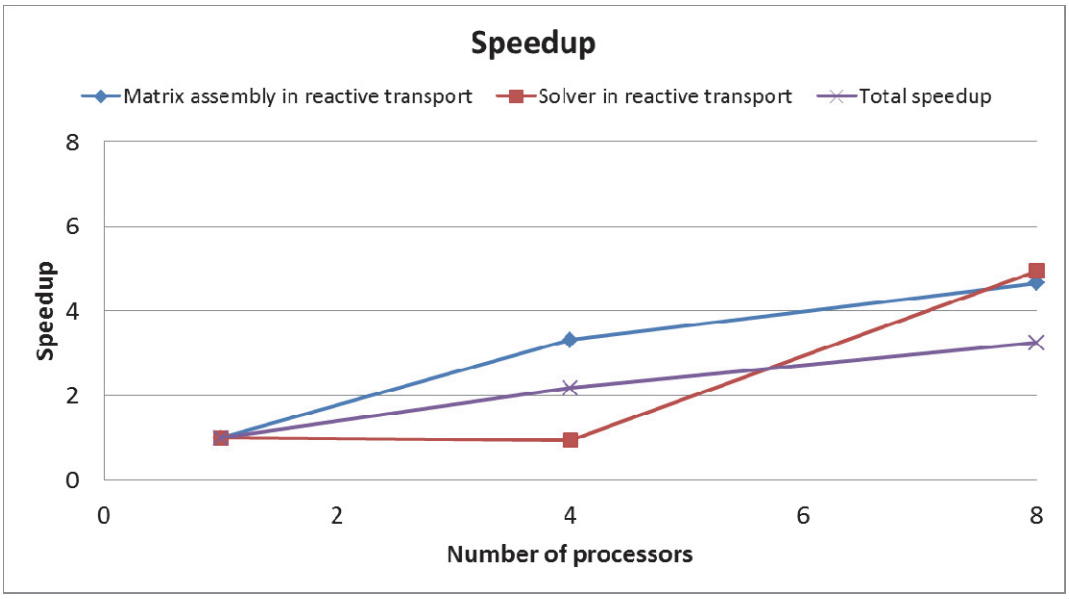

Figure 32: Speedup of MPI Parallel Version for Case I, Executed on the WestGrid Orcinus Cluster

3.5.2 Case II: Uranium Remediation by Lactate Injection

3.5.2.1 Solver Statistics

The solver statistics for Case II are given in Table 13. The number of linear solver iterations, Newton iterations and time steps generally increase as the number of processors increases due to the use of the parallel solver and the domain decomposition method.

Table 13: Solver and Runtime Statistics of Distributed-memory Parallelization for Case II*

Number of Processors	8	16	32	64	128
Number of Newton iterations in flow equations	44828	44890	44900	44898	44934
Number of solver iterations in flow equations	52851	54415	54249	55206	55725
Number of Newton iterations in reactive transport equations	243005	243906	243191	243339	244003
Number of solver iterations in reactive transport equations	599914	611589	621545	632550	653349
Number of time steps	44825	44885	44897	44893	44928
Total runtime (hours)	83.51	44.97	23.09	12.94	7.38

* Performance testing based on simulation time 12 hours.

3.5.2.2 Parallel Speedup

The speedup due to parallelization is significant for this case (Figure 33). Speedups for matrix assembly in both flow and reactive transport problems scale almost linearly with the number of processors. The speedup of the solver for the reactive transport does not increase much for more than 32 processors and the speedup of the solver for the flow problem does not increase beyond 16 processors. For the flow equations, the performance of the solver even deteriorates because of the limited size of the flow matrix (61165 nonzeros), which is much smaller in size than the matrix (6134805 nonzeros) for the reactive transport equations. The speedup for matrix assembly of the reactive transport equations increases up to 109 when using 128 processors and the parallel efficiency is 85%. The total speedup for Case II is 92 when using 128 processors and the parallel efficiency is 72%. The speedup is mainly hindered by the solver due to the problem size and the matrix properties. The output also affects parallel performance, because parallel output does not scale as well as the numerical solution of the problem.

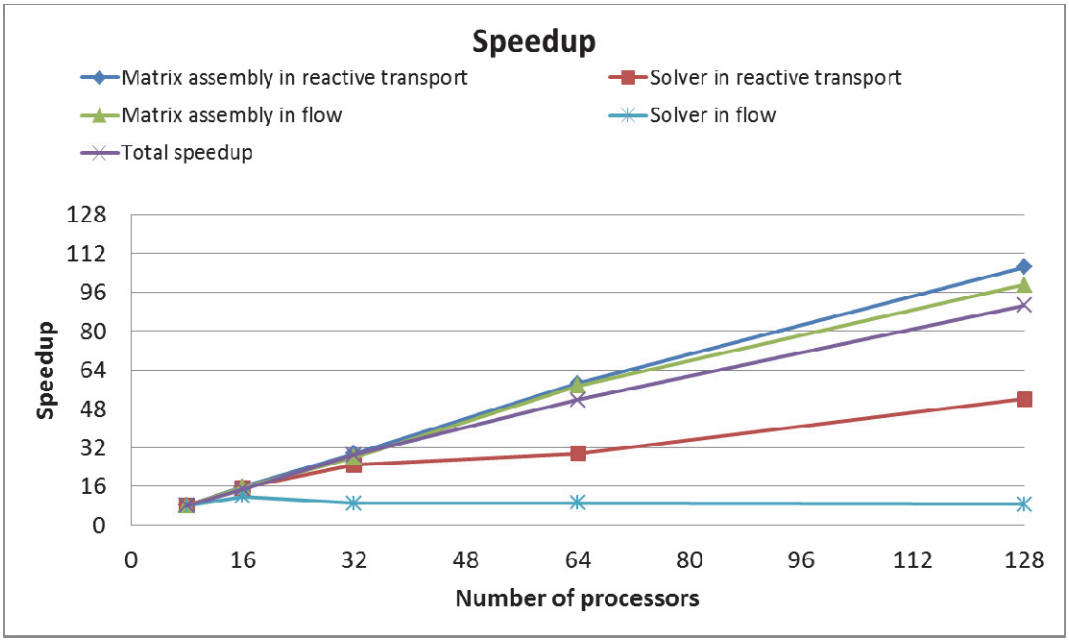

Figure 33: Speedup of MPI Parallel Version for Case II, Executed on the WestGrid Orcinus Cluster

3.5.3 Case III: Flow and Reactive Transport in a Hypothetical Sedimentary Basin

3.5.3.1 Solver Statistics

The solver statistics for Case III are given in Table 14. The number of linear solver iterations and Newton iterations fluctuates as the number of processors increases.

Table 14: Solver and Runtime Statistics of Distributed-memory Parallelization for Case III*

Number of Processors	8	16	32	64	128
Number of Newton iterations in flow equations	26980	27181	26750	26843	26662
Number of solver iterations in flow equations	84291	84442	85111	85568	81144
Number of Newton iterations in reactive transport equations	31505	31505	31441	31441	31489
Number of solver iterations in reactive transport equations	90280	90271	90284	90291	90727
Number of time steps	6504	6504	6504	6504	6504
Total runtime (hours)	8.11	4.33	2.22	1.21	0.74

* Performance testing based on simulation time 1000 years.

3.5.3.2 Parallel Speedup

Similar to Case II, the speedup is also significant for this case (Figure 34). Speedups for matrix assembly of both flow and reactive transport problems scale almost linearly with the number of processors. The solver for the reactive transport equations scales well up to 128 processors while the solver for the flow equations is scalable up to 32 processors. The performance of the solver deteriorates for the flow equations because of its small problem size. The speedup for matrix assembly in both flow equations and reactive transport equations can reach up to 114 when using 128 processors and the parallel efficiency is 89%. The total speedup is 88 using 128 processors and the parallel efficiency is 68%. The speedup is mainly hindered by the solver due to the relatively small problem size and matrix properties. As for Case II, the output also affects the parallel performance.

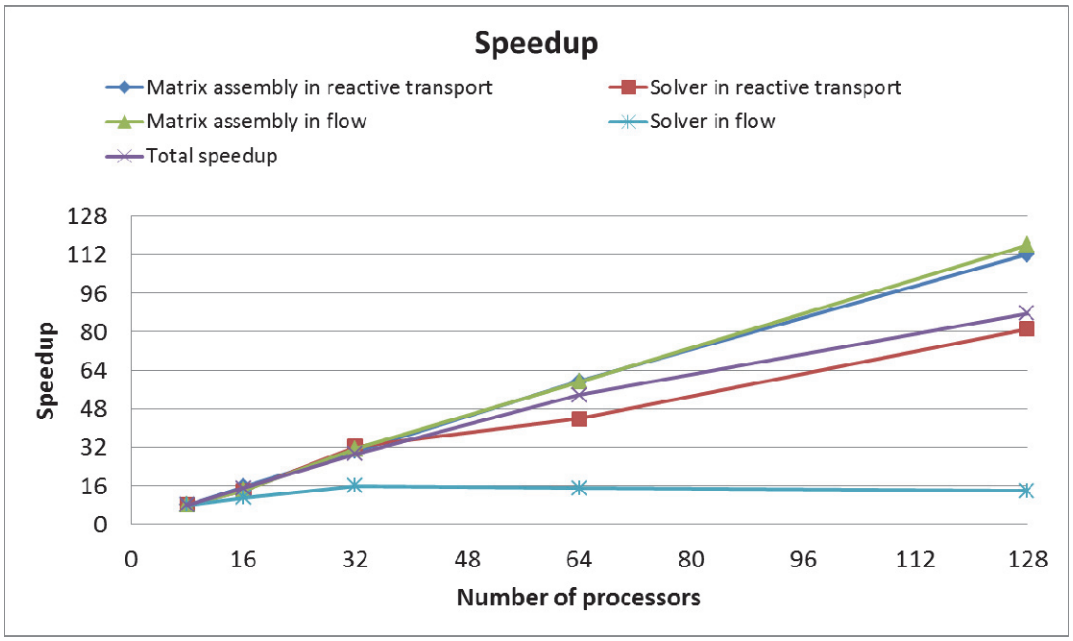

Figure 34: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Orcinus Cluster

3.5.4 Summary of Distributed-Memory Parallel Performance

The distributed-memory version is fully parallelized; however, the scalability of different parts of the code varies with an increasing number of processors. Generally, matrix assembly for both the flow equations and reactive transport equations is quite scalable with parallel efficiencies higher than 85%. The performance of the solver depends on the problem size and matrix properties. The parallel efficiency of the solver for the reactive transport equations is up to 62% using 128 processors; however, the parallel efficiency of the solver for the flow equations does not scale well when using more than 32 processors for this problem size (Case II and Case III). The total parallel efficiency is 68% using 128 processors.

The bottleneck for the distributed-memory parallel version is the scalability of the solver and the output efficiency because these two parts are not as scalable as the matrix assembly. The results suggest that the distributed-memory version is suitable for mid-size to large-size problems. Generally, given the same number of processors, better speedup can be obtained for larger size problems.

3.6 HYBRID PARALLEL PERFORMANCE

The hybrid parallel version of ParMIN3P-THCm is fully parallelized for the sections of the code using MPI but is only partially parallelized for the sections involving OpenMP (e.g., output). PETSc is used as the default parallel solver. For Case I, as it is a small case, the parallel performance is tested for up to 96 processors while for Case II and Case III, the parallel performance is tested for up to 768 processors. Code performance is compared between the hybrid parallel version and the distributed-memory version (i.e. MPI parallel version).

3.6.1 Case I: Complex Cement/Clay Interactions

3.6.1.1 Solver Statistics

The solver and runtime statistics for Case I are given in Table 15. The number of linear solver iterations and Newton iterations fluctuates with an increasing number of processors. Generally, more iterations are needed when the number of processors increases.

Table 15: Solver and Runtime Statistics for Hybrid Parallelization for Case I*

Number of Processors	1	4	8	12	24	48	96
Number of Newton iterations in flow equations	0	0	0	0	0	0	0
Number of solver iterations in flow equations	0	0	0	0	0	0	0
Number of Newton iterations in reactive transport equations	97960	95889	103581	96942	125236	139849	104001
Number of solver iterations in reactive transport equations	2186952	2171330	2260365	2206782	2422811	2592327	2175160
Number of time steps	7298	7162	7647	7233	9110	10017	7719
Total runtime (minutes)	117	38	26	20	16	11	6

* Performance testing based on simulation time of 200 years.

3.6.1.2 Parallel Speedup

For this small-scale simulation the speedup is significant when using less than 12 processors. As shown in Figure 35, the speedup of the matrix assembly continues to increase gradually as the number of processors increases. Due to the small problem size, the solver achieves almost no speedup for more than 12 processors, negatively affecting the total speedup. The total speedup is around 6 when 12 processors are used and 11 when using 48 processors.

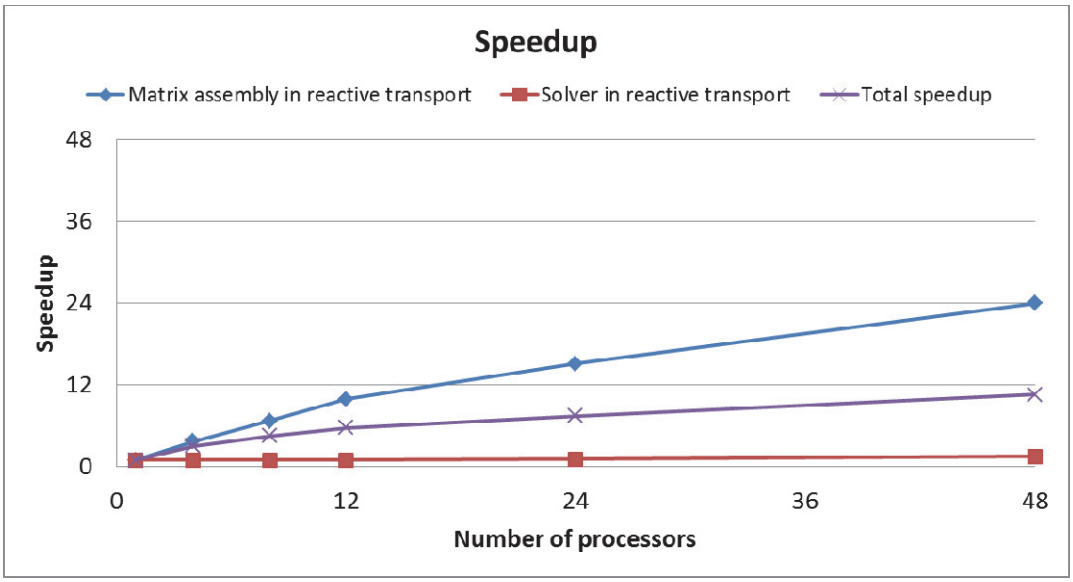

Figure 35: Speedup of Hybrid Parallel Version for Case I, Executed on the WestGrid Jasper Cluster

The hybrid parallel version performs similar to the MPI parallel version, as shown in Figure 36. However, the MPI version can only scale up to 48 processors as it is unable to effectively deal with additional domain decomposition for a larger number of processors. The speedup of the hybrid parallel version is slightly improved in comparison to the MPI parallel version, but not significantly.

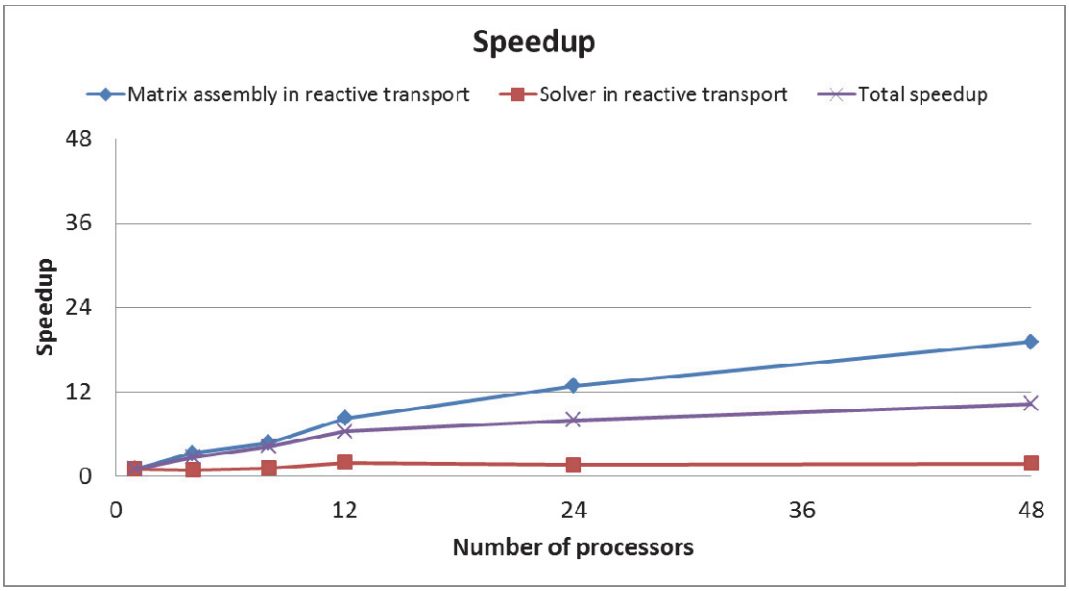

Figure 36: Speedup of MPI Parallel Version for Case I, Executed on the WestGrid Jasper Cluster

A direct comparison of total speedup between the hybrid version and the MPI version is depicted in Figure 37. The speedup of the MPI parallel version is sligthly superior to the hybrid parallel version for less than 24 processors. However, as the number of processors increases, the speedup of the hybrid parallel version becomes better than that of the pure MPI parallel version, implying that it is more scalable.

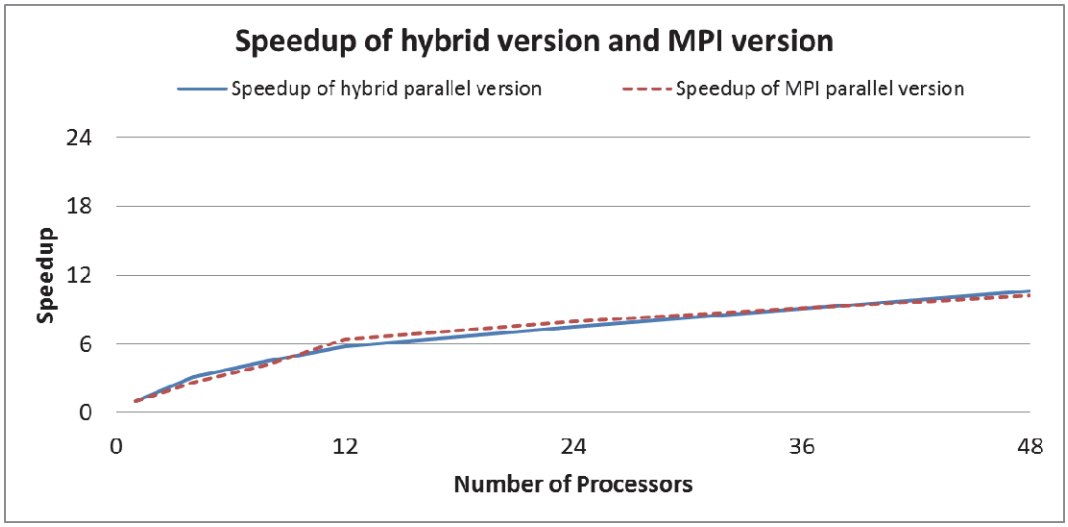

Figure 37: Comparison of Total Speedup of Hybrid Parallel Version and MPI Parallel Version for Case I, Executed on the WestGrid Jasper Cluster

3.6.2 Case II: Uranium remediation by lactate injection

3.6.2.1 Solver Statistics

The solver statistics for Case II for the hybrid parallel version are summarized in Table 16. The number of linear solver iterations and Newton iterations generally increases with an increasing number of processors.

Table 16: Solver and Runtime Statistics for Hybrid Parallelization for Case II

Number of Processors	12	24	48	96	192	384	768
Number of Newton iterations in flow equations	140	140	140	140	140	140	140
Number of solver iterations in flow equations	1347	2358	2563	2410	2547	2554	2840
Number of Newton iterations in reactive transport equations	904	904	901	901	901	905	912
Number of solver iterations in reactive transport equations	3859	3864	4413	4419	4418	4484	4363
Number of time steps	140	140	140	140	140	140	140
Total runtime (seconds)	1034	598	297	150	77	41	25

3.6.2.2 Parallel Speedup

The speedup that can be achieved is significant for this case, as shown in Figure 38. The speedup for the matrix assembly of the reactive transport problem is almost linear with an increasing number of processors, up to 768 processors. The solver for the reactive transport equations achieves a near-linear speedup for up to 384 processors. Speedups for matrix assembly and solution of the flow equations are not as substantial as for the reactive transport equations, due to the smaller size of the flow problem. The total speedup is around 500 when using 768 processors, implying a parallel efficiency of 65%.

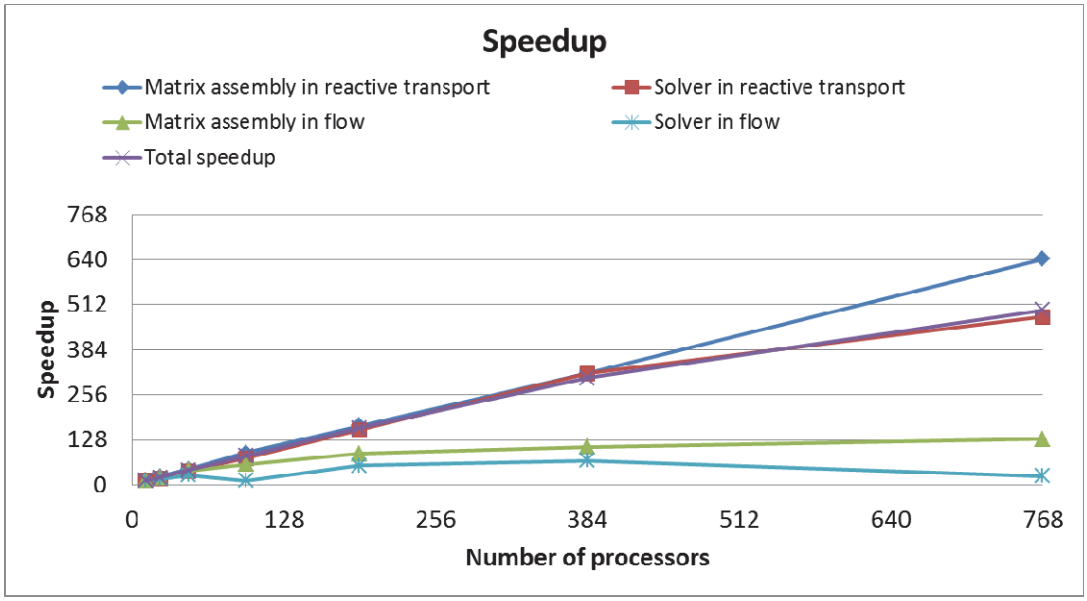

Figure 38: Speedup of Hybrid Parallel Version for Case II, Executed on the WestGrid Jasper Cluster

The hybrid parallel version shows better scalability than the MPI parallel version when the number of processors exceeds a specific threshold, as shown in Figure 39. For the hybrid parallel version, the matrix assembly is scalable for up to 768 processors for the reactive transport equations, but only up to 384 processors for the flow equations. The solver is scalable up to 192 processors for the reactive transport equations, but only for 24 proessors for the flow equations. The total speedup is around 260 when using 768 processors (Figure 39), implying a parallel efficiency of around 34%.

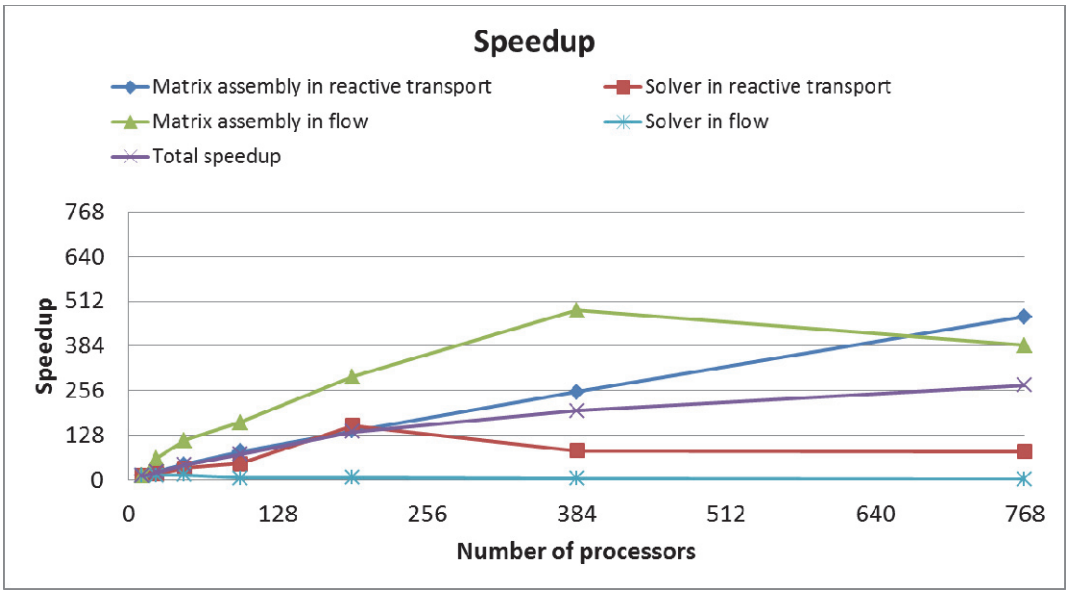

Figure 39: Speedup of MPI Parallel Version for Case II, Executed on the WestGrid Jasper Cluster

A direct comparison between the total speedups between the hybrid version and the MPI version is provided in Figure 40. The speedup of the MPI version is slightly better than that of the hybrid version for less than 48 processors, e.g., the speedup is 42 for hybrid version and 43 for MPI version. However, as the number of processors increases, speedup of the hybrid parallel version becomes substantially superior to the MPI version, implying that the hybrid version is more scalable.

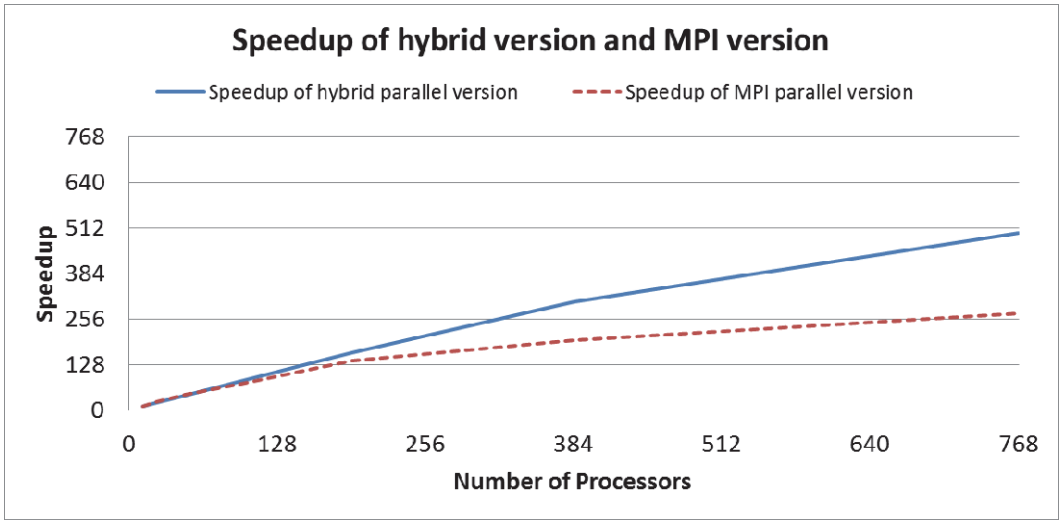

Figure 40: Comparison of Total Speedup of the Hybrid Parallel Version and the MPI Parallel Version for Case II, Executed on the WestGrid Jasper Cluster

3.6.3 Case III: Flow and reactive transport in a hypothetical sedimentary basin

3.6.3.1 Solver Statistics

The solver statistics for Case III for the hybrid parallel version are provided in Table 17. As for the previous test cases, the number of linear solver iterations and Newton iterations generally increase with an increasing number of processors.

Table 17: Solver and Runtime Statistics for Hybrid Parallelization for Case III

Number of Processors	12	24	48	96	192	384	768
Number of Newton iterations in flow equations	633	636	633	646	646	649	660
Number of solver iterations in flow equations	1717	1833	1700	1728	1721	1729	1762
Number of Newton iterations in reactive transport equations	825	825	825	825	825	826	836
Number of solver iterations in reactive transport equations	2257	2257	2257	2263	2263	2279	2296
Number of time steps	205	205	205	205	205	205	204
Total runtime (seconds)	996	549	279	152	80	46	29

3.6.3.2 Parallel Speedup

The speedup achieved is also significant for this case, as shown in Figure 41. The speedups for both matrix assembly and solution of the reactive transport equations are almost linear as the number of processors increases, up to 768 processors. The speedups for the matrix assembly and for the solution of the flow equations are less than the corresponding speedups for reactive transport equations, due to the smaller size of the flow problem. The total speedup is around 410 when using 768 processors, implying a parallel efficiency of 54%.

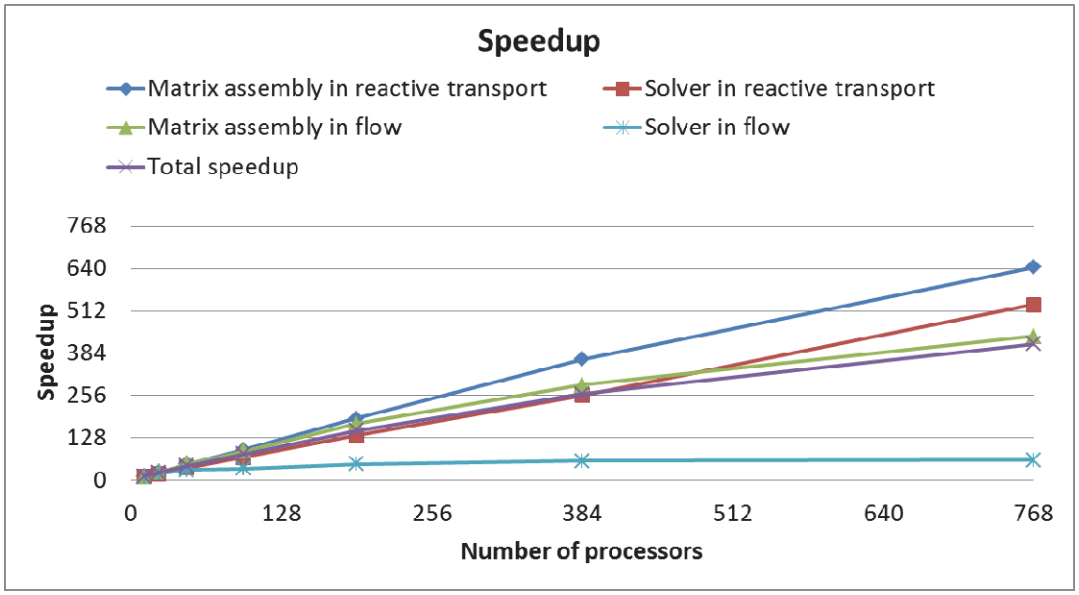

Figure 41: Speedup of Hybrid Parallel Version for Case III, Executed on WestGrid Jasper Cluster

As for Case I and Case II, the results demonstrate that the hybrid parallel version is more scalable than the MPI version, if the number of processors exceeds a specific threshold, as shown in Figure 42. The matrix assembly in the hybrid version scales well up to 768 processors for the reactive transport equations; however, for the flow equations good scaling can only be observed up to 384 processors. The solver is scalable up to 768 processors for the reactive transport equations, but only up to 24 processors for the flow equations. For the MPI version, there is no total speedup for more than 384 processors. The total speedup is around 128 using 384 or 768 processors and the parallel efficiency is around 34% or 17%, respectively.

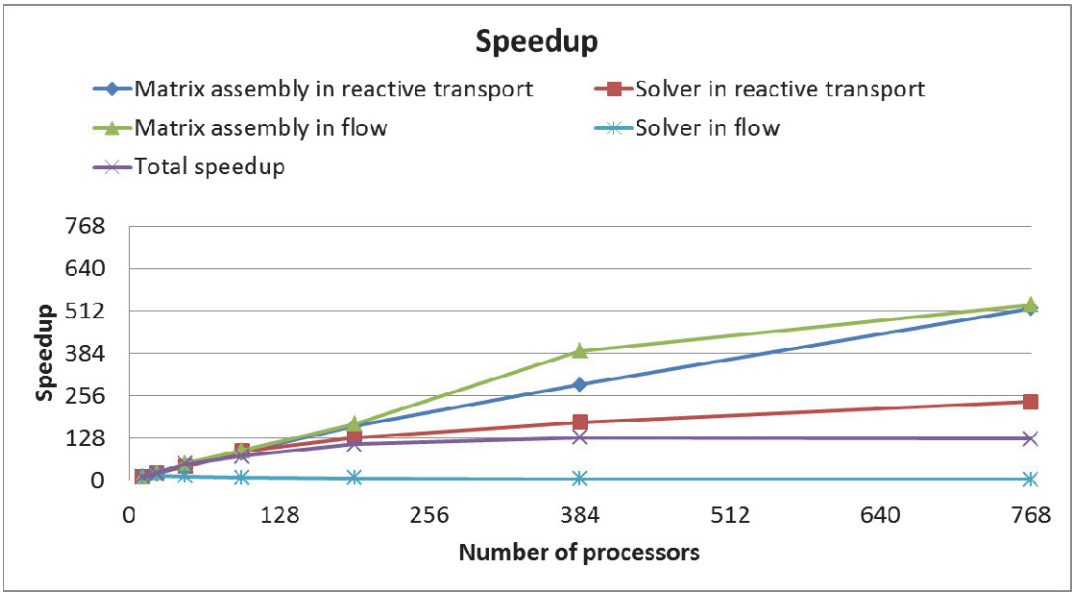

Figure 42: Speedup of the MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster

A direct comparison of total speedup between the hybrid version and the MPI version is provided in Figure 43. The speedup of the MPI version is marginally better than that of the hybrid version for less than 48 processors. However, as the number of processors increases, speedup of the hybrid version is substantially superior in relation to the MPI version, implying that it is more scalable.

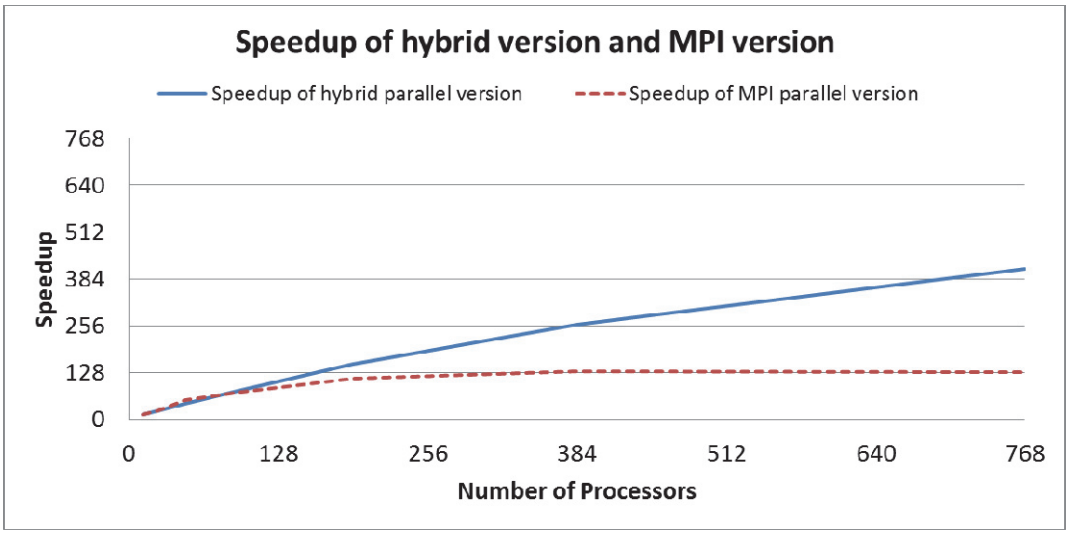

Figure 43: Comparison of Total Speedup between the Hybrid Parallel Version and the MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster

3.6.4 Summary of Hybrid Distributed-Shared-Memory Parallel Performance

In the hybrid parallel version of ParMIN3P-THCm, the MPI part is fully parallelized while the OpenMP part is only partially parallelized. The scalability of different parts of the code is quite different as the number of processors increases. Generally, the assembly and solution of the reactive transport equations are much more scalable than the corresponding tasks for the flow equations. The performance of the solver depends on the problem size and matrix properties. For the reactive transport equations, the parallel efficiency for matrix assembly and solver reach up to 83% and 70%, respectively, when using 768 processors. The total parallel efficiency is hindered somewhat by the solution of the flow problem, which is not as scalable as the reactive transport problem due to its smaller scale and lower number of unknowns. The total parallel efficiency, considering input and output, averages around 54% when using 768 processors.

The MPI parallel version is generally superior to the hybrid version when using a small number of processors (e.g., less than 96). However, as the number of processors increases, the hybrid version tends to provide better speedup. The results indicate that the MPI parallel version is most suitable for simulations utilizing a small number of processors. However, when the speedup of the MPI parallel version does not further increase for a larger number of processors, the hybrid parallel version should be used. The performance test for the hybrid version is based on a mid-size problem (Case II and Case III). The speedup should be more substantial for larger scale simulations. However, it can be difficult to secure longer runtimes and a larger number of processors on the WestGrid clusters.

3.7 SYSTEM SCALABILITY

The aforementioned parallel performance tests have shown that ParMIN3P-THCm generally performs well as the number of processors increases. However, the tests also revealed that the speedup has an upper limit for a fixed size problem. In this section, a simulation speedup analysis for different problem sizes is presented for Case III.

The selected case focused on flow and reactive transport processes in a hypothetical sedimentary basin. Four different spatial discretizations were selected for the analysis. The total degrees of freedom for these four cases are 405,000, 6,480,000, 25,920,000 and 103,680,000, respectively. Speedups for flow and reactive transport solutions, other operations (e.g., input and output), and total speedup are depicted in Figure 44 to Figure 47.

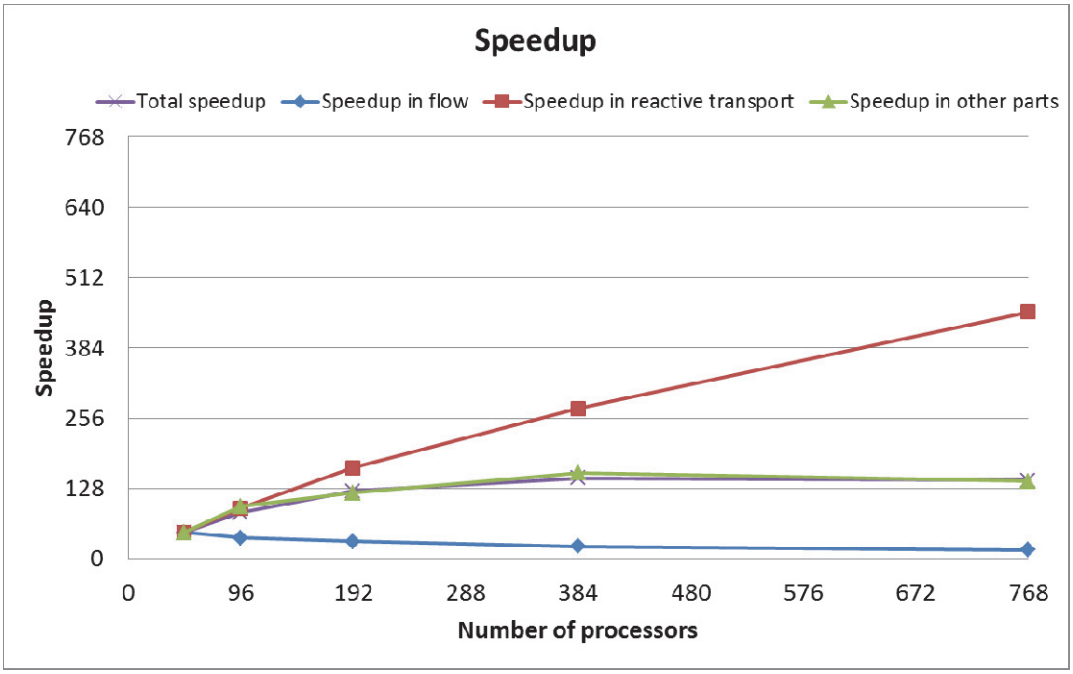

Figure 44: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster, Total Degrees of Freedom is 405,000

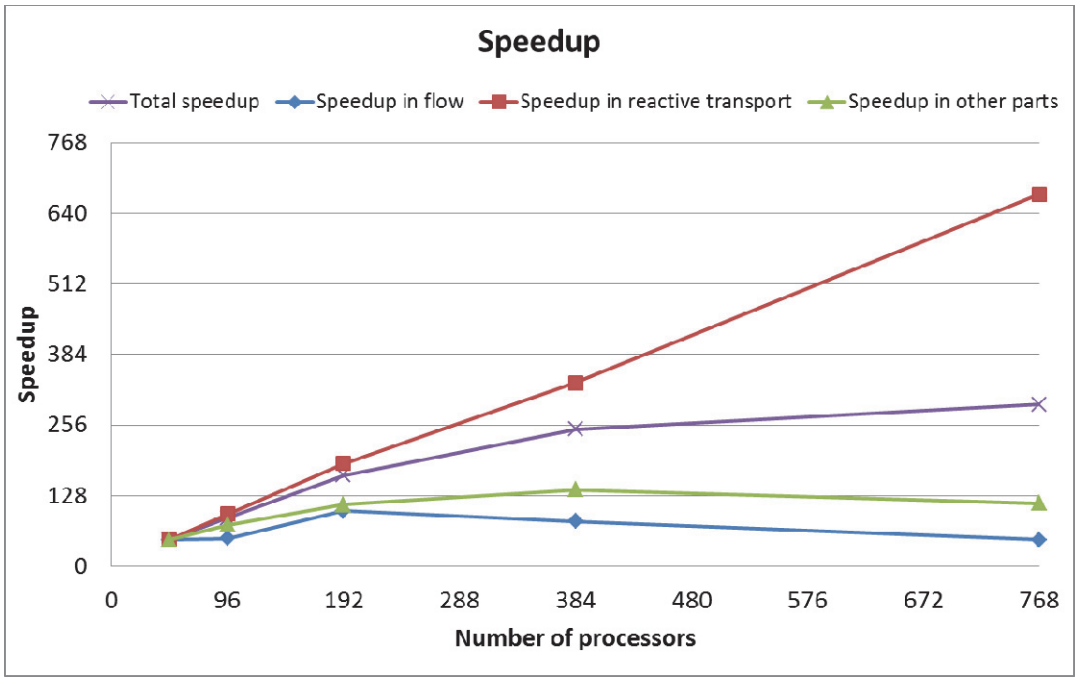

Figure 45: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster, Total Degrees of Freedom is 6,480,000

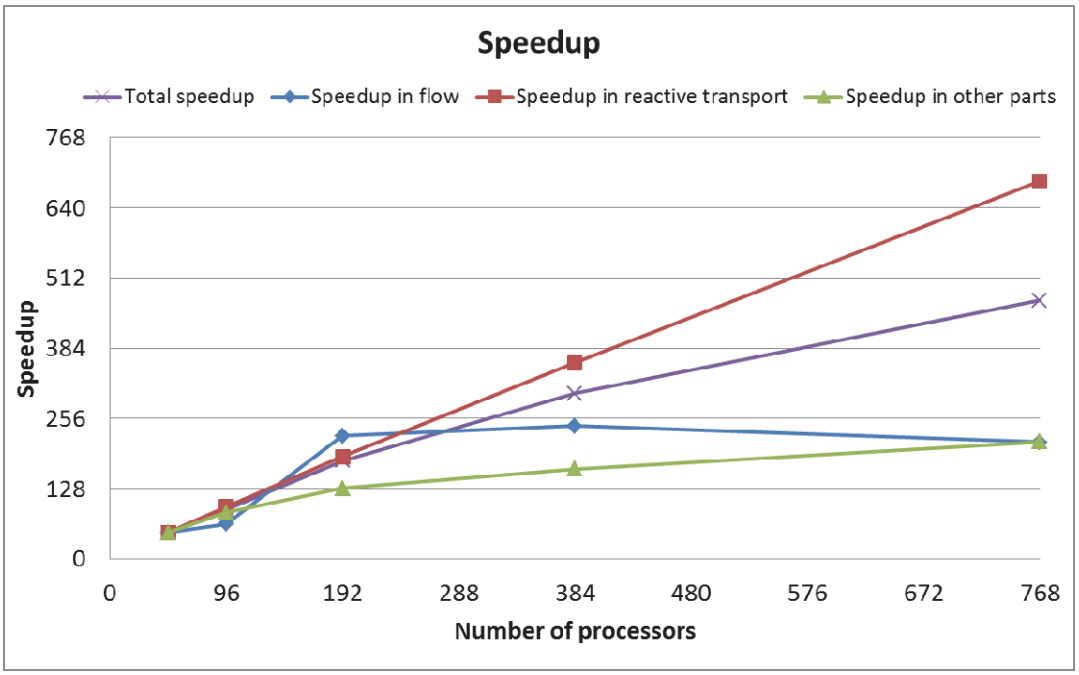

Figure 46: Speedup of MPI Parallel Version for Case III, Executed on WestGrid Jasper Cluster, Total Degrees of Freedom is 25,920,000

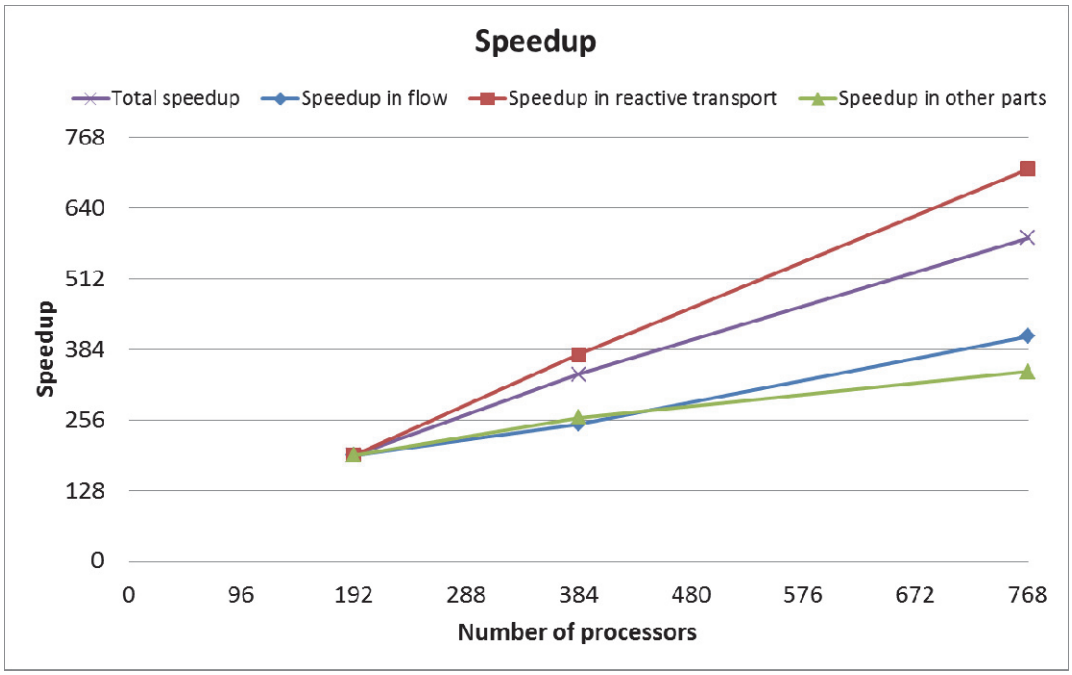

Figure 47: Speedup of MPI Parallel Version for Case III, Executed on the WestGrid Jasper Cluster, Total Degrees of Freedom is 103,680,000

An important conclusion that can be drawn from this analysis is that the total speedup tends to approach linear speedup when the degrees of freedom per processor is larger than 135,000. For the reactive transport problem, the speedup tends to be ideal when the degrees of freedom per processor is larger than 33,750. As expected from prior analysis, the speedup for solution of the flow problem is not as scalable as for the reactive transport problem because of the smaller size of the flow problem. The speedup of other operations is seriously affected by system input and output, which tends to require intensive communication. Because most of the execution time is spent on the reactive transport problem, this analysis documents very good scalability for large reactive transport simulations.

4. SUMMARY AND CONCLUSIONS

A parallel version of the reactive transport code MIN3P-THCm has been developed to facilitate simulations of large-scale long-term computationally intensive problems by using methods of high-performance computing. Parallelization of the new code, ParMIN3P-THCm, was achieved through the domain decomposition method using the PETSc toolkit. Three parallel versions, including a shared-memory version, a distributed-memory version and a hybrid distributed-shared-memory version, were developed, suitable for different kinds of problems and computer architectures.

The code has demonstrated an excellent speedup for reactive transport simulations for up to 8 processors on local shared-memory workstations, 768 processors on the WestGrid supercomputer for the distributed-memory parallel version, and 768 processors on the WestGrid

supercomputer for the hybrid parallel version. The code has shown strong scalability for modelling large scale reactive transport problems with problem sizes up to 100 million unknowns.

Detailed performance testing for the three parallel versions has been conducted. For the shared-memory version, the CPU-intensive tasks, such as matrix assembly for the reactive transport equations including computing of the local mass conservation equations, show good speedups; however, other tasks cannot achieve good speedup due to the parallel overhead. For the distributed-memory version, the code is fully parallelized, but the scalability of different parts of the code varies substantially with an increasing number of processors. Generally, the CPU-intensive tasks for both the flow and reactive transport equations scale well with parallel efficiencies higher than 85%. The performance of the solver depends on the problem size and matrix properties. The total parallel efficiency reaches up to 72% when using 128 processors. For the hybrid version, the MPI part is fully parallelized while the OpenMP part is only partially parallelized. For the reactive transport equations, the parallel efficiency for matrix assembly and solver is up to 83% and 70%, respectively when using 768 processors. The total parallel efficiency is hindered by the solution of the flow problem, which is not as scalable as the reactive transport problem due to its smaller size. The total parallel efficiency, considering input and output, is around 54% when using 768 processors.

The CPU-intensive tasks in the code are quite scalable and the performance of the solver and other parts (e.g. input and output) depends on various aspects of the problem (e.g., problem size, matrix properties, and output). The major bottleneck for the parallel version is the scalability of the flow problem, which is usually not as scalable as the reactive transport part for an increasing number of processors. The parallel efficiency of the solver and data input/output are also important factors that affect the performance.

The total speedup tends to be ideal and near-linear when the degrees of freedom per processor is larger than 135,000. For the reactive transport solution, the speedup tends to be ideal with near-linear performance when the degrees of freedom per processor is larger than 33,750. The speedup for the flow problem and other operations (e.g. input and output) are not as scalable as the reactive transport problem, because of the smaller size of the flow problem and communication requirements for system input and output. However, since most of the execution time is spent in reactive transport, ParMIN3P-THCm shows a strong scalability for large-scale reactive transport simulations.

The results suggest that it is most efficient to use the shared-memory version for cases that do not require much solver time, as well as system input and output. In addition, the shared-memory version is restricted to small and mid-size problems, because increased bandwidth provides a bottleneck for shared-memory machines.

The MPI parallel version is generally superior in comparison to the hybrid parallel version for a relatively small number of processors (e.g., less than 96). As the number of processors increases, the hybrid version generally provides better speedup for the test cases considered. The results suggest that the MPI parallel version should be used for solution of problems with a small number of processors. When the speedup of the MPI parallel version no longer increases with the number of processors, the hybrid parallel version should be used instead. To date, the performance testing for the hybrid version was based on mid-size problems. It is expected that better speedup and higher parallel efficiency can be obtained for larger scale simulations. To date, model testing with more than 1000 processors was not possible because of a lack of computational resources.

The performance analysis of the new code has shown that a speedup greater than 100 is easily achievable with ParMIN3P-THCm. As a result, it is now possible to carry out 2D reactive transport simulations that require weeks of CPU time (on a single processor machine) within hours. The new code will also allow refining model discretization in both space and time and will facilitate 3D simulations that were impractical to carry out with the sequential version of MIN3P-THCm.

ACKNOWLEDGEMENTS

This research is supported by NWMO under project GS60. The authors would like to acknowledge the contributions of the PETSc development team, especially Barry Smith, Jed Brown, and Satish Balay. Without their help, ParMIN3P-THCm would not have achieved its high scalability in such a short development time. The authors would also like to acknowledge WestGrid (www.westgrid.ca) and Compute Canada (www.computecanada.ca) for providing computing hardware, software and technical support. The development team gives many thanks to Roman Baranowski and Brent Gawryluik for their patient technical support on WestGrid Orcinus supercomputer, and Masao Fujinaga for his support on WestGrid Jasper supercomputer. The authors are also indebted to the Argonne National Lab (www.anl.gov) for providing a scholarship to Danyang Su to attend the "Argonne Training Program on Extreme-Scale Computing (ATPESC) 2014", which provided intensive hands-on training on the key skills, approaches, and tools to design, implement and execute applications on current high-end computing systems.

REFERENCES

Adams, M., P. Colella, D.T. Graves, J.N. Johnson, N.D. Keen, T.J. Ligocki, D.F. Martin, P.W. McCorquodale, D. Modiano, P.O. Schwartz, T.D. Sternberg and B. Van Straalen. 2014. Chombo Software Package for AMR Applications - Design Document, Lawrence Berkeley National Laboratory Technical Report LBNL-6616E. https://commons.lbl.gov/display/chombo/Chombo+-+Software+for+Adaptive+Solutions+of+Partial+Differential+Equations

Balay, S., S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, K. Rupp, B.F. Smith and H. Zhang. 2014a. PETSc Web page. http://www.mcs.anl.gov/petsc.

Balay, S., S. Abhyankar, M.F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, K. Rupp and B.F. Smith and H. Zhang. 2014b. PETSc Users Manual. http://www.mcs.anl.gov/petsc.

Balay, S., W.D. Gropp, L.C. McInnes and B.F. Smith. 1997. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries, in Modern Software Tools in Scientific Computing (editors Arge, E., A. M. Bruaset and H. P. Langtangen), 163-202, Birkhauser Press.

Bea Jofre, S. A, K.U. Mayer and K.T.B. MacQuarrie. 2011. Modelling Reactive Transport in Sedimentary Rock Environments – Phase II. MIN3P Code Enhancements and Illustrative Simulations for a Glaciation Scenario. Nuclear Waste Management Organization Technical Report NWMO TR-2011-13.

Beisman, J.J., R.M. Maxwell, A.K. Navarre-Sitchler, C.I. Steefel and S. Molins. 2015. ParCrunchFlow: an efficient, parallel reactive transport simulation tool for physically and chemically heterogeneous saturated subsurface environments. Computational Geosciences, 19(2), 403-422, doi: 10.1007/s10596-015-9475-x.

Charlton, S.R., and D.L. Parkhurst. 2011. Modules based on the geochemical model PHREEQC for use in scripting and programming languages. Computers & Geosciences, 37, 1653-1663.

Clement, T. P. and C. D. Johnson. 2012. RT3D: Reactive Transport in 3-Dimensions. Groundwater Reactive Transport Models, 96-111 (16), doi: 10.2174/978160805306311201010096.

Gebali, F. 2011. Algorithms and Parallel Computing, Wiley Series on Parallel and Distributed Computing, Albert Zomaya, Series Editor.

Hammond, G. E., P. C. Lichtner, C. Lu and R.T. Mills. 2012. PFLOTRAN: Reactive Flow & Transport Code for Use on Laptops to Leadership-Class Supercomputers. Groundwater Reactive Transport Models, 141-159(19), doi: 10.2174/978160805306311201010141.

Hao, Y., Y. Sun and J.J. Nitao. 2012. Overview of NUFT: A Versatile Numerical Model for Simulating Flow and Reactive Transport in Porous Media. Groundwater Reactive Transport Models, 212-239 (28), doi: 10.2174/978160805306311201010212.

Kolditz, O., S. Bauer, L. Bilke, N. Bottcher, J. Delfs, T. Fischer, U. Gorke, T. Kalbacher, G. Kosakowski, C. McDermott, C. Park, F. Radu, K. Rink, H. Shao, H. Shao, F. Sun, Y. Sun, A. Singh, J. Taron, M. Walther, W. Wang, N. Watanabe, Y. Wu, M. Xie, W. Xu and B. Zehner. 2012. OpenGeoSys: an open-source initiative for numerical simulation of thermo-hydro-mechanical/chemical (THM/C) processes in porous media Environmental Earth Sciences, Springer-Verlag, 67, 589-599

Kollet, S. J. and R.M. Maxwell. 2006. Integrated surface-groundwater flow modelling: A free-surface overland flow boundary condition in a parallel groundwater flow model, Advances in Water Resources, (29)7, 945-958.

Lagneau, V. and J.V.D. Lee. 2010. HYTEC results of the MoMas reactive transport benchmark. Computational Geosciences, Springer Verlag (Germany), 14, 435-449. https://hal-mines-paristech.archives-ouvertes.fr/hal-00505360.

Marty, N. C.M., O. Bildstein, P. Blanc, F. Claret, B. Cochepin, E.C. Gaucher, D. Jacques, J.E. Lartigue, S. Liu, K.U. Mayer, J.C.L. Meeussen, I. Munier, I. Pointeau, D. Su and C.I. Steefel. 2015. Benchmarks for multicomponent reactive transport across a cement/clay interface, Computational Geosciences, 19, 635-653.

Mayer, K.U., E.O. Frind and D.W. Blowes. 2002. Multicomponent reactive transport modelling in variably saturated porous media using a generalized formulation for kinetically controlled reactions. Water Resources Research, 38, 1174, doi: 10:1029/2001WR000862.

Mayer, K.U., and K.T.B. MacQuarrie. 2010. Solution of the MoMaS reactive transport benchmark with MIN3P - Model formulation and simulation results, Computational Geosciences, 14, 405-419, doi:10.1007/s10596-009-9158-6.

Mayer, K.U., M. Xie, D. Su and K.T.B. MacQuarrie. 2014. MIN3P-NWMO: A Three-dimensional Numerical Model for Multicomponent Reactive Transport in Variably Saturated Porous Media.

McIntosh, J. and L. Walter. 2005. Volumetrically significant recharge of Pleistocene glacial meltwaters into epicratonic basins: Constraints imposed by solute mass balances. Chemical Geology, 222, 292-309.

Meeussen, J.C. 2003. ORCHESTRA: An object-oriented framework for implementing chemical equilibrium models. Environmental Science & Technology, 37, 1175–1182.

Nieplocha, J., B. Palmer, V. Tipparaju, M. Krishnan, H. Trease and E. Apra. 2006. Advances, Applications and Performance of the Global Arrays Shared Memory Programming Toolkit, International Journal of High Performance Computing Applications, 20(2), 203-231.

Parkhurst, D.L. and C.A.J. Appelo. 2013. Description of input and examples for PHREEQC version 3—a computer program for speciation, batch-reaction, one-dimensional transport, and inverse geochemical calculations, U.S. Geological Survey Techniques and Methods, book 6, chap. A43,497 p., available only at http://pubs.usgs.gov/tm/06/a43

Prommer, H. and V.E.A. Post. 2010. PHT3D, A Reactive multicomponent transport model for saturated porous media. User's Manual v2.10 (2010). http://www.pht3d.org.

Samper, J., C. Yang, L. Zheng, L. Montenegro, T. Xu, Z. Dai, G. Zhang, C. Lu and S. Moreira. 2012. CORE2D V4: A Code for Water Flow, Heat and Solute Transport, Geochemical Reactions, and Microbial Processes. Groundwater Reactive Transport Models, 160-185 (26), doi: 10.2174/978160805306311201010160.

Schenk, O. and K. Gärtner. 2011. PARDISO User Guide Version 4.1.2. http://www.pardiso-project.org/index.php?p=manual.

Şengör, S. S., K. U. Mayer, J. Greskowiak, C. Wanner, D. Su and H. Prommer. 2015. A reactive transport benchmark on modelling biogenic uraninite re-oxidation by Fe(III)-(hydr)oxides, Computers & Geosciences, 19, 569-583.

Simunek, J., D. Jacques, M. Sejna and M.T. van Genuchten. 2012, The HP2 Program for HYDRUS (2D/3D): A coupled code for simulating two-dimensional variably-saturated water flow, heat transport, and biogeochemistry in porous media, Version 1.0, PC Progress, Prague, Czech Republic, 76.

Steefel, C.I. 2009. CrunchFlow: Software for modelling multicomponent reactive flow and transport user's manual. Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 USA.

Toselli, A. and O.B. Widlund. 2005. Domain Decomposition Methods – Algorithms and Theory, pp 4-5, doi: 10.1007/b137868, Springer Berlin Heidelberg.

van der Lee, J., L. De Windt, V. Lagneau and P. Goblet. 2003. Module-oriented modelling of reactive transport with HYTEC. Computers & Geosciences, 29, 265–275.

van Leer, B. 1977. Towards the Ultimate Conservative Difference Scheme IV. A New Approach to Numerical Convection, Journal of Computational Physics, 23, 276-299.

Wei, X., W. Li, H. Tian, H. Li, H. Xu and T. Xu. 2015. THC-MP: High performance numerical simulation of reactive transport and multiphase flow in porous media, Computers & Geosciences, 80, 26-37.

Wheeler, M. F., S. Sun and S.G. Thomas. 2012. Modelling of Flow and Reactive Transport in IPARS. . Groundwater Reactive Transport Models. 42-73(32), doi: 10.2174/978160805306311201010042.

White, M.D. and M. Oostrom. 2006. STOMP subsurface transport over multiple phases version 4.0 user's guide. Pacific Northwest National Laboratory, Washington.

Xu, T., E. Sonnenthal, N. Spycher, G. Zhang, L. Zheng and K. Pruess. 2012. TOUGHREACT: A Simulation Program for Subsurface Reactive Chemical Transport under Non-isothermal Multiphase Flow Conditions. Groundwater Reactive Transport Models, 74-95 (22), doi: 10.2174/978160805306311201010074.

Yeh, G.T. and V.S. Tripathi. 1990. HYDROGEOCHEM: A coupled model of HYDROlogical transport and GEOCHEMical equilibrium of multi component systems, ORNL 6371, Oak Ridge National Laboratory, p. 37831. Oak Ridge National Laboratory, Oak Ridge.

Yeh, G. T., V.S. Tripathi, J.P. Gwo, H.P. Cheng, J. –R.C. Cheng, K.M. Salvage, M.H. Li, Y. Fang, Y. Li, J. T. Sun, F. Zhang and M.D. Siegel. 2012. HYDROGEOCHEM: A Coupled Model of Variably Saturated Flow, Thermal Transport, and Reactive Biogeochemical Transport. Groundwater Reactive Transport Models, 3-41(36), doi: 10.2174/978160805306311201010003.