TECHNIQUES OF HIGH PERFORMANCE RESERVOIR SIMULATION FOR UNCONVENTIONAL CHALLENGES

A Dissertation

by

YUHE WANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	John Killough
Committee Members,	Akhil Datta-Gupta
	Michael King
	Yalchin Efendiev
Head of Department,	Dan Hill

December 2013

Major Subject: Petroleum Engineering

ABSTRACT

The quest to improve the performance of reservoir simulators has been evolving with the newly encountered challenges of modeling more complex recovery mechanisms and related phenomena. Reservoir subsidence, fracturing and fault reactivation etc. require coupled flow and poroelastic simulation. These features, in turn, bring a heavy burden on linear solvers. The booming unconventional plays such as shale/tight oil in North America demand reservoir simulation techniques to handle more physics (or more hypotheses). This dissertation deals with three aspects in improving the performance of reservoir simulation toward these unconventional challenges.

Compositional simulation is often required for many reservoir studies with complex recovery mechanisms such as gas inject. But, it is time consuming and its parallelization often suffers sever load imbalance problems. In the first section, a novel approach based on domain over-decomposition is investigated and implemented to improve the parallel performance of compositional simulation. For a realistic reservoir case, it is shown the speedup is improved from 29.27 to 62.38 on 64 processors using this technique.

Another critical part that determines the performance of a reservoir simulator is the linear solver. In the second section, a new type of linear solver based the combinatorial multilevel method (CML) is introduced and investigated for several reservoir simulation applications. The results show CML has better scalability and performance empirically and is well-suited for coupled poroelastic problems. These

results also suggest that CML might be a promising way of precondition for flow simulation with and without coupled poroelastic calculations.

In order to handle unconventional petroleum fluid properties for tight oil, the third section incorporates a simulator with extended vapor-liquid equilibrium calculations to consider the capillarity effect caused by the dynamic nanopore properties. The enhanced simulator can correctly capture the pressure dependent impact of the nanopore on rock and fluid properties. It is shown inclusion of these enhanced physics in simulation will lead to significant improvements in field operation decision-making and greatly enhance the reliability of recovery predictions.

DEDICATION

To my family

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the guidance and help from several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study. In this humble acknowledgement, I would like to express my gratitude and appreciation to all of them.

First and foremost, I would like to convey my utmost gratitude to my advisor, Dr. Killough for this sincerity, inspiration and encouragement that I will remember forever. Being this student is one of my most fortunate things I have ever had in my life. What I have learned from Dr. Killough helps shape my career and will continuously put positive influence on my future.

I am heartily grateful to Dr. Eduardo Gildin, who was my advisor in the early stage of my PhD study. It was him who introduced me to the area of reservoir simulation.

I am thankful to Dr. King, Dr. Datta-Gupta and Dr. Efendiev for serving as my committee and valuable comments and suggestions that helped shaped the dissertation.

I would like to thank Landmark/Halliburton for the internship opportunities, especially for my mentors, Qinghua Wang and Graham Fleming.

I would like to thank my colleagues of Killough group and all my dear friends. It is you guys who made my life in College Station enjoyable.

Finally, my special thank goes to Sammi and Vanguard Orient (Beijing) Technology Company for the support and encouragement during my four years' stay at Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION AND STUDY SCOPES

1.1 Introduction

Reservoir simulation is a technique to mimic or infers the behavior of fluid flow, such as oil, gas and water, in a petroleum reservoir through the use of mathematical model numerically. It has been a proven technology to be routinely used in petroleum asset management. Reservoir simulator was born as an efficient tool for reservoir engineers to better understand and mange assets. However, like any numerical simulation approach, reservoir simulation is inherently complex and computational intensive and easily becomes inefficient if more grids, complex recovery mechanisms, and/or complex geometry are necessary to accurately describe the complex phenomena occurring in the subsurface. For example, shown in **Figure 1.1** is a multi-reservoir compositional model with about 10 million cells. The simulation of this model could take considerable amount of time, which can prolong the project time significantly. The quest to improve the performance of reservoir simulators has been evolving with the newly encountered challenges of modeling more complex recovery mechanisms and related phenomena. Reservoir subsidence, fracturing and fault reactivation etc. require coupled flow and poroelastic simulation. Shown in **Figure 1.2** is an example of reservoir model with coupled flow and poroelastics. These features, in turn, bring a heavy burden on linear solvers. The booming of unconventional plays such as shale/tight oil has greatly changed the energy outlook of North America. Shown in **Figure 1.3** is the 2013

EIA forecast of US domestic crude oil production by source. To economically develop such unconventional plays require technology advances. On of such need is to the demand of reservoir simulation techniques to handle more physics (or more hypotheses).

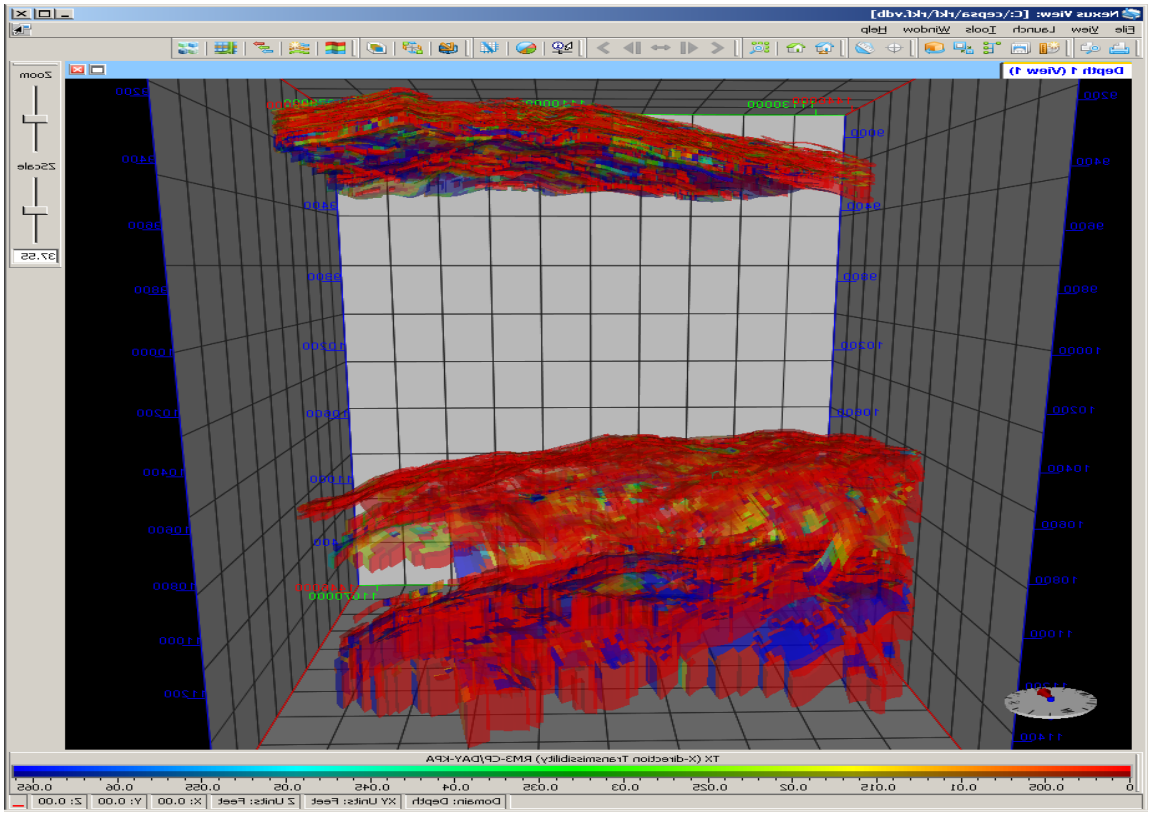

Figure 1.1 10 million cell multi-reservoir compositional model

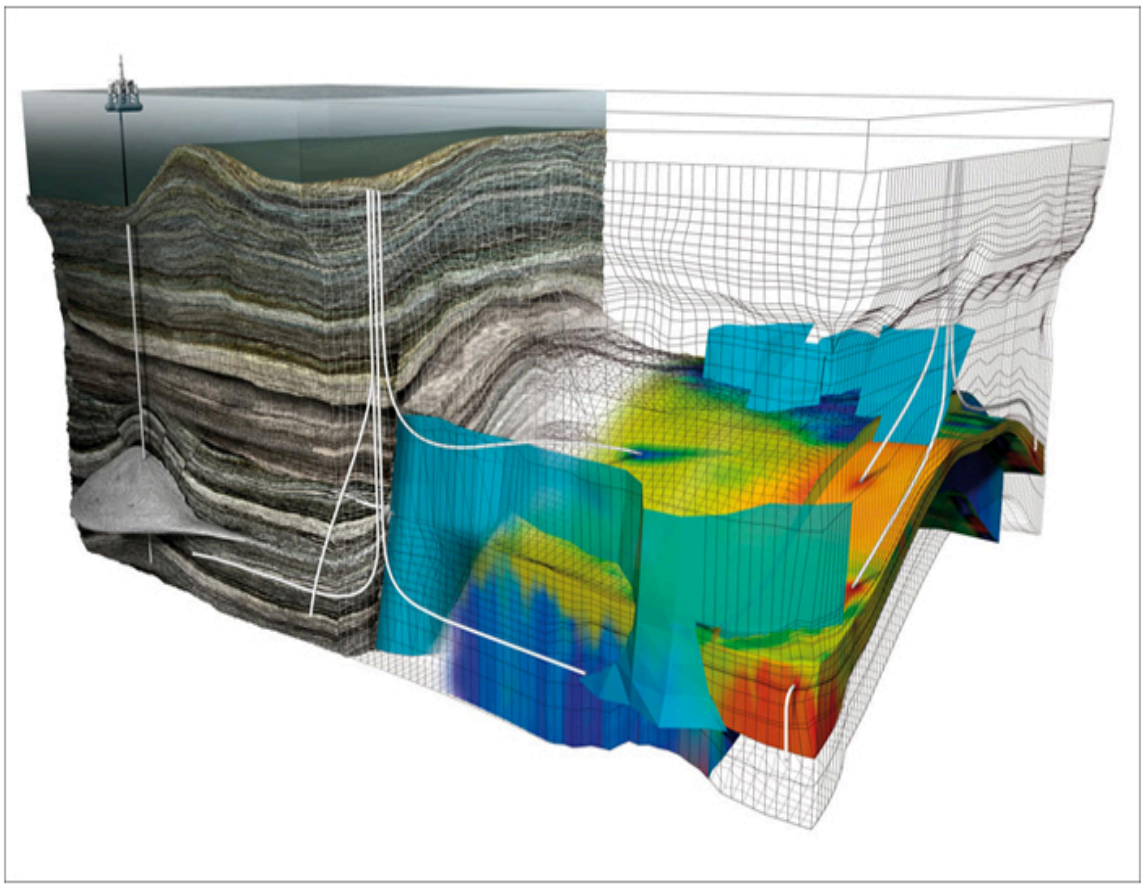

Figure 1.2 A reservoir model with couple flow and poroelastices (Schlumberger)

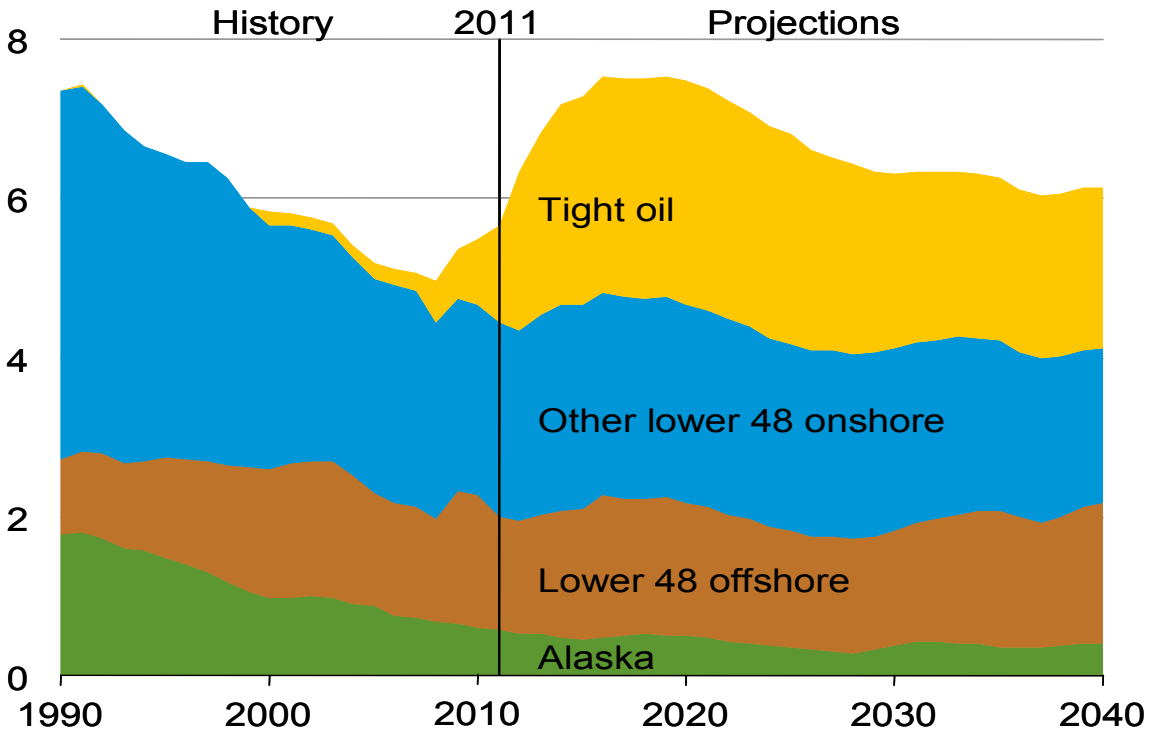

Figure 1.3 US domestic crude oil production by source, 1990-2040 (MMbbl/day) (EIA 2013)

1.2 Study Scopes and Outlines

This dissertation deals with three aspects in efforts to improve the reservoir simulation technique toward these challenges. The three aspects are: 1) load balancing of parallel compositional simulation; 2) solver preconditioning using combinatorial multilevel method; and 3) compositional modeling of tight oil considering dynamic nanopore properties.

Compositional simulation is often required for many reservoir studies. As mentioned above, it is time consuming and the cost grows dramatically with an increase in the number of components. Because of this, reservoir studies requiring compositional simulation often become a bottleneck in the engineering process. With the advances in

high performance computing, execution of compositional simulation in parallel seems to be the apparently feasible way to tackle its computational demand. Although running reservoir simulation in parallel sounds extremely attractive, developing an efficient parallel reservoir simulator is far more challenging than developing the underlying serial reservoir simulator. For decades there have remained many open problems associated with high performance computing and reservoir simulation.

Among the various challenges of efficiency and scalability of parallel compositional simulation, load imbalance is a major obstacle that has not been fully addressed and solved. In Chapter II, a novel approach is investigated and implemented to improve the performance of parallel compositional simulation which often suffers severe dynamic load imbalance problems. This new approach is based on domain over-decomposition. It over-decomposes the reservoir model to assign each processor a bundle of subdomains. Processors treat these bundles of subdomains as virtual processes that can be dynamically migrated across processors in the run-time system. This technique is shown to be capable of achieving better overlap between computation and communication and cache efficiency. For a realist simulation problem, it is shown that domain over-decomposition together with a load balancer can improve speedup from 29.27 to 62.38 on 64 physical processors.

The linear solver is another very critical component that determine the robustness and efficiency of a reservoir simulator. For large-scale black-oil simulation, the solution of the resulting linear system usually consumes up to 90% of the total execution time. For problem with coupled reservoir displacement or poroelastics, the burden on linear

solver is even higher. In Chapter III, a new type of linear solver based the combinatorial multilevel method (CML) is introduced and investigated for several reservoir simulation applications. CML is applied as the preconditioner in IMPES, fully implicit and sequential implicit formulations for flow simulation, and also applied in coupled flow and poroelastic simulation for both pressure and poroelastic preconditioning. CML is compared with commonly used ILU(0) and two Algebraic MultiGrid (AMG) preconditioners, namely Ruge Stüben AMG and AggreGation based MultiGrid (AGMG). The results show CML has better scalability and performance empirically and is well-suited for coupled poroelastic problems. These results also suggest that CML might be a promising way of preconditioning for flow simulation with and without coupled poroelastic calculations.

The recent advances in massive hydraulic fracturing techniques have enabled the oil industry to economically extract hydrocarbon from ultra-tight, unconventional resources, such as shale gas, liquid rich shale and tight oil. In spite of the great commercial success, there still remain many open questions encountered in field practices, such as the abnormal production behavior observed (long lasting low GOR even the pressure around the wellbore is believed to be below bubble point pressure) in Bakken oil. One probably hypothesis of this abnormal behavior is the fluid properties in the confined nanopore space deviate from the corresponding bulk measurements in which zero vapor-liquid interface curvature is assumed. A typical shale/tight oil reservoir such as the Bakken has a matrix pore size at the nanoscale. At such small scales the confined hydrocarbon phase behavior deviates from bulk measurements due to

the effect of capillary pressure. In addition, compaction of the pore space can bring about order of magnitude changes for tight oil formation properties during pressure depletion further exacerbating these deviations. Without considering these facts, a conventional reservoir simulator will likely not be able to explain the inconsistent produced GOR observed in the field compared to simulated results. The effect of these inaccuracies on ultimate recovery estimation can be devastating to the underlying economics. In Chapter IV, an improved reservoir simulator is developed which can rigorously model the dynamic confinement effect, such as suppression of bubble point pressure, increase of formation volume factor, reduction of oil viscosity and enhancement of critical gas saturation as well as their interaction with pore space compaction. The enhanced simulator can correctly capture the pressure-dependent impact of the nanopore structure on rock and fluid properties. As a result, the problem of inconsistent GOR is resolved and the history matching process is greatly facilitated. It is shown that inclusion of these enhanced physics in the simulation will lead to significant improvements in field operation decision-making and greatly enhance the reliability of recovery predictions.

CHAPTER II

LOAD BALANCING OF PARALLEL COMPOSITIONAL SIMULATION USING RESERVOIR MODEL OVER-DECOMPOSITION METHOD*

The quest for efficient and scalable parallel reservoir simulators has been evolving with the advancement of high performance computing architectures. Among the various challenges of efficiency and scalability, load imbalance is a major obstacle that has not been fully addressed and solved. The reasons that cause load imbalance in parallel reservoir simulation are both static and dynamic. Robust graph partitioning algorithms are capable of handling static load imbalance by decomposing the underlying reservoir geometry to distribute a roughly equal load to each processor. However, these loads determined by a static load balancer seldom remain unchanged as the simulation proceeds in time. This so-called dynamic imbalance can be further exacerbated in parallel compositional simulations. The flash calculations for equations of state in complex compositional simulations not only can consume over half of the total execution time but also are difficult to balance merely by a static load balancer. The computational cost of flash calculations in each grid block heavily depends on the dynamic data such as pressure, temperature, and hydrocarbon composition. Thus, any static assignment of grid blocks may lead to dynamic load imbalance in unpredictable

---

manners. A dynamic load balancer can often provide solutions for this difficulty. However, traditional techniques are inflexible and tedious to implement in legacy reservoir simulators. In this paper, we present a new approach to address dynamic load imbalance in parallel compositional simulation. It over-decomposes the reservoir model to assign each processor a bundle of subdomains. Processors treat these bundles of subdomains as virtual processes or user-level migratable threads which can be dynamically migrated across processors in the run-time system. This technique is shown to be capable of achieving better overlap between computation and communication for cache efficiency. We employ this approach in a legacy reservoir simulator and demonstrate reduction in the execution time of parallel compositional simulations while requiring minimal changes to the source code. Finally, it is shown that domain over-decomposition together with a load balancer can improve speedup from 29.27 to 62.38 on 64 physical processors for a realistic simulation problem.

2.1 Introduction

High performance computing including parallel computing plays a vital role in many areas of engineering, such as defense, energy and financial engineering. Nowadays, these devices are being widely used by domain application engineers and scientists to solve a variety of commercially and scientifically interesting computationally intensive problems. Many of the techniques utilized are associated with solving the discretized partial differential equations that describe the underlying physics. Reservoir simulation, which mimics or infers the behavior of fluid flow in a petroleum reservoir system through the use of mathematical models, is one of the methods that are

widely used in petroleum upstream development and production. Reservoir simulator was born as an efficient tool for reservoir engineers to better understand and manage assets. However, like any numerical simulation tool, reservoir simulation is inherently computationally intensive and easily becomes inefficient if larger and larger grids or more components are necessary to describe accurately the complex phenomena occurring in the subsurface of the earth. Therefore, execution of reservoir simulation on parallel computers seems be the apparently feasible way to tackle the computational demand of reservoir simulation. Although running reservoir simulation in parallel sounds extremely attractive, developing an efficient parallel reservoir simulator is far more challenging than developing the underlying serial reservoir simulator. For decades there have remained many open problems associated with high performance computing and reservoir simulation.

The quest for efficient and scalable parallel reservoir simulators has been evolving with the advancement of high performance computing architectures. The mainframes of decades ago quickly gave way to workstations, clusters, and finally PCs as technology advanced and costs were dramatically reduced. Each of these evolutionary steps led to significant changes in reservoir simulators. The era of serial reservoir simulation was replaced by vectorized and finally parallelized simulation. An elusive goal of reservoir simulation has been the ability to efficiently utilize massively parallel processing.

In the past the majority of effort has been spent on developing robust parallel linear solvers (Killough and Wheeler 1987; Cao et al. 2005; Fung and Dogru 2007). As

Graphic Processing Units (GPUs) have become more and more popular in the oil industry (Foltinek et al. 2009; Appleyard et al. 2011; Klie et al. 2011; Liu et al. 2012; Bayat and Killough, 2013), it is expected that the reservoir simulation community will soon have a GPU accelerated linear solver commercially implemented for reservoir simulation.

Massively parallel linear solver development has by no means been completed; however, little effort has been spent on investigating another important aspect of high performance simulation - load balancing. Load imbalance has become a major obstacle for parallel performance. The reasons that cause load imbalance in parallel reservoir simulation are both static and dynamic. Robust graph partitioning algorithms are capable of handling static load imbalance by decomposing the underlying reservoir geometry to distribute roughly equal load to each processor. This approach works well to distribute the load for parallel linear solver. Metis from the University of Minnesota is one of most popular tool for graph partitioning (Karypis and Kumar 1999) and has been applied in parallel linear solver packages and parallel reservoir simulation both commercially and academically (Shuttleworth et al. 2009; Zhang et al. 2001). In fully implicit black oil simulation, where the linear solver can often take over 90% of the total execution time, graph partitioning often can cure the load imbalance problem. However, this is not the case for compositional reservoir simulation. In fully compositional simulation using IMPES formulation, the time spent on equation of state computations can be more than 70% of the total computational time. It is noted that in fully implicit compositional simulation, the linear solver would dominate the computation. However, IMPES

formulation is still often used for large-scale fully compositional model. Thus, in this paper, we consider the fully compositional simulation using IMPES formulation. Although the underlying grid is divided equally (roughly) to each processor, the loads determined by a static load balancer such as Metis seldom remain unchanged for the equation of state computations. The changing of load as simulation running - the so-called dynamic load imbalance - is difficult to balance merely by a static load balancer. The reason is that each of the underlying grid blocks has an independent phase behavior calculation. The computational cost of the associated flash calculations in each grid block heavily depends on the dynamic data such as pressure, temperature, and fluid composition. It is well known that flash calculations can vary tremendously with time in each grid block. The cost can become very high when phase changes happen in a grid block or when the fluid mixture is near the critical point. Since the conditions that cause expensive flash calculations are difficult to predict a priori, any static assignment of grid blocks may lead to dynamic load imbalance in unpredictable manners. Removing this load imbalance in compositional reservoir simulation remains mostly an open problem. A dynamic load balancer is clearly needed to alleviate this difficulty. Admitted, dynamic load imbalance is very complex with sources from multiple parts. Besides the flash calculations mentioned above, well opening/shutting and associated hydraulics, property calculations and Jacobian construction, the linear solver in the Adaptive Implicit formulation and reporting may also bring some degree of dynamic load imbalance.

Sherman (1992) proposed a dynamic load-balancing scheme for parallel compositional simulation based on the Linda coordination language. However, this

parallel programming model is not widely used as opposed to MPI (Massage Passing Interface). Killough and Anguille et al. (1995) developed an improved load-sharing and receiver-initiated dynamic load balancing algorithm for parallel compositional simulation and reported substantial improvement. Little or no research has been reported over the past decade in this area. In essence, to optimize the parallel performance by communicating information from one processor to another relies on message passing. The overhead, or the time involved in message passing creates extra elapsed time which is added to the total computational time. Thus, in principle, all load-balancing schemes can be treated as a compromise between the reduction of load imbalance and minimization of the overhead. However, this improvement is likely to degrade with more processors since this mechanism requires a significant amount of data exchange between processors. As a model is scaled to more processors, this overhead may offset any potential gain by a load-balancing scheme. Moreover, implementing dynamic load balancing requires intimate knowledge of the underlying reservoir simulator source code. It can be quite tricky and cumbersome to determine which variables must be exchanged between processors for legacy comprehensive reservoir simulators.

In this paper, we present a novel approach to address the dynamic load imbalance issue in parallel compositional simulation. This new methodology over-decomposes the underlying reservoir model into mini-subdomains. Based on the Charm++ infrastructure, a bundle of mini-subdomains are assigned to the available physical processors. Processors treat these bundles of mini-subdomains as virtual processes or user-level migratable threads, which can be dynamically migrated across processors in the runtime

system. To our best knowledge, this is the first adaptation of domain over-decomposition or processor virtualization to reservoir simulation.

Our approach can seamlessly handle static and dynamic load imbalance in a uniform fashion. Further more, this new approach requires minimal changes to the original MPI based reservoir simulator. The main contribution of this paper is to demonstrate that domain over-decomposition implemented as virtual processes can be applied to improve parallel performance of MPI based compositional reservoir simulations that suffer from load imbalance issues. The rest of the paper is organized as follows. We first present the legacy comprehensive reservoir simulator used in this study and the main features of Charm++ and AMPI. Next, we describe the parallelization of equation of state calculation using MPI and how we adapted it to exploit processor virtualization followed by results from our experiments. Conclusions and outlooks are provided at last.

2.2 Background

2.2.1 A Commercial Comprehensive Reservoir Simulator

A commercial comprehensive reservoir simulator (simulator hereafter) written by Fortran 90 and C is applied as the test bed in this study (Dean and Lo, 1988). Based on fully implicit and IMPES formulations, the simulator is capable of performing black oil, limited compositional and fully compositional simulation for single and dual porosity reservoirs. It utilizes 3D radial and corner point grid system and supports multiple levels of local grid refinements. Wellbore parameters are treated implicitly during the simulation process and wells can be connected into surface networks. Geomechanics is

also supported. Stress, strains, and displacements can be calculated throughout the simulation in a fully coupled fashion with flow calculations. The linearized equation system is solved by iterative solvers with preconditioning and acceleration options. In a word, this is a comprehensive reservoir simulator with production quality. The hope is to demonstrate that the technique introduced below is applicable to real-strength reservoir simulators.

2.2.2 Charm++ and Processor Virtualization

Charm++ (Kale et al. 2008) is an objective-oriented parallel programming library for C++/C/Fortran. It aims to improve productivity in parallel code development and enhance parallel scalability. Charm++ is message-driven. It does not block the processors while waiting for messages to be received. Based on migratable objects, Charm++ uses the idea of processor virtualization. In this framework, the programmer decomposes a domain into N subdomains to execute on P processors. In ideal case, we should have N >> P. In the programmer's point of view, it is seen that the program is running using N subdomains. The Charm++ runtime system maps those subdomains or more specifically Charm++ objects to the P available processors. **Figure 2.1** provides a schematic illustration of the basic idea of processor virtualization. The ratio of N over P is the so called processor virtualization ratio. The mapping is dynamic and the subdomains can migrate across processors during program running. This unique capability is utilized by the underlying intelligent Charm++ runtime system, which provides potential for better overlap between computation and communication as well as cache utilization.

The combination of a natural encapsulation mechanism and an intelligent runtime system has made Charm++ suitable for parallel code development over a range of computing architectures, from personal computers to large-scale parallel clusters, from multicore CPUs to massively-parallel GPUs. In addition, it has been applied to scale real-world applications to thousands of processors on several scientific and engineering fields, such as quantum chemistry (Bohm et al. 2008), computational cosmology (Jetley et al. 2008), rocket simulation (Jiao et al. 2005) and weather forecasting (Rodrigues et al. 2010, 2010).

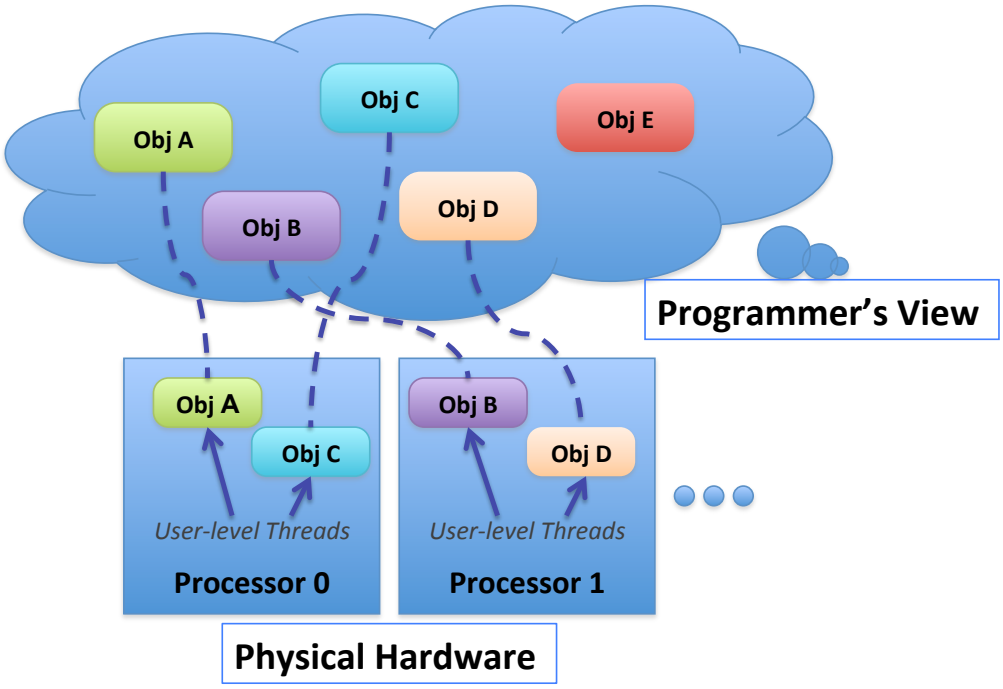

Figure 2.1 Programmer view vs. real implementation of over-decomposed objects

2.2.3 Adaptive MPI

Adaptive MPI (AMPI) is an implementation of the MPI standard on top of Charm++ (Huang et al. 2003, 2006; Zheng et al. 2006). As abovementioned, the developer only sees the virtual processors while the mapping of virtual processors to physical processors is handled by the Charm++ runtime system. It is illustrated in **Figure 2.2** that in AMPI the original MPI processes from a programmer's perspective are embedded in a Charm++ object as user-level threads. These user-level threads are not only migratable between physical processors but also have very short context switch times. In the context of AMPI, the N MPI tasks in a MPI code are referred as Virtual Processors (VPs). A VP is assigned as a user level thread and a bundle of VPs share one physical processor. Without any programming effort, the overlap between computation and communication is automatically achieved by having more VPs than real physical processors. When one VP is blocked from communication, the Charm++ scheduler picks up the next VP to execute. More specifically, when some VPs of a physical processor are waiting for messages to be received, other VPs can continue their execution in this particular physical processor. As a result, performance can be improved without any source code change. Since smaller subdomains may fit into cache if the over-decomposition is enough, better cache utilization is expected in this situation. Therefore, it is natural to see legacy MPI code will benefit substantially from AMPI and this potentially significant benefit comes with few catches and without major modifications of the original MPI code.

However, to utilize AMPI in practice, one must pay attention to global and static variables. Those variables are problematic for a multi-threaded programming model such as OpenMP and AMPI. In MPI, since only one thread exists in the allocated process's address space, global and static variables are safe. But, if a single instance of a global or static variable is shared by more than one thread in the single address space, incorrect results are likely to be generated. In other words, VPs residing on a particular physical processor will access the same copy of global and static variables. If those variables are to be read and updated, conflicts will occur and correct results cannot be guaranteed. Thus, one has to privatize global and static variables. There are a few approaches for this privatization, which will be discussed in following sections.

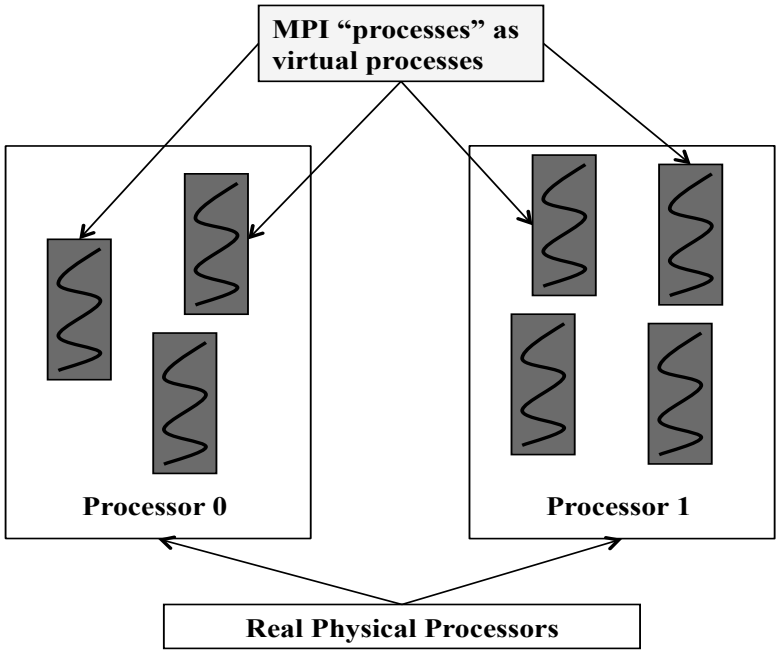

Figure 2.2 MPI "processes" are implemented as virtual processes (user-level threads) in AMPI (adapted from Huang et al. (2006))

2.2.4 Native Load Balancing in AMPI and Charm++

Charm++ provides a native and powerful infrastructure for load balancing. Charm++ is based on the idea that load behaviors from the recent past provide sound predication for loads in the near future. By actually measuring the load information at runtime, it migrates VPs from heavily loaded processors to lightly loaded processors. Thus, Charm++ use a measurement based load balancing technique. Several different load balancing policies have been made available by Zheng (2005). In addition, a new load balancer can be written simply by using the Charm++ API. A particular load balancer is selected in the command line at execution time.

The specific statement to invoke a load balancer in AMPI is MPI_Migrate(). When MPI_Migrate() is called, VPs may migrate between processors, if it is determined that such migration will improve parallel performance. Obviously, the frequency of calling MPI_Migrate() is determined by the compromise between the overhead and performance degradation caused by load imbalance.

2.3 Adaptation of Reservoir Simulator to AMPI

Generally speaking, adapting an MPI program to AMPI is a simple process. But as explained in the previous section, one must pay attention to the global and static variables. Since the original simulator is serial, the first required step is to parallelize a portion or all of the simulator using MPI. In this section, we present the parallelization of a portion of the equation of state of simulator, possible ways of global and static variables privatization, and different load balancing schemes.

2.3.1 Equation of State Parallelization

To serve as a proof of concept study, in this paper we only parallelize the equation of state computations which can take more than 70% of the total execution time. The idea of parallelization of this portion of the compositional simulator is rather straightforward. Since the flash calculations for each grid block are independent from other grid blocks, no subdomain boundary data exchange is needed. Basically, at the equation of state subroutine, we do the follow four steps:

1. Divide the key input and output arrays according to the domain decomposition setup. We use 2-D domain decomposition, i.e. the reservoir is divided in X and Y directions while keeping the Z direction undivided. This domain decomposition setting is reasonable since in general the Z direction extent is far less than the X and Y directions.
2. Let the master processor distribute necessary information to the slave processors.
3. Each processor performs equation of state computations independently.
4. Let the master processor gather computed information from the slave processors. Once the master has finish gathering, all processors then exit the equation of state subroutine.

Although it seems to be an easy parallelization according to the above four steps, it is not the case for a comprehensive reservoir simulator. The equation of state and other involved code in this parallelization is about 20K lines. Moreover, the grid blocks are reordered inside flash calculation according to their fluid types. **Table 2.1** lists the eight fluid types. Thus, we could not expect a big DO loop over all grid blocks, which can be

parallelized easily as shown in **Table 2.2**. *N* is the total number of grid blocks that are to be flashed. *numproc* is the number of requested processors. So, *N_local* is the number of grid blocks that are assigned to each available processor. Note that, to shorten the presentation, it is assumed that *N* can be divided by *numproc* exactly. *A* and *A_local* represent the arrays which are to be updated by flash calculations. *A_local* is allocated to each available processor to store information computed. *A* is the global array to collect information from each available processor. To be concise yet complete, we omit the actually arguments of MPI and subroutine calls. As abovementioned, since the grid blocks are reordered according to fluid types, we construct and parallelize a derived type on top of the actual equation of state routine to govern the data structure in the flash calculations. In the derived type, pointers are set up to trace the grid blocks to be updated in each subdomain. In this setting, we do not need to worry about the grid reordering inside of the flash calculation.

Table 2.1 Fluid types in the flash calculation

Fluid Type	Meaning
1	Two phase jacobian constant (won't be updated)
2	Two phase jacobian will be updated
3	One phase checked by negative flash
4	One phase checked by stability test calculation
5	One phase not tested (was single-phase last call and do not have two phase neighbor)
6	Aquifer cells
7	Zero PV cells (for well calculation, new well and inactive)
8	One phase cells (converged from stability test)

Table 2.2 Pseudo Fortran MPI code for flash calculation

```
N_local = N/numproc
! Broadcast variables and arrays to slave processors
CALL MPI_BCAST()
DO i = 1,N_local
   A_local(i) = flash()
ENDDO
IF (myid == Master) THEN
! put A_local computed by master to global A
   A = A_local
   DO i = 1,numproc-1
      CALL MPI_RECV()
! put A_local computed by slaves to global A
      A = A_local
   ENDDO
ELSEIF (myid /= 0) THEN
   CALL MPI_SEND()
ENDIF
```

Since we only parallelize a portion of the simulator, we let the master processor perform the rest of the simulation. Special care must be paid to places where GOTO statements are used such as when the newton iteration is not converged and time step reduction is performed. Otherwise, a deadlock condition will be likely occurring. **Table 2.3** lists a situation in which a deadlock might happen. No deadlock will happen if label 1000 is inside the IF_Master structure. However, when label 1000 is outside of IF_Master and slave processors need to perform computation after this label, deadlock will happen if trigger is true. This is because only the master processor gets the signal to go to label 1000 while other slave processors are hanging there. If this situation happens, the program will be waiting with slave processors at label 1000 and the program will

stall. Since there is no OMP Flush like function in MPI to flush all the processes to have the same view of a variable, to fix this issue we should broadcast the triggers to all the processors and let all processors go to label 1000 explicitly. A quick fix for **Table 2.3** is provided in **Table 2.4**.

Table 2.3 Pseudo Fortran code that may cause deadlock

```
IF_Master : IF (myid == Master) THEN
   Perform computation
   IF (trigger == .TURE.) THEN
      GOTO 1000
   ENDIF
ENDIF IF_Master
```

Table 2.4 Pseudo Fortran code to fix deadlock

```
IF_Master : IF (myid == Master) THEN
   Perform computation
ENDIF IF_Master
CALL MPI_BCAST(trigger)
IF (trigger == .TURE.) THEN
   GOTO 1000
ENDIF
```

It is readily seen that the slowest processor determines the time spent in equation of state. If one or few processors contain grid blocks that have phase changes or are at critical condition, those busy processors will dominate the computation time while other processors are idle. It is difficult to predict which processor will be busy. One may argue that a processor that has wells will be busy; but such processors will not always be the busiest due to the processes occurring around the moving injectant-displacement front.

This front movement is generally unknown *a priori* (since predicting where the front will go and how long it will take is what the simulator is designed for). One may also argue that we can estimate where the fluid will go roughly from the permeability field, but decomposing a real world reservoir according to the permeability will complicate the coding to an even greater extent. Moreover, even we could assign more processors to high permeability channels; this will not help since we still do not know when the front arrives at a certain place *a priori*. The only feasible and unified approach for this issue is to use finer grained computation and migrate those fine-grained units when necessary in a smart run-time system, which Charm++ can provide.

2.3.2 Domain Over-Decomposition

As explained in the previous section, processor virtualization is required to enhance parallel performance of an existing MPI code. It is not redundant to emphasize again that one can just decompose the domain as if as many as processors that are required are available. In other words, the MPI code does not need to be changed at all in this aspect. For example, as shown in **Figure 2.3**, one just decomposes the domain as if there are 16 processors. But in fact, these are 16 VPs that are to be mapped to the 4 physical processors. In **Figure 2.3**, the thick lines divide the physical processors while the thin lines separate the VPs. In this example, the processor virtualization ratio is 4. One should also appreciate that such over-decomposition may enhance cache utilization. As shown in **Figure 2.5**, subdomain 3 might fit in cache while subdomain 0 might not. Note that over-decomposition is applicable if and only if the results are independent of

the number of MPI tasks and one should be beware of the increase in memory usage due to over-decomposition.

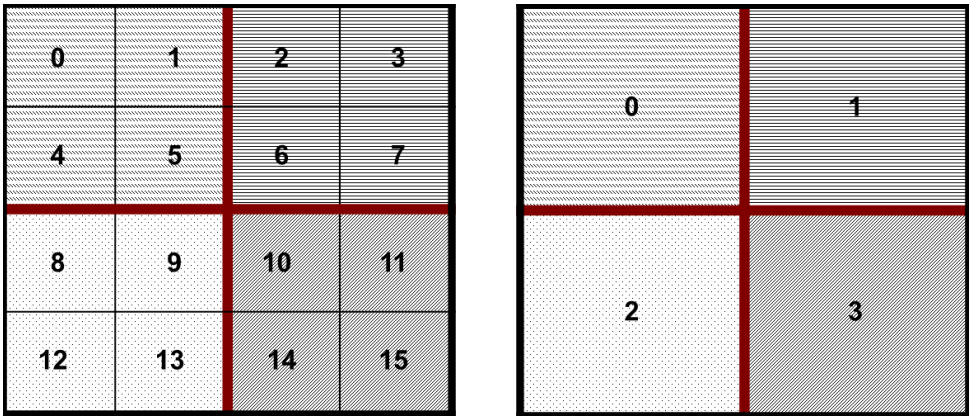

Figure 2.3 Domain over-decomposition (On the left, the domain is over-decomposed into smaller subdomains. The same grid fill pattern denotes the same physical processor (adapted from AMPI manual))

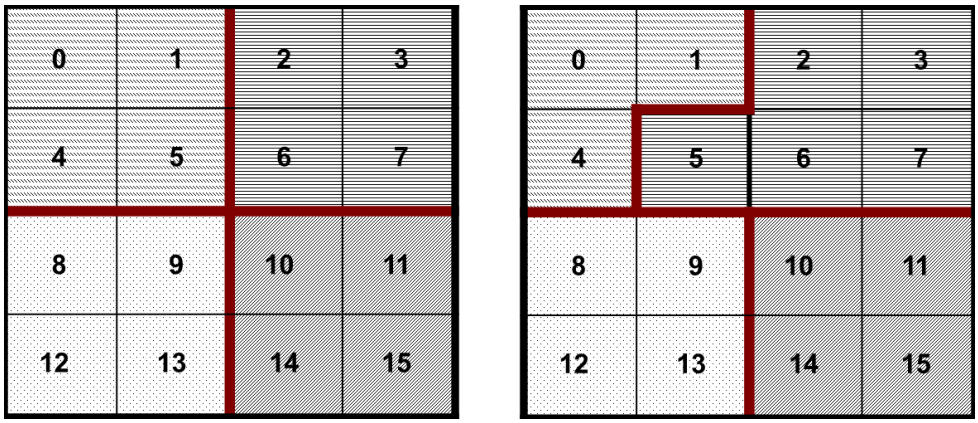

Figure 2.4 Migration of VP (adapted from AMPI manual)

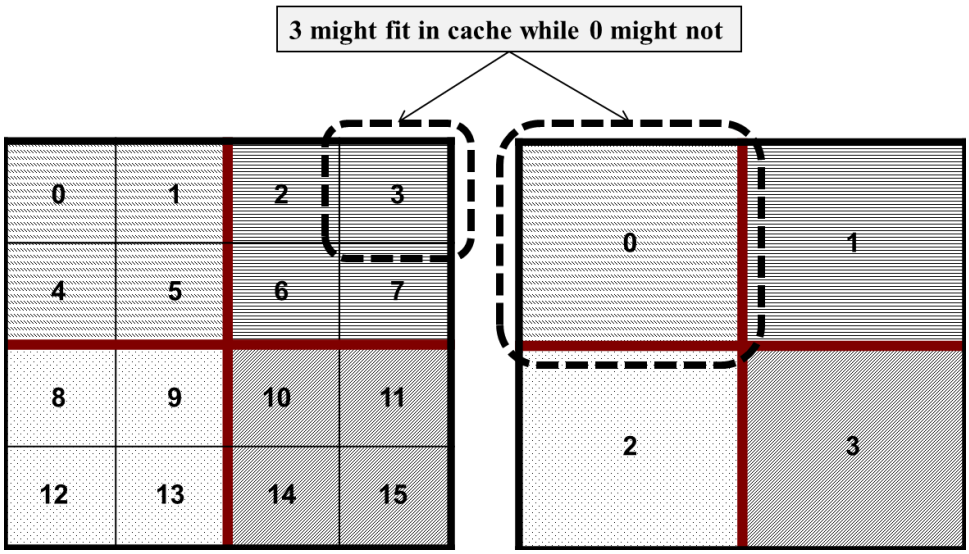

Figure 2.5 Subdomain from over-decomposition might fit in cache (adapted from AMPI manual)

2.3.3 Variable Privatization

As illustrated in **Figure 2.3, 2.4** VPs share one physical processor. We have discussed that such a scenario will cause problems if global and static variables are used. Thus, these variables have to be privatized. It is noted that, in Fortran, module variables, saved subroutine variables, and common blocks belong to this category. Fortunately, since we only execute the equation of state in parallel, the number of variables to be privatized is not significant. **Table 2.5** lists the number of global, static variables and common blocks. Currently, there are 4 approaches to privatize global and static variables with different mechanisms and applicability:

Table 2.5 Number of variables to privatize

Globals	Static	Commons
124	4	0

2.3.3.1 Manual Change

The thought behind manual change is to pass the information those global variables carry as subroutine arguments, since subroutine arguments are passed on a stack which are not shared across threads. We could gather global variables together in a single derived type in Fortran 90. This derived type is allocated by each thread dynamically. We then set up a pointer to this derived type. The pointer is passed across the subroutine as an argument. This mechanism will make sure each thread owns a private copy of the global variables. Static variables can be handled in the same way. As the name suggests, this privatization requires manual changes of all of these variables, which can be very tedious and bug-prone if the number of global variables is significant. Thus, it is clearly not a good approach for a large legacy reservoir simulator.

2.3.3.2 Source-to-source Transformation

There is a way to do the manual change automatically. A tool called Photran by Zheng et al. (2011) is available to transform the source code to change global and static variables in objects and then pass the objects across subroutines. Photran works by constructing abstract syntax trees of the program. However, at the time of preparation of this paper, Photran is only in beta phase. In our limited experiments, this tool works well for simple example code but is tends to be unstable or even crash for large code.

Moreover, the readability of the transformed code by Photran decreases to some extent. Therefore, it is not recommended to apply Photran until a stable version is released.

2.3.3.3 Automatic Global Variables Swapping

AMPI provides a build-in compilation flag that can automatically privatize global variables on systems that support Executable and Linkable Format (ELF). ELF is now a standard for objective files in Unix-like systems. It works as it maintains a Global Offset Table (GOT) for global variables and switches GOT contents at thread context switching. It is very straightforward to apply this approach. All one need to do is to add the flag –swapglobals at compilation and link time, which will enforce that each VP has its own view of a certain global variable. It works for C/C++/Fortran and x86 and x86_64 platforms. However, the drawback is that it does not handle static variables and has a context switch overhead that grows with the number of global variables. For static variables, we could replace saved local variables with a module, which transforms static variables to global variables. Since in this study the number of global and static variables is not high, in this paper we apply this approach.

2.3.3.4 Privatization Based on Thread Local Storage (TLS)

It is easy to appreciate that when the number of global and static variables is significant, which is often the case for a comprehensive parallel reservoir simulator, the overhead caused at context switch by –swapglobals may become excessive. Thus, clearly automatic global variables swapping is not the silver bullet for a comprehensive parallel reservoir simulator. Rodrigues et al. (2010) developed a better privatization strategy based on TLS (Thread Local Storage). TLS is designed for thread safety. This

approach works by allocating one instance of the variable per thread. To utilize TLS to privatize global and static variables, one just need to add C directive "__thread" before all global and static variables. This simple modification will make these variables have thread local storage duration, which means a unique instance of a particular global or static variable is created for each thread that uses it and is destroyed when the thread terminates in a multi-threaded environment. Privatization based on TLS not only has no context switch overhead but also can handle both global and static variables. However, unfortunately, only C/C++ compilers have implemented TLS at this time and there is no such directive in Fortran. As a workaround for Fortran program, one can write a GFortran patch file to modify GFortran to adopt the TLS method for global and static variables privatization. How to modify GFortran compiler is certainly beyond the scope of this paper. At the time when this paper was prepared, a patch file for GCC 4.5 is being developed for our future study based on (Rodrigues et al. 2010; Rodrigues 2012). This approach is recommended if one want to apply AMPI to a comprehensive parallel reservoir simulator written in Fortran.

In summary, although we apply automatic global variable swapping for variable privatization, the TLS based privatization might be the only applicable strategy for a comprehensive parallel reservoir simulator at this time. Even if one has to have a modified GFortran for Fortran MPI code, this obstacle is not significant to surmount if one can adsorb or find expertise on compiler writing. One must keep in mind that global variable swapping and TLS are not supported by all platforms. For example, Blue Gene/P and Mac OS X support neither of them and Cray/XT only supports TLS. In fact,

only the first two methods are applicable for all platforms. Therefore, when a stable Photran is debuted, the adaptation of MPI code to AMPI might become truly trouble-free.

2.3.4 Dynamic Load Balancer

As stated previously, Charm++ and AMPI can migrate VPs across processors to balance load. As illustrated in **Figure 2.4**, when Processor 0 becomes overloaded while Processor 1 is under loaded, VP 5 will be migrated to Processor 1 if a load balancer is invoked. To utilize the dynamic load balancing strategy of AMPI, one only need to insert MPI_Migrate() calls at a certain frequency and setup the dynamic load balancer at compilation and runtime in the command line. In the reservoir simulator we can call MPI_Migrate() every *nt* time steps as follows:

```
IF (mod(time_step, nt) == 0) CALL MPI_Migrate()
```

We adapt the methodology of Rodrigues et al. (2010) to quantify imbalance by how much the load in the most loaded processor is above the average load. An imbalance threshold can also be set to trigger migrations.

Charm++ provides several dynamic load balancers that consider computational and/or communication load. The selection of a specific load balancer depends on the application itself. If an application only has computational load and no communication traffic, a balancer that only takes computational load into account will be enough. Otherwise, a balancer based both on computational and communication loads must be chosen.

Since inter-subdomain data exchange is not necessary for equation of state computations, we may choose dynamic load balancers that only handle computational

load. Two balancers are selected in this study: GreedyLB and RefineLB. GreedyLB uses a greedy algorithm to assign the heaviest object to the least loaded processor till a balance is reached. RefineLB not only moves objects from overloaded processors to under loaded ones but also limits the number of objects migrated. In general, RefineLB is useful when only a few VPs migrations are sufficient to reach balance.

Although the case application in this paper does not need ghost cells along subdomain boundaries to exchange data, it is worth mentioning the dynamic load balancers that also take communication traffic into consideration. This kind of balancer is required for a general parallel reservoir simulator. The following discussion will also serve as a reference for our future study. The build-in balancers in this perspective are GreedyCommLB, RefineCommLB, RecBisectBfLB and MetisLB. GreedyCommLB extends GreedyLB to take the communication graph into account. RefineCommLB applies the same idea as RefineLB but takes communication into account. RecBisectBfLB recursively partition the communication graph until the number of partitions is equal to the number of processors. However, RecBisecBfLB does not explicitly guarantee that communication traffic is minimized. MetisLB uses Metis to partition the communication graph. The selection of these balancers is application dependent and heuristic. A systematic comparison between these balancers is recommended to understand their behavior for a particular application. It is noted that none of these balancers explicitly consider the spatial relationship between VPs. Thus, neighboring VPs might be distributed to different processors or even far away nodes in runtime. This is clearly not an optimal scenario since there might be communication

between neighboring VPs. Thus, a balancer that considering the spatial relationship between VPs is preferred. Rodrigues et al. (2010) developed a new balancer based on Hilbert curve to consider such situations.

2.4 Example

In this section, we provide a case study for applying the domain over-decomposition and dynamic load balancing techniques for parallel equation of states computation of compositional simulation. We begin with describing the compositional reservoir model used in this case study and showing results of MPI execution on 64 physical processors. Next, we study the effects of domain over-decomposition and load balancing on this case model. We first show the results of domain over-decomposition without load balancing by simply varying the processor virtualization ratio. We then bring the effect of load balancing by dynamically migrating VPs during simulation execution.

All the simulations are performed on an IBM iDataplex cluster with Intel 8-way Nehalem and 12-way Westmere processors. All the cores are at 2.8 GHz and nodes are connected via 4X QDR infiniband. The development tools used are Intel Fortran and C compilers and Open MPI.

2.4.1 Reservoir Model

As depicted in **Figure 2.6**, a 3D reservoir model using a corner point grid is used as the test case. It contains 15 vertical layers and each layer has 256×256 grid blocks. Thus, it has totally 983040 grid blocks. This reservoir has about areal dimension of about 3760 acres and 150 ft thickness. 14 gas injectors and 14 producers are perforated

throughout all of the layers. Porosity is generated by sequential Gaussian simulation and permeability is obtained by correlation. This reservoir model contains 12 components except water. The properties of these components are provided in **Table 2.6**. In **Table 2.6**, MW, TC, PC, W, ZC, VPARM are molecular weight, critical temperature, critical pressure, Pitzer accentric factor, critical Z-factor, and volumetric shift parameter, respectively. A 20-year production/injection period is simulated for this model.

The IMPES formulation is applied in this case. The key parameters used to control the equation of state convergence are residual norm of flash calculations, maximum residual of constant K-flash, and change in K-values for stability calculations, which are set to be 1.0e-6, 1.0e-6, and 1.0e-8, respectively. The jacobian is set to be recalculated only at selected grid blocks and K-value is set such that starting values cannot be borrowed from neighbor grid blocks. The timing summary for this reservoir model in sequential execution is listed in **Table 2.7**. We can see that, for this case, the equation of state calculations consumes almost 80% of the total computational time. It is this part of computation that we parallelize in this study.

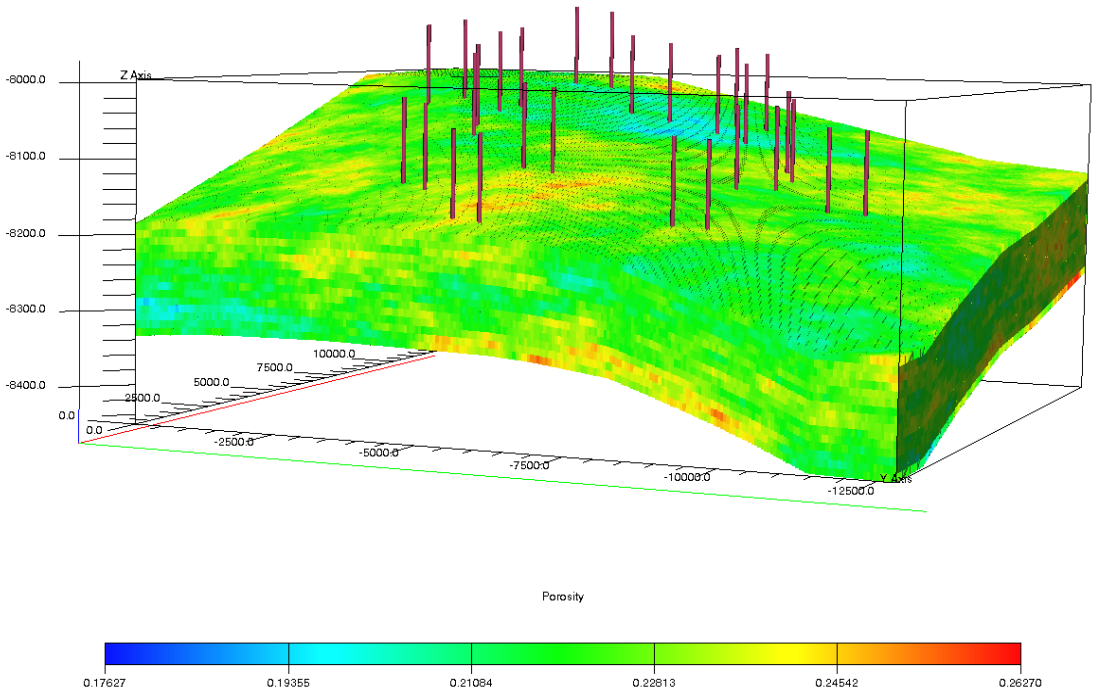

Figure 2.6 3D Reservoir model (axes unit in ft)

Table 2.6 Properties of Components

Component	MW	TC (Rankine)	PC (psi)	W	ZC	VPARM
CO_2	44.010	547.650	1071.30	0.2250	0.2750	0.02700
C1	16.040	343.040	667.800	0.0130	0.2900	-0.11800
C2	30.070	549.760	707.800	0.0986	0.2850	-0.10700
C3	44.100	665.680	616.300	0.1524	0.2770	-0.08477
C4	58.120	765.320	550.700	0.2010	0.2740	-0.06858
C5	72.150	845.370	488.600	0.2539	0.2690	-0.04103
C6+	94.200	975.920	458.680	0.2695	0.2663	-0.00076
C8+	116.00	1087.87	408.080	0.3328	0.2589	0.05973
C11+	169.50	1223.01	305.180	0.4856	0.2546	0.08719
C15+	232.60	1353.35	248.530	0.6436	0.2691	0.09684
C20+	328.00	1458.35	227.270	0.7926	0.3165	-0.06104
C30+	628.00	1670.24	168.570	1.0536	0.3676	-0.13829

Table 2.7 Percent of time spent in each portion of sequential execution of the test case

Portion of simulator	Percent of time spent
Linear Solver	14%
Equation of State	79%
Coefficient Setup	6%
Others	1%

2.4.2 Performance of MPI

As has been discussed in previous sections, MPI implementation of a parallel equation of state is very likely to suffer dynamic load imbalance issues. Subdomains with phase changes or at critical conditions will dominate the execution time. To test the performance of an ordinary MPI implement with static decomposition, we run the case reservoir model using 64 physical processors. The model is divided into 64 subdomains using 2D domain decomposition, i.e. 8×8 decomposition. Depicted in the left picture of **Figure 2.7** is the gas saturation distribution of the top layer at 7300 days. The small squares that are separated by dash lines correspond to the decomposed subdomains. The saturation map is believed to be a strong indicator for processor load in the equation of state calculations. Shown in the right picture of **Figure 2.7** is the gray-scale-coded load map at 7300 days for the 8×8 set of physical processors. The load scale has a range from 0 to 1. The darker the color is the higher the load is for a particular processor. Indeed, there is a clear correlation between the saturation map and load map.

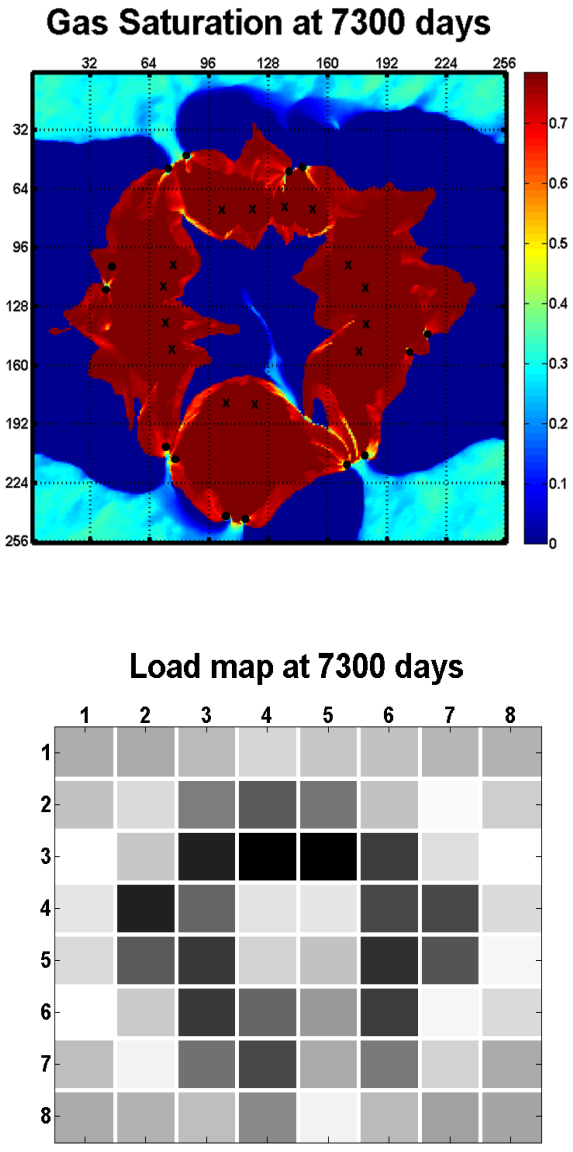

Figure 2.7 Upper: Gas saturation at 7300 days; Lower: Load map at 7300 days

In addition, the load distribution of physical processors does not remain unchanged as simulation running. We explicitly show the load map at 30, 365, 1825, 3650, 5475 and 7300 days in **Figure 2.8**. To better quantitatively understand the load imbalance, we also provide the column plots at these time snapshots in **Figure 2.9**. Note that, the loads are normalized to the highest processor at each snapshot. Initially, at 30 days, the dark processors correspond to the subdomains that contain wells. Other processors are roughly equally loaded. This is easy to be understood since at 30 days most of the states changes happen near the wells. The load maps become complex at later time snapshots. Although the load maps do seem to be similar at 1825, 3650, 5475 and 7300 days, the imbalances and high-to-low load ratios are different. The imbalance is quantified by how much the load in most loaded processor is above the average load. The imbalances and high-to-low load ratios at these snapshots are listed in **Table 2.8**. It should be noted that, since a star topology is applied such that the master processor scatters data to and gathers results from all of the slave processors, the master processor will have significantly more message send and receives. For this reason, the load results are chosen to only reflect the actual equation of state computations. It can be seen from **Table 2.8** that imbalance and high-to-low load ratio tends to increase with time.

Table 2.8 Imbalance and high-to-low load ratio at selected days

Time (days)	Imbalance	High-to-low load ratio
30	46%	2.08
365	36%	2.21
1825	48%	3.42
3600	52%	4.45
5475	53%	4.98
7300	55%	5.60

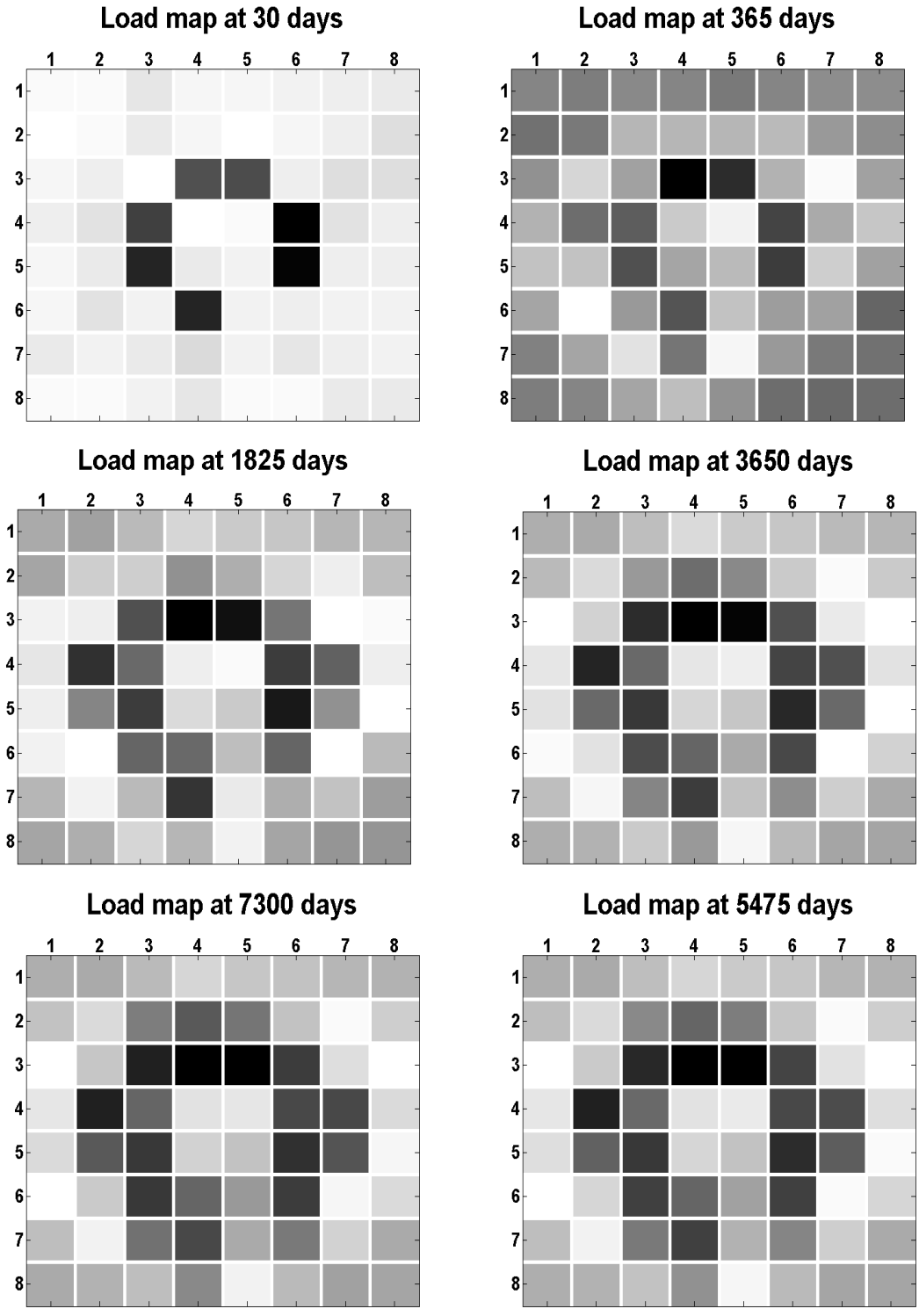

Figure 2.8 Load map recorded at various time snapshots

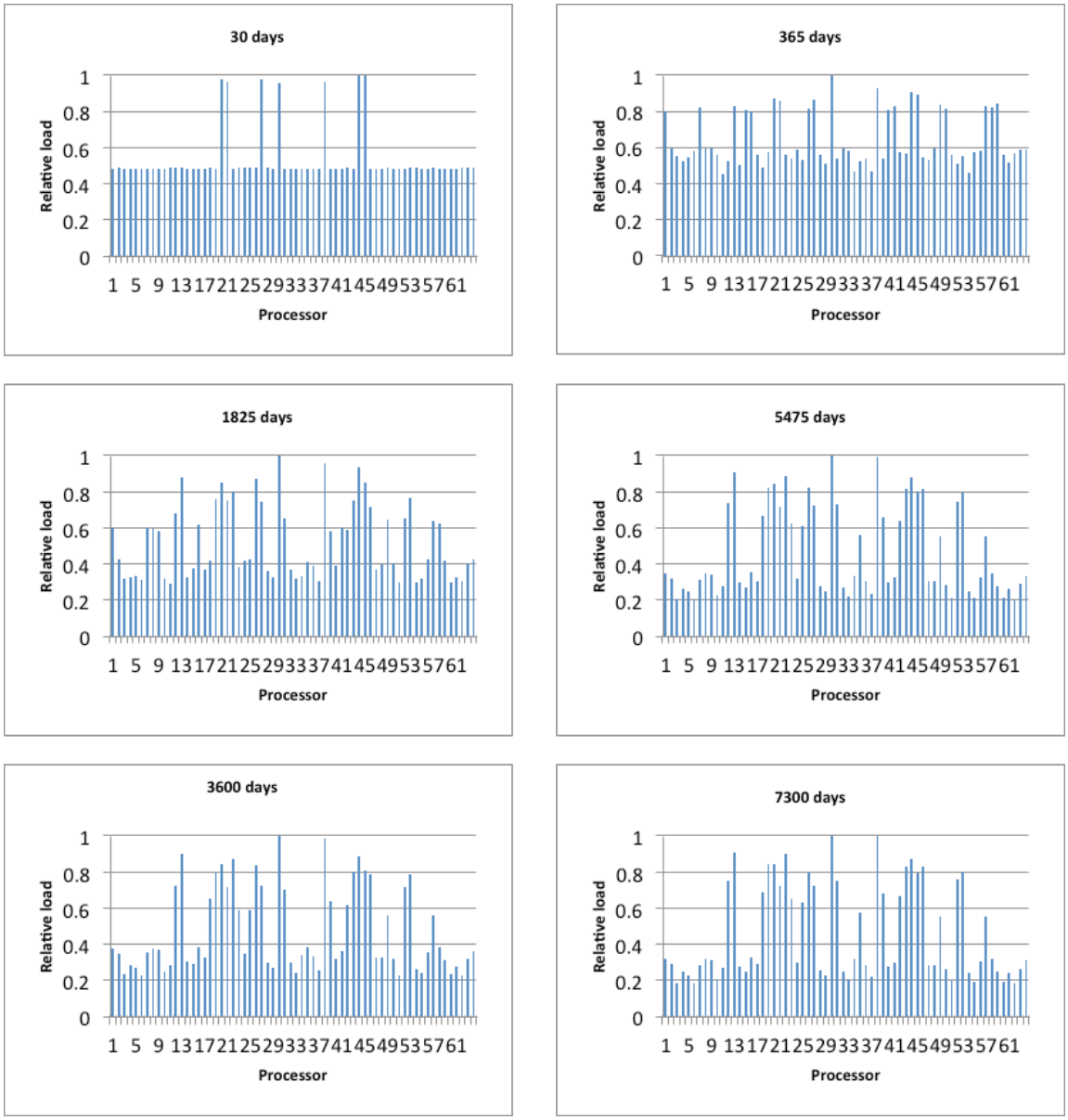

Figure 2.9 Column plots of load at various time snapshots

2.4.3 Performance of Processor Virtualization

We first evaluate the effect of domain over-decomposition or processor virtualization on the case model. To perform this comparison we simply vary the number of VPs using a fixed number of physical processors, i.e. 64 processors in this case. It is expected that overhead due to handling extra user-level threads and communications will be brought in by increasing the processor virtualization ratio. However, there may still be benefits due to the implicit overlap of computation and communication and better cache utilization. For example, in this case model, when the master user-level thread blocks a receiving message, other VPs can still execute on the same physical processor; while in ordinary MPI execution, this physical processor will be held for receiving.

We compare the execution timings of equation of state of the case model at 7300 days with 64, 256, 1024, 2048, 4096 virtual processors, respectively. We consider the elapsed time during actual equation of state computation. As it can been seen in **Table 2.9**, the execution time is reduced by 19.78% with 256 VPs, 23.23% with 1024 VPs, 24.48% with 2048 VPs, and 16.27% with 4096 VPs. It seems that there is an optimal number of VPs. The use of more VPs does not improve the performance further. To understand this behavior, we provide the column plot of the actual time of equation of state computation using different VPs in **Figure 2.10**. As we can see from **Figure 2.10**, the load imbalance pattern remains unchanged for different processor virtualization ratios. The reduction in computational time is believed to be the result of better cache utilization since the subdomain size becomes much smaller if high processor virtualization ratio is applied. There are a number of steps in the stability test and flash

calculations. Each step may require reading previous results. In the over-decomposed setting, the results and data of an entire subdomain during the whole process may fit into the cache. But, in a conventional domain decomposition setting, those data may have to be stored in faraway memory which will cause latency. The reduction of time in data scattering and gathering is the result of overlap of computation and communication when the master VP performs sending and receiving of messages. Thus, the observed effects of processor virtualization are the combination of results from better cache utilization and overlap of computation and communication. However, the performance improvement is limited by the server load imbalance in equation of state computations.

Table 2.9 Execution time of equation of state on 64 processors

Virtualization configuration	Execution Time (s)
No virtualization	1960.5905
256 VPs	1572.3643
1024 VPs	1505.1414
2048 VPs	1480.5817
4096 VPs	1641.8273

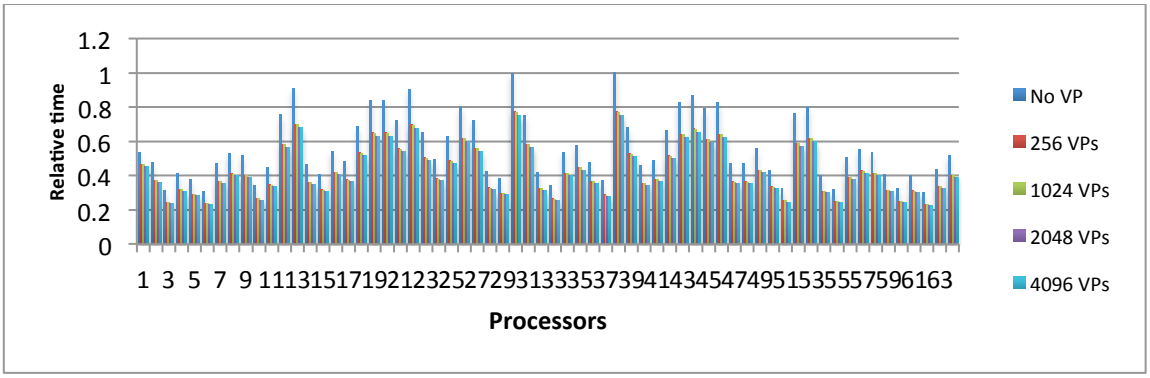

Figure 2.10 Column plots of load for different number of virtual processors

2.4.4 Performance of Dynamic Load Balancing

As revealed in last section, merely domain over-decomposition can only improve the performance to some extent (24.48% in this case). A dynamic load balance scheme must be applied for better performance enhancement. Fortunately, load balancing becomes much easier in the virtualized implementation of MPI, since the over-decomposed VPs are ready to be migrated to mitigate load imbalance. Thanks to the particular feature of equation of state, there is only computation involved and no communication needed between VPs. As a result, a dynamic load balancer that only handles computation should be enough for this kind of application. For this reason, we choose GreedyLB and RefineLB to balance the load during parallel equation of state computation. Listed in **Figure 2.11** are the timing results of each processor before and after implementation of the abovementioned load balancers. The reference timing without load balancer is shown in **Figure 2.11a**. Clearly, only a few of processors dominate the execution time, while

Table 2.10 Load balancing overhead

Load balancer	Overhead (s)
GreedyLB	40.85
RefineLB	4.21

Table 2.11 Speedup improvements for 64 processors

Configuration	Speedup
MPI	29.27
2048 VPs	38.73
2048 VPs + GreedyLB	62.38

others are idle. The results of GreedyLB and RefineLB are provided in **Figure 2.11b** and **Figure 2.11c**, respectively. Apparently, the performances of GreedyLB and RefineLB are distinct. GreedyLB balances the load very well. Indeed, by design GreedyLB algorithm always grabs the heaviest unassigned VPs and assigns it to the currently least loaded physical processor until a balance is reached. In contrast, as shown in **Figure 2.11c**, RefineLB only balances the load to some extent and the imbalance trend seems to be unchanged, since the RefineLB algorithm tries to minimize the number of VPs to be migrated. RefineLB is only useful when only a small number of VP migrations is enough to reach balance. Nevertheless, GreedyLB is an $O(NlogN)$ algorithm, which implies that it is more expensive than RefineLB. This is confirmed by measuring the overhead cost of the load balancing scheme and VP migrations. It can be seen from

Table 2.10 that GreedyLB is about 10 times more expensive than RefineLB in this example case.

However, these overheads become negligible when comparing with the total execution time. This is attributed to the smart runtime system and overlap of VP migrations and computations, which help to hide migration overhead. To finalize our experiments, we provide the speedup improvement after applying the new technique introduced in this paper. It can be seen from **Table 2.11** that the speedup using 64 physical processors is improved from 29.27 to 62.38.

2.5 Conclusions

In this paper, it is shown that load imbalance is the key performance limiter for parallel compositional simulation when using the IMPES formuation for reservoir simulation. This kind of load imbalance is often dynamic and highly unpredictable. Thus, a dynamic load balancing scheme must be implemented to improve parallel performance. However, adaptation of dynamic load balancers to an established legacy reservoir simulator can be challenging and requires a substantial amount of development time. This paper provides a promising shortcut to mitigate load imbalance yet with high efficiency. Built upon Charm++ and AMPI, this approach over-decomposes the underlying reservoir model into small chunks. A bundle of these chunks is then mapped to each physical processor as virtual processes or user-level threads. Based on this unique idea, we develop a new parallel equation of state computation capability on a legacy commercial comprehensive reservoir simulator. It is shown that domain over-decomposition and GreedyLB load balancer in together help to improve the speedup

from 29.27 to 62.38 on 64 physical processors. This is because, by design, domain over-decomposition not only brings overlapping between communication and computation and better cache utilization, but also provides a natural framework for dynamic load balancing.

It should be noted that it is due to the particular feature of equation of state computation that GreedyLB provides excellent performance. Generally, inter-subdomain data exchange is not needed in this application. Thus, a balancer without considering communication is enough. If the abovementioned approach is applied to general reservoir simulations, clearly, other balancers should be implemented to considering inter-subdomain communication. A full adaption to a comprehensive reservoir simulator should be straightforward if an MPI version is in hand. The key in a successful implementation is the correct treatment of global and static variables.

This technique is only adapted to the parallel equation of state in a reservoir simulator. The impact on the linear solver should also be investigated before full implementation. Based on limited experiments, it appears that domain over-decomposition can improve Jacobi, Gauss-Seidel and CG iterative solvers' parallel performances by 10%-20%. However, further research needs to be performed for more complex parallel preconditioners and linear solvers.

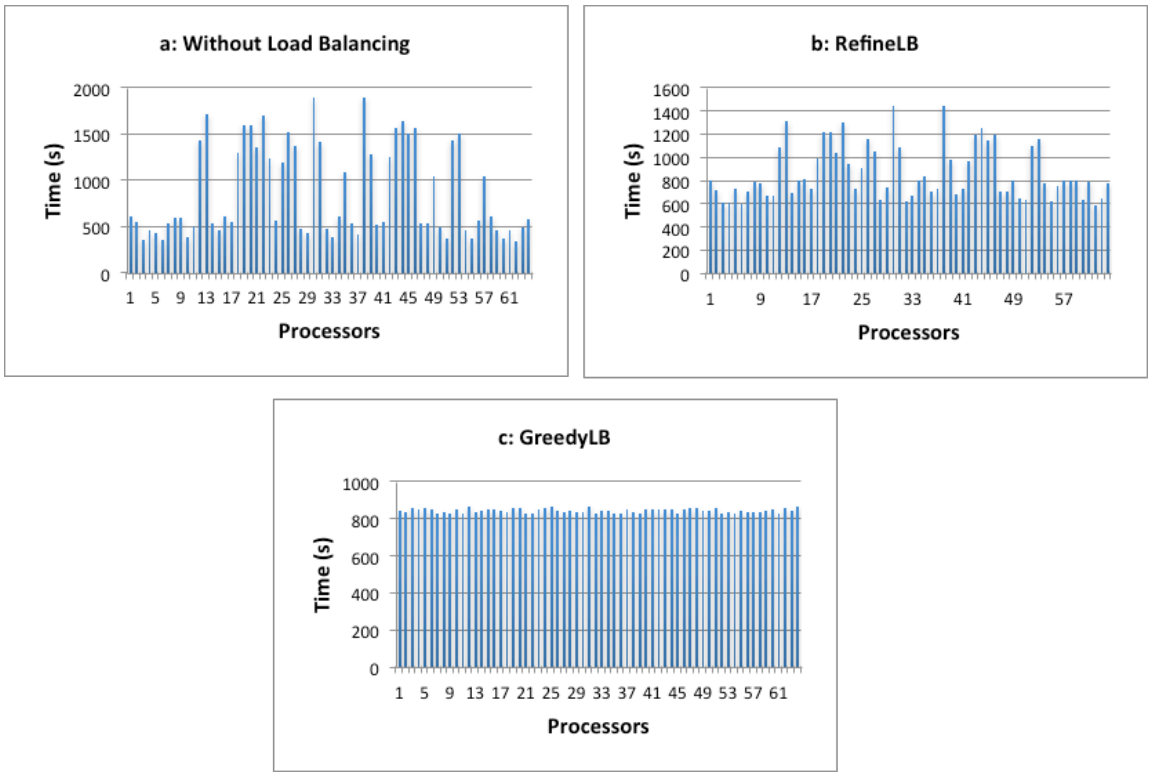

Figure 2.11 a: Time column plot without load balancer; b: Time column plot with RefineLB; c: Time column plot with GreedyLB

CHAPTER III

SOLVER PRECONDITIONING USING THE COMBINATORIAL MULTILEVEL METHOD*

The purpose of this chapter is to present the first preliminary study of the recently introduced Combinatorial Multilevel (CML) method for solver preconditioning in large-scale reservoir simulation with coupled geomechanics. The CML method is a variant of the popular Algebraic Multigrid (AMG) method yet with essential differences. The basic idea of this new approach is to construct a hierarchy of matrices by viewing the underlying matrix as a graph and by using the discrete geometry of the graph (Koutis 2007; Koutis et al 2009). In this way, the CML method combines the merits of both geometric and algebraic multigrid methods. The resulting hybrid approach not only provides a simpler and faster set-up phase compared to AMG, but the method can be proved to exhibit strong convergence guarantees for arbitrary symmetric diagonally-dominant matrices. In addition, the underlying theoretical soundness of the CML method contrasts to the heuristic AMG approach, which often can show slow convergence for difficult problems.

This new approach is investigated for both pressure and displacement preconditioners in the multi-stage preconditioning technique. We present results based on several known benchmark problems and provide a comparison of performance and complexity with the widespread preconditioning schemes used in large-scale reservoir simulation. An adaptation of CML for unsymmetric matrices is shown to exhibit excellent convergence properties for realistic cases.

---

* Reproduced with permission from "Solver Preconditioning Using the Combinatorial Multilevel Method" by Wang, Y. and Killough, J. 2013. Paper SPE 163589 presented at the SPE Reservoir Simulation Symposium, The Woodlands, TX, USA, 18-20 Feb.

3.1 Introduction

Reservoir simulation, which mimics or infers the behavior of fluid flow in a petroleum reservoir system through the use of mathematical models, is a practice that is widely used in petroleum upstream development and production. Reservoir simulation was born as an efficient tool for reservoir engineers to better understand and manage assets. However, like any numerical simulation tool, reservoir simulation is inherently computational intensive and easily becomes inefficient if more grids, coupled physics, and/or complex geometry are necessary to accurately describe the complex phenomena occurring in the subsurface. Mathematically speaking, reservoir simulation solves a system of discretized partial differential equations (PDEs) which describe the underlying physics. Due to stability constraints, an implicit formulation is required at least for the pressure system. Details about the numerical analysis for choosing an implicit formulation (or more specifically, the backward Euler method) can be found in the classic literature of Aziz and Settari (1979). However, as a recent exception, Piault and Ding (1993) attempted a fully explicit scheme in a reservoir simulation on a massively parallel computer and showed acceptable results. They adopted the Dufort and Frankel scheme (Dufort and Frankel 1953) which is unconditionally stable but numerically inconsistent. This scheme is of order of $\Delta t^2 / \Delta x^2$ accuracy, which clearly implies the truncation error can be significant if Δt does not approach *0* faster than Δx. In essence, implicit formulation is the only unconditionally stable and consistent scheme and is adopted by all commercial reservoir simulators. As a result, a linear solver is inevitable for reservoir simulation due to this implicit formulation.

There are four main streams of formulations applied in reservoir simulation: IMPES, fully-implicit, AIM, sequential implicit. Of these, fully-implicit is the most robust formulation but the resulting coupled system matrix is numerically challenging and computationally expensive. In the fully-implicit formulation, pressure, saturation/mass, and/or temperature are to be solved simultaneously. The generated system matrix is highly non-symmetric and not positive definite, which brings great challenges for applying robust and efficient preconditioners and liner solvers. This situation is further exacerbated for large-scale models with highly heterogeneous coefficients and unstructured gridding. Since, generally speaking, in black-oil simulation the solution of linear system ($Ax = b$) usually consumes up to 90% of the total execution time, linear solver performance enhancement means significant reservoir simulator speedup.

Many problems in petroleum extraction require understanding of fluid flow and its interaction with formation displacements. Fluid extraction and/or injection in deformable a reservoir formation modifies the *in-situ* stress field which may cause surface subsidence, fault activation, wellbore instability, thermal fracturing, etc. in geomechanically weak formations. To understand these problems one needs to perform coupled flow and displacement simulation. Generally, there are three approaches to couple flow simulation with poroelastic calculation: explicitly coupling, iterative coupling and fully-implicit coupling (Gai et al. 2003; Dean et al. 2006, Lu et al. 2007). In the explicit coupling scheme displacements are solved at selected time-steps. In iterative coupling flow and displacement are solved sequentially and then iteratively

coupled at each time step. For fully-implicit coupling, flow and displacements are solved simultaneously through a full system matrix that contains flow and displacement contributions. Fully-implicit coupling is the most stable approach of these and can have second-order convergence for nonlinear iterations. However, the resulting coupled matrix becomes even more challenging to be solved efficiently compared with the fully-implicit flow simulation without rock deformation.

The quest for robust and efficient linear solvers in the fully-implicit formulation and fully-coupled flow and displacement is one of the main themes focused on by simulator developers in the petroleum industry. Matrix scaling and reordering and variants of the ILU method remain the state-of-the-art of most reservoir simulators. However, the convergence rate is not independent of problem size and can be slow for difficult matrices. Inspired by the different characteristics of pressure and saturation parts of the full system matrix, two-stage preconditioning method was developed as a 'divide-and-conquer' using the CPR (Constraint Pressure Residual) approach of Wallis (1983, 1985). Cao et al. (2005) extended CPR to a general multi-stage preconditioning framework for fully implicit flow simulation. Under this framework, the full matrix system is decomposed into different sub-blocks to deal effectively with the specific algebraic characteristic of each subset of equations. For instance, the pressure part is mainly elliptical and nearly symmetric while the saturation part is mainly hyperbolic and non-symmetric. As the result, it may be more efficient to customize a preconditioning method for different sub-blocks. The nearly symmetric pressure sub-block is usually diagonal dominant with positive diagonal and non-positive off-diagonal entries. Thus, it

is generally positive (semi-) definite. This algebraic character enables us to apply the popular AMG approach for pressure preconditioning. Since the first application of the multigrid method in reservoir simulation by Behie and Forsyth (1983), AMG has become a very attractive option for the pressure solution. A number of implementations have been reported with promising performance. Generally, the convergence rate is independent of matrix size and scales linearly with matrix size. Stüben et al. (2007) developed efficient AMG implementations for fully-implicit and sequential implicit formulations. Klie et al. (2007) designed deflation AMG preconditioners for highly ill-conditioned reservoir simulation problems. The elliptic displacement sub-block resulted from coupled flow and geomechanics modeling is symmetric positive (semi-) definite, which makes multigrid applicable. White and Borja (2011) applied AMG as sub-preconditioner for fully coupled flow and geomechanics. Alpak and Wheeler (2012) implemented a supercoarsening multigrid solver for poroelasticity in 3D coupled flow and geomechanical modeling.

However, AMG is based on heuristics, especially for the classic Ruge-Stüben AMG (Ruge and Stüben 1987). Although it often exhibits impressive performance in practice, it does not offer guarantees on the speed of convergence especially for challenging matrices. In this chapter, a recently developed Combinatorial Multilevel (CML) method (Koutis et al. 2007) is introduced to reservoir simulation problems. CML has provable convergence properties and sound theoretical machinery. It not only offers convergence guarantees for SDD (Symmetric Diagonally Dominant) matrices with

arbitrary weights, but also has lower grid, operator and computational complexities comparing with other variants of AMG methods.

To our best knowledge, this is the first implementation of CML in reservoir simulation with coupled geomechanics. The contribution of this paper is that it adapts CML into the multistage preconditioning solution technique and provides performance comparisons with other popular preconditioners using challenging benchmarks. The paper is organized as follows. First, we briefly describe the multistage precondition framework and discuss the applicability of CML in this framework. Second, we present the CML algorithm. Third, we show comparisons using several case experiments. Finally, the conclusions and outlook are provided.

For the example cases CML results are compared with ILU(0) and two popular variants of AMG, Ruge-Stüben AMG and aggregation-based AMG (Notay 2010, 2012; Napov and Notay 2012). We use a Matlab/C implementation of CML (Koutis et al. 2009) in a comprehensive reservoir simulator. To compare with aggregation-based AMG, we choose the AGMG package (v3.2) (Notay 2012). For Ruge-Stüben AMG (RS_AMG hereafter), we use an implementation that is available as part of the PyAMG package (Bell et al. 2011). For all of the cases, the convergence criterion is set to $\|b - Ax\| / \|b\| < 1.0^{-6}$. All of the experiments were performed on a 64-bit Mac OS X 10.7 system with a 2.3 GHz dual-core Intel Core i5 processor and 8 GB DDR3 memory.

3.2 Solution Technique – Multistage Preconditioning

The multistage preconditioning framework was introduced to fully-implicit reservoir simulation by Cao et al. (2005). To keep the presentation concise, we describe the key algorithmic steps of two-stage preconditioning. The extension to multistage is straightforward. To solve the following linear system

$$Ax = b$$

where A is the coupled system matrix that contains pressure and saturation sub-blocks, we perform the following steps:

1. Map total residual to the constraint decoupled pressure residual, r_P. Several possible mappings are available for this stage, for example, an IMPES-like decoupling or a simple algebraic decoupling.
2. Solve the decoupled pressure system using a linear solver of selection to obtain $x_P = \widetilde{A}_P^{-1} r_P$. This is the first-stage preconditioning.
3. Update the total residual $r_{updated}$ using newly computed pressure x_P, $r_{updated} = r - AWx_P$. W is a mapping matrix to map x_P to the total solution vector.
4. Solve the fully coupled system using a selected linear solver to obtain $x = M^{-1} r_{updated} + Wx_P$.

The 4 steps are repeated until convergence or stopping criterion is reached. Note that, in step 2, a preconditioned linear solver is applied to solve the decoupled pressure system. M^{-1}in step 4 acts as the second stage preconditioner. As a result, a nested iteration is formed such that a pressure sub-block is solved at the inter iteration while

M^{-1}acts as a global smoother at the outer iteration. In practice, ILU, Gauss-Sediel, or block SOR is often an effective choice of M^{-1}. But these traditional preconditioners might not work well with the pressure sub-block that is mainly elliptic and cannot scale with the matrix size. Stüben et al. (2007) discussed the algebraic properties of the decoupled pressure sub-block and concluded that AMG is a favorable choice of preconditioner.

It is natural to appreciate that the multistage preconditioning methodology can also be applied to coupled flow and geomechanics simulation. We provide here a solution strategy that merges IMPES and fully coupled flow and poroelastic calculations. Indeed, the coupling between flow and poroelastic calculations is through pressure only. In this strategy, pressure and poroelastic calculations are coupled via GCR acceleration (Eisenstat et al. 1983). To solve

$$Ax = b$$

where A is now the full system containing pressure and displacement sub-blocks, we perform the following steps:

1. Map total residual to the constraint-decoupled pressure residual, r_P.
2. Solve the decoupled pressure system using a linear solver of choice to obtain $x_P = \widetilde{A}_P^{-1} r_P$. This is the first stage preconditioning.
3. Update displacement residual r_D using newly computed pressure solution x_P.
4. Solve the displacement system using a selected linear solver to obtain $x_D = \widetilde{A}_D^{-1} r_D$. This is the second stage preconditioning.

5. Map the constraint solution to the total estimate of pressure vector.
6. Update increment residual vector.
7. Make the increment residual vector orthogonal to previous increment direction.
8. Calculate step size.
9. Update solution and residual vectors.

The 9 steps are repeated until convergence or stopping criterion is reached. Using this approach a nested iteration is formed. There are two inner iterations for pressure and displacement. The outer iteration couples flow and displacement using GCR. AMG has been implemented as a sub-preconditioner for the mainly elliptic pressure sub-block and elliptic displacement sub-block. In the abovementioned places where AMG has been implemented, we now replace the solver with CML. The algebraic characteristics of both the pressure and displacement matrices should favor CML solution. In the following section, we will briefly describe the CML method.

3.3 The Combinatorial Multilevel Method

Before describing the algorithm of CML, we first show two examples in which aggregation based and classical AMG have convergence troubles. The first matrix comes from a maximum flow in network problem (Livne 2012). The resulting matrix is highly ill-conditioned with condition number about 10^{19}. Its sparse matrix plot is provided in **Figure 3.1**. Note that all the sparse matrix plots in the paper are generated using the CSPY tool of CSparse package (Davis 2006). Zero entries are white. Entries with tiny absolute value are light orange and entries with large magnitude are black. In the midrange, it ranges from light green to deep blue.

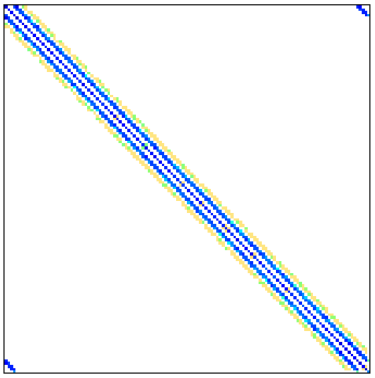

Figure 3.1 Sparse matrix plot of the first example

Table 3.1 Number of CG iterations of the first example

Method	Iterations
CML	28
AGMG	N/A
RS_AMG	N/A
ILU(0)	2726

The iteration counts of the CG accelerator are listed in **Table 3.1**. The relative residual reduction is plotted in **Figure 3.2**. N/A denotes AGMG and RS_AMG do not converge in 10000 iterations. Clearly, it can be seen that CML can readily solve this problem as opposed to AGMG and RS_AMG. The reason for this is that unlike AGMG and RS_AMG, CML is not limited by indefinite matrices.

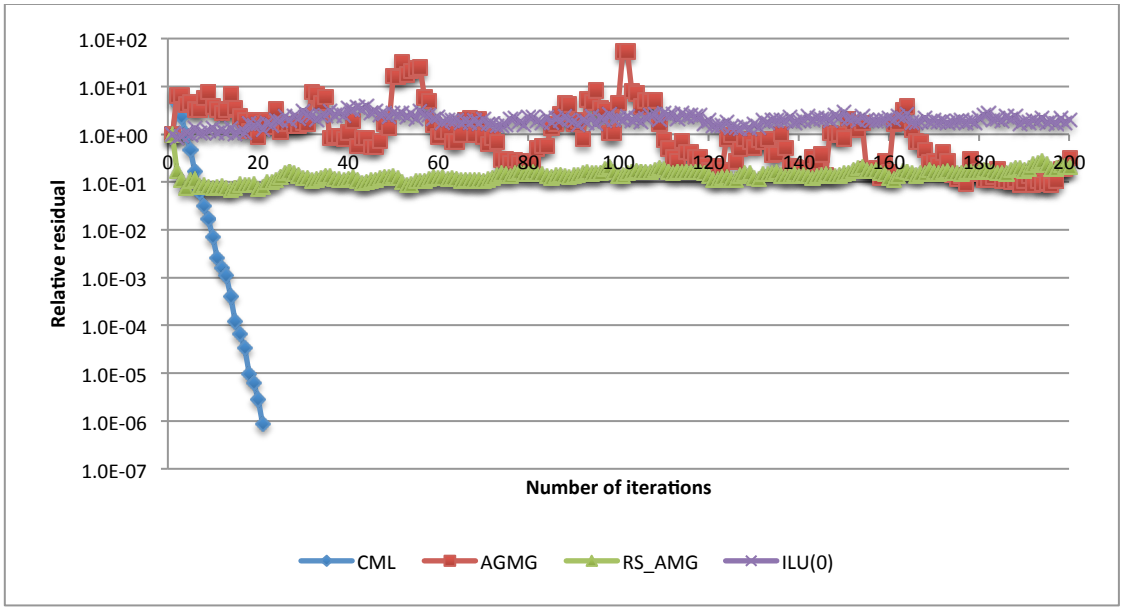

Figure 3.2 Relative residual reduction of the first example

The second example is extracted from a matrix of a reservoir simulation application (Beckner et al. 2006; Diyankov et al. 2007). The original matrix is highly unsymmetric and describes a coupled system with more than one unknown per gridblock. We convert the matrix to a symmetric matrix by extracting a connected graph of the original matrix. The resultant matrix is SPD. Its sparse plot is provided in **Figure 3.3**. As listed in **Table 3.2** and plotted in **Figure 3.4**, for this SPD matrix, CML shows the fastest convergence while AGMG exhibits convergence difficulties.

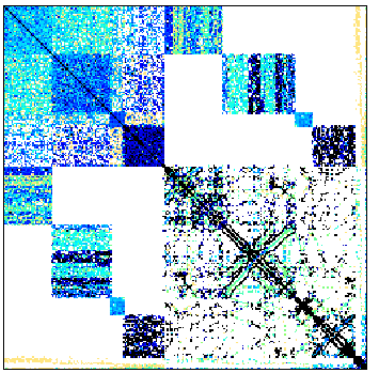

Figure 3.3 Sparse matrix plot of the second example

Table 3.2 Number of CG iterations of the second example

Method	Iterations
CML	18
AGMG	634
RS_AMG	97
ILU(0)	1187

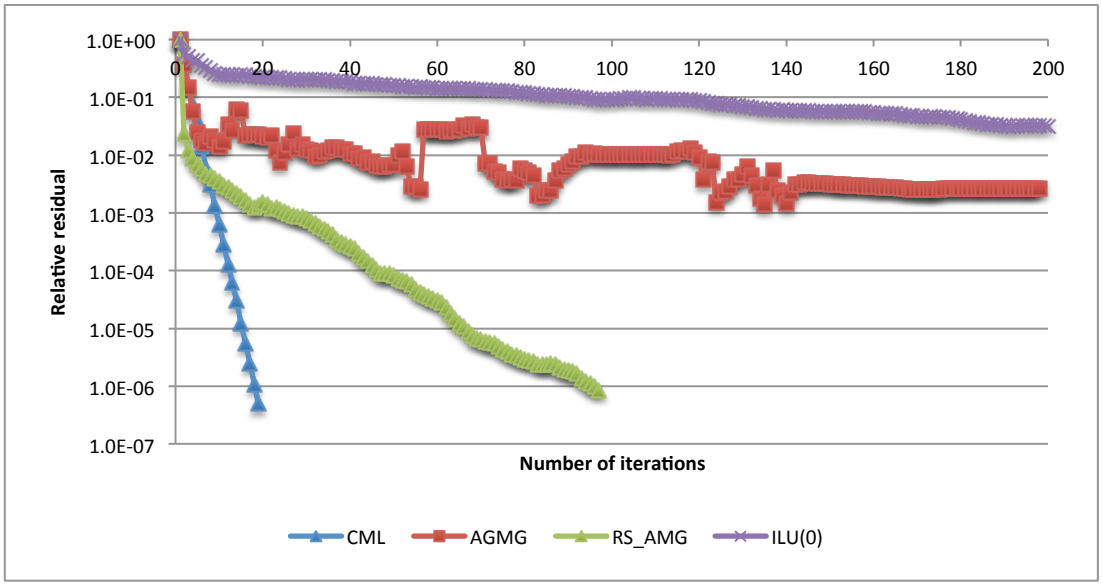

Figure 3.4 Relative residual reduction of the second example

The above excellent performance of CML can be attributed to its provable convergence properties and sound theoretical machinery for SDD matrices. In this section, we describe the underlying principles of CML. As its name suggested, CML is inspired by the popular multigrid algorithems, yet with two significantly different distinguishing features. CML features a uniquely different coarsening strategy that is faster than various AMG approaches and is easier to implement. The second feature is that CML is a truly black-box solver while AMG has several algorithmic input options that may be crucial for convergence, especially for the classic AMG (for instance, the strength parameter). Note that aggregation-based AMG has a much better black-box feature than classic AMG. Although in practice the time spent in the set-up phase is generally negligible compared to the iteration phase, such timing can reflect the efficiency of the hierarchy construction and can also suggest an easy implementation. We focus on describing the two-level approach. Extension to multilevel is straightforward. To keep the presentation concise, we simplify the algebra and only present the key ingredients of CML. Algorithmic details with proofs can be found in Koutis (2007) and Koutis et al. (2009).

To provide a quick sense of CML and how it is related to the multigrid approach, **Table 3.3** lists the two-level CML algorithm to solve $Ax = b$, where A is a laplacian matrix. It should be noted that any SDD matrix could be converted to laplacian with lightweight transformation (Gremban 1996). When there are positive off-diagonal entries, we generally can merge the positive off-diagonal entries to the diagonal. For

reservoir simulation, the decoupled pressure matrix sometime has positive off-diagonal entries though the number of rows having positive off-diagonal entries is small comparing with the matrix dimension. It can be seen that this algorithm resembles the simple form of the two-level method. In practice, we need to implement this algorithm by call it recursively. Provided in **Table 3.4** is the popular V-cycle multilevel approach. At this point, we haven't described how CML construct the hierarchy of coarse matrices. It is the constructed hierarchy of matrices with the associated restriction matrices that distinguishes CML with other variants of AMG.

Table 3.3 Two-level CML algorithm

Two-level CML
Input: laplacian A, vector b, current solution x^k $n \times m$ restriction matrix R Output: Updated solution x^{k+1} 1. // Jacobi pre-smoothing $x^k_{presmoothed} = \left(I - D^{-1}A\right)x^k + D^{-1}b$ **;** 2. // Restriction $r^k = b - Ax^k_{presmoothed}$ **;** $r^k_{coarse} = R^T r^k$; 3. // Solve using coarse level $A_{coarse} = R^T AR$; $x_{coarse} = A^{-1}_{coarse} r^k_{coarse}$; 4. // Correction $x^{k+1} = x^k_{presmoothed} + Rx_{coarse}$; 5. // Jacobi smoothing $x^{k+1}_{postsmoothed} = \left(I - D^{-1}A\right)x^{k+1} + b$

Table 3.4 V-cycle CML algorithm

V-cycle CML

Procedure CML(*level*, A_h, x_h, b_h)
if *level* = *coarsest* then
solve coarsest directly
else
1. //Jacobi smoothing
$x^k_{presmoothed} = \left(I - D^{-1}A\right)x^k + D^{-1}b$
2. // Restriction
$r^k = b - Ax^k_{presmoothed}$
$r^k_{coarse} = R^T r^k$
3. call CML(*level-1*,A_H,v_H,f_H)
4. // Correction
$x^{k+1} = x^k_{presmoothed} + Rx_{coarse}$;
5. // Jacobi smoothing
$x^{k+1}_{postsmoothed} = \left(I - D^{-1}A\right)x^{k+1} + b$
endif
endprocedure

Before providing the key ingredient of the construction of restriction matrix, let's first introduce a few definitions. It should be noted, CML is developed based on the laplacian matrix and each laplacian is associated with a corresponding graph. We build the restriction matrix by exploring the geometric properties of the underlying graph. This is reason why CML is said to bring geometric information into the algebraic operations. The following description of the construction of the restriction is based on (Koutis et al. 2011). For a laplacian matrix *A*, its corresponding graph is denoted as *G*. The total weight incident to node *v* in G is defined as

$$tw(v) = \sum_{u \in N(v)} w(u,v)$$

We then define the weighted degree of a node *v* in *G* as

$$wd(v) = \frac{tw(v)}{\max_{u \in N(v)} w(u,v)}$$

The degree of a node *v* in *G* is the number of neighbor nodes that are adjacent to *v*. *w(u,v)* is the weight connecting *u* and *v* in G. The average weighted degree of *G* is

$$awd(G) = \left(1/n\right) \sum_{v \in V} wd(v)$$

We can decompose the graph into disjoint cluster V_i using the Decompose-Graph algorithm provided in **Table 3.5**. The first step is to mark nodes *v* in *G* if $wd(v) > \alpha \cdot awd(A)$. α is a constant and set to be larger than 4. We group the marked nodes to form a set called *W*. The nodes inside *W* are then relabeled as *w*. Apparently, $W \subseteq V$. In the second step, we remove some of the edges for each $v \in V$ by only keeping the incident edge with the largest weight. As a result, we get a set *F*. In the third step, we search the nodes *w* in set *W* whose total incident weight is smaller than $tw_A(w)/awd(A)$. In the forth step, we remove the edges in *F* that are contributed by *w* found in Step 3. At the last step, we then construct disjoint clusters V_i from *F*.

Table 3.5 Decompose-Graph algorithm

Algorithm: Decompose-Graph
Input: Graph $A = (V, E, w)$ Output: Disjoint Clusters V_i 1. Find a set of nodes, W, such that $wd(v) > \alpha \cdot awd(A)$ α is a constant and >4 2. Construct $F \subset A$ by keeping the incident edge with the largest weight for each node $v \in V$ 3. Find nodes $w \in W$ such that the total weight incident to node w is smaller than $tw_A(w)/awd(A)$ 4. In F, remove the edges contributed by w found in Step 3 5. Construct disjoint clusters V_i from F

Using the Decompose-Graph algorithm, we can partition a graph G with n vertices into m disjoint clusters V_i ($i = 1, \cdots, m$). The restriction matrix R, which is of size $n \times m$, is constructed as $R_{i,j} = 1$ if vertex i is in cluster j and $R_{i,j} = 0$ if vertex i is not in cluster j.

In principle, the basic idea of this new approach is to construct a hierarchy of matrices by viewing the underlying matrix as a graph and using the discrete geometry of the graph. In this way, the CML method combines the merits of both geometric and algebraic multigrid methods, which provide strong convergence guarantees for SDD matrices.

3.4 Case Experiments

As it can be seen from the previous section, the core of the CML method is built for SDD matrices with general weights. In other words, theoretically, it only guarantees

convergence for this class of matrices. In practice, we extend CML to also handle the nearly symmetric pressure sub-block matrix that is derived from black-oil simulation. In the case experiments, we first test on an incompressible system using the SPE 10th comparative project model (Christie and Blunt 2001) whose resulting pressure sub-block matrices are SDD and not indefinite. For this case, we provide the comparison of performance and complexity of CML, AGMG, RS_AMG and ILU(0). Next, based also on the SPE 10th comparative project model, we perform an experiment using a black-oil system that generates a nearly-symmetric pressure sub-block matrix. In addition, we also provide tests on a series of matrices from unstructured gridding (Beckner et al. 2006; Diyankov et al. 2007). Finally, we test the performance of these methods on a finite element displacement sub-block matrix using a test instance from the University of Florida sparse matrix collection (Davis 1994). Since the time spent in the set-up phases of these preconditioners are negligible comparing to the iteration phases, the number of iterations of a chosen accelerator is a strong indicator of the performance of the respect preconditioner. We also introduce a notation of iteration cost by multiplying iteration counts by computational complexity to quantify the effective work consumed by each preconditioner.

3.4.1 Incompressible Oil-water System

We build a 5-spot SPE 10 problem by defining one water injector at the center and four producers at the four corners. The compressibility of oil, water and rock is neglected to make an incompressible system. Note that the four preconditioners are written using different programming languages with unknown code optimization levels.

We thus choose not to compare the elapsed time. Since the number of iterations taken to converge is strongly correlated to the time, we use the number of iterations as a more fair comparison criterion. Listed in **Table 3.6** are the number of iterations taken by CML, AGMA, RS_AMG, and ILU(0). CG is used as the accelerator. Obviously, the three variants of AMG outperform ILU(0) by orders of magnitude. More importantly, CML is clearly the winner over AGMG and RS_AMG. The relative residual reductions of the full SPE 10 model (85 layers) are plotted in **Figure 3.6**. It can be seen from **Figure 3.6** that, the relative residual of CML decreases linearly in log scale while AGMG and RS_AMG deviate from log linear reduction to some extent. It is worth mentioning that there are a few algorithm knobs that can affect the performance of RS_AMG dramatically, such as the method used to determine the strength of connection between unknowns of the underlying matrix. A careless choice may even destroy the convergence of RS_AMG.

Table 3.6 Number of Iterations for incompressible oil-water system

Method	Number of Iterations
CML	28
AGMG	43
RS_AMG	59
ILU(0)	2726

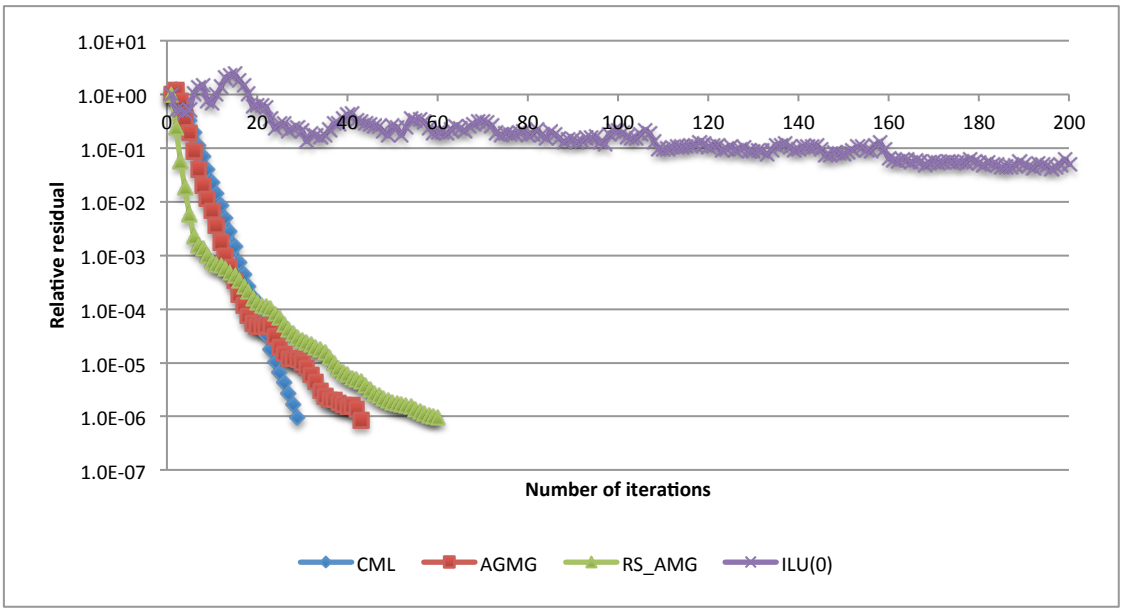

Figure 3.5 Relative residual reduction for the incompressible oil-water system

As abovementioned, aggregation-based and classic AMG can scale linearly with matrix size. To compare the scalability of CML, AGMG and RS_AMG, we perform experiments by adding the layers of the SPE 10 model. To test how these methods scale with matrix size, we record elapsed time of each method and compare the normalized time that is taken to be the time taken by the multi-layer model divided by the time taken by a single layer model for each method respectively. The comparison results are shown in **Figure 3.6**. Since ILU(0) scales badly with matrix size, the plots in **Figure 3.6** are cut to better shown the differences among CML, AGMG and RS_AMG. It can be seen from **Figure 3.6** that CML scales linearly as matrix size. Moreover, some supper linear scalability is exhibited, as the normalized time of 85 layers is 81.83. Neither AGMG nor RS_AMG shows strict linear scalability. In addition, for AGMG and RS_AMG the deviation from linear scalability seems to be enlarged when passing layer 35 (transition

from non-fluvial Tarbert formation to channelized Upper Ness formation). This is believed to be attributable to the significant weight change of the underlying matrix graph. In contrast, thanks to the proved ability to handle general weights, CML is shown to be insensitive to this change.

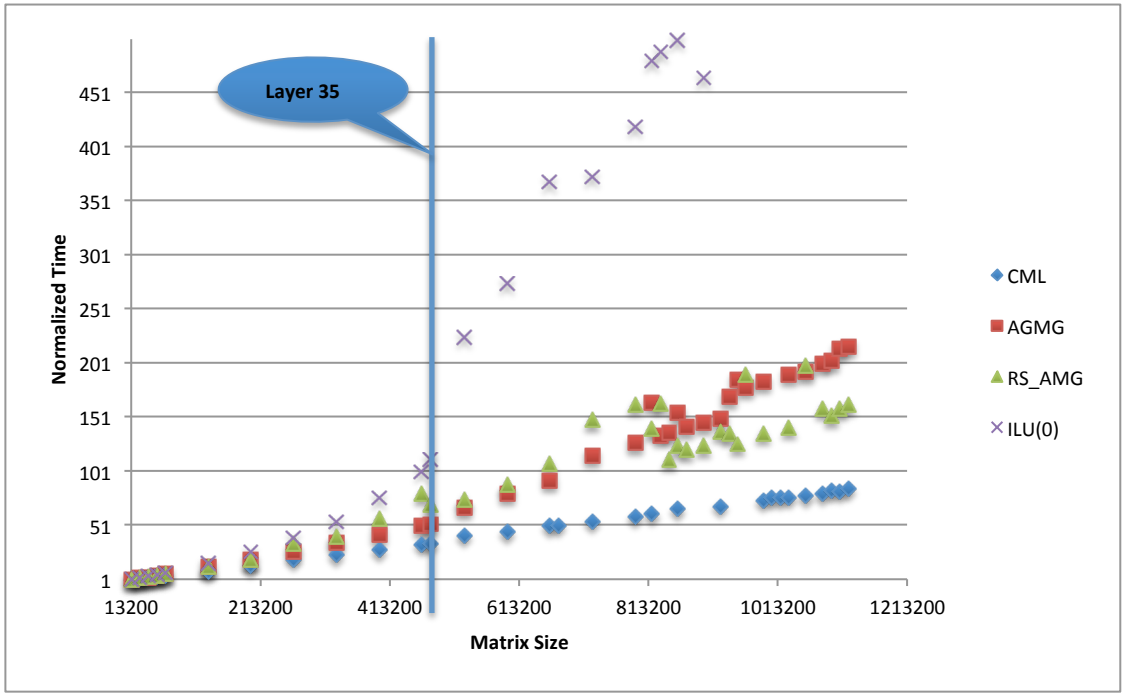

Figure 3.6 Normalized time vs. matrix size

To make the comparison complete, complexities of these methods should be analyzed. Although these variants of multilevel preconditioners significantly enhance the convergence rate, they require extra cost in each iteration compared to ILU(0). We first provide the grid complexity and operator complexity of the three multilevel methods (**Figure 3.7**, **Figure 3.8**). Grid complexity is defined as the total number of grid points of

all levels divided by the number of original fine grid points. Operator complexity is similarly defined as the total number of nonzero entries of all levels divided by the number of nonzero entries of original fine grid. Clearly, smaller grid and operator complexities are favored. As it can be seen from **Figure 3.7** and **Figure 3.8**, CML preserves the smallest grid and operator complexities among the three approaches. In addition, the grid and operator complexities of AGMG and RS_AMG show variations as matrix dimension increases while the variations are almost flat for CML. RS_AMG has the worst grid and operator complexities. RS_AMG applies a heuristic approach to mimic the grid coarsening of geometric multigrid using the connection strengths of matrix entries, while CML and AGMG use an agglomerative clustering technique. Since the number of nonzero entries determines the number of floating point operations in preconditioning, the computational complexity is directly related to grid and operator complexity.

Figure 3.9 shows the estimated computational complexity of CML, AGMG, RS_AMG, and ILU(0). Computational complexity is defined as the number of floating point operations a preconditioner consumes per iteration (normalized by the number of nonzero entries of the original fine matrix). As expected, ILU(0) has the lowest

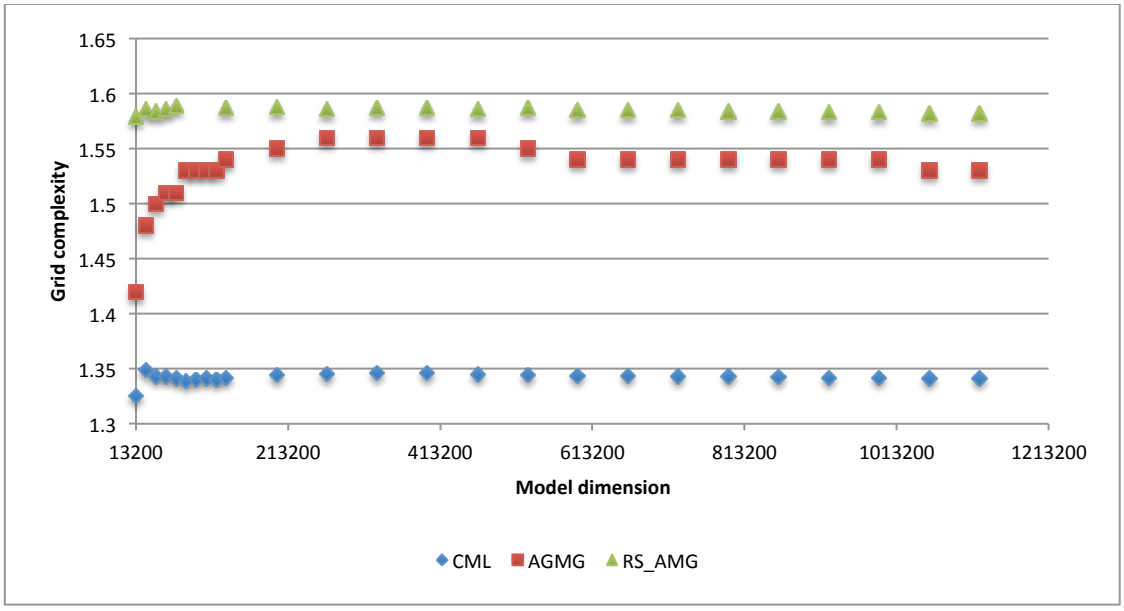

Figure 3.7 Grid complexity

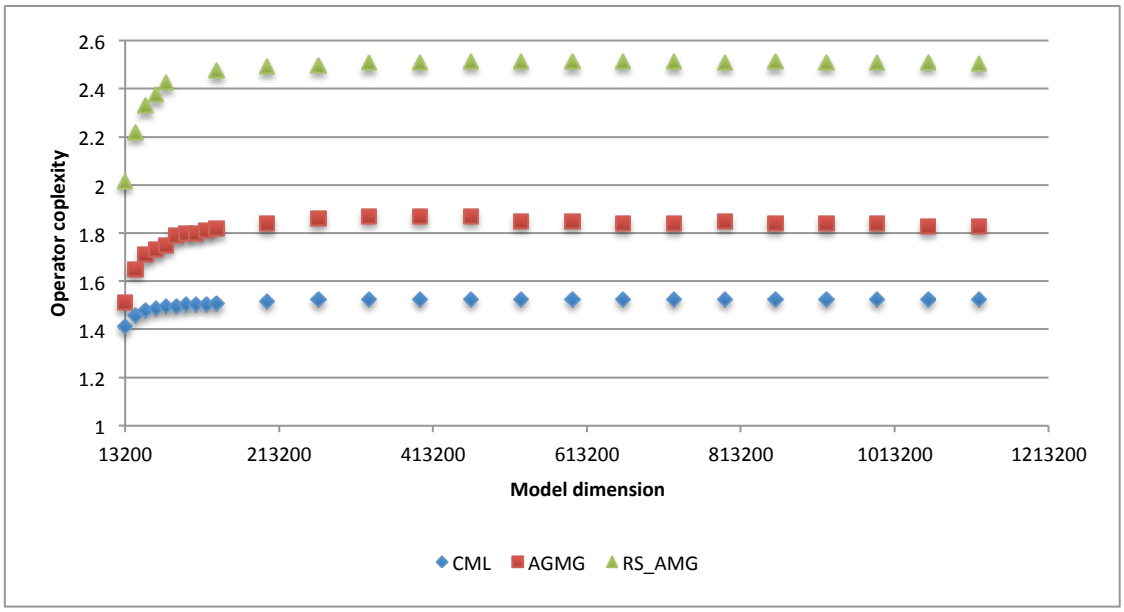

Figure 3.8 Operator complexity

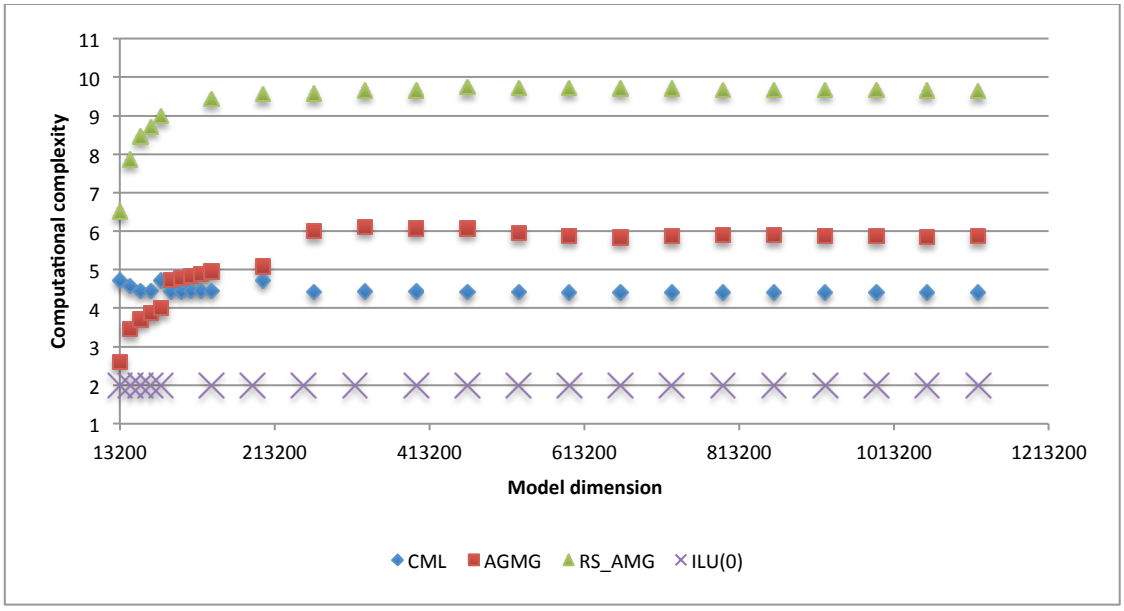

Figure 3.9 Computational complexity

Table 3.7 Iteration costs for the incompressible oil-water system

Method	Iter cost
CML	123.39
AGMG	252.41
RS_AMG	568.76
ILU(0)	5452.0

computational complexity among the 4 methods. The computational complexity of CML is lower than AGMG except when the grid dimension is smaller than about 100,000. RS_AMG has the most expensive computational complexity. Since the computational complexity is the extra work per iteration, we introduce the effective iteration counts as the performance indicator. The iteration costs of the 85-layer model are listed in **Table 3.7**. It can be seen that, the advantage of CML over AGMG and RS_AMG is further increased when these factors are taken into account.

3.4.2 Black-oil System

Since the pressure sub-block matrix is generally nearly symmetric, it is more interesting to test the performance of CML on this class of matrices. We extend the incompressible system to a black-oil system. Watts (1981) proposed an approach to build a symmetric approximation of the resulting nearly symmetric pressure matrix. He then reformulated the solution process by adding an outer iteration such that (to solve $Ax = b$):

$$
\begin{aligned}
&r = b - Ax^k \\
&while\left(\text{not converged}\right) \\
&\quad \Delta x = A_S^{-1} r; \\
&\quad x^{k+1} = x^k + \Delta x; \\
&\quad r = b - Ax^{k+1}; \\
&end
\end{aligned}
$$

where A_S is the symmetric approximation of A. Results of Watts' work indicated that 2 or 3 iterations is generally enough for convergence. The original work was attempting to use the conjugate gradient method as the accelerator for the linear solver. With the

development of non-symmetric accelerators such as GMRES or BICGSTAB a few years later (Saad and Schultz 1986; Van der Vorst 1992), the requirement for matrix symmetry was eliminated. We instead build a hierarchy of matrices using A_S and use this hierarchy as a preconditioner for GMRES. The hope is that the slight non-symmetry does not change much of the spectrum of A. Moreover, we attempt to directly build a preconditioner using the nearly symmetric A with CML. Similarly, we compare the convergence of CML, AGMG, RS_AMG and ILU(0) using full SPE 10 model. **Table 3.8** lists the iteration counts and iteration costs of each method. **Figure 3.10** provides the relative residual reduction history. CML_unsymm denotes that we apply CML directly to A while CML_symm means we apply CML to A_S. GMRES(10) is applied as the accelerator. CML still outperform AGMG, RS_AMG and ILU(0). But surprisingly, CML_unsymm has the best performance. Its residual tends more to decrease log linearly than others. The mathematical justification could not be provided at this point. A hypothesis of CML_unsymm's excellent performance is that the nearly symmetric A is still (semi-) positive definite and structure pattern of A is symmetric. In addition, support theory for preconditioning, which is the foundation of CML, might be able to be extended to more general matrices. Boman and Hendrickson (2003) discussed the extension of support theory to general matrices and Huang (2012) generalized support theory for preconditioning to non-symmetric matrices. Understanding of the performance of CML on nearly symmetric matrices and the development of a multilevel method based on support theory for slightly non-symmetric matrices clearly requires further research.

Table 3.8 Number of iterations and costs for the black oil system

Method	Number of iterations	Iter cost
CML	18	77.22
AGMG	28	106.40
RS_AMG	42	455.70
ILU(0)	539	1078.0

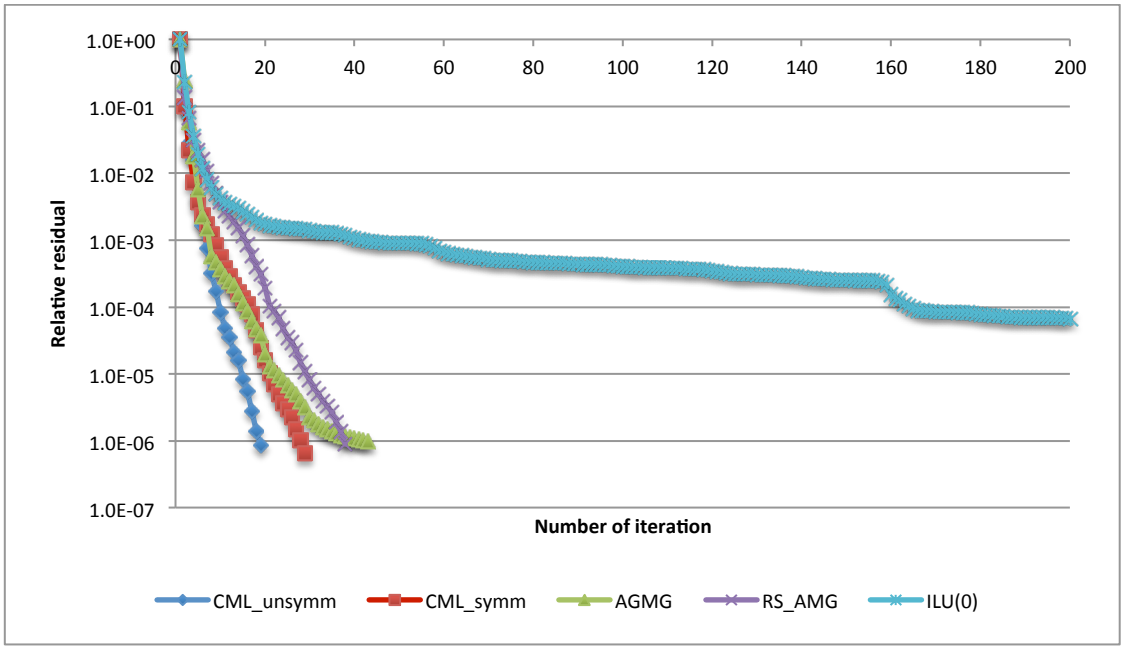

Figure 3.11 Relative residual reduction for the black oil system

In addition, compared to the incompressible system, the performance of these non-symmetric preconditioners seems to become better in this black-oil case. This can be understood by realizing that the underlying matrices become strictly diagonal dominant due to the effect of compressibility.

To assess the applicability of the solvers in unstructured reservoir simulation, we perform further experiments using a series of matrices from unstructured reservoir

models (Beckner et al. 2006; Diyankov et al. 2007). Unfortunately, there is no publicly available information of these models except the system matrices alone. We can only infer the underlying information by viewing the matrices. To apply CML, AGMG and RS_AMG, we extract the nearly symmetric pressure-like matrices since the original matrices appear to be from coupled systems. **Figure 3.11** lists the sparse matrix plots of the extracted matrices. Clearly, these were derived from unstructured gridding. **Table 3.9** shows the performance of the various solvers. It can be seen from **Table 3.9** that CML performs better for larger matrices (SBO-3). For small-sized matrices, it seems that CML is not as efficient as AGMG and is even worse than ILU(0) for SEO-1, CI-1 and CIT-1. For SBO-4 and CI-1, the iteration counts of CML and AGMG are close. The reason why CML has worse efficiency than AGMG is that the computational complexity of CML is about two times higher than AGMG. As we have seen in **Figure 3.10**, the computational complexity of AGMG is lower than CML when the matrix dimension is small and the computational complexity of CML is flat as the matrix dimension increases. This also implies that CML tends to perform better for large-scale matrices. It should be noted that for the 6 test instances, CML is applied directly to the slightly unsymmetric matrices. Hence, these results also suggest that CML is applicable for this type problem making it a promising alternative method for large-scale flow simulation.

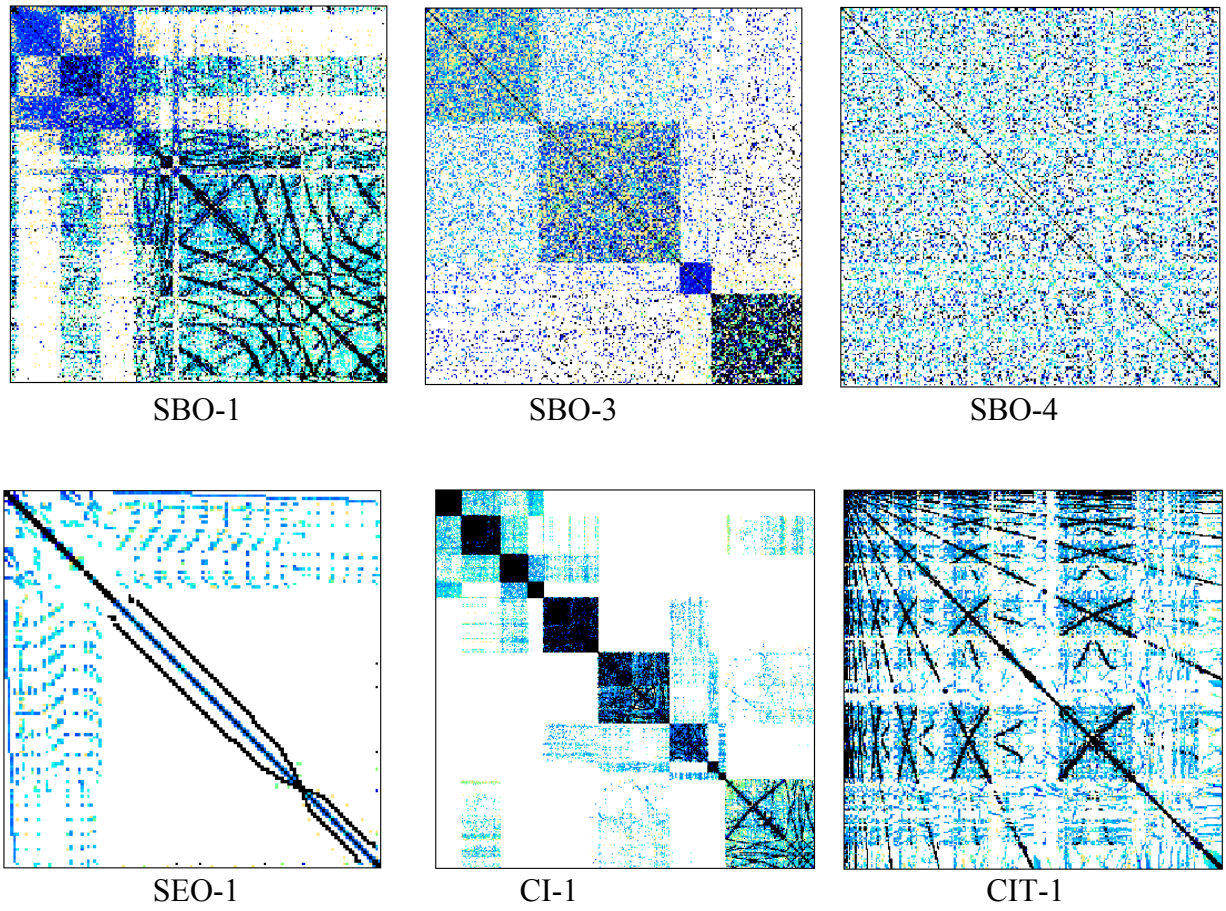

Figure 3.12 Sparse matrix plot for the unstructured example

Table 3.9 Number of iterations and costs for the unstructured example

	Size (unknown#, nonzero#)	CML		AGMG		RS_AMG		ILU(0)	
		# of Iter	Iter cost	# of Iter	Iter cost	# of Iter	Iter cost	# of Iter	Iter cost
SBO-1	(21692, 144986)	14	60.20	24	78.96	48	350.4	81	162
SBO-3	(2165051,1849317)	14	56.20	40	136.9	49	401.3	326	652
SBO-4	(61956, 486010)	28	114.2	25	67.50	68	685.4	175	350
SEO-1	(21498, 185700)	10	28.40	6	9.720	4	35.44	6	12
CI-1	(13500, 88860)	17	65.96	13	16.25	5	52.75	11	22
CIT-1	(4359, 28041)	26	112.1	24	85.44	18	144.7	32	64

3.4.3 Displacement Computation in Coupled Flow and Geomechanics

The underlying matrix for displacement computation is symmetric. Hence, we can directly apply CML as preconditioner if the displacement matrix is diagonal dominant. If diagonal dominance cannot be preserved, neither CML nor AMG guarantees convergence. Since there is no well-documented benchmark problem for coupled flow and geomechanics, we instead test directly on a benchmark matrix from the University of Florida sparse matrix collection (Davis 1994). The test case comes from a coupled flow and geomechanical study of CO_2 sequestration in a depleted gas reservoir (Ferronato et al. 2010).

The 3D view of the depleted gas reservoir and its 2D planar view are shown in **Figure 3.12**. The finite element discretization has about 250,000 nodes and more than 1,250,000 elements. A number of local and regional faults exist in this reservoir, which is shown as solid line the **Figure 3.12**. The data results from the application of commercial reservoir simulator for flow simulation while the study the fault activation and ground deformation is derived from a geomechanical simulation. The resulting sparse matrix is plotted in **Figure 3.13**. Note that it has been reordered by the reverse Cuthill-McKee (RCM) algorithm.

The iteration counts and costs are listed in **Table 3.10** and iteration reduction histories are plotted in **Figure 3.14**. Obviously, CML is again the winner among the four approaches. In addition, similar to the incompressible system case, the residual reduction of CML decreases log linearly. For this case, AGMG exhibits a poor performance and is even worse than the plain ILU(0).

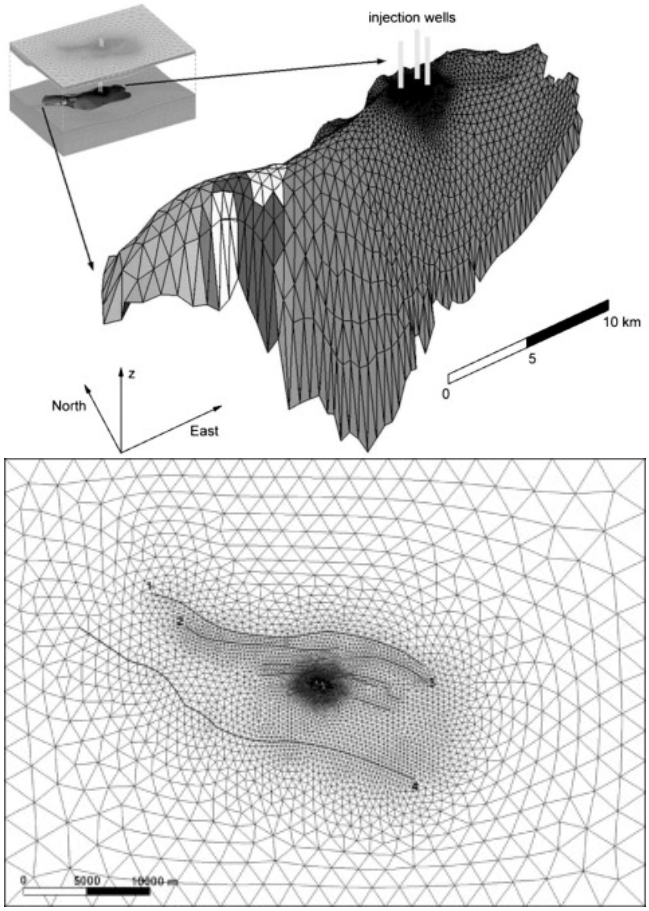

Figure 3.12 3D (upper) and 2D (lower) view of the depleted gas reservoir for CO_2 sequestration (From Ferronato et al. 2010)

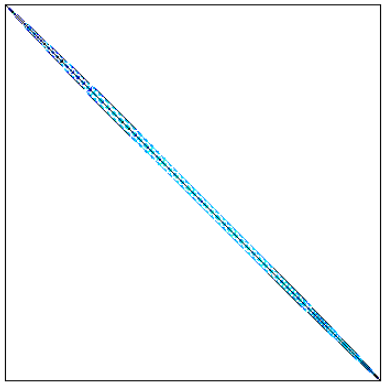

Figure 3.13 Sparse matrix plot of displacement matrix

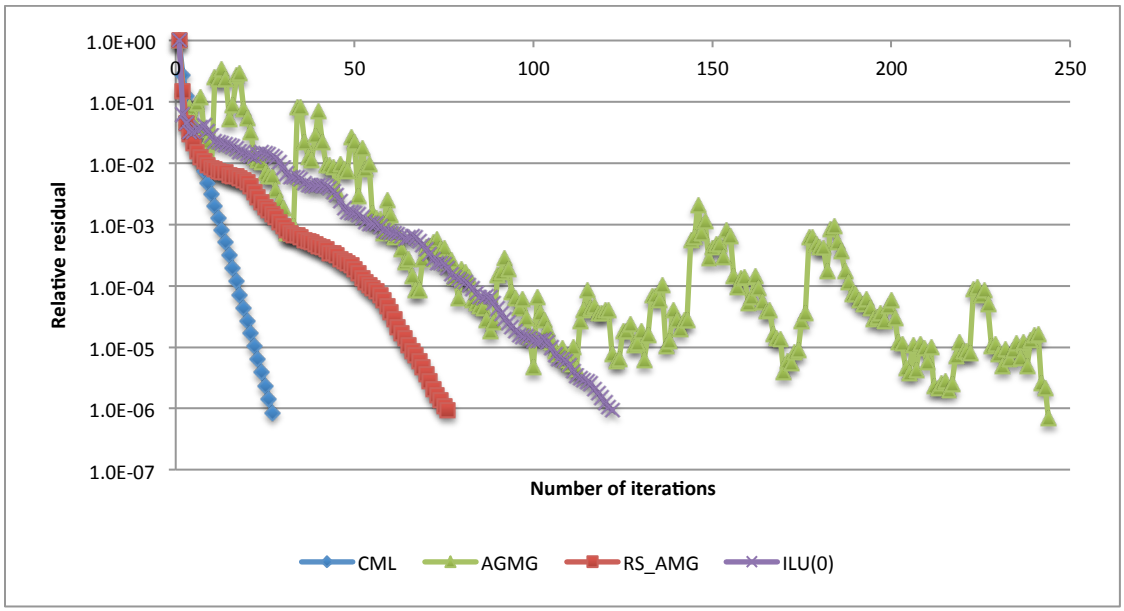

Figure 3.14 Relative residual reduction for the displacement example

Table 3.10 Number of iterations and costs for the displacement example

Method	Number of iterations	Iter cost
CML	26	78.52
AGMG	243	634.23
RS_AMG	75	318.75
ILU(0)	121	242.00

3.5 Conclusions

In summary, this paper introduces a new multilevel preconditioner, CML, to reservoir simulation and coupled geomechanics. CML is rooted in support theory and Steiner preconditioner and is integrated with the popular multilevel approach. The resulting algorithm has a unique matrix hierarchy building machine that tends to bring geometry information back into the algebraic operations thus introduces proven strong convergence guarantees for SDD matrices with general weights.

We implement CML into the multistage preconditioning framework for reservoir simulation and coupled geomechanics. Specifically, CML is applied for pressure and poroelastic displacement preconditioning. We perform experiments on a series of examples and compare the performance of CML with AGMG, RS_AMG, and ILU(0). From the results, we have the following findings:

1. CML has better scalability than AGMG and RS_AMG. Through the incompressible system example, we show only CML can scale strictly linearly using the SPE 10 model.
2. CML has lower grid and operator complexity than AGMG and RS_AMG, which reveals it has better hierarchy building machinery.

3. Although without theoretical justification yet, it is shown that CML can be directly applied to nearly symmetric pressure-like matrices. Its performance is superior to AGMG and RS_AMG for large-scale matrices. Handling nearly symmetric matrices robustly and efficiently is a prerequisite for pressure preconditioning in reservoir simulation application. CML is shown to be capable in this aspect through our experiments.

This preliminary study shows CML is a promising alternative for pressure and displacement preconditioning in reservoir simulation and coupled geomechanics, especially for large-scale models. A relative unpleasant aspect of CML is, however, the theoretical support for nearly symmetric matrices is not available yet. Although the current algorithm is shown to work with pressure-like matrices in reservoir simulation and has better performance than AGMG and RS_AMG for large models, we need to justify or develop new algorithms based on CML. Indeed, one of the purposes of this paper is to bring attention to this new way of pressure and displacement preconditioning and to serve as an introduction for further research in this area.

CHAPTER IV

COMPOSITIONAL MODELING OF TIGHT OIL USING DYNAMIC NANOPORE PROPERTIES*

A typical tight oil reservoir such as the Bakken has matrix pore sizes ranging from 10 nm to 50 nm. At such small scales the confined hydrocarbon phase behavior deviates from bulk measurements due to the effect of capillary pressure. In addition, compaction of pore space can bring about order of magnitude changes for tight oil formation properties during pressure depletion further exacerbating these deviations. Without considering these facts a conventional reservoir simulator will likely not be able to explain the inconsistent produced GOR observed in the field compared to simulated results. The effect of these inaccuracies on ultimate recovery estimation can be devastating to the underlying economics.

This chapter presents a compositional tight oil simulator that rigorously models pressure dependent nanopore-impacted rock and fluid properties, such as suppression of bubble point pressure, decrease of liquid density, and reduction of oil viscosity as well as their interactions with pore space compaction. The cubic Peng-Robinson equation of state is used for phase behavior calculations. Capillary pressure is evaluated by standard Leverett J-function for porous media. Modifications to the stability test and two-phase

---

split flash calculation algorithms are provided to consider the capillarity effect on vapor-liquid equilibrium.

The simulator can capture the pressure-dependent impact of the nanopore structure on rock and fluid properties. As a result, the problem of inconsistent GOR is resolved and the history matching process is greatly facilitated. It is shown that inclusion of these enhanced physics in the simulation will lead to significant improvements in field operation decision making and greatly enhance the reliability of recovery predictions.

4.1 Introduction

The recent advances in massive hydraulic fracturing techniques have enabled the oil industry to economically extract hydrocarbon from ultra-tight, unconventional resources, such as shale gas, liquid rich shale and tight oil. The success in North America has stimulated the development of unconventional plays worldwide. For example, a marine shale play in southern China has showed large potential and attracted great attention (Wei et al. 2012; 2013a, b). However, despite the great success and potential, the understanding of fluid flow mechanism in shale and properties in confined pore space is still poor. The flow mechanism in the shale matrix is complicated by organic and inorganic portions of the matrix with distinct wettabilities. Yan et al. (2013 a, b, c, d) proposed a micro-model to model single-phase gas and two-phase gas-water flow in shale matrix block by considering different flow mechanisms in organic and inorganic nanopores and upscaled the single-phase gas flow to well-scale modeling via the apparent permeability approach. On the other hand, the fluid properties in the confined nanopore space deviate from the corresponding bulk measurements in which

zero vapor-liquid interface curvature is assumed. This assumption is generally held when the vapor-liquid equilibrium takes in place in PVT cells. But, when the fluid is confined into pore spaces of nano-size, the significant interfacial curvature may cause a large capillary pressure difference between liquid and vapor phases. The effect of capillary pressure on vapor-liquid equilibrium is not new to the oil industry. A number of researchers have conducted both experimental and theoretical investigations with general conclusions that capillarity effect on vapor-liquid equilibrium is negligible for conventional reservoirs (Leverett 1941; Sigmund et al. 1973, 1982; Shapiro and Stenby 1997; Shaprio et al. 2000). Perhaps due to this reason, essentially all the current commercial simulators assume no pressure difference between vapor and liquid phases during flash calculations.

However, ignoring capillarity in vapor-liquid equilibrium might not be a valid assumption for unconventional reservoirs. A typical tight oil reservoir such as the Bakken has matrix pore size ranging from 10 nm to 50 nm. At such small scales, the confined hydrocarbon phase behavior is believed to deviate from bulk measurements due to the extra capillarity effect. Rock wettability is another factor to consider when dealing with capillary pressures. Wang et al. (2012) performed a wettability survey of the Bakken formation and reported that the Bakken formation is oil-wet. A series of studies of confined fluid properties for a Bakken field shows that for this type of reservoirs, the bubble point pressure can be suppressed significantly by considering the capillary effect (Nojabaei et al. 2012; Honarpour et al. 2012; Pang et al. 2012; Du and Chu 2012; Chu et al. 2012). In addition, compaction of pore space can bring about order of magnitude

changes for tight oil formation properties during pressure depletion further exacerbating these estimates. In their approaches, confined PVT tables are constructed from a separate modified flash calculation program and applied as inputs using a commercial reservoir simulator.

It has been known that the properties of petroleum fluids and reservoir rock are closely related to the effect of capillary pressure on vapor-liquid equilibrium. However, on the other hand, a standard and reliable measurement of confined fluid properties in ultra-tight rocks is still challenging and not available (Du and Chu 2012). In this sense, the findings and conclusions in this topic are still only supported by theoretical derivation or hypotheses. As quoted "The only concept assured is that the confined PVT properties are substantially different from the corresponding bulk properties and such variations have significant impact on well performance and ultimate recovery in unconventional reservoirs" (Du and Chu 2012).

Reservoir models containing hydraulic fractured wells are needed to model production behavior and perform recovery predictions. Such models are complicated by massive hydraulic fractures. It is natural to realize that this kind of system contains fluid properties with confined and unconfined effects, which need to be explicitly modeled. Besides, there is a contradictive effect for the rock compaction. Rock compaction makes the confinement greater, which will increase driving energy and mobility by decreasing viscosity. But it also reduces the permeability that will reduce the mobility. These factors must be considered when conducting reservoir studies for tight reservoirs.

This chapter incorporates the extended vapor-liquid equilibrium calculations into a fully compositional commercial simulator. The pore space can be dynamically updated during pressure depletion via rock compaction tables. In this way, a more rigorous treatment is included to model the combined effects.

4.2 Assumptions

Since we deal with the situation of phase behavior in confined space of nano size, (10 nm – 50 nm), we need to examine the applicability conventional thermodynamics formulation in petroleum fluid properties. The conventional formulation is based on a bulk representation of fluid, or more precisely, homogenous fluid, of which the average particle density is constant. When the fluids are in confined space, such as nanopores, the wall fluid interactions will bring significant effects on the fluid structure. The fluid in confined space may become inhomogeneous, of which the average particle density varies spatially (Evans, 2009). Based on Firoozabadi (2013), the boundary between homogenous and inhomogeneous fluid is about 10 nm. For pore size greater than 10 nm, it is appropriate to assume the fluid is still homogeneous. Since the smallest pore size we deal with for Bakken reservoir is 10 nm, the development in this chapter is based on the conventional thermodynamics for hydrocarbon reservoirs.

It has been mentioned that wettability has great effect on the effect of capillary pressure on vapor-liquid equilibrium and Bakken reservoir is oil wet. In this study, we further assume the rock surface is completely oil wet with contact angle being zero.

4.3 Approach

In this section, we provide the model of vapor-liquid equilibrium with capillarity effect, extended vapor-liquid flash calculation with implementations and evaluation of capillary pressure for tight oil reservoirs.

4.3.1 Capillarity Effect on Vapor-Liquid Equilibrium (VLE)

The fundamental of capillarity effect on vapor-liquid equilibrium (VLE) is the separation of two multicomponent phases by a curvature interface. Such effect can be readily revealed by **Eq. 4.1** and **Eq. 4.2** (the equality of the chemical potentials in the liquid and vapor phases). P^L and P^V are the phase pressures of liquid and vapor phases, respectively. P_c is the capillary pressure between them. μ_i is the chemical potential of component i at the respective temperature, phase pressure and mole fraction. m is the number of components in the system. Note that temperature is generally not affected by capillary pressure. Capillary effected VLE presents in the porous petroleum reservoirs, though it is generally ignored because this effect is negligible for conventional reservoirs.

$$P^V - P^L = P_c \tag{4.1}$$

$$\mu_i^L\left(T, P^L, x\right) = \mu_i^V\left(T, P^V, y\right) \quad i = 1,2,\ldots,m \tag{4.2}$$

Eq. 4.1 and **Eq. 4.2** provide the full constituting system for capillarity VLE. Fundamentally, the magnitude of capillary pressure is determined by the geometry of the capillary system and the wettability of medium surfaces. It should be noted that this constituting system is established based on continuous bulk vapor and liquid phases,

which are separated by a curvature interface. However, the vapor and liquid phases are not necessarily continuous in porous rocks, especially for ultra-tight reservoirs, such as Bakken. Nevertheless, it can be shown that the pressures and compositions in different isolated vapor or liquid regions are equal when the system is at equilibrium (Bedrikovetsky 1993). The analysis is based on the assumption that the fluid is homogenous, which is the same assumption that has been indicated in 4.2.

4.3.1.1 Extended VLE Flash Calculation

The VLE flash calculation implemented in compositional simulators generally involves a stability test and a two-phase split calculation. The stability test is first performed to test if a single phase is stable. Only under the circumstance that the single phase is tested to be unstable, the two-phase split calculation then will be performed. Michelsen (1982 a, b) provides the algorithm details and implementation practices for isothermal stability test and the two-phase split calculation. However, the algorithm and implementation is developed and designed assuming the vapor and liquid phase pressures are equal, i.e. no capillary pressure. Essentially, the flash calculation in all current commercial simulators applies this assumption. In the following, the classic stability test and two-phase split algorithm is extended to consider the capillary pressure effect.

4.3.1.1.1 Stability test using Gibbs free energy approach

This section shows how the standard stability test based on tangent plane distance analysis can be extended to consider the capillarity effect. For the original system to be stable, **Eq. 4.3** should hold. $f_i(y)$is the fugacity of the incipient phase and

$f_i(z)$ is the fugacity of the original system. If the original system is liquid, then the incipient phase is vapor and vice versa. The detailed derivations to obtain **Eq. 4.3** can be found in Michelsen (1982 b).

$$\sum_i^m y_i\left[\ln f_i(y) - \ln f_i(z)\right] \geq 0 \tag{4.3}$$

Since

$$f_i(z) = z_i \varphi_i(z) P^L \tag{4.4}$$

$$f_i(y) = y_i \varphi_i(y) P^V \tag{4.5}$$

where φ_i is the fugacity coefficient.

After substituting **Eq. 4.4** and **Eq. 4.5** into **Eq. 4.3**, we then have **Eq. 4.6**.

$$\begin{aligned} &\sum_i^M y_i\left[\ln\left(y_i \varphi_i(y) P^V\right) - \ln\left(z_i \varphi_i(z) P^L\right)\right] \\ &= \sum_i^M y_i\left[\ln y_i + \ln \varphi_i(y) - \ln z_i - \ln \varphi_i(z) + \left(\ln P^V - \ln P^L\right)\right] \geq 0 \end{aligned} \tag{4.6}$$

Eq. 4.6 shows that the capillary term ($\ln P^V - \ln P^L$) is naturally incorporated into the stability test. This term is normally ignored for conventional compositional simulation because of the fact that the capillary pressure between vapor and liquid phases is small. But, when the pore spaces are confined into nano scale, the capillary pressure should not be ignored.

In implementation, it is more convenient to let

$$h_i = \ln z_i + \ln \varphi_i(z) + \ln P^L \tag{4.7}$$

This is because that **Eq. 4.7** is independent of y and can be pre-computed. Let $Y_i = e^{-k} y_i$,

where $k = \ln y_i + \ln \varphi_i(y) - h_i + \ln P^V$. Then we have

$$\ln Y_i + \ln \varphi_i(y) - h_i + \ln P^V = 0 \tag{4.8}$$

Eq. 4.8 is the final form used to test stability. The fugacity coefficient is calculated using the cubic Peng-Robinson equation of state. Successive substitution and/or the Newton-Raphson method can be used to solve this nonlinear equation. The details about the derivations and solution method are provided in Appendix I.

4.3.1.1.2 VLE two-phase split calculation

The VLE two-phase split calculation is based on equality of chemical potentials or fugacities and mass balance (**Eq. 4.9 – 4.12** and **Eq. 4.1 – 4.2**). F is number of moles of original system or feed. L and V are the number of moles of liquid and vapor phases, respectively. x_i and y_i are the mole fraction of liquid and vapor phases. z_i is the mole fraction of the feed phase. To solve this set of equations, we need another mass balance constraint, **Eq. 4.12**, where K_i is the equilibrium ratio or K-value, defined as in **Eq. 4.13**. **Eq. 4.12** is called Rachford-Rice equation.

$$Fz_i = x_i L + y_i V \tag{4.9}$$

$$\sum_i^m x_i = 1 \tag{4.10}$$

$$\sum_i^m y_i = 1 \tag{4.11}$$

$$\sum_{i=1}^m \frac{(K_i - 1)z_i}{1 + \alpha(K_i - 1)} = 0 \tag{4.12}$$

$$K_i = \frac{y_i}{x_i} = \frac{\varphi_i^L P^L}{\varphi_i^V P^V} \tag{4.13}$$

It can be seen from **Eq. 4.13** that the capillarity comes in place in terms of the modified K-value. Note that conventionally the K-value is evaluated as $K_i = \varphi_i^L / \varphi_i^V$ by assuming the liquid and vapor pressures are equal. This set of equations can be solved using successive substitution and/or the Newton-Raphson method. The details about the derivations and solution method are provided in Appendix II.

4.3.1.2 Evaluation of Capillary Pressure for Tight Porous Media

The previous two sections provide the extended VLE flash calculation considering the capillarity effect. To complete the solution process of stability test and VLE split calculation, we need to evaluate the capillary pressure. Capillary pressure can be evaluated by the well-known Young-Laplace equation (**Eq. 4.14**) for a sufficiently narrow tube. Nojabaei et al. (2012), Du and Chu (2012), Chu et al. (2012), Honarpour et al. (2012) and Pang et al. (2012) applied this approach to calculate the capillary pressures.

$$P_c = \frac{2\sigma \cos\theta}{r} \tag{4.14}$$

Noted, Nojabaei et al. (2012) points out that, for tight oil rock, the capillary pressure computed using Young-Laplace equation is much less than actual measurement because of the very low interfacial tension value calculated by Macleod-Sugden correlation (Pederson 2007). Firoozabadi (2013) also points out that interfacial tension becomes function of pore size when the pore is reduced to nano size and the Macleod-Sugden

correlation is not applicable anymore. This finding reveals the importance of having reliable capillary pressure and /or interfacial tension measurements for shale/tight rocks. Since the system dealt with is saturated porous rock, it is more reasonably to apply the Leverett J-function approach (**Eq. 4.15**), which is based on measured reference capillary pressures.

$$J(S) = \frac{P_c(S)\sqrt{k/\phi}}{\sigma \cos\theta} \tag{4.15}$$

During the iteration process of stability test and VLE split calculations, capillary pressure values are looked up using the saturation results of the previous iteration.

Although **Eq. 4.15** is derived strictly only for the ideal case, it is commonly applied to other types of rocks. In this paper, it is assumed that this standard approach also applies to ultra-tight reservoir rocks. The dependence or scaling factor $J(S)$ is supposed to be known for a particular type of rock, which comes from lab measurement. Unfortunately there are no well-documented capillary pressure measurements for the Bakken reservoir yet.

In this study, the porosity of the Bakken reservoir is fixed as 0.06. Based on Kozeny-Carman equation (Kozeny 1927; Carman 1937) and correlations from Nelson (1994), the corresponding permeabilities and pore radius is provided in **Table 4.1**. Based on the capillary pressure data of a similar rock from Crain's petrophysical handbook, the corresponding capillary pressures are listed in **Table 4.1** also. Note that, the capillary pressures are obtained from various correlations without any calibration for Bakken rock. However, these values should be close to real values, at least based on hypothesis.

Table 4.1 Pore radius, permeability and capillary pressure

Pore Raduis (nm)	Permeability (md)	Capillary Pressure (psi)
50	0.0070	102.20
40	0.0046	127.75
30	0.0027	170.33
20	0.0012	255.50
10	0.0003	511.00

4.3.2 Dynamic Compaction of Nanopores

It is natural to expect that, as the pore space being compacted during pressure depletion, the impact of pore size on the fluid properties becomes more significant. By considering pore size reduction due to reservoir depletion, the reservoir is likely to experience even more reduction in bubble point pressure throughout the life of the reservoir. Such further reduction in bubble point pressure will keep the fluid in single-phase oil phase with reduced viscosity and density and compressibility, which will favor the driving energy and flow capacity. On the other hand, the compaction will reduce the permeability of the rock, which of course will decrease the mobility. Thus, compaction has two contradictory effects. And the combined effect will be determined by rigorous compositional simulation with compaction. Dynamic rock compaction generally can be incorporated into a reservoir simulator via rock compaction tables. A table look-up approach is performed to obtain permeability reduction ratios when the pressure is updated. For the Bakken reservoir in this study, the rock compaction table used is listed in **Table 4.2**.

Table 4.2 Rock compaction table of Bakken

Pressure Change (psi)	Permeability Reduction Ratio
-5180	0.489
-4450	0.500
-3700	0.511
-2960	0.532
-2220	0.588
-1480	0.675
-740	0.791
0	1.0

4.4 Results

4.4.1 Confined Phase Behavior

This section provides the results of confined phase behavior of Bakken oil. The compositional data of Bakken oil are listed in **Table 4.3** and **4.4** (Nojabaei et al. 2012). At the reservoir temperature around 240 °F, Bakken oil resides in the black oil region. Hence, the flash calculation is simplified to the case that the original or feed system is single-phase oil and the saturation pressure is the bubble point pressure. **Figure 4.1** shows the bubble point pressure lines of Bakken oil with capillarity effect at different pore space sizes, ranging from 10 nm to 50 nm. The bubble point pressures are evaluated using rigorous stability tests. Capillary pressures are calculated using the Young-Laplace equation, **Eq. 4.14**. As mentioned in the previous section, this approach could underestimate the capillary pressure to a large extent because the inaccurate interfacial tension. Using this approach, for a 10 nm pore size, the suppression of bubble point pressure is about 140 psi, which is much lower than suspected initially. For porous rock,

the Leverett J-function approach should be more appropriate to evaluate capillary pressures than Young-Laplace equation. Besides, this approach provides a way to calibrate against measurements of particular rock. In the following, the Leverett J-function approach is applied for all cases. **Table 4.5** provides the bubble point pressures calculated using the Young-Laplace equation and the Leverett J-function. We can see there are significant differences between the two approaches. It also suggests the importance of having capillary pressure data of good quality for the Bakken reservoir.

Table 4.3 Bakken oil composition data

Component	Mole Fraction	Critical Pressure (psia)	Critical Temperature (°R)	Acentric Factor	Mole Weight	Parachor
C1	0.36736	667.80	343.04	0.0130	16.04	74.8
C2	0.14885	707.80	549.76	0.0986	30.07	107.7
C3	0.09334	616.30	665.68	0.1524	44.10	151.9
C4	0.05751	550.70	765.32	0.2010	58.12	189.6
C5-C6	0.06406	461.29	875.48	0.2684	78.30	250.2
C7-C12	0.15854	363.34	1053.25	0.4291	120.56	350.2
C13-C21	0.07330	249.61	1332.10	0.7203	220.72	590.2
C22-C80	0.03704	190.12	1844.49	1.0159	443.518	1216.8

Table 4.4 Bakken oil binary interaction table

	C1	C2	C3	C4	C5-C6	C7-C12	C13-C21	C22-C80
C1	0	0.005	0.0035	0.0035	0.0037	0.0033	0.0033	0.0033
C2	0.0050	0	0.0031	0.0031	0.0031	0.0026	0.0026	0.0026
C3	0.0035	0.0031	0	0	0	0	0	0
C4	0.0035	0.0031	0	0	0	0	0	0
C5-C6	0.0037	0.0031	0	0	0	0	0	0
C7-C12	0.0033	0.0026	0	0	0	0	0	0
C13-C21	0.0033	0.0026	0	0	0	0	0	0
C22-C80	0.0033	0.0026	0	0	0	0	0	0

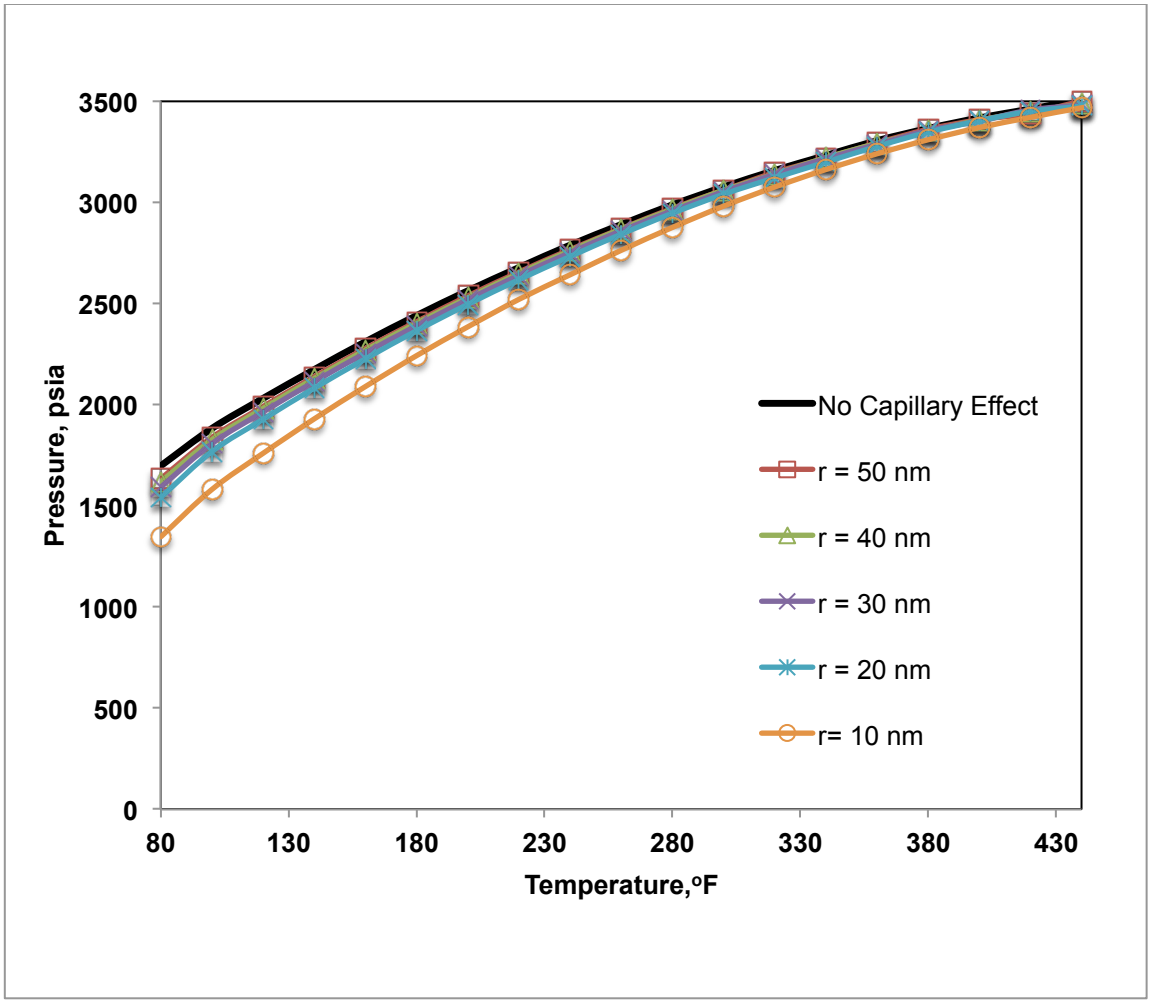

Figure 4.1 Bubble point pressure lines of Bakken oil using Young-Laplace equation

Table 4.5: Bubble point pressure of Bakken oil at 240 °F

Pore Radius (nm)	Bubble Point Pressure (psi)	
	Young-Laplace	Leverett J-Function
50	2766 psi	2512 psi
40	2761 psi	2450 psi
30	2751 psi	2345 psi
20	2732 psi	2145 psi
10	2641 psi	1588 psi

The physical implication of bubble point pressure suppression is that more light components remain in the oil phase compared to a system having unsuppressed bubble point pressure at the same temperature and pressure. Apparently, this in turn reduces the viscosity and density. **Table 4.6** provides the confined viscosity and density at 240 °F and 1500 psia. The Lorenz-Bray-Clark correlation is used for the viscosity calculation. Clearly, viscosity and density are reduced as the pore space is confined from 50 nm to 10 nm. The implication is that confinement increases the driving energy and flow capacity of the tight oil reservoir, which favors the extraction of more liquid. Otherwise, the light component will easily escape from the oil phase leave the more valuable but heavier components (C6-C12) underground. To evaluate the effect of confined fluid properties on production behavior and reservoir recovery, reservoir simulation is conducted using a fully compositional commercial simulator with extended VLE flash calculation routines.

Table 4.6: Confined fluid properties of Bakken oil at 240 °F and 1500 psia

	Bubble point Pressure (psi)	Viscosity (cp)	Density (lb/ft3)
No Capillary Effect	2788	0.41	39.17
50 nm	2512	0.363	38.5
40 nm	2450	0.351	38.35
30 nm	2345	0.331	38.04
20 nm	2145	0.29	37.3
10 nm	1588	0.18	34.4

4.4.2 Reservoir Simulation

A fully compositional commercial simulator (Dean and Lo 1988; Tang and Zick 1993; Fleming 2012) is extended to accommodate the extended VLE flash calculation. The aim is to rigorously model the effect of capillarity influenced VLE on production behavior and recovery prediction, which also including the effect of dynamic reservoir compaction. This investigation may be helpful in improving the understanding of abnormal production behavior observed in the Bakken field, such as long-lasting, relatively constant producing GOR even when the pressure near well has dropped below bubble point pressure (measured in lab). The reality is further complicated by multistage hydraulic fractured wells. Fluid is only confined in the tight matrix while being unconfined in fractures. The whole system will have different compositional models for matrix and fractures. Fortunately, the extended VLE flash approach can model the whole system in a uniform fashion. The capillary pressure for the grid representing fractures will be very small since the porosities and permeabilities for these grids are much higher than matrix grids. Thus, the fluid properties in fractures can be maintained unconfined, while fluid in matrix will be confined.

4.4.2.1 1D Core Size Model

We first provide an example using a 1D core size model, with 1 grid in the X and Z direction and 50 grids in the Y direction. The model has dimension of 0.5 ft in the X and Z direction and 3.28 ft in the Y direction. The model contains no fractures with homogenous initial permeability (0.002 md) and porosity (0.06). The reservoir temperature is 240 °F and initial pressure is 6840 psi. A sink is assigned to the first grid. Bottom-hole pressure is constrained at1500 psi. Initially, production is controlled by oil flow rate. Three scenarios are modeled: 1) no capillary pressure effect on VLE; 2) with capillary effect on VLE but without reservoir compaction; 3) with capillary effect on VLE and reservoir compaction. The production responses are plotted in **Figure 4.2** and **Figure 4.3**. **Figure 4.2** provides the cumulative oil production along with pressure decline and **Figure 4.3** provides the producing GOR along with pressure decline. Clearly, cumulative oil productions of cases considering the capillarity effected VLE are higher than the case without considering the capillarity effect. It also reveals that although reservoir compaction makes the oil phase thinner and bubble point pressure lower, its cumulative oil production is less than the case considering only the capillarity effect. This should be attributed to the fact that the reduction of mobility due to reduction of permeability offsets the increase of mobility due to reduction of viscosity. It also suggests that reservoir compaction should be considered in every reservoir simulation study for tight oil. The producing GOR of cases considering capillarity effects is much lower than the case without capillarity effect. In addition, the dynamic reservoir compaction further lowers the producing GOR.

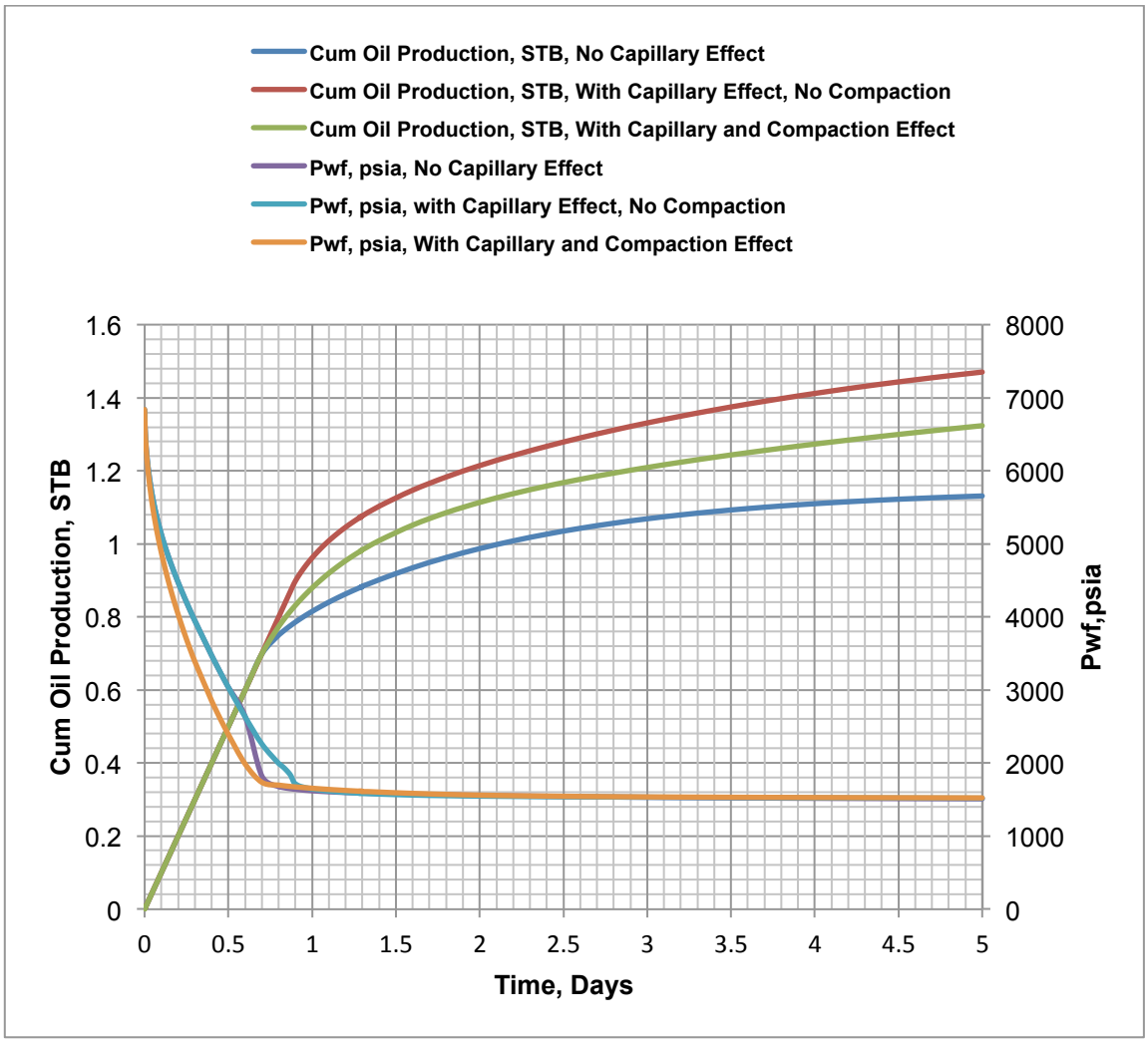

Figure 4.2 Cumulative oil production and pressure depletion of 1D model

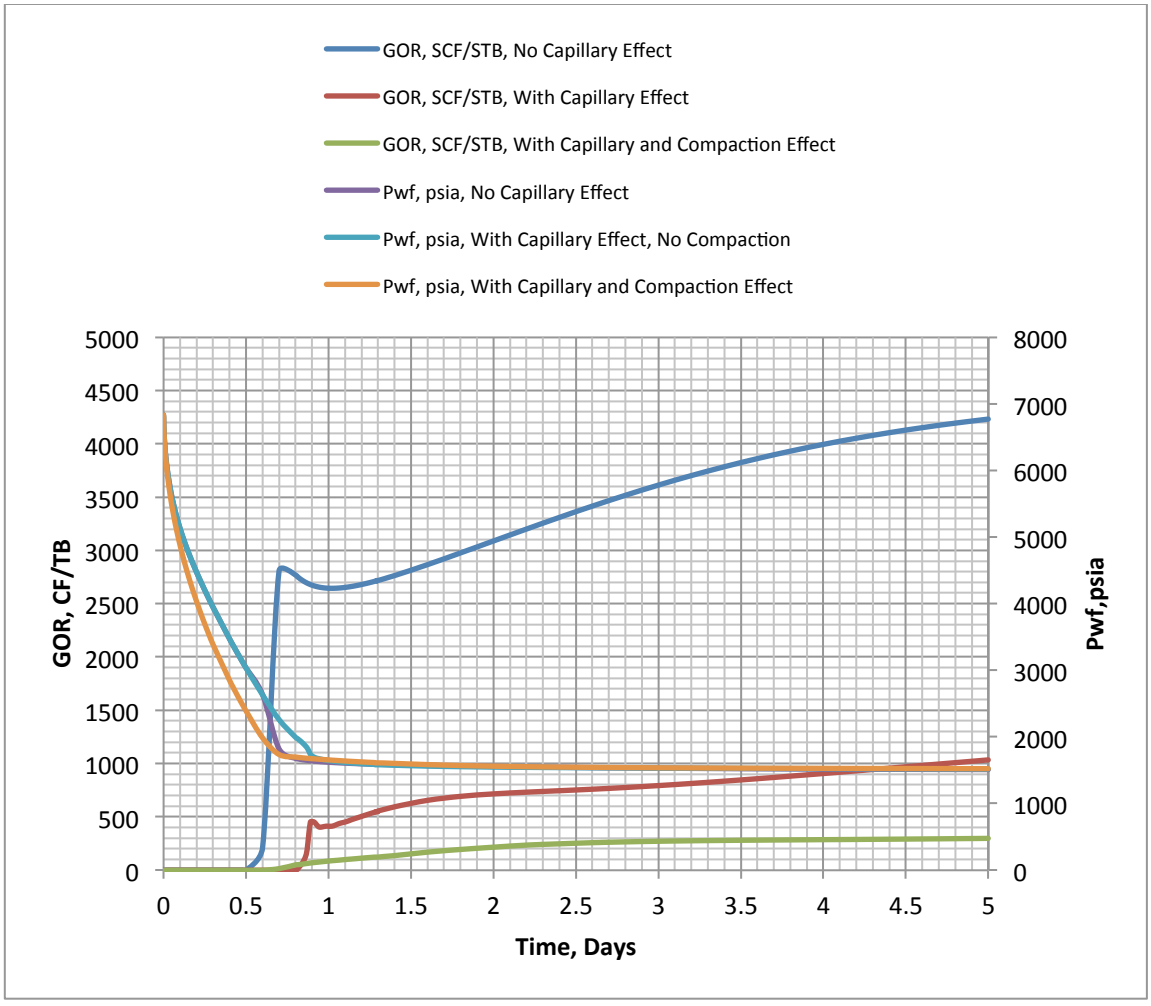

Figure 4.3 Producing GOR and pressure depletion of 1D model

4.4.2.2 Horizontal Well Model with Multiple Hydraulic Fractures

The second case is a horizontal well model with multi-stage hydraulic fractures (**Figure 4.4**). The 35X67X10 model has extent of 1000 ft in both X and Y direction and 20 ft in Z direction. Four hydraulic fractures are equally spaced in the Y direction. The production is initially constrained by oil flow rate and later by bottom hole pressure of 1900 psi. As discussed in previous sections, this model contains fluid with both confined

and unconfined properties. The two fluid systems are modeled in a unified fashion using distinct capillary pressures. The well production is mainly fed by the hydraulic fractures, which are in turn fed by the tight matrix. Note that the production and GOR are calculated at in-situ reservoir conditions when fluid enters the well. Although fluid that is directly connected to the well is unconfined, the volume of such fluid is very small comparing with its feed source, which is the confined fluid contained in tight matrix. Similarly, three scenarios are compared. Results are shown in Figure **4.5** and **4.6**. We can see the case with capillarity effect and no compaction has the highest cumulative oil production. The case with capillarity and compaction effect is in the middle. The pressure decline rate of the compaction case is smaller than the other two cases. Note that for the 1D core size model, the pressure decline rate of compaction is the highest. This might be because that for a hydraulic fractured well, the production is controlled by the fractures. The compaction reduced the conductivity of fracture, which makes the pressure decline smaller than the cases without considering compaction. It is interesting that there is a step-wise increase and decrease of the producing GOR of the case with compaction. This might be because that, as the reservoir being compacted, the bubble point pressure is reducing.

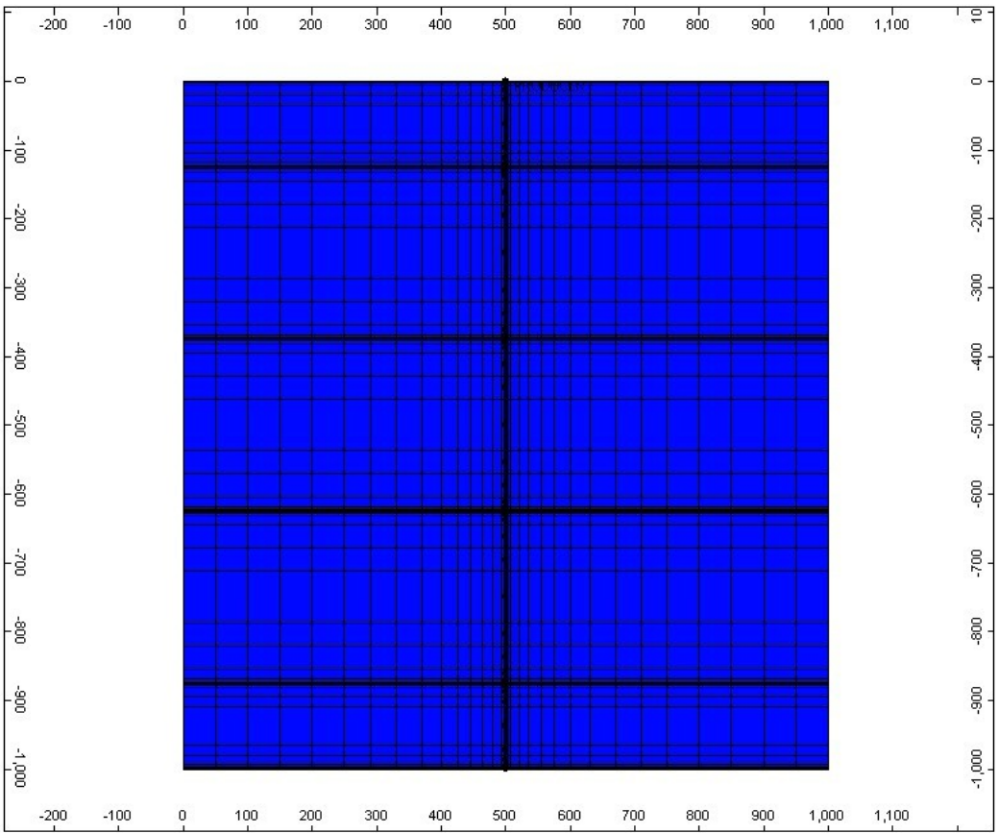

Figure 4.4: Top view of a horizontal well model with four hydraulic fractures (scale in feet)

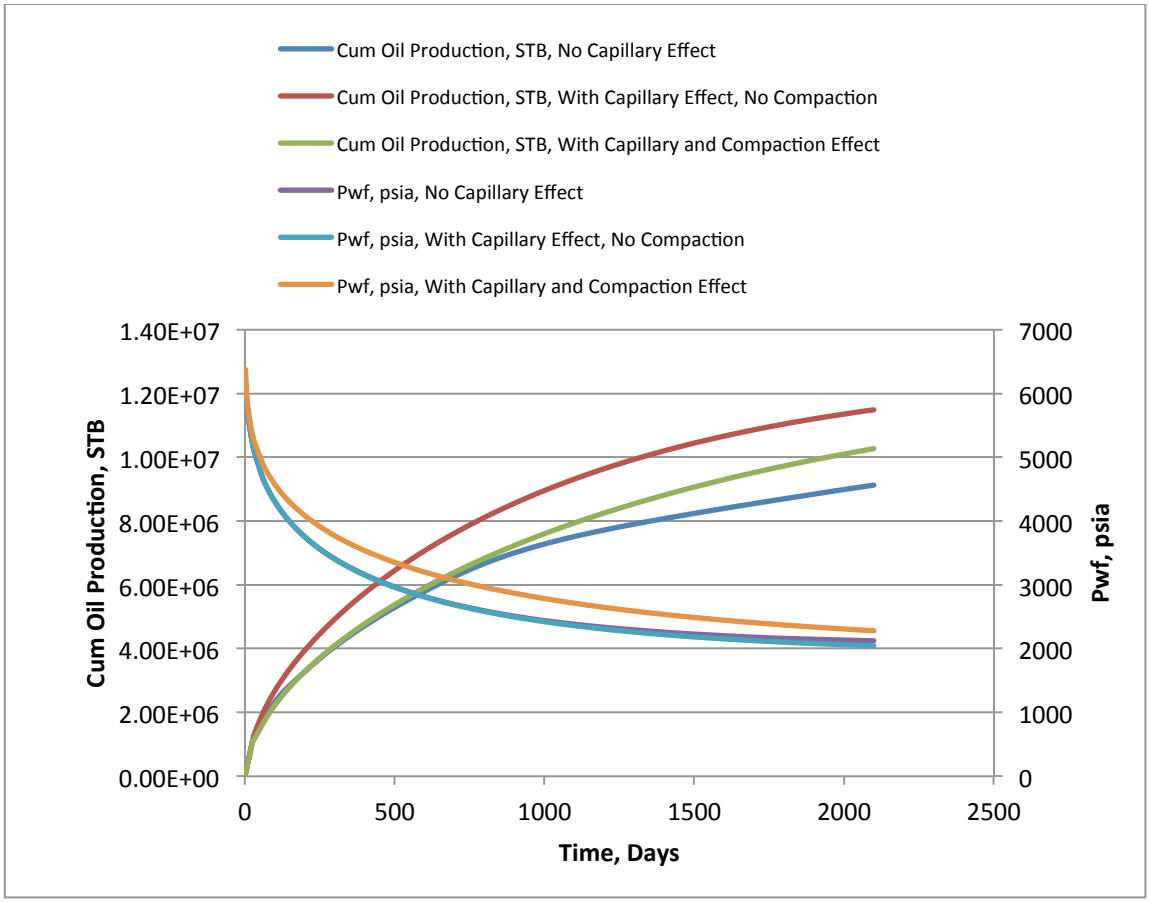

Figure 4.5 Cumulative oil production and pressure depletion of horizontal well model

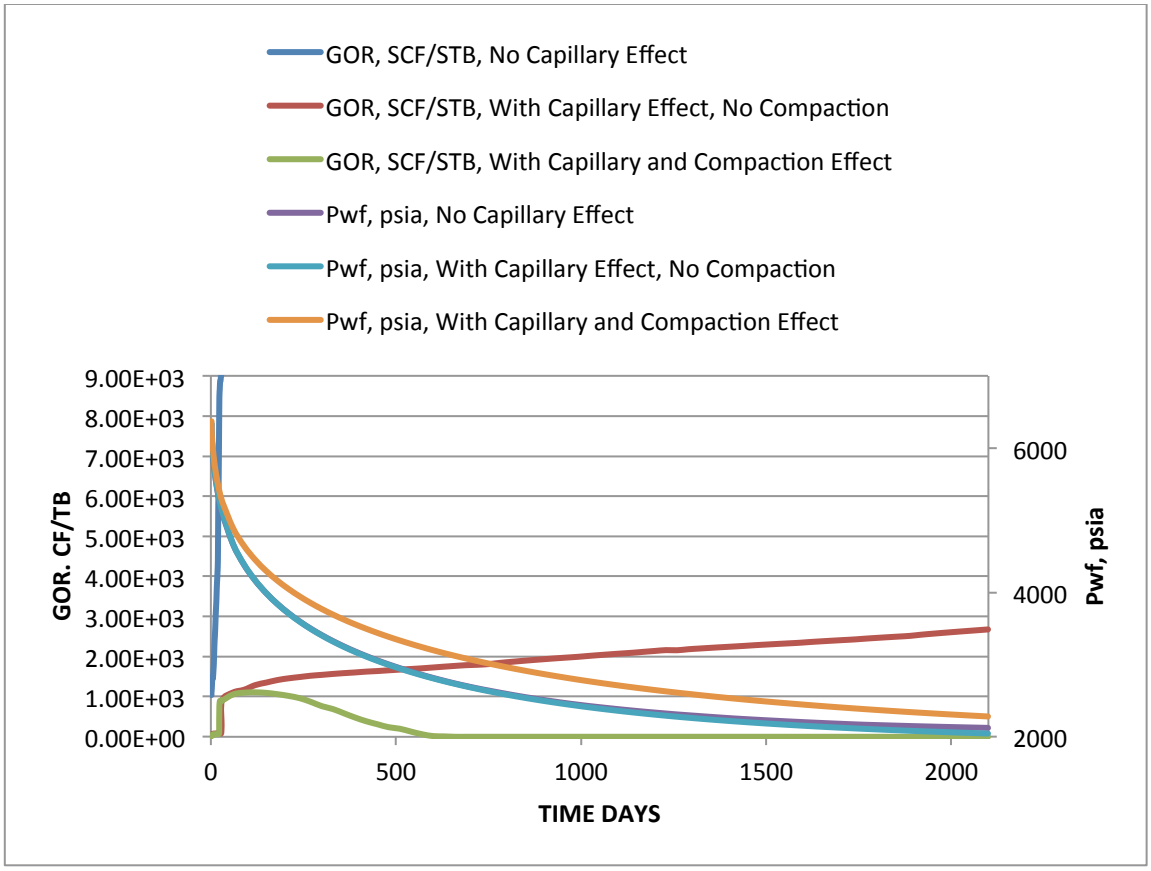

Figure 4.6 Producing GOR and pressure depletion of horizontal well model

4.5 Conclusions

Based on the assumption that the fluid is homogenous for pore size larger than 10 nm, we evaluate the effect of capillarity on VLE using the conventional thermodynamics for hydrocarbon reservoirs. The capillary pressure effect on the vapor-liquid equilibrium of reservoir fluids becomes significant when the pore size reduces to the nano scale. For oil-wet reservoirs, the confinement effect suppresses the bubble point pressure, which in turn favors the single-phase oil production by increasing the driving energy and decreasing the viscosity of the fluids. In order to model such confined fluid properties in reservoir simulation, the VLE flash calculation algorithms need to be extended to consider capillary pressure difference between vapor and liquid phase. Leverett J-function should be the approach used to evaluate capillary pressures for tight oil. However, good quality capillary pressure measurements are needed to calibrate the scaling factor for a particular tight oil reservoir, such as Bakken. For tight oil reservoirs, which do not have pressure maintenance strategies, oil production is believed to be quite sensitive to reservoir compaction during pressure depletion. Thus, rock compaction should be considered in any reservoir study of tight oil. Rock compaction further reduces the pore size. As a result, the confinement effect becomes larger. However, compaction also reduces the permeability. Thus, simulation studies should consider the combined effect of capillarity and compaction on production.

CHAPTER V

SUMMARY AND RECOMMENDATIONS

In Chapter II, dynamic load imbalance, which is the key performance limiter of parallel compositional simulation when using the IMPES formulation, is investigated. Since such imbalance pattern is generally unpredictable, a dynamic load-balancing scheme is needed to improve the parallel efficiency. The implementation or design of dynamic load balancing schemes for parallel reservoir simulation can be very challenging and requires a substantial amount of development time. In this chapter, a promising shortcut is presented to mitigate the dynamic load imbalance problem. Inspired by the domain over-decomposition concept, and based on Charm++ and AMPI, the approach over-decomposed the underlying reservoir model into small chunks and bundles of these chunks are then mapped to each physical processor as virtual processes. There are two main attractions of AMPI. First, if we have a parallelized reservoir simulator using MPI, the adaptation to use AMPI is not cumbersome. But, special care must be taken to treat the global variables. There is an on-going research in the Charm++ community to improve the approach to handle the issues related to global variables. As discussed in Chapter II, the methodology of treating global variables for using AMPI may not have much room to improve. The future effort may be related to the workflow of easing the process to handle global variables.

As noted in Chapter II, the GreedyLB is recommend only because we deal with the parallelization of equation of state. When we deal with a fully paralleled reservoir

simulator, there will be inter-subdomain communications. As a result, other balancers that consider the inter-subdomain communication overhead should be applied instead. Fortunately, the Charm++ infrastructure provides a series of balancer for user to pick, which greatly eases the development effort. However, it should be keep in mind that, the operation of virtual processor assignment and migration and balancing is very high-level without explicitly exploring the underlying physics. On one hand, this mechanism could shorten the development phase, since the design of balancing based on the underlying physics and implementation could be very complex. However, on the other hand, this high-level mechanism may overlook some issues, which may bring a significant overhead.

One of such issues happens in the linear solver portion of a simulator. The parallelization of linear solver need exchange boundary cell data between subdomains. If we over-decompose the domain into many smaller chunks, the massage passing overhead could kill the performance gained by using over-decomposition. This problem becomes very important especially for fully implicit simulation, where the linear solution part often consumes the majority part of time. Further research should be conducted in this area to investigate the solutions. Based on limited experiments, for the best cases, it appears that domain over-decomposition can improve Jacobi, Gauss-Seidel and CG method's parallel performance by 10-20%.

Overall, domain over-decomposition is a promising way of improving the parallel performance of reservoir simulators. If we have a fully parallelized reservoir simulator using MPI, it is recommended to test the performance using AMPI. As

emphasized, generally speak, the cost of adaption to AMPI is lowers than trying to design and add a dynamic load balancer inside the simulator.

In Chapter III, the recently developed CML method is introduced to reservoir simulation application. The overall performance of CML is promising, especially for the case to handle the matrix resulting from coupled geomechanics. Keep in mind, there is no a linear solver that can be optimal for all the cases encountered in reservoir simulation. The properties of matrices resulting reservoir simulation change with different models and scenarios. The major draw back of CML is that it is designed and proved based on symmetric matrices. Its performance is expected to degrade with the increase of degree of asymmetry. For the coupled geomechanics, the matrix from poroelastic displacement is symmetric. We have shown CML has much better performance in this case using a benchmark matrix. If for some cases, CML do not have better performance than other methods for the flow problem, the recommendation of choice of solver for coupled poroelastic displacement would be CML.

In the current implementation of CML, the smoother is Jacobi. Other smoother may need to be implemented to improve its performance. Further research is needed in this area.

In Chapter IV, the conventional VLE formulation is extended to consider the effect of capillary pressure between gas and oil phases. When the pore size is reduces to the nano scale, the effect of capillarity on VLE should not be ignored. The extended VLE formulation is developed based on the conventional Gibbs stability test and two phase split calculation. It should be emphasized that this work is based on the

assumption that the conventional homogenous thermodynamics, which is the ground for the conventional VLE formulation, is not violated for the nanopore scale between 10 nm to 50 nm. As indicated in Chapter IV, 10 nm seems to the limit to use conventional VLE formulation.

The approach taken to evaluate the effect of dynamic nanopore on tight oil recovery is based on fully compositional simulation. Since Bakken oil is apparently in the black oil region, fully compositional simulation may not be needed in practice. To perform black oil simulation for Bakken oil, we first need to use a confined PVT table to consider the effect of capillarity. Second, we need to establish the relation between confined PVT table and compaction table, since as the pore space changes the confinement effect changes (in other words, the PVT changes). As a future research recommendation, it is necessary to perform comparison of the results of fully compositional simulation and black oil simulation to consider the dynamic nanopore effects.

Since Bakken oil is in the black oil region, Chapter IV simplifies the saturation pressure to be the bubble point pressure. If the fluid is in volatile oil or gas condensate region (e.g. Eagle Ford liquid rich shale), the saturation pressure will be the dew point pressure. The same approach can be applied to model the effect of dynamic nano-pore properties for Eagle Ford liquid rich shale if we ignore the effect of adsorptions. Adsorption can complicate the analysis in great extent. The adsorption of vapor and liquid at the shale surface can induce highly structured particle density distributions close to the surface, which will make the fluid inhomogeneous (Evans 2009). Further

research is needed to investigate the applicable and practical approach to handle this situation.

REFERENCES

Adaptive MPI Manual. V 1.0. Parallel Programming Laboratory, University of Illinois at Urbana-Champaign.

Alpak, F.O. and Wheeler, M.F. 2012. A Suppercoarsening Multigrid Method for Poroelasticity in 3D Coupled Flow and Geomechanics Modeling. *Computational Geoscience* **16**(4): 953-974.

Anguille, L., Killough, J.E., Li, T.M.C., Toepfer, J.L. 1995. Static and Dynamic Load-balancing Strategies for Parallel Reservoir Simulation. Paper SPE 29102 presented at the SPE Symposium on Reservoir Simulation, San Antonio, Texas, USA, 12-15 February.

Appleyard, J.R., Appleyard, J.D., Wakefield, M.A., Desitter, A.L. 2011. Accelerating Reservoir Simulators Using GPU Technology. Paper SPE 141402 presented at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 21-23 February.

Aziz, K. and Settari, A. 1979. *Petroleum Reservoir Simulation*. Applied Science Publishers Ltd, London.

Bayat, M., and Killough, J. 2013. An Experimental Study of GPU Acceleration for Reservoir Simulation. Paper SPE 163628 presented at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 18-20 February.

Beckner, B.L., Usadi, A.K., Ray, M.B., Diyankov, O.V. 2006. Next Generation Reservoir Simulation Using Russian Linear Solvers. Paper SPE 103578 presented at

the SPE Russian Oil and Gas Technical Conference and Exhibition, Moscow, Russia, 3-6 October.

Bedrikovetsky, P. 1993. *Mathematical Theory of Oil and Gas Recovery: with Applications to Ex-USSR Oil and Gas Fields*. Kluwer Academic Publishers, The Netherlands.

Behie, G.A. and Forsyth, P.A. 1983. Multigrid Solution of the Pressure Equation in Reservoir Simulation. *SPE J* **23**(4): 623-632. SPE-10492-PA.

Bell, W.N., Olson, L.N., Schroder, J.B. 2011. PyAMG: Algebraic Multigrid Solvers in Python v2.0. http://www.pyamg.org.

Bohm, E., Bhatele, A., Kale, L.V., Tuckerman, M.E., Kumar, S., Gunnels, J.A., Martyna, G.J. 2008. Fine Grained Parallelization of the CaParrinello ab initio MD Method on Blue Gene/L. *IBM journal of Research and Development: Applications of Massively Parallel Systems*, **52**(½): 159-174.

Boman, E.G., Hendrickson, B. 2003. Support Theory for Preconditioning. *SIAM J Matrix Anal Appl,* **25**(3): 694-717.

Cao, H., Tchelepi, H.A., Wallis, J.R., Yardumian, H. 2005. Parallel Scalable Unstructured CPR-Type Linear Solver for Reservoir Simulation. Paper SPE 96809 presented at the SPE Annual Technical Conference and Exhibition, Dallas, Texas, 9-12 October.

Carman, P.C. 1937. Fluid Flow through Granular Beds. *Transaction, Institution of Chemical Engineers*, London, **15**: 150–166.

Christie, M.A. and Blunt, M.J. 2001. Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. Paper SPE 66599 presented at the SPE Reservoir Simulation Symposium, Houston, Texas, USA, 11-14 February.

Chu, L., Ye, P., Harmawan, I., Du, L., Shepard, L. 2012. Characterizing and simulating the non-stationariness and non-linearity in unconventional oil reservoirs: Bakken application. Paper SPE 161137 presented at the SPE Canadian Unconventional Resources Conference, Calgary, Alberta, Canada, 31 Oct-1 November.

Davis, T.A. 1994. University of Florida Sparse Matrix Collection, *ACM Transactions on Mathematical Software*, **38**(1): 1-25. http://www.cise.ufl.edu/research/sparse/matrices

Davis, T.A. 2006. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia.

Dean, R.H., Gai, X., Stone, C.M., Minkoff, S.E. 2003. A Comparison of Techniques for Coupling Porous Flow and Geomechanics. Paper SPE 79709 presented at the SPE Reservoir Simulation Symposium, Houston, Texas, USA, 3-5 February.

Dean, R.H. and Lo, L.L. 1988. Simulation of Naturally Fractured Reservoirs. *SPE RE* **3**(2): 633-648.

Diyankov, O.V., Koshelev, S.V., Kotegov, S.S., Krasnogorov, I.V., Kuznetsova, N.N., Pravilnikov, V.Y., Beckner, B.L., Maliassov, S.Y., Mishev, I.D., Usadi, A.K. 2007. Sparsol – Sparse Linear Systems Solver. *J Numer Math* **0**(0): 1-16.

Du, L. and Chu, L. 2012. Understanding Anomalous Phase Behavior in unconventional oil reservoirs. Paper SPE 161830 presented at the SPE Canadian Unconventional Resources Conference, Calgary, Alberta, Canada, 31 Oct–1 November.

Dufort, E.C., Frankel, S.P. 1953. Stability Conditions in the Numerical Treatment of Parabolic Equations. *Math Tables and other Aids to Computation*, **7**(43): 135-152.

Evans, R. 2009 Density Functional Theory for Inhomogeneous Fluid I: Simple Fluids in Equilibrium. Lectures at 3rd Warsaw School of Statistical Physics, Kazimierz Dolny, Poland, 27 June-3 July.

Eisenstat, S.C., Elman, H.C., Schultz, M.H. 1983. Vibrational Iterative Methods for Nonsymmetric Systems of Linear Equations. *SIAM J Numer Anal*, **20**:345-357.

Ferronato, M., Gambolati, G., Janna, C., Teatini, P. 2010. Geomechanical Issues of Anthropogenic CO_2 Sequestration in Exploited Gas Fields. *Energy Conversion and Management*, **51**(10): 1918-1928.

Firoozabadi, A. 2013. Private Communication

Fleming, G. 2012. Private communication.

Foltinek, D., Eaton, D., Mahovsky, J., Moghaddam, P., McGarry, R. 2009. Industrial-scale Reverse Time Migration on GPU Hardware. SEG Annual Meeting, Extended Abstract, 2009-2789.

Fung, L.S.K., Dogru, Ali H. 2007. Parallel Unstructured Solver Methods for Complex Giant Reservoir Simulation. Paper SPE 106237 presented at the SPE Reservoir Simulation Symposium, Houston, Texas, USA, 26-28 February.

Gai, X., Dean, R.H., Wheeler, M.F., Liu, R. 2003. Coupled Geomechanical and Reservoir Modeling on Parallel Computers. Paper SPE 79700 presented at the SPE Reservoir Simulation Symposium, Houston, Texas, USA, 3-5 February.

Gremban, K. 1996. Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems. PhD dissertation, Carnegie Mellon University.

Honarpour, M.M., Nagarajan, N.R., Organgi, A., Arastech, F. Yao, Z. 2012. Characterization of Critical Fluid, Rock, and Rock-fluid Properties-impact on Reservoir Performance of Liquid-rich Shales. Paper SPE 158042 presented at the SPE Annual Technical Confference, San Antonio, Texas, USA, 8-10 October.

Huang, C., Zheng, G., Kumar, S., Kale, L.V. 2006. Performance Evaluation of Adaptive MPI. In Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, New York, New York, USA, 29-31 March.

Huang, C., Lawlor, O., Kale, L.V. 2003. Adaptive MPI. In Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing, College Station, Texas, USA, 2-4 October.

Huang, Y. 2012. Generalization of Support Theory for Preconditioning. Fifth International Conference on Information and Computing Science. Liverpool, UK, 24-25 July.

Jetley, P., Gioachin, F., Mendes, C., Kale, L.V., Quinn, T.R. 2008. Massively Parallel Cosmological Simulations with ChaNGa. In Proceedings of IEEE International Parallel and Distributed Processing Symposium. Miami, Florida, USA, 14-18 April.

Jiao, X., Zheng, G., Lawlor, G., Alexander, P., Campbell, M., Heath, M., Fiedler, R. 2005. An Integration Framework for Simulations of Solid Rocket Motors. In 41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference, Tucson, Arizona, USA, 10-13 July.

Kale, L.V., Bohm, E., Mendes, C.L., Wilmarth, T., Zheng, G. 2008. Programming Petascale Applications with Charm++ and AMPI. in Petascale Computing: Algorithms and Applications. Bader, D., Ed. Chapman & Hall/CRC Press, pp. 421-441.

Karypis, G. and Kumar, V. 1999. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J Sci Comput*, **20**(1): 359-392.

Killough, J.E., Wheeler, M.F. 1987. Parallel Iterative Equation Solvers: an Investigation of Domain Decomposition Algorithms for Reservoir Simulation. Paper SPE 16021 presented at the SPE Symposium on Reservoir Simulation, San Antonio, Texas, USA, 1-4 February.

Klie, H., Sudan, H., Li, R., Saad, Y. 2011. Exploiting Capabilities of Manycore Platform in Reservoir Simulation. Paper SPE 141265 presented at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 21-23 February.

Klie, H., Wheeler, M.F., Clees, T., Stüben, K. 2007. Deflation AMG Solvers for Highly Ill-Conditioned Reservoir Simulation Problems. Paper SPE 10582 presented at the SPE Reservoir Simulation Symposium, Houston, USA, 26-28 February.

Koutis, I. 2007. Combinatorial and Algebraic Tools for Optimal Multilevel Algorithms. PhD Dissertation, Carnegie Mellon University.

Koutis, I., Miller, G., Tolliver, D. 2009. Combinatorial Preconditioners and Multilevel Solvers for Problems in Computer Vision and Image Processing. In: Proceeding ISVC '09 Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I, Las Vegas, Nevada, USA, 30 November-2 December.

Koutis, I. CMG Solver. http://www.cs.cmu.edu/~jkoutis/cmg.html

Kozeny, J., 1927. Ueber kapillare Leitung des Wassers im Boden. *Sitzungsber Akad Wiss Wien*, **136**(2a): 271-306.

Kuila U., Prasad M., 2011. Surface Area and Pore-size Distribution in Clays and Shales. Paper SPE 146869 presented at the SPE Annual Technical Conference and Exhibition, Denver, Colorado, USA, 30 October–2 November.

Leverett, M.C. 1941. Capillary Behavior in Porous Solids. *Trans AIME* **142**(1): 159-172

Liu, H., Yu, S., Chen, Z., Hsieh, B., Shao, L. 2012. Parallel Preconditioners for Reservoir Simulation on GPU. Paper SPE 152811 presented at the SPE Latin America and Caribbean Petroleum Engineering Conference, Mexico City, Mexico, 16-18 April.

Liven, O.E. 2012. Lean Algebraic Multigrid MATLAB Software, Release 2.1.1. http://lamg.googlecode.com

Lu, B., Alshaalan, T.M., Wheeler, M.F. 2007. Iteratively Coupled Reservoir Simulation for Multiphase Flow. Paper SPE 110114 presented at SPE Annual Technical Conference and Exhibition, Anaheim, USA, 11-14 November.

Michelsen, L.M. 1982a. The Isothermal Flash Problem. Part I. Stability. *Fluid Phase Equilibria*, **9**(1): 1-19.

Michelsen, L.M. 1982b. The Isothermal Flash Problem. Part II. Phase-split Calculation. *Fluid Phase Equilibria*, **9**(1): 21-40.

Nelson, P.H. 1994. Permeability-porosity Relationships in Sedimentary Rocks. *The Log Analyst*, **35**(3): 38-62

Nojabaei, B., Johns, R.T. and Chu, L. 2012. Effect of Capillary Pressure on Fluid Density and Phase Behavior in Tight Rocks and Shales. Paper SPE 159258 presented at the SPE Annual Technical Conference and Exhibition, San Antonio, Texas, USA, 8-10 October.

Notay, Y. 2010. An Aggregation-based Algebraic Multigrid Method. *Electron Trans Numer Anal*, **37**:123-146.

Napov, A., Notay, Y. 2012. An Algebraic Multigrid Method with Guaranteed Convergence Rate. *SIAM J Sci Comput* **34**(2): A1079-A1109.

Notay, Y. 2012. Aggregation-based Algebraic Multigrid for Convection-diffusion Equations. *SIAM J Sci Comput* **34**(4): A2288-A2316.

Notay, Y. AGMG Software and Documentation; http://homepages.ulb.ac.be/~ynotay/AGMG.

Pang, J., Zuo, J.Y., Zhang, D., Du, L. 2012. Impact of Porous Media on Saturation Pressure of Gas and Oil in Tight Reservoirs. Paper SPE 161143 presented at the SPE Canadian Unconventional Resources Conference, Calgary, Alberta, Canada, 31 October-1 November.

Pedersen, K.S. and Christensen, P.L. 2007. *Phase Behavior of Petroleum Reservoir Fluids*. CRC Press, Taylor & Francis Group, Boca Raton, FL.

Piault, E., Ding, Y. 1993. A Fully Explicit Scheme in Reservoir Simulation on a Massively Parallel Computer. Paper SPE 25274 presented at the SPE Symposium on Reservoir Simulation, New Orleans, Louisiana, USA, 28 February-3 March.

Rodrigues, E.R., Navaux, P.O.A., Paneta, J., Mendes, C.L., Kale, L.V. 2010. Optimizing an MPI Weather Forecasting Model via Processor Virtualization. In Proceeding HiPC, Dona Paula, India, 19-22 December.

Rodrigues, E.R., Navaux, P.O.A., Paneta, J., Fazenda, A., Mendes, C.L., Kale, L.V. 2010. A Comparative Analysis of Load Balancing Algorithms Applied to a Weather Forecast Model. In Proceeding SBAC-PAD '10 Proceedings of the 2010 22nd International Symposium on Computer Architecture and High Performance Computing. Petrópolis, Rio de Janeiro, Brazil, 27-30 October.

Rodrigues, E.R., Navaux, P.O.A., Panetta, J., Mendes, C.L. 2010. A New Technique for Data Privatization in User-level Threads and Its Use in Parallel Applications. In ACM 25th Symposium on Applied Computing (SAC), Sierre, Switzerland. 22-26 March.

Rodrigues, E.R. 2012. Thread local storage enabled GFortran. Private Communication.

Ruge, J.W. and Stüben, K. 1987. Algebraic Multigrid (AMG). In Multigrid Methods, Volume 3 of Frontiers in Applied Mathematics. McCormick, S.F., Ed. SIAM, Philadelphia, PA, pp. 73-130.

Saad, Y., Schultz, M.H. 1986. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J Sci Stat Comput* **7**(3): 856-869.

Sigmund, P.M., Dranchuk, P.M., Morrow, N.R., Purvis, R.A., 1973. Retrograde Condensation in Porous Media. *SPE J* **13**(2): 93-104.

Shapiro, A.A., and Stenby, E.H. 1997. Kelvin Equation for a Non-ideal Multicomponent Mixture. *Fluid Phase Equilibria* **134**(1-2): 87-101.

Shapiro, A.A., Potsch, K., Kristensen, J.G., Stenby, E.H. 2000. Effect of Low Permeable Porous Media on Behavior of Gas Condensates. Paper SPE 65182 presented at SPE European Petroleum Conference, Paris, France, 24-25 October.

Sherman, A.H. 1992. A Hybrid Approach to Parallel Compositions Reservoir Simulation. Paper OTC 6829 presented at the Offshore Technology Conference, Houston, Texas, USA, 4-7 May.

Shuttleworth, R., Maliassov, S., Zhou, H. 2009. Partitioners for Parallelizing Reservoir Simulations. Paper SPE 119130 presented at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, 2-4 February.

Stüben, K., Clees, T., Klie, H., Lu, B., Wheeler, M.F. 2007. Algebraic Multigrid Methods (AMG) for the Efficient Solution of Fully Implicit Formulations in Reservoir Simulation. Paper SPE 105832 presented at the SPE Reservoir Simulation Symposium, Houston, Texas, USA, 26-28. February.

Tang, D.E. and Zick, A.A. 1993. A New Limited Compositional Reservoir Simulator. Paper SPE 25255 presented at the SPE Symposium on Reservoir Simulation, New Orleans, LA, USA, 28 February-3 March.

Van der Vorst, H.A. 1992. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. *SIAM J Sci Stat Comput* **13**(2): 631–644

Wallis, J.R. 1983. Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration. Paper SPE 12265 presented at the

SPE Reservoir Simulation Symposium, San Francisco, California, USA, 15-18 November.

Wallis, J.R., Kendall, R.P., Little, T.E. 1985. Constrained Residual Acceleration of Conjugate Residual Methods. Paper SPE 13563 presented at the SPE Reservoir Simulation Symposium, Dallas, Texas, USA, 10-13 February.

Wang, D., Butler, R., Zhang, J., Seright, R. 2012. Wettability Survey in Bakken Shale with Surfactant-formulation Imbibition. *SPE RE* **15**(6): 695-705.

Wei, C., Wang, H., Sun, S., Xiao, Y., Zhu, Y., Qin, G. 2012. Potential Investigation of Shale Gas Reservoirs, Southern China. SPE paper 162828 presented at the SPE Canadian Unconventional Resources Conference, Calgary, Alberta, Canada, 31 October-1 November.

Wei, C., Qin, G., Guo, W., Yan, B., Killough, J.E., Wang, H., Liu, H. 2013a. Characterization and Analysis on Petrophysical Parameters of a Marine Shale Gas Reservoir. Paper SPE 165380 presented at the SPE Western Regional & AAPG Pacific Section Meeting, Monterey, California, USA, 19-15 April.

Wei, C., Wang, Y., Qin, G., Li, Q., Killough, J.E. 2013b. Experimental and Numerical Studies on the Micro-fractures and its Significance Toward Production of Shales: a Case Study. SPE paper 165327 presented at the SPE Eastern Regional Meeting, Pittsburgh, Pennsylvania, USA, 20-22 August.

White, J.A., Borja, R.I. 2011. Block-preconditioned Newton-Krylov Solvers for Fully Coupled Flow and Geomechanics. *Comput Geosci* **15**(4): 647-659.

Yan, B., Wang, Y., and Killough, J. 2013a. Beyond Dual-porosity Modeling for the

Simulation of Complex Flow Mechanisms in Shale Reservoirs. Paper SPE 163651 presented at the 2013 SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 18-20 February.

Yan, B., Killough, J., Wang, Y. et al. 2013b. Novel Approaches for the Simulation of Unconventional Reservoirs. Paper SPE 168786 presented at the Unconventional Resources Technology Conference, Denver, Colorado, USA, 12-14 August.

Yan, B., Alfi, M., Wang, Y., Killough, J.E. 2013c. A New Approach for the Simulation of Fluid Flow in Unconventional Reservoirs through Multiple Permeability Modeling. Paper SPE 166173 presented at the SPE Annual Technical Conference and Exhibition, New Orleans, Louisiana, USA, 30 September-2 October.

Yan, B. 2013. A Novel Approach for the Simulation of Multiple Flow Mechanisms and Porosities in Shale Gas Reservoirs. M.S. Thesis, Texas A&M University, College Station.

Zhang, K., Wu, Y.S., Pruess, K., Elmroth, E. 2001. Parallel Computing Techniques for Large-scale Reservoir Simulation of Multi-component and Multiphase Fluid Flow. Paper SPE 66343 presented at the SPE Reservoir Simulation Symposium, Houston, Texas, USA, 11-14 February.

Zheng, G. 2005. Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing. Ph.D. Dissertation, University of Illinois at Urbana-Champaign.

Zheng, G., Lawlor, G.O.S., Kale, L.V. 2006. Multiple Flows of Control in Migratable Parallel Programs. In 2006 International Conference on Parallel Processing Workshops (ICPPW'06). Columbus, Ohio, USA, 14-18 August.

Zheng, G., Negara, S., Mendes, C., Rodrigues, E., Kale, L. 2011. Automatic Handling of Global Variables for Multi-threaded MPI programs. In Proceeding ICPADS '11 Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems. Tainan, Taiwan, 7-9 December.

APPENDIX I

STABILITY TEST WITH CAPILLARY PRESSURE EFFECT

The following derivations are based on Michelsen (1982a).

I.1 Stability Criterion

Given temperature T_0 and pressure P_0 and M component mixture with component mole fractions ($z_1, z_2, ..., z_m$). The Gibbs energy of the mixture is

$$G_0 = \sum_i^M n_i \mu_i^0 \tag{I.1.1}$$

where μ_i^0 is the chemical potential of component i in the mixture and n_i is the number of moles of component i in the mixture. Now, let's assume this mixture is divided into two phases with mole number $N - \varepsilon$ and ε, respectively. The amount ε of the second phase is infinitesimal. Let the mole fractions in Phase II be $y_1, y_2, \ldots, y_M$. The change in Gibbs energy is then

$$\Delta G = G_I + G_{II} - G_0 = G(N - \varepsilon) + G(\varepsilon) - G_0 \tag{I.1.2}$$

If we apply Taylor series expansion to G_I, discarding second order term in ε, we have

$$G(N - \varepsilon) = G(N) - \varepsilon \sum_i^M y_i \left(\frac{\partial G}{\partial n_i} \right)_N = G_0 - \varepsilon \sum_i^M y_i \mu_i^0 \tag{I.1.3}$$

Then changes in Gibbs free energy is given by

$$\Delta G = \varepsilon \sum_i^M y_i \left(\mu_i(y) - \mu_i^0 \right) \tag{I.1.4}$$

The stability of the original mixture requires that its Gibbs free energy is at the global minimum. A criterion for stability is that

$$F(y) = \sum_i^M y_i\left(\mu_i(y) - \mu_i^0\right) \geq 0 \tag{I.1.5}$$

for all trial compositions y. All minima of $F(y)$ are located in the interior of the permissible region ($y_i \geq 0, \sum_i^M y_i = 1$). It is usually convenient to work with fugacity. The fugacities are given by

$$f_i(z) = z_i \phi_i(z) P^L \tag{I.1.6}$$

$$f_i(y) = y_i \phi_i(y) P^V \tag{I.1.7}$$

Note that, it is assumed the original phase is liquid and trial phase is vapor. The opposite setting (original phase is vapor and trial phase is liquid) is equivalent. Since

$$\mu_i = RT \ln(f_i) \tag{I.1.8}$$

We then have

$$\begin{aligned} g(y) = \frac{F(y)}{RT^0} &= \sum_i^M y_i\left[\ln f_i(y) - \ln f_i(z)\right] \\ &= \sum_i^M y_i\left[\ln\left(y_i \phi_i(y) P^V\right) - \ln\left(z_i \phi_i(z) P^L\right)\right] \\ &= \sum_i^M y_i\left[\ln y_i + \ln \phi_i(y) - \ln z_i - \ln \phi_i(z) + \left(\ln P^V - \ln P^L\right)\right] \geq 0 \end{aligned} \tag{I.1.9}$$

The last term ($\ln P^V - \ln P^L$) is normally ignored because of the small capillary pressure between vapor and liquid. But it is included here to include the effect of high capillary pressure, which is hypothesized to encounter in confined pore space. Let

$$h_i = \ln z_i + \ln \phi_i(z) + \ln P^L \tag{I.1.10}$$

Then we have,

$$g(y) = \sum_i^M y_i \left(\ln y_i + \ln \phi_i(y) - h_i + \ln P^V \right) \geq 0 \tag{I.1.11}$$

In the above equation, h_i is independent of y (z remains unchanged). So, this term can be pre-computed without iterating. The stationary criterion is

$$\ln y_i + \ln \phi_i(y) - h_i + \ln P^V = k \quad i = 1,2,\ldots,M \tag{I.1.12}$$

Let $Y_i = e^{-k} y_i$, then

$$\ln Y_i + \ln \phi_i(y) - h_i + \ln P^V = 0 \quad i = 1,2,\ldots,M \tag{I.1.13}$$

Since $\sum_i^M y_i = 1$, then

$$e^k \sum_i^M Y_i = 1 \tag{I.1.14}$$

As a result,

$$y_i = Y_i \Big/ \sum_i^M Y_i \tag{I.1.15}$$

This equation suggests that Y_i can be interpreted as mole numbers. Apparently, we can see that the solutions of the nonlinear equation (**Eq. I.1.13**) can be used to examine the stability analysis. For a system of fixed composition z_i, the system is stable when $k \geq 0$. Thus, in terms of Y, this system is stable when $\sum_i^M Y_i \leq 1$ and is unstable when $\sum_i^M Y_i > 1$.

Successive substitution and/or newton's method can be used to solve the nonlinear equation (**Eq. I.1.13**).

I.2 Successive substitution

The updating of Y is simply

$$Y_i^{new} = \exp\left[h_i - \ln\phi_i(y) - \ln P^V\right] \qquad i = 1,2,\ldots,M \tag{I.2.1}$$

I.3 Newton's method

Although successive substitution is very easy to implement, it is not efficient and requires significant number of iterations. Thus successive substitution method is not desired for reservoir simulation, since we need to solve **Eq. I.1.13** for each grid block**.** To overcome this drawback, we use Newton's method which only requires substantially less number of iterations. To use Newton's method, we first write the residual as

$$R(y_i) = \ln Y_i + \ln\phi_i(y) - h_i + \ln P^V = 0 \qquad i = 1,2,\ldots,M \tag{I.3.1}$$

Then the updating formulations are as follows

$$y^{k+1} = y^k - \left(J^k\right)^{-1} R^k \tag{I.3.2}$$

$$J^k \Delta y = -R^k \tag{I.3.3}$$

The jacobian is given by

$$J_{ij} = \frac{\partial R_i}{\partial y_j} \qquad i = 1,2,\ldots,M; j = 1,2,\ldots,M \tag{I.3.4}$$

In matrix format, we have

$$\begin{pmatrix} \frac{\partial R_1}{\partial y_1} & \frac{\partial R_1}{\partial y_2} & \cdots & \frac{\partial R_1}{\partial y_M} \\ \frac{\partial R_2}{\partial y_1} & \frac{\partial R_2}{\partial y_2} & \cdots & \frac{\partial R_2}{\partial y_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial R_M}{\partial y_1} & \frac{\partial R_M}{\partial y_2} & \cdots & \frac{\partial R_M}{\partial y_M} \end{pmatrix} \begin{pmatrix} \Delta y_1 \\ \Delta y_2 \\ \vdots \\ \Delta y_M \end{pmatrix} = -\begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_M \end{pmatrix} \tag{I.3.5}$$

In implementation, we instead use the K-value, K_i, or its logarithm as the independent variable. The formulations using $\ln K_i$ is below. Now R is treated as a function of $\ln K_i$, such as $R(\ln K_i) = 0$. Similarly, we have

$$\ln K_i^{(k+1)} = \ln K_i^{(k)} - \left(J^{(k)}\right)^{-1} R^{(k)} \tag{I.3.6}$$

$$J^{(k)} \Delta \ln K = -R^{(k)} \tag{I.3.7}$$

$$J_{ij} = \frac{\partial R_i}{\partial \ln K_j} \qquad i = 1,2,\ldots,M; j = 1,2,\ldots,M \tag{I.3.8}$$

In matrix format, we have

$$\begin{pmatrix} \frac{\partial R_1}{\partial \ln K_1} & \frac{\partial R_1}{\partial \ln K_2} & \cdots & \frac{\partial R_1}{\partial \ln K_M} \\ \frac{\partial R_2}{\partial \ln K_1} & \frac{\partial R_2}{\partial \ln K_2} & \cdots & \frac{\partial R_2}{\partial \ln K_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial R_M}{\partial \ln K_1} & \frac{\partial R_M}{\partial \ln K_2} & \cdots & \frac{\partial R_M}{\partial \ln K_M} \end{pmatrix} \begin{pmatrix} \Delta \ln K_1 \\ \Delta \ln K_2 \\ \vdots \\ \Delta \ln K_M \end{pmatrix} = -\begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_M \end{pmatrix} \tag{I.3.9}$$

After applying chain rule, we have

$$\frac{\partial R_i}{\partial \ln K_j} = \frac{\partial R_i}{\partial y_j} \frac{\partial y_j}{\partial \ln K_j} \tag{I.3.10}$$

Since $Y_i = z_i K_i$ and $Y_i = e^{-k} y_i$, we then have

$$y_i = z_i e^k K_i \tag{I.3.11}$$

Note that $z_i e^k$ is independent of y_i or $\ln K_i$. So,

$$\frac{\partial y_j}{\partial \ln K_j} = \frac{\partial z_j e^k K_j}{\partial \ln K_j} = z_j e^k \frac{\partial K_j}{\partial \ln K_j} = z_j e^k K_j = y_j \tag{I.3.12}$$

As a result,

$$\frac{\partial R_i}{\partial \ln K_j} = \frac{\partial R_i}{\partial y_j} \frac{\partial y_j}{\partial \ln K_j} = y_j \frac{\partial R_i}{\partial y_j} \tag{I.3.13}$$

and finally in matrix form, we have

$$\begin{pmatrix} y_1 \frac{\partial R_1}{\partial y_1} & y_2 \frac{\partial R_1}{\partial y_2} & \cdots & y_M \frac{\partial R_1}{\partial y_M} \\ y_1 \frac{\partial R_2}{\partial y_1} & y_2 \frac{\partial R_2}{\partial y_2} & \cdots & y_M \frac{\partial R_2}{\partial y_M} \\ \vdots & \vdots & \ddots & \vdots \\ y_1 \frac{\partial R_M}{\partial y_1} & y_2 \frac{\partial R_M}{\partial y_2} & \cdots & y_M \frac{\partial R_M}{\partial y_M} \end{pmatrix} \begin{pmatrix} \Delta \ln K_1 \\ \Delta \ln K_2 \\ \vdots \\ \Delta \ln K_M \end{pmatrix} = - \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_M \end{pmatrix} \tag{I.3.14}$$

It should be noted that although Newton's method converges fast, global minima could not be guaranteed. In simulator implementation, we generally first perform a few iterations of successive substation to a bigger tolerance. Then we use Newton's method to converge to the final tolerance.

APPENDIX II

TWO-PHASE SPLIT CALCULATION WITH CAPILLARY PRESSURE EFFECT

II.1 Governing Equations

For given F mole of feed (original single-phase phase fluid) with mole fraction of z_i ($i = 1,2,\ldots,m$) in a close system, the original phase splits into L mole liquid with mole fraction of x_i ($i = 1,2,\ldots,m$) and V mole vapor with mole fraction of y_i ($i = 1,2,\ldots,m$). We have the following energy and mass balances.

$$f_i^L\left(T,P^L,x_i\right) = f_i^V\left(T,P^V,y_i\right) \tag{II.1.1}$$

$$Fz_i = x_iL + y_iV \tag{II.1.2}$$

Besides, we have

$$\sum_i^m x_i = 1 \tag{II.1.3}$$

$$\sum_i^m y_i = 1 \tag{II.1.4}$$

To solve these equations, we first define V-L equilibrium ratio, K-value as

$$K_i = \frac{y_i}{x_i} \quad i = 1,2,\ldots,m \tag{II.1.5}$$

From **Eq. II.1.2**, we have

$$x_i = \frac{z_i}{1+\left(K_i-1\right)\alpha} \tag{II.1.6}$$

$$y_i = \frac{K_i z_i}{1+(K_i-1)\alpha} \tag{II.1.7}$$

Where $\alpha = V/F$.

Substitute **Eq. II.1.6** and **Eq. II.1.7** into **Eq. II.1.3** and **Eq. II.1.4**, we have

$$h(\alpha) = \sum_i^m \frac{(K_i-1)z_i}{1+\alpha(K_i-1)} \tag{II.1.8}$$

Eq. II.1.8 is called the Rachford-Rice equation. Since

$$f_i^V = y_i \phi_i^V P^V \qquad i = 1,2,\ldots,m \tag{II.1.9}$$

$$f_i^L = x_i \phi_i^L P^L \qquad i = 1,2,\ldots,m \tag{II.1.10}$$

From **Eq. II.1.9**, **Eq. II.1.10** and **Eq. II.1.1**, we then have

$$K_i = \frac{y_i}{x_i} = \frac{\phi_i^L P^L}{\phi_i^V P^V} \qquad i = 1,2,\ldots,m \tag{II.1.11}$$

Note that, normally, the pressure terms in the above equation are canceled out if we assume the pressures of vapor and liquid are equal.

II.2 Successive Substitution

The successive substitution generally follows the 6 steps

Step 1: Guess initial value of K_i. Stability test generally give good estimate of K_i

Step 2: Solve the Rachford-Rice using Newton-Raphson method

Step 3: Calculate x_i and y_i

Step 4: Calculate ϕ_i^L and ϕ_i^V using Peng-Robinson equation of state for instance

Step 5: Update K_i

$$K_i^{new} = K_i^{old} \frac{f_i^L}{f_i^V}$$

Step 6: Check convergence

$$\sqrt{\sum_i^m \left(1 - \frac{f_i^V}{f_i^L}\right)^2}$$

II.3 Newton's Method

Let $g_i = \frac{\left(f_i^L - f_i^V\right)}{1-\alpha}$, then we instead solve the nonlinear equation $g_i\left(y_i, \alpha\right) = 0$. In matrix format, we have

$$\begin{pmatrix} \frac{\partial g_1}{\partial y_1} & \frac{\partial g_1}{\partial y_2} & \cdots & \frac{\partial g_1}{\partial y_{m-1}} & \frac{\partial g_1}{\partial \alpha} \\ \frac{\partial g_2}{\partial y_1} & \frac{\partial g_2}{\partial y_2} & \cdots & \frac{\partial g_2}{\partial y_{m-1}} & \frac{\partial g_2}{\partial \alpha} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial g_{m-1}}{\partial y_1} & \frac{\partial g_{m-1}}{\partial y_2} & \cdots & \frac{\partial g_{m-1}}{\partial y_{m-1}} & \frac{\partial g_{m-1}}{\partial \alpha} \\ \frac{\partial g_m}{\partial y_1} & \frac{\partial g_m}{\partial y_2} & \cdots & \frac{\partial g_m}{\partial y_{m-1}} & \frac{\partial g_m}{\partial \alpha} \end{pmatrix} \begin{pmatrix} \Delta y_1 \\ \Delta y_2 \\ \vdots \\ \Delta y_{m-1} \\ \Delta \alpha \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{m-1} \\ g_m \end{pmatrix} \tag{II.3.1}$$

The updating is simply

$$y_i^{n+1} = y_i^n + \Delta y_i \qquad i = 1,2,\ldots,m-1 \tag{II.3.2}$$

$$y_m^{n+1} = 1 - \sum_i^{m-1} y_i^{n+1} \tag{II.3.3}$$

$$x_i^{n+1} = \frac{z_i - \alpha^{n+1} y_i^{n+1}}{1 - \alpha^{n+1}} \qquad i = 1,2,\ldots,m \tag{II.3.4}$$