The Thesis Committee for Hector Emilio Barrios Molano
Certifies that this is the approved version of the following thesis:

Development of a Framework for Parallel Reservoir Simulation

APPROVED BY

SUPERVISING COMMITTEE:

Kamy Sepehrnoori, Supervisor

Lee Chin

Development of a Framework for Parallel Reservoir Simulation

by

Hector Emilio Barrios Molano

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University Of Texas At Austin

August 2017

To my beloved parents and brother

Acknowledgments

I would like to express my deepest appreciation and gratitude to my supervisor, Dr. Kamy Sepehrnoori for his continuous guidance, support and encouragement through this research.

I am indebted to Dr. Lee Chin, my second reader, for providing valuable comments and feedback on my thesis.

I would like to specially thank Dr. Francisco Marcondes for his valuable comments and advice through my thesis. Also, I would like to thank Mojtaba Ghasemi Doroh for his help and knowledge sharing on UTCOMPP.

I would like to thank Dr. Chowdhury Mamun for his careful review and comments of my thesis.

I sincerely appreciate the help and support provided by the TACC staff.

Furthermore, I would like to thank PGE staff for their administrative and technical support; specially to Tim Guinn, John Cassibry and Amy Stewart.

I appreciate the financial support provided by the members of the Reservoir Simulation Joint Industry Project (RSJIP) at the Center for Petroleum and Geosystems Engineering at The University of Texas at Austin.

I am very grateful to all my friends and officemates for the great time that has been grad school.

Development of a Framework for Parallel Reservoir Simulation

Hector Emilio Barrios Molano, M.S.E.
The University of Texas at Austin, 2017

Supervisor: Kamy Sepehrnoori

Parallel reservoir simulation is a topic of special interest to reservoir engineers and reservoir simulator developers. Parallel reservoir simulators provides several advantages over non-parallel reservoir simulators, such as

- Capability to run bigger models.
- Capability to have simulation results faster by using several processing units at once.
- Not limited to single computer memory. Memory available increases as more computers are used.

All these are compelling reasons for reservoir engineers. However, for reservoir simulator developers, the creation of a parallel reservoir simulator is a more complex task than non-parallel simulators. Problems related to parallel implementation such as parallel communication, model division among processors, and the management of data distributed among processors, among others should be addressed and solved on top of the already complex task of simulator development. Hence, development time for parallel reservoir simulators is more time intensive than the traditional development on single processor computers.

The objective of this work is to separate the development focus of parallel reservoir simulators in two: parallel development and reservoir simulator development. To achieve such separation, a parallel framework was developed. The framework developed in this work implements and handles the parallel complexity and provides easy

to use programming interfaces to accelerate the development of new parallel reservoir simulators or the parallelization of existing ones.

The University of Texas Compositional Simulator (UTCOMP) was used with the framework to create a new parallel reservoir simulator. Several cases were used to verify accuracy, to assert usability and to test parallel performance on our new parallel reservoir simulator. The parallel reservoir simulator developed in this work has all of UTCOMP's features and is able to run models with up to 102.4 million cells using up to 1024 processors.

Table of Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

Reservoir simulation is widely used in reservoir development. A reservoir model serves as a virtual sandbox in which many scenarios can be evaluated. The results from a reservoir simulation study aid in the selection of the best scenarios to be applied to the real reservoir. These reservoir studies can require many simulation runs (hundred or thousands). Each of these simulations is computationally expensive and requires long time to finish. Additionally, over the years there is an increasing trend in reservoir model size (could be multi million cells for full field high resolution cases) and complexity of the processes simulated (for example enhanced oil recovery processes). Consequently, the time required to finish a single simulation run could be in the order of days or weeks. Given that a reservoir study could require hundreds of simulations; if a single simulation takes days to finish, it could make the reservoir study unpractical. A solution to this problem is to use parallel reservoir simulation. Parallel reservoir simulation makes use of several computer processors at the same time. The reservoir model is divided into several pieces and each piece is assigned to a computer processor. The actual time required to finish the simulation is thus reduced because the work load is divided into several processors.

From the point of view of reservoir simulator development, generation of a parallel reservoir simulator requires the following additional points of consideration during its development:

- Define the best way to divide the reservoir model.
- Allocate memory for each processor.
- Transfer data among processors
- Give and retrieve information from processors.

- Display results from scattered data among processors.

These added requirements create an additional level of complexity to the already daunting task of reservoir simulator development; hence, increases simulator development time.

The objective of this work is to create a framework that provides the necessary tools to facilitate removal of the complexity in the development of a parallel reservoir simulator. The resulting framework needs to include the following capabilities:

- Completely in Fortran language
- Modular
- Extensible
- Use of simulator code already developed

Furthermore, We applied our new framework to UTCOMPP a previously parallelized version of the UTCOMP simulator to create a new parallel compositional reservoir simulator with improved parallel features. Several cases are created to assert the usability and the parallel performance of the new simulator.

1.1 Brief Description of Chapters

Chapter 2 sets the background information for parallel computing, parallel reservoir simulation and literature review about parallel reservoir simulation.

Chapter 3 describes in detail the methodology used to develop the framework of this study.

Chapter 4 gives an overview of UTCOMP simulator and the changes made to UTCOMP in order to make it work with the framework.

Chapter 5 shows the cases and the results used for verification and parallel performance testing.

Chapter 6 summarizes and concludes the present work; it also gives recommendations for future works.

Chapter 2

Background and Literature Review

2.1 Parallel Computing

Parallel computing in a broad sense is the use of multiple compute resources simultaneously to solve a computational problem. In computation and memory intensive applications like engineering problems, parallel computation is focused on the use of multiple processing units (processors) aiming to decrease wall time. The problem is divided into smaller parts and each part is assigned to one processor. Each processor solves its assigned part and communicates the solution to the other processors (Figure 2.1).

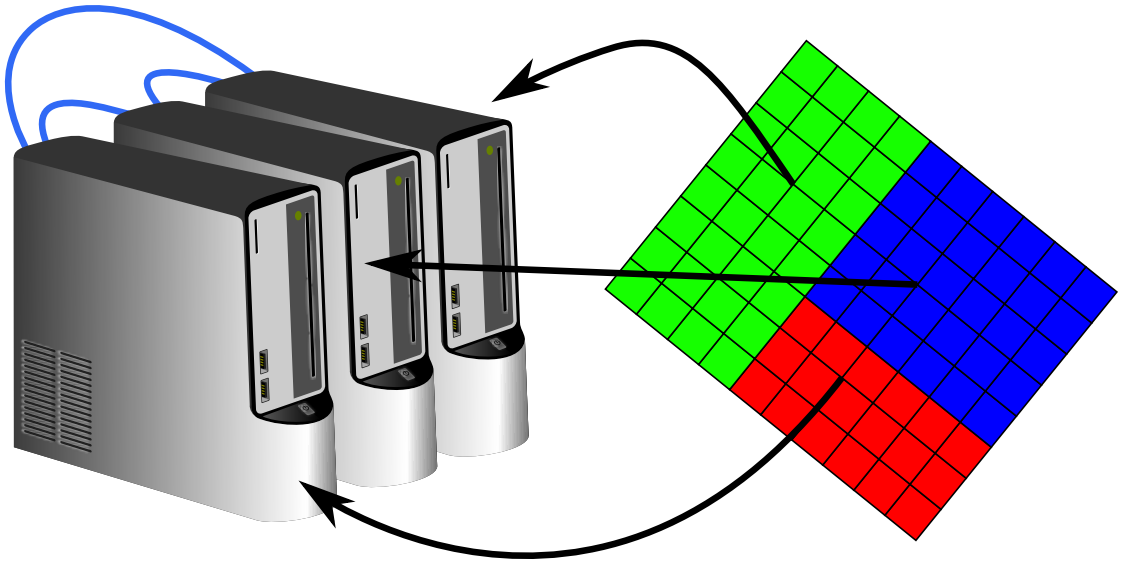


Figure 2.1: Schematic of parallel computing

2.1.1 Why Parallel Computing?

If we wanted to build the most powerful single computer computer, sooner or later our performance will be limited by physical limits such as the speed of light and the effectiveness of heat dissipation. Additionally, the cost of advanced single processor

computers increases more rapidly than their power (Gropp, Lusk, and Skjellum 2014). Parallel computing provides solutions to this problems; this is one of the reasons parallel computing has become a mainstream technology found from smartphones up to the world's largest supercomputers.

Parallel computing provides the following advantages (Willmore 2013)(Blaise 2015a):

- Decreased time to solve problems. We are using more resources, and hence, we have more computing power available than using one computer; thus the time required to solve a problem will shorten.
- Increased capability. Using multiple computers increases the amount of memory available. It is possible to solve problems that cannot fit in a single computer's memory.
- Low price/performance ratio. Parallel computers can be built from commodity components; this makes the price/performance ratio lower than a single computer with similar performance.

2.2 Performance of Parallel Applications

2.2.1 Performance Measurements for Parallel Performance

It is necessary to have a set of measurements that can be used as a basis to quantify the degree of performance of a parallel application. Additionally, these measurements can be used to identify parallel inefficiencies and optimize the parallel application performance. Furthermore, these measurements are valuable when comparing two or more parallel applications (Malony 2011).

2.2.1.1 Time measurements

Timing of parallel applications is one of the basic measurements performed. Time can be reported in different ways, depending on which part of the application is timed. *Total execution time* represents the elapsed time from the start of the application to its end. *CPU time* is the time when the process is being executed by the CPU; this excludes times like input/output time or idle time. There are also application specific

times that focus on certain parts of the application. These application specific times help to identify which part of the code could represent bottlenecks.

2.2.1.2 Speedup

Speedup, defined in equation 2.1, expresses the performance improvement as parallelism increases (Malony 2011). T_s is the serial execution time of the application; T_p is the execution time taken when p processors are used. If $S_p > 1$ there is performance improvement; if $S_p < 1$ there is a performance degradation. If a program is ideally parallel then $S_p = p$. In the case $S_p > p$ it is called superlinear speedup. Superlinear speedup can be obtained if the amount of memory per processor required to solve the problem can fit in the cache (Gustafson 1990).

$$S = S_p = \frac{T_s}{T_p} \tag{2.1}$$

2.2.1.3 Efficiency

Efficiency is defined as the ratio of speedup to the number of processors (equation 2.2). This represents a return on investment on parallelism. For example, it is not the same to have a speedup of 10 using 16 processors than the same speedup with 64. Using 64 processors represents a higher investment over 16 processors to obtain the same speedup; this is reflected on the efficiency $E_{10} = 0.625$ and $E_{64} = 0.156$.

$$E_p = \frac{S_p}{p} \tag{2.2}$$

2.2.2 Theoretical Limits on performance

2.2.2.1 Amdahl's Law

For a given parallel program, there is a serial fraction of the program that cannot be parallelized f_s; for example, output printing. The rest of the program can be parallelized f_p. Amdahl's law (Amdahl 1967) gives the speedup for a program run with p processors as

$$S_p = \frac{1}{f + \frac{1-f}{p}} \tag{2.3}$$

This gives a limit on the maximum speed up that can be obtained for a fixed size problem. This limit will be highly dependent on the serial fraction of the program (see Figure 2.2). It is worth noting that Amdahl's law treats the cost of communication between processors as negligible (Gustafson 2011).

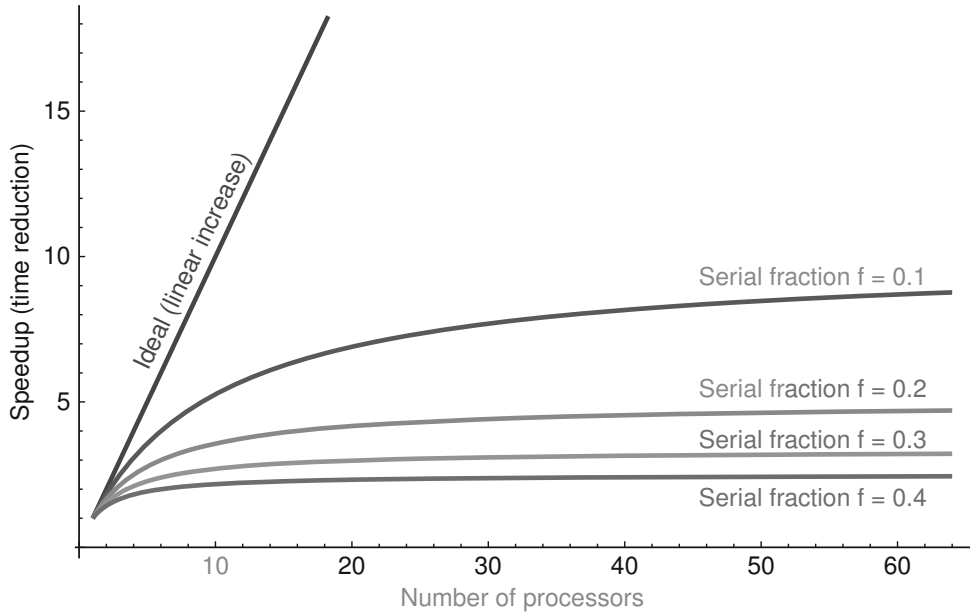


Figure 2.2: Speedup from Amdahl's law for different serial fractions (Gustafson 2011)

2.2.2.2 Gustafson's Law

Gustafson's law (Gustafson 1988) is defined in equation 2.4; where f is the serial fraction of the program and the program is run in P processors. This gives a limit on maximum speedup when the computational work increases with increasing number of processors. This theoretical limit, contrary to Amdahl's law, is not limited by the serial faction of the program. Figure 2.3 shows speedups computed using Gustafson's law for different serial fractions.

$$S_P = f + P(1 - f) = P - f(P - 1) \tag{2.4}$$

2.2.3 Scalability

Scalability refers to the capability of the parallel program to achieve good speedups with increasing number of processors. *Strong scalability* is associated with

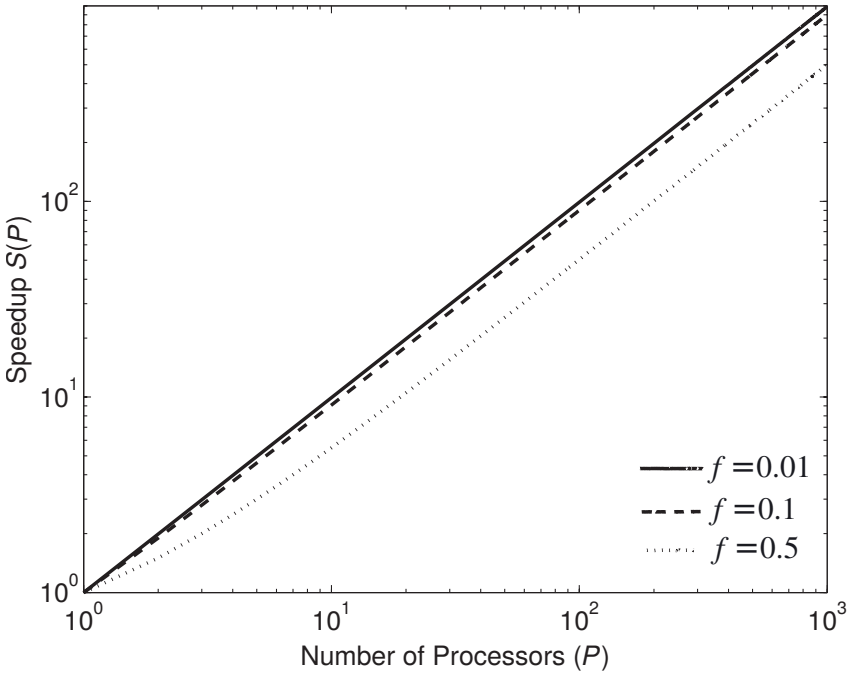


Figure 2.3: Speedup from Gustafson's law for different serial fractions (Gebali 2011)

Amdahl's law. It measure the speedup of a fixed problem with increasing number of processors. A strong scalability test solves the question of how quickly the results from a particular input data can be obtained by increasing the number of processors. *Weak scalability* is associated with Gustafson's law. It measures the elapsed time for an input data that increases proportionally to the number of processors. A weak scalability test gives an estimate of the biggest model that we could run with our parallel program in a specific hardware.

2.3 Parallel Architectures

Parallel computers can have different architectures. Before describing the most common parallel architectures, we will talk about the terminology used when referring to parallel computers.

2.3.1 General Parallel Computer Terminology

There are some terms commonly associated with parallel computing. Sometimes the same term refers to different things. Figure 2.4 shows the terms used in parallel computing.

- *Core* also referred to as *processor* is the unique execution unit.
- *Multicore processor* or *CPU* or *Processor* or *Socket* is the processing unit formed by multiple cores.
- *Computing node* or simply *node* is a complete computer in a box. It usually contains multiple multicore processors.
- *Supercomputer* or *High performance clusters* is the ensemble of several computing nodes connected through a network.

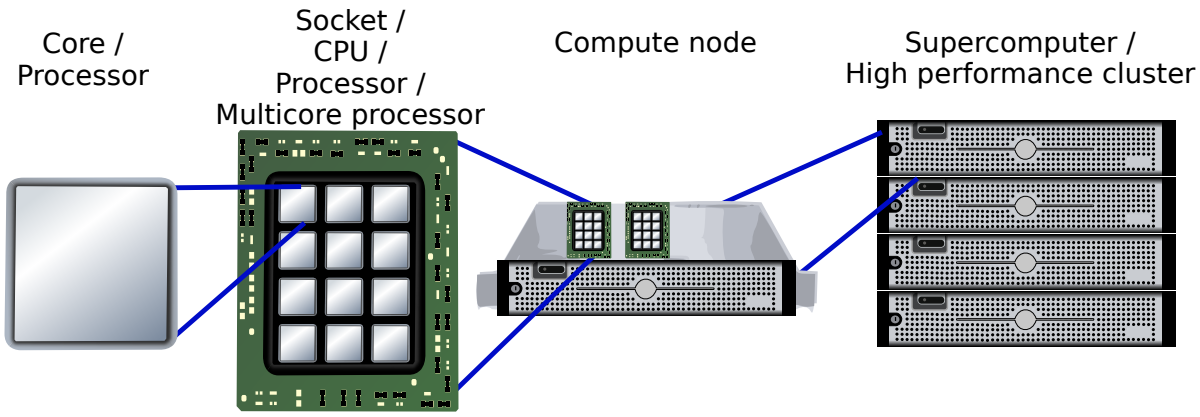


Figure 2.4: Different terms used in parallel computing

2.3.2 Flynn's Taxonomy

Flynn (Flynn 1972) created a widely used classification of parallel computer architectures based on the concurrency in processing instruction streams and data streams (Figure 2.5):

- SISD - single instruction, single data stream, only one instruction is executed on one data stream at a time; for example minicomputers, single processor/core PCs.
- SIMD - single instruction, multiple data stream, all processing elements execute the same instruction on a different data element; for example, array processors and vector processors.

- MISD - multiple instruction, single data stream, multiple instructions are executed on the same data stream; the output of one processing element is the input of the next one; for example GPUs and data flow machines.
- MIMD - multiple instruction, multiple data stream, every processing element may execute a different execution stream over a separate data stream; for example multicore or multithreaded multiprocessors.

2.3.3 Shared Memory

A shared memory parallel computer is composed by multiple independent processors that share a common memory address space and communicate with each other via memory (Ceze 2011). Figure 2.6 shows an shared memory system. In this kind of systems the communication between processors is done implicitly through read and write operations to a common memory address. The big advantage of this architecture is that the data does not need to be divided and all processors have access to the same data. The main disadvantage is that the communication is done implicitly; thus it is up to the developer to explicitly ensure adequate access to memory for all processors. Another problem of this architecture is that the hardware cost increases abruptly with increased memory capacity.

2.3.4 Distributed Memory

A distributed memory parallel computer is composed of multiple processors, each with its own memory address space. The processors are connected via a network (Snir 2011). Figure 2.7 shows a distributed memory system. Distributed memory systems can be built from commodity components, for example, a beowulf cluster (Sterling et al. 1999); or from specialized hardware like the most powerful supercomputers of today. The main advantage of this kind of architecture is its low price/performance ratio compared with shared memory systems. The main disadvantage is that the data needs to be divided among processors making the development of applications more complex.

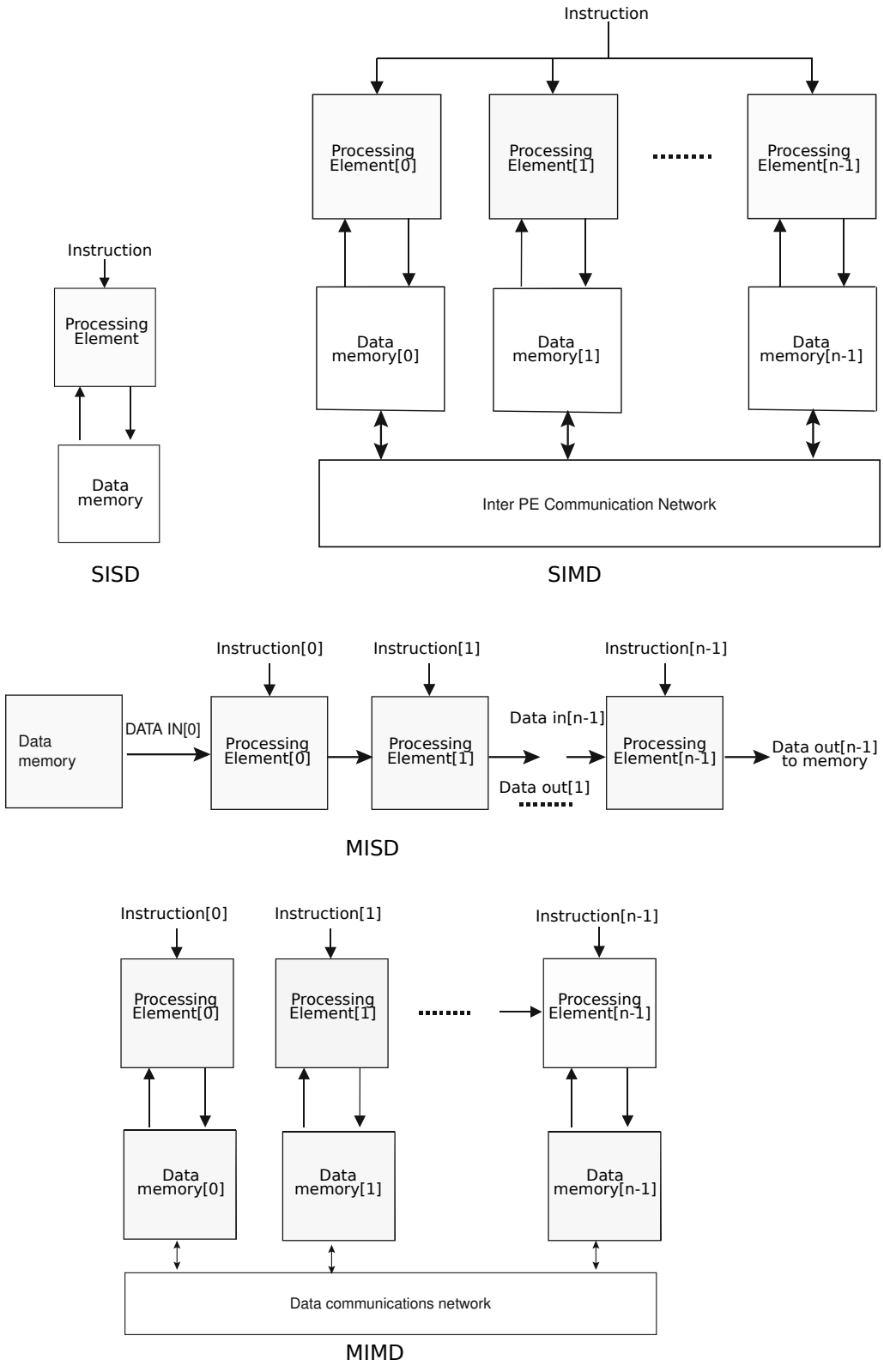


Figure 2.5: Flynn's taxonomy (Flynn 2011)

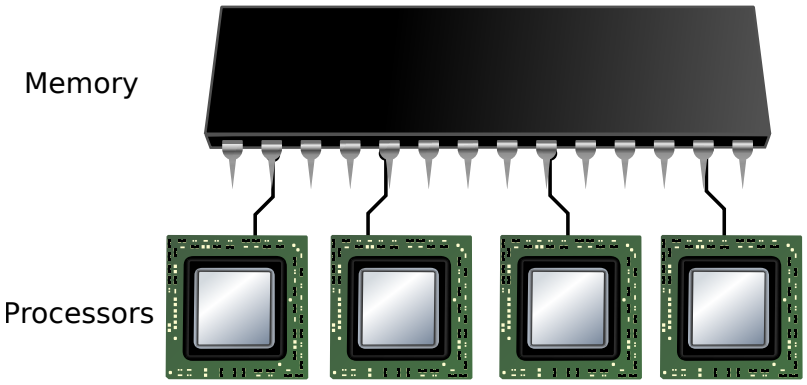


Figure 2.6: A shared memory system

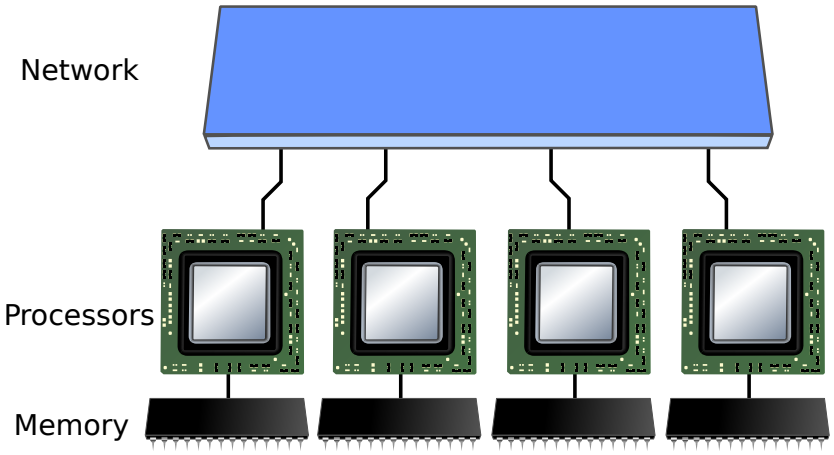


Figure 2.7: A distributed memory system

2.3.5 Hybrid Systems

Hybrid systems try to take the best of each architecture. A hybrid system consists of multiple shared memory systems connected through a network.

2.3.5.1 Multicore Systems

This kind of architecture became commonplace with the commercial availability of processors with multiple processing units (cores). This architecture consists of multiple systems with multiple core processors. Each system has its own memory address. Communication between cores outside the local system is done via network. Figure 2.8 shows a multicore system. The main disadvantage is that communication among cores becomes increasingly complex.

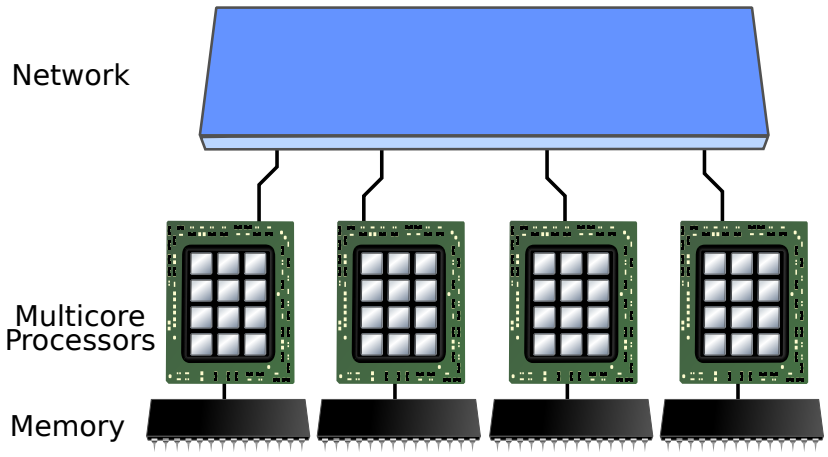


Figure 2.8: A multicore system

2.3.5.2 Accelerated Systems

This is an extension to the multicore architecture. Each system that composes the parallel system contains an accelerator (Figure 2.9). An accelerator is a self-contained system with its own memory and many simple processing units optimized to work in parallel that may contribute to vector-style parallelism. An accelerator could be a Many Integrated Core (MIC) or a General Purpose Graphical Processing Unit (GPGPU). The advantage of an accelerated system is that the accelerator boosts the performance of the system. The problem with this kind of systems is that there is the additional complexity to load and retrieve data and execution orders to and from

the accelerator.

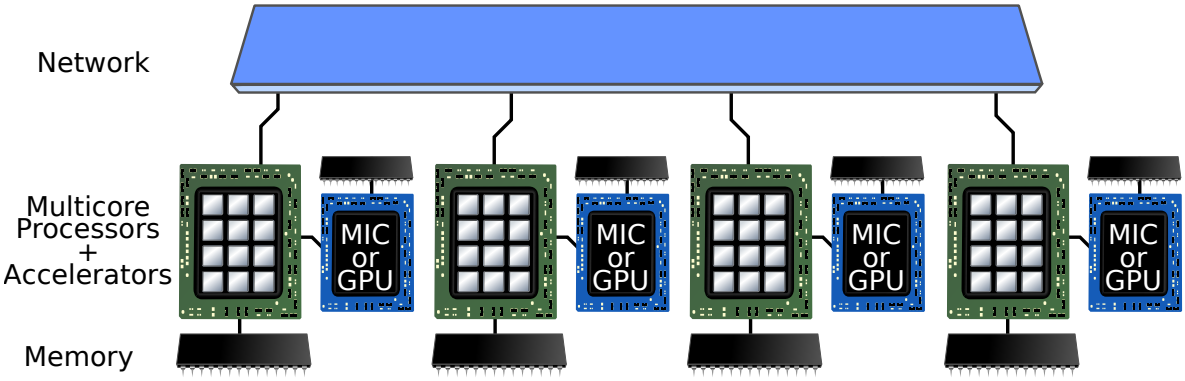


Figure 2.9: An accelerated system

2.4 Parallel Programming

In order to make use of parallel computers, we need to create programs that can be executed in those machines. Following there is a description of the available programming approaches for the architectures described earlier.

2.4.1 Shared Memory

Open Multi-Processing (OpenMP) is an application program interface (API) used to develop parallel programs in shared memory systems. It can be used with Fortran, C and C++. It provides several compiler directives, a runtime library, and environment variables (Chapman and LaGrone 2011). OpenMP uses a multithreading method called the *fork-join* model in which the program initially starts as a single process and whenever a portion of the program can be executed in parallel the process is "forked" into several processes and once the parallel part is done all processes "join" into a single process (see Figure 2.10) (Blaise 2015b). This approach to parallel programming is very convenient, specially when there is already a base code that runs serially. This is because there is no need to change dramatically the code to parallelize it. OpenMP Architecture Review Board (2015) describe the complete OpenMP API and Chapman, Jost, and Van Der Pas (2008) provides a more in depth explanation on how to use OpenMP.

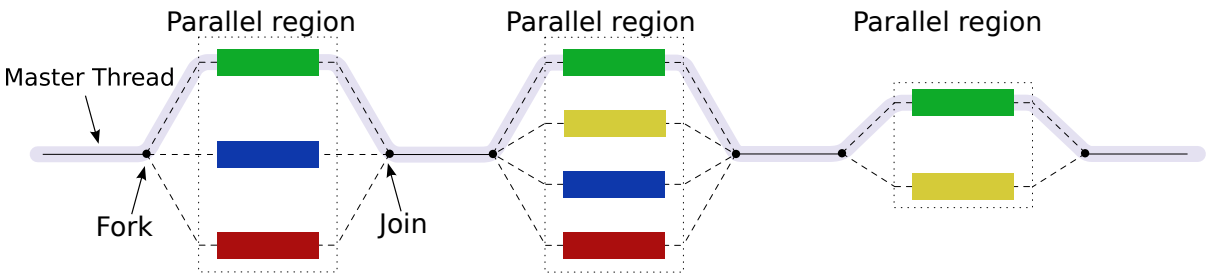


Figure 2.10: Example of fork-join model (Blaise 2015b)

2.4.2 Distributed Memory

For distributed memory systems, the most used approach is message passing. In message passing the processors communicate by sending and receiving messages. When a processor sends data from its local memory to the local memory of another, it requires operations from both processors (Gropp, Lusk, and Skjellum 2014). Message Passage Interface (MPI) is an standardization of the message passage model and defines the application program interface (API) (Gropp 2011). The API defines interfaces for Fortran and C languages. The MPI standard has been implemented in several libraries, such as OpenMPI, MPICH, Intel MPI, and Cray MPI, among others.

Another model used for distributed memory systems is remote memory access (RMA) or one-sided communication which is a half way between the shared-memory model and the message passage model (Gropp, Hoefler, et al. 2014). In RMA operations only one process specifies source and destination of the communication usually defined as *put/get* operations. To take fully advantage of this approach the underlying hardware needs to support put and get operations. RMA operations were introduced in MPI-2 standard and extended in MPI-3 (Message Passing Interface Forum 2015).

2.4.3 Multicore Systems

Multicore systems can be programmed using solely MPI. For this case even the cores within the same node will communicate through message passing instead of a shared memory approach.

Another approach is to use MPI to handle internode communication and OpenMP for parallelization whitin each node.

2.4.4 Accelerated Systems

For accelerated systems there is the need to program the accelerator on top of the multicore system. There are many platforms for programming accelerators; some of them are vendor-dependent. Few of this platforms use programming languages specific to the accelerator hardware; thus, there is the overhead for the developer to learn the new language and to port existing code to the accelerator platform. Intel's MIC can be programmed using MPI or OpenMP. GPUs can be programmed using CUDA or OpenCL.

2.5 Parallel Reservoir Simulation

Development of parallel reservoir simulators has been advancing in tandem with the new technological advancements in parallel computing. Since the early days of parallel vector processor systems, reservoir simulator developers have been evaluating, porting, developing, and optimizing parallel reservoir simulators to these parallel machines. These works can be categorized into linear, non-linear solvers and preconditioners, new simulator development, conversion of non parallel simulators, and development of parallel frameworks.

Chien, Wasserman, et al. (1987) is the earliest work on parallel reservoir simulators. They developed a general purpose reservoir simulator in a multiple vector processor machine. They used microtasking which creates several independent tasks in a shared memory environment. Killough and M. F. Wheeler (1987) studied domain decomposition in iterative linear system of equation solver algorithms and performed tests on multitasking and microtasking based shared memory parallel systems. Scott et al. (1987) studied matrix creation, sparse matrix solution, and several SOR algorithms in multiple MIMD computers with shared and distributed memory. Kårstad, Henriquez, and Korsell (1988) converted a previously vectorized reservoir simulator to be used in a parallel machine that used macrotasking shared memory model. Barua and Horne (1989) proposed a method for solution of non-linear system of equations combining an iterative solution of jacobian and a quasi-Newton method. Parallel tests of the method were performed in a 16 processor MIMD shared memory system. Mayer (1989) evaluated the applicability of reservoir simulation runs in a SIMD computer. The computer consisted of a front end node that stores the code and gives instructions

to the other nodes, and of up to 65636 extremely simple 1-bit processor nodes; the processor nodes performed operations given by the front end node in their local data. Van Daalen et al. (1989) parallelized Bosim, an IMPEC serial reservoir simulator to run in a distributed MIMD computer. The communication was performed by the Occam library, a native message passing library for the parallel system used. This simulator was further developed by Meijerink et al. (1991).

Briens et al. (1990) developed the Sequential Staging of Tasks algorithm to solve large linear systems of equations in parallel. Tests were performed in a parallel computer with 6 vector processors. J. A. Wheeler and Smith (1990) created a 3d, two phase implicit simulator; they tested it in a 16 vector processors distributed memory parallel system. Each processor was given a portion of the reservoir model to solve. Killough and Bhogeswara (1991) adapted a commercial three phase, EOS based compositional reservoir simulator to work in a 32 processor distributed memory parallel system. Wallis, Foster, and Kendall (1991) developed a preconditioner based on domain decomposition and nested factorization and a linear solver based on the generalized conjugate residual method for parallel shared memory computers. Cheshire and Bowen (1992) developed a parallel solver based on nested factorization and domain decomposition similar to (Wallis, Foster, and Kendall 1991) with the difference of being simpler and avoiding the use of complex numbering schemes. Rutledge et al. (1992) developed a three phase, 3d, IMPES compositional simulator for a SIMD computer with up to 65536 1-bit processors. They were able to run up to 2 million grid blocks. Sherman (1992) proposed to use dynamic load balance for the phase behavior calculations and static load balance for the rest of the simulation for MIMD parallel computers. Chien and Northrup (1993) developed a vectorized, parallel processed algorithm for local grid refinements and the adaptive implicit scheme. The algorithm was tested in a shared memory system. Kremer and Ramé (1994) studied the use of the Fortran D parallel language to make the process of parallelization of existing code easier. As a test case, the subroutine `DISPER` from the UTCOMP simulator was converted from vectorized to parallel. The test was run in a distributed memory system. Ghori et al. (1995) ported the UTCOMP simulator to a distributed memory parallel system using message passing for communication between processors. Additionally, UTCOMP was also ported to a distributed memory system with a shared memory programming model.

Michielse (1995) evaluated a parallel multigrid method using Parallel Virtual Machine (PVM) software. PVM allows several heterogeneous computers connected to a network to be treated as a single parallel computer. Rame and Delshad (1995) modified UTCHEM, a highly vectorized chemical compositional simulator to run in several distributed memory parallel computers. They performed a domain decomposition of the reservoir model and each processor was responsible for solving its assigned part. Communication between processors was done using message passing libraries native for each parallel machine used.

After this point, the introduction of MPI and later of OpenMP made it possible to use a standard for distributed and shared memory parallel programming instead of using native and non-standard libraries. Parashar et al. (1997) and P. Wang, Yotov, et al. (1997) developed a fully implicit equation of state compositional parallel simulator. They separated the reservoir simulator development from the parallel development. P. Wang, Yotov, et al. (1997) showed the mathematical formulation and the numerical solution techniques used in the simulator. And Parashar et al. (1997) described the parallel solving environment, called Integrated Parallel Accurate Reservoir Simulator (IPARS). This environment used different programming languages such as C and FORTRAN. The communication between processors was done by MPI. They also integrated Distributed Adaptive Grid Hierarchy (DAGH), an object oriented data management infrastructure into IPARS framework. P. Wang, Balay, et al. (1999) developed the General Purpose Adaptive Simulator (GPAS), a fully implicit parallel equation of state compositional simulator. They used Peng-Robinson EOS and PETSc linear solvers. The simulator was developed under the IPARS framework. Test cases used up to 4 million cells in a dry gas injection process. Abate, P. Wang, and Sepehrnoori (2001) used the GPAS simulator to evaluate the suitability of using clusters of PCs for parallel reservoir simulation. GPAS has been extended for a surfactant phase behavior chemical model (John et al. 2004), a fully implicit chemical flood model (Han et al. 2007), a multiple-interacting-continua dual porosity model (Naimi-Tajdar et al. 2007), a coupled finite element method geomechanics model (Pan, Sepehrnoori, and Chin 2007), an asphaltene precipitation model (Fazelipour, G. A. Pope, and Sepehrnoori 2008), and a three phase extended chemical model (Delshad et al. 2009).

W. Liu et al. (2000) developed a preconditioned Krylov subspace method with

hybrid preconditioner based on domain decomposition to solve linear systems of equations. This method was implemented into a fully implicit, 3d, three phase black oil reservoir simulator. MPI was used for communication. Tests were performed in shared memory and distributed memory systems. DeBaun et al. (2005) created a parallel reservoir simulator based on an object-oriented approach. Parallel processing was handled by a framework that uses MPI for communication. The simulator unifies the treatment of structured and unstructured grids. Atan, Kazemi, and Caldwell (2006) developed a multiscale, multimesh reservoir simulator that worked with shared memory parallel computers using OpenMP. The tests used up to 32 processors. Löf, Gerritsen, and Thiele (2008) developed a streamline parallel simulator for shared memory systems using OpenMP. Tests used up to 16 threads. Khait (2009) adapted Constrained Pressure Residual algorithms for shared memory multicore machines using OpenMP. Yuan, Delshad, and M. F. Wheeler (2010) developed a parallel reservoir simulator with comprehensive polymer property model based on IPARS framework. Tarman et al. (2011) developed an automatic domain decomposition framework for parallel reservoir simulation. Such framework applied several domain decomposition algorithms and selects the best resultant division. The framework was applied to a commercial reservoir simulator. Ghasemi Doroh (2012) developed a parallel version of the UTCOMP simulator using IPARS framework. Yu et al. (2012) developed a GPU based parallel preconditioner and GMRES algorithm coupled with a black oil simulator. Dogru, Fung, and Sindi (2013) developed a parallel reservoir simulator that makes use of multi-paradigm parallelization using MPI, OpenMP, and GPU directives. Maliassov, Beckner, and Dyadechko (2013) developed a three phase black oil reservoir simulator on unstructured meshes using the Trillinos project as a framework for parallel development. Y. Wang and Killough (2014) converted a MPI based parallel compositional reservoir simulator into a Charm++ (an object oriented parallel library based on MPI) based parallel simulator. They performed dynamic load balance by overdecomposing the domains into subdomains that can be dynamically migrated across processors.

Beckner et al. (2015) developed a general parallel reservoir simulator that combines different fluid systems and transport equations. Parallel development was done by a propietary data layer. Tests used up to 14 million cells and up to 16000 proces-

sors. Guan et al. (2015) developed a parallel reservoir simulator with a black oil model using the Parallel eXtension Framework (PXF). PXF is written in C++ and uses MPI for communication. Tests used up to 10000 processors and up to 120 million cells. H. Liu et al. (2015) created a parallel framework written in C, based on MPI and OpenMP for structured grids. They applied the framework in the development of a black oil reservoir simulator and tested it in simple cases with up to 140 million cells using up to 2048 processors. Based on this framework K. Wang et al. (2016) developed a multi continuum multiphase parallel reservoir simulator, and Zhong et al. (2016) developed a parallel thermal reservoir simulator. Fung and Du (2016) developed a framework to simulate unconventional reservoirs using a multiconnected multicontinuum system. The framework works in parallel computers by dividing an unstructured grid using a domain decomposition algorithm based on graphs. Shuhong et al. (2016) developed a black oil fully implicit reservoir simulator with a shared memory parallel algebraic multigrid preconditioner. Tests used up to 8.9 million cells in a two phase model and up to 8 threads.

Chapter 3

Proposed Framework

Our approach to alleviate the complexity of implementing the parallel aspects in a parallel reservoir simulator is to divide the development focus in parallel development and reservoir simulator development. This approach makes sense since most of the parallel development is independent of the reservoir simulator development.

The separation in development focus depends on the definition of a parallel framework. This framework implements and handles the parallel complexity and provides the reservoir simulator developer with high level access through defined subroutines. Also, the grouping of all parallel functions in one place makes it easy to reuse them for future simulator development.

Additionally, some of the characteristics the proposed framework should have are:

- *Modularity*; the proposed framework should be constructed in a modularized fashion. Each module can be modified without requiring to modify the other modules.
- *Maintainability*; each of the framework's modules should be easy to maintain.
- *Extensibility*; the framework should allow for an easy way to add new functionality.
- *Code compatibility*; many years of experience and effort have been invested in the development of reservoir simulators. To be able to easily reuse code already developed in parallel reservoir simulators, few modifications should be needed in order to make them compatible with the framework.

The next Section discusses the parallel development problems encountered during development of parallel reservoir simulators. Then, there is the description of the solution approach used within the framework. Followed by a description of the

framework and its organization, the final Section is a summary of current capabilities of the framework.

3.1 Parallel Development Problems

In this Section, we discuss the most important problems related to parallel implementation that arise when developing a parallel reservoir simulator. Our main aim is to have our framework capable of working on distributed memory parallel systems; thus, the problems are described in that context.

3.1.1 Parallel Communication

One of the main problems in developing parallel applications is defining the model of communication among processors. This model of communication usually is dependent on the architecture of the parallel machine; there are several standards for communication models described in Section 2.4.

3.1.2 Reservoir Model Division

One of the first steps during a simulation performed in parallel is the division or decomposition of the reservoir (domain) into small pieces. How to perform such division is a problem that needs to be solved; there are many ways to divide the domain; see Figure 3.1. The aim of a domain decomposition algorithm is to evenly distribute each domain so that the memory requirements for each processor are fairly equal, as well as the amount of computational work required to solve each domain. If the domain decomposition is not performed correctly, it could lead to load imbalance, for which the computational work is not evenly distributed and some processors will finish their assigned part earlier and will need to wait for the processor with a heavier load. Figure 3.2 shows an example of work load distribution for an unbalanced and for a balanced case. Load imbalance could be more severe in reservoirs with lots of inactive cells in which big parts of the reservoir model are not considered for computation.

Additionally, each processor needs to be aware of how many neighbors it has, and how the communication pattern among domains is going to be performed. This needs to be computed once the domain decomposition is done.

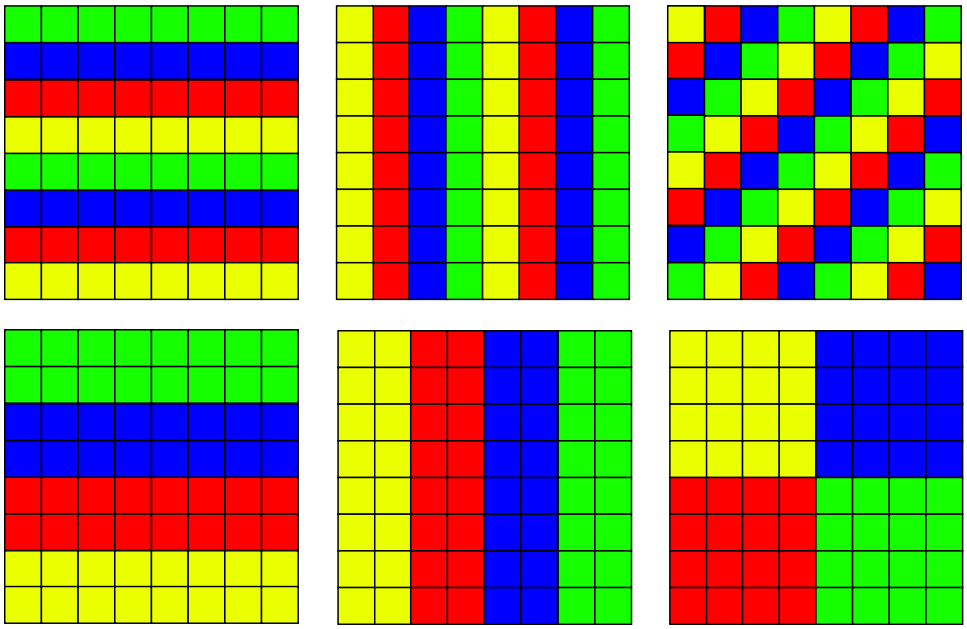


Figure 3.1: Examples of domain decomposition in 2d

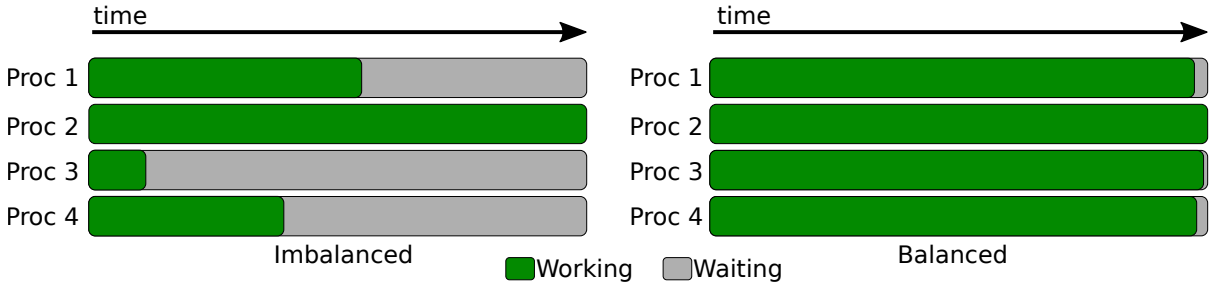


Figure 3.2: Example of two load distribution

3.1.3 Spatially Distributed Properties

Spatially distributed properties are associated with the reservoir grid, such as porosity, permeability, pressure, and compositions among others. The definition and management of these properties is greatly impacted by the parallel division of the reservoir, specifically these are the problems:

- Memory allocation for each reservoir domain in each processor.
- Association of data to a property.
- Use of data stored in a property during the simulation
- Retrieving a property from scattered to output.
- Data communication for properties.

3.1.4 Input Processing

If not done properly, the processing of input data could become a bottleneck and cause a negative impact on the performance of the parallel application. This issue is particularly important when the reservoir model contains millions of cells. Also, a way to input complex input data that is flexible and that can be adapted to further needs is required.

3.1.5 Output Processing

Output processing could become a bottleneck, specially when generating output of spatially distributed properties for visualization.

3.2 Solutions to the Parallel Development Problems

In this Section, we show the approach taken to solve each of the problems related to parallel development.

The language used for the development of the framework is Fortran. Fortran is a programming language that has been used by scientific and engineering communities since its introduction in 1957. Fortran is selected because of the excellent performance of computationally intensive programs. Standard Fortran has been in continuous revision (90/95, 2003, 2008, 2015), adding new features to the language,

such as memory management, derived types, object oriented programming, among others; always taking into account compatibility with prior versions (Metcalf, Reid, and Cohen 2011). We make use of these new capabilities during the development of the framework. Fortran 77 and older are usually referred to as *FORTRAN*, whereas the new Fortran standards (90/95, 2003, 2008, 2015) are referred to as *modern Fortran*.

Additionally, a lot of reservoir simulators has been written in Fortran, some in old standards (i.e. Fortran 77). As each new Fortran standard thrives to remain compatible with prior versions of the standard, there is no need to rewrite or convert previous written code into a different language. Such conversion would be very expensive in terms of human labor and time spent. Many programs were developed and have matured through decades of continuous enhancement and testing, some are very domain specific requiring specific knowledge just to convert them. Moreover, usually the process of conversion is not straightforward.

3.2.1 Parallel Communications

For this work, we use the MPI standard to communicate between processors. Specifically, we use the version of the MPI standard equal to or higher than 3.0 (currently the newest is 3.1) because there is better Fortran support through the module `mpi_f08`. Also there is type and argument number checking for MPI subroutines (Gropp, Hoefler, et al. 2014).

The advantages of using MPI is that it can be used in distributed and shared memory systems. It provides point to point (two-sided) and remote memory access (one-sided) communication models. It is portable across several hardware systems. Furthermore, there are many libraries that implement the MPI standard, such as OpenMPI, Intel MPI, and Cray MPI, among others.

MPI provides low level functions to perform the communications. In our framework we built functions easier to use and focused on the specific problem of reservoir simulation on top of MPI.

3.2.2 Reservoir Model Division

We used Cartesian grids throughout this work because of its simple structure and because it serves as a first implementation for a grid within the framework that

can be later extended or replaced.

We define each grid division (domain) to be a rectangular block, this is because less information is required to define the dimension of a domain. However, our approach to domain distribution is irregular; hence, the size and location of each domain is not fixed and one domain could have many domain neighbors.

In order to perform the domain decomposition, we use the Recursive Coordinate Bisection (RCB) algorithm (Berger and Bokhari 1987). RCB divides the domain into two parts with equal load using a straight cutting plane orthogonal to the axis; the model's cells are assigned to each processor based on each cell's position relative to the cutting plane. Each domain is divided recursively until the number of subdomains required is obtained. Advantages of this method are: it maintains geometric locality of cells within a processor, the resulting domains are regular and easy to describe and it is a fast algorithm. This method requires the coordinates (i,j,k) of each cell as an input in order to perform load balancing. Optionally, RCB can use a weight factor; this factor represents the amount of computational load for each cell. We use this weight factor to tell the RCB algorithm which cells are inactive and have a low impact on the load for the domain (equation 3.1). This factor could be modified further to include more properties like porosity or permeability. Figure 3.3 shows an example of domain decomposition using RCB in a grid with inactive cells. The impact on resulting domains can be observed when inactive cells are taken into account.

$$\text{Weight} = \begin{cases} 1.0 & \text{Active cell} \\ \\ 0.01 & \text{Inactive cell} \end{cases} \tag{3.1}$$

We use the implementation of the RCB algorithm from the Zoltan library. Zoltan[1] is a library that provides a suite of partition algorithms and dynamic load balancing; besides RCB algorithm also provides space filling curves, graph and hypergraph based partitioning algoritms (Devine et al. 2002). Zoltan has interfaces with Fortran, C and C++.

The method of communication between domains used during this work is through ghost layers. Each domain allocates an additional layer on its boundaries.

[1] `http://www.cs.sandia.gov/zoltan/`

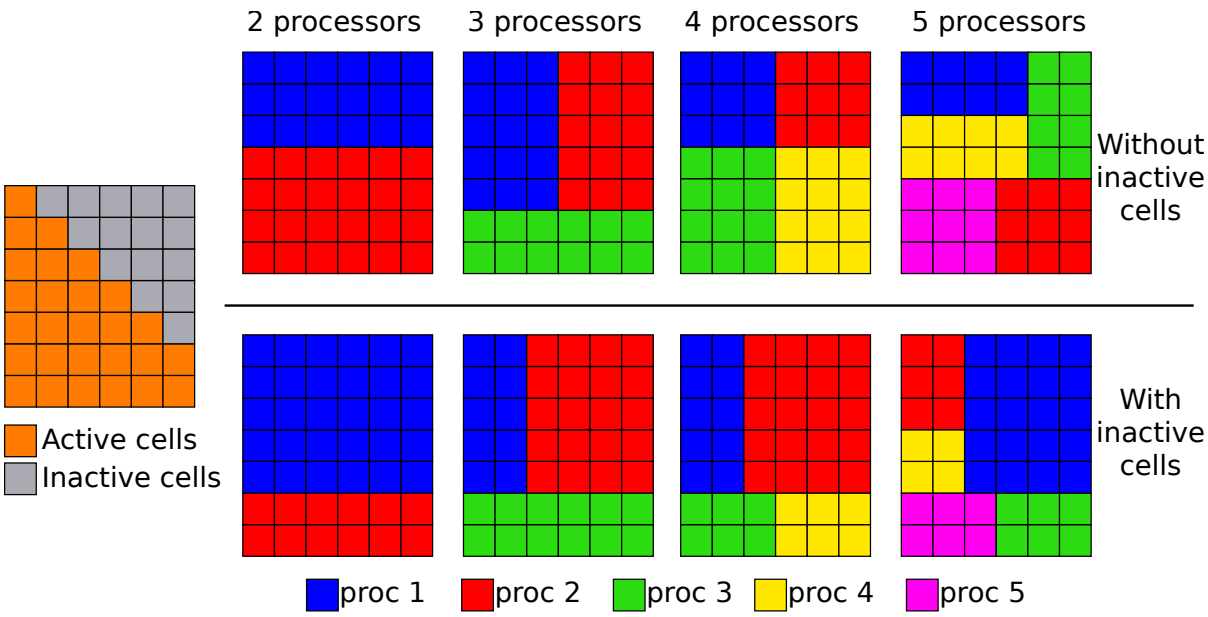


Figure 3.3: Example of domain decomposition using RCB algorithm with and without taking into account the inactive cells

These layers represent the neighboring cells owned by other processors required for local cells during the simulation. Every time the values of a property are updated in a neighbor processor, their values need to be updated in the ghost cells. Figure 3.4 shows an example of ghost layers.

After using the RCB algorithm for domain decomposition, each processor can compute the cells it owns. However, there is no information regarding how many neighbors each processor has, and how many cells should be sent to and received from each neighbor. We defined a communication setup process that all processors use to obtain this information. The process is summarized in the following steps:

- Each processor counts the number of cells that are in the edge of its domain which are not physical reservoir boundaries; hence, the edges that have a neighbor. This counting is done without repeating cells, `num_local_exportable`.
- Each processor counts the number of ghost cells that will be required from other processors, `num_local_ghost`.
- Distribute the values of `num_local_exportable` and `num_local_ghost` to all processors. Compute `num_global_exportable` and `num_global_ghost` as the sum

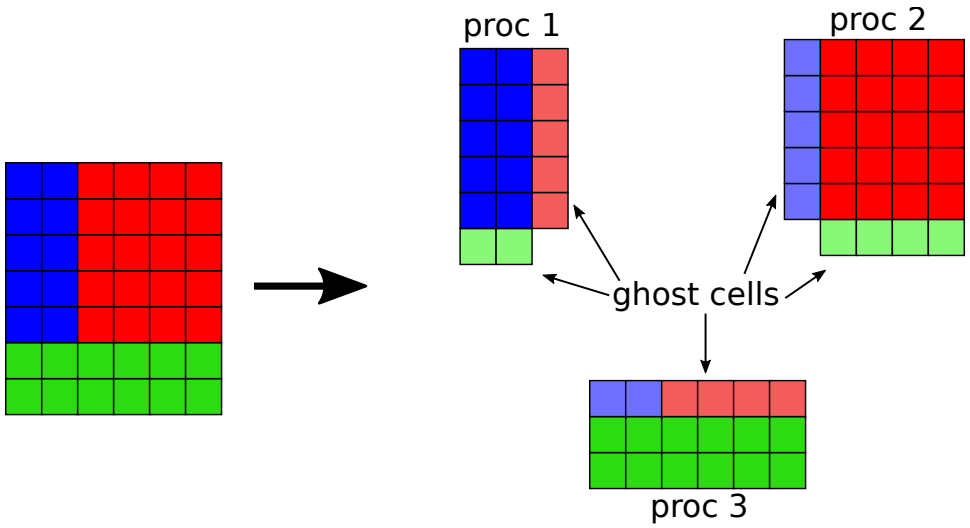


Figure 3.4: Example of domains whit ghost layers

of all exportable and ghost cells from all processors.

- Each processor creates a list of local cells that can be exported (cell in the edge of domain) to other processors `local_exportable_list`. This list contains i, j, k coordinates and the processor owner id; the size of the list is `num_local_exportable`.
- All processor gather the `local_exportable_list` from all processors into a global list of exportable cells `global_exportable_list`.
- Each processor creates a local list of ghost cells that will be required by its own domain. This is done by increasing by 1 the coordinate in the direction of the domain edge that is not a reservoir boundary (example, i+1 in the domain edge in i direction), `local_ghost_list`. This list contains i, j, and k coordinates of the ghost cell, processor id that needs the ghost cell, local index, and processor owner of cell. The owner of the cell is taken by looking in `global_exportable_list` for a cell with the same i, j, k coordinates.
- All processor gather the `local_ghost_list` from all processors into a global list of ghost cells `global_ghost_list`.
- Each processor creates a local list of cells it needs to export `local_export_list`. This list is filled by selecting the cells owned by the processor in `global_ghost_list`.

This list could contain several entries for the same cell, because cells in the corners need to be exported to more than one processor.

- Each processor uses `local_ghost_list` to count the number of neighbor processors and the number of ghost cells per neighbor processor.
- Each processor creates a list of general ghost cell information `ghost_info`. This list contains the id of the neighbor processor, id of processor needing the cell (its own id), the number of cells owned by the neighbor (number of cells that will be received), the start and end index on receiver buffer in which cells from neighbor will be placed.
- For each entry in `ghost_info`, send entry to neighbor and receive entry from neighbor and place it into `export_info`. `export_info` contains owner processor id (own id), processor needing the cell (neighbor id), number of cells that will be sent, start and end index on the neighbor processor's buffer where the cells sent will be placed.

An example of the results of the communication setup process for a two dimensional grid with three domains (Figure 3.5) is shown in Tables 3.1 and 3.2.

We defined a Fortran derived type `GridCart_type` that contains all information and type bounded procedures required to define, allocate, divide, and set up communication pattern for a Cartesian grid. By doing this, the whole process is encapsulated and there is a simple interface that the developer can use independently of the underlying implemented algorithms. Listing 3.1 shows an example using `GridCart_type`.

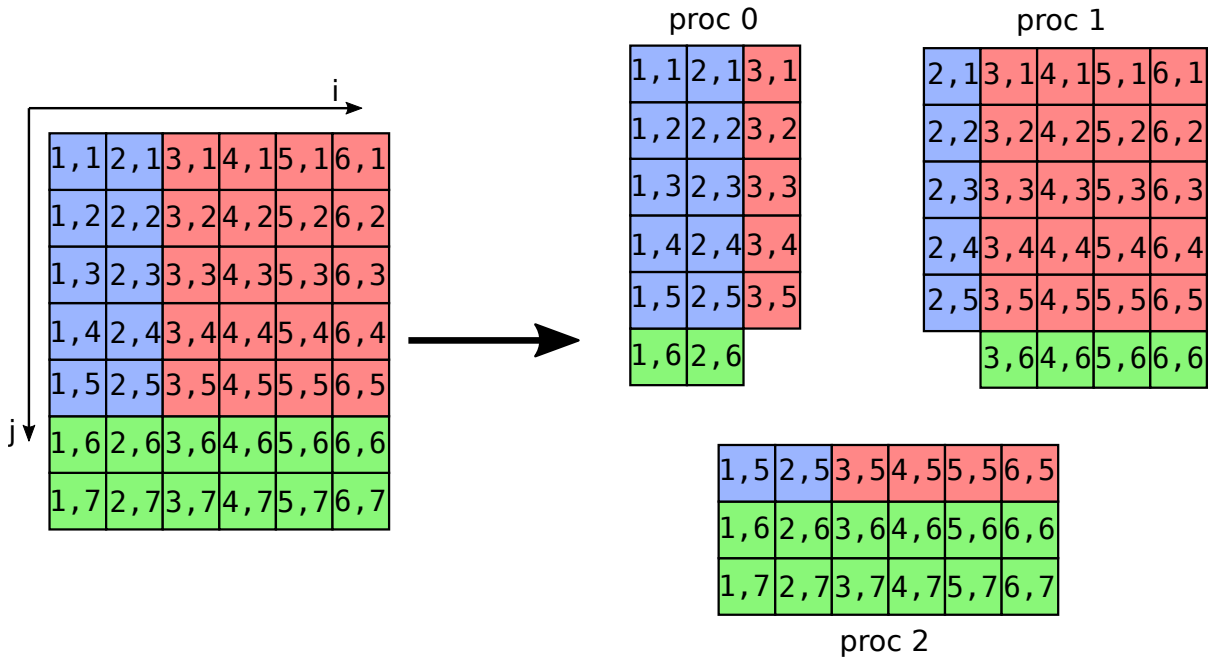


Figure 3.5: Grid and cells coordinates shown as example of communication setup between processors, results are shown in Tables 3.1 and 3.2

```
1  use iso_fortran_env
2  use mpi_f08
3  use Core_mod, only : GridCart_type
4  integer(kind=int32)          :: nx, ny, nz
5  integer(kind=int32), pointer :: keyout(:,:,:)
6  type(GridCart_type)          :: grid
7  nx = 6
8  ny = 7
9  nz = 1
10 !creates a cartesian grid
11 grid = GridCart_type(nx,ny,nz,MPI_COMM_WORLD)
12 !creates a grid property, to use during grid division
13 call grid%CreateProp("keyout",1_int32,keyout)
14 !divide grid
15 call grid%Divide()
16 !deallocate properties used during grid division
17 call grid%DeallocAllProps()
18 !nullify pointer
19 keyout => null()
```

Listing 3.1: Example use of `GridCart_type`

Table 3.1: list of export cells and ghost cells for each processor in example

	local_export_list						local_ghost_list					
				dest		owner				dest		owner
Proc	i	j	k	proc	indx	proc	i	j	k	proc	indx	proc
0	2	1	1	1	1	0	3	1	1	0	1	1
	2	2	1	1	2	0	3	2	1	0	2	1
	2	3	1	1	3	0	3	3	1	0	3	1
	2	4	1	1	4	0	3	4	1	0	4	1
	2	5	1	1	5	0	3	5	1	0	5	1
	1	5	1	2	1	0	1	6	1	0	6	2
	2	5	1	2	2	0	2	6	1	0	7	2
1	3	1	1	0	1	1	2	1	1	1	1	0
	3	2	1	0	2	1	2	2	1	1	2	0
	3	3	1	0	3	1	2	3	1	1	3	0
	3	4	1	0	4	1	2	4	1	1	4	0
	3	5	1	0	5	1	2	5	1	1	5	0
	3	5	1	2	3	1	3	6	1	1	6	2
	4	5	1	2	4	1	4	6	1	1	7	2
	5	5	1	2	5	1	5	6	1	1	8	2
	6	5	1	2	6	1	6	6	1	1	9	2
2	1	6	1	0	6	2	1	5	1	2	1	0
	2	6	1	0	7	2	2	5	1	2	2	0
	3	6	1	1	6	2	3	5	1	2	3	1
	4	6	1	1	7	2	4	5	1	2	4	1
	5	6	1	1	8	2	5	5	1	2	5	1
	6	6	1	1	9	2	6	5	1	2	6	1

Table 3.2: General information for export cells and ghost cells for each processor in example

	export_info					ghost_info				
	processor			dest buff indx		processor			recv buff indx	
Proc	owner	dest	# cells	start	end	owner	dest	# cells	start	end
0	0	1	5	1	5	1	0	5	1	5
	0	2	2	1	2	2	0	2	6	7
1	1	0	5	1	5	0	1	5	1	5
	1	2	4	3	6	2	1	4	6	9
2	2	0	2	6	7	0	2	2	1	2
	2	1	4	6	9	1	2	4	3	6

3.2.3 Spatially Distributed Properties

To solve this problem, we defined *Parallel Arrays (PArray)*. *Parallel Arrays* are arrays that are distributed among processors. They take the results from the grid division and allocate the memory needed in each processor. Access to the data contained in a Parallel Array is done through Fortran pointers.

Parallel Arrays also provide a function to update ghost cells. This process is summarized as:

- *Pack* uses `local_export_list` to fill the export buffer using the values from the export cells stored in PArray memory (Figure 3.6).
- *Exchange* sends parts of the export buffer to neighbors according to `export_info`. Additionally, it receives data from neighbors into the ghost buffer according to `ghost_info` (Figure 3.7).
- *Unpack* uses `local_ghost_list` to copy the values from the ghost buffer to PArray memory (Figure 3.8).

We defined a Fortran derived type `PArray_type` that contains all data and procedures used to allocate, deallocate, access, and update spatially distributed properties. The advantage of enclosing everything in a derived type is to encapsulate the information and define a simple interface independent of the underlying implementation. Listing 3.2 shows an example using `PArray_type`. Here are the features of `PArray_type`:

- A PArray can have up to 6 dimensions (3 spatial + 3 extra).
- One PArray can allocate integer, real or logical values, no need for specific definition.
- Simple interface to memory management (`Alloc`, `Dealloc`).
- Easy way to access and modify values through pointers (`Show`).
- Easy way to update ghost cells (`Exchng`).

```
use iso_fortran_env
use mpi_f08
use Core_mod, only : GridCart_type, PArray_type
integer(kind=int32)          :: nx, ny, nz
integer(kind=int32), pointer :: keyout(:,:,:)
type(GridCart_type)          :: grid
type(PArray_type)            :: parray3d,parray4d
integer(kind=int32), pointer :: ipoint3d(:,:,:)
real(kind=real64),   pointer :: rpoint4d(:,:,:,:)
nx = 6
ny = 7
nz = 1
!creates and divide cartesian grid
grid = GridCart_type(nx,ny,nz,MPI_COMM_WORLD)
call grid%Divide()
call grid%DeallocAllProps()
!create PArrays
parray3d = PArray_type("Array3D_name",[0,0,0],grid)
parray4d = PArray_type("Array4D",[2,0,0],grid)
!allocate integer and real PArrays
call parray3d%Alloc(1_int32)
call parray4d%Alloc(0.123_real64)
!associate pointers to PArray values
call parray3d%Show(ipoint3d)
call parray4d%Show(rpoint4d)
!exchange ghost cells among processors
call parray3d%Exchng()
call parray4d%Exchng()
!deallocate PArray and nullify pointers
call parray3d%Dealloc()
call parray4d%Dealloc()
ipoint3d => null()
rpoint4d => null()
```

Listing 3.2: Example use of `PArray_type`

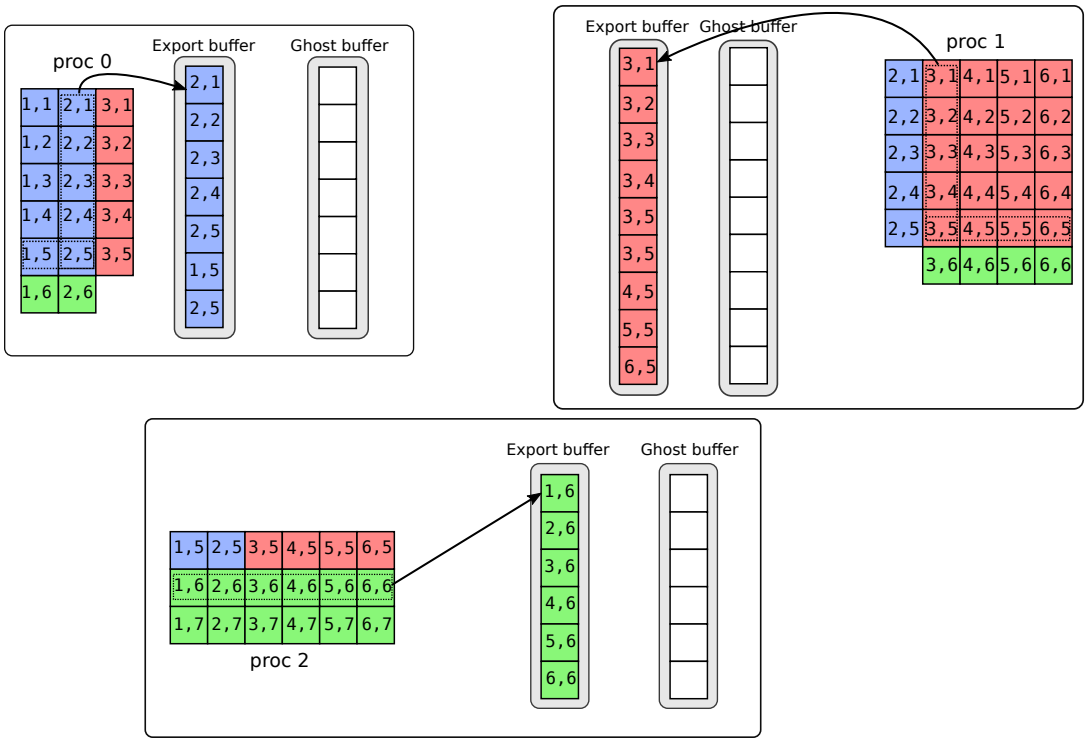


Figure 3.6: Packing of export cells into export buffer

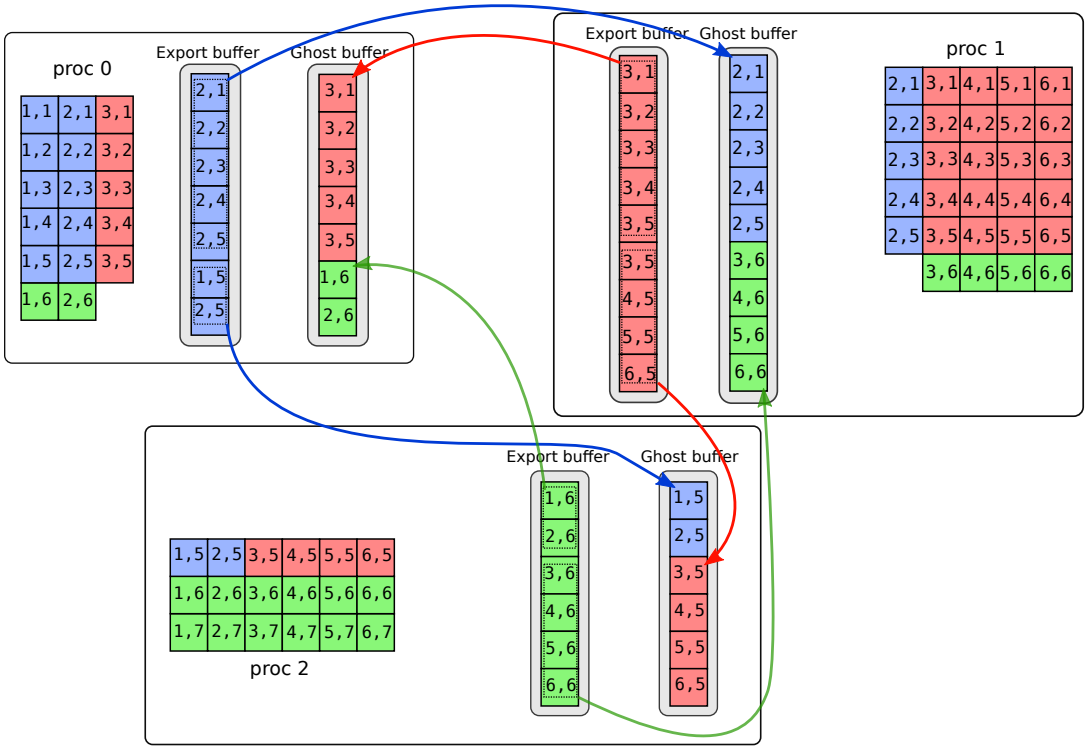


Figure 3.7: Exchange of ghost cells among processors

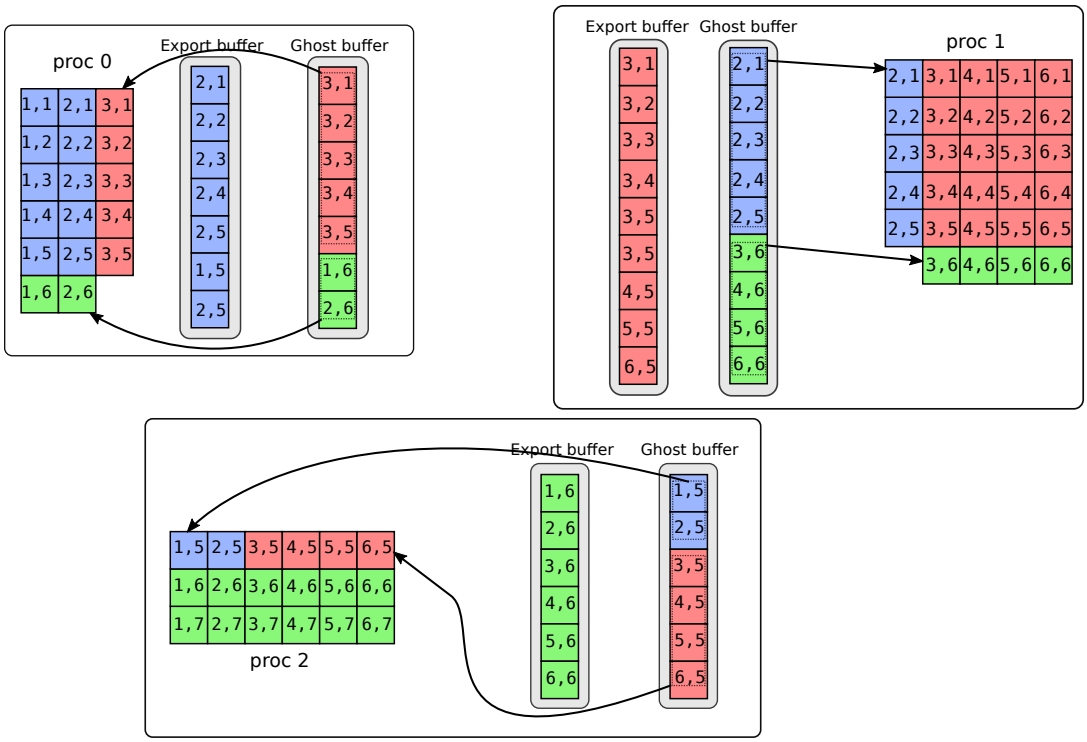


Figure 3.8: Unpacking of ghost cells from ghost buffer

3.2.4 Input Processing

The input processing for the framework makes use of Lua[1]. Lua is a scripting language designed to be embedded within applications. Lua has been used as a configuration or an extension language. It is written in ANSI C; hence, it is highly portable. Lua provides automatic memory management, dynamic typing, easy string handling, among others. Lua aims to be simple, fast, efficient and portable (Ierusalimschy, de Figueiredo, and Filho 1996). Since its first version in 1993, Lua has been in continuous development (Ierusalimschy, de Figueiredo, and Celes 2007) at PUC-Rio, the latest version is 5.3. For a more in depth description of the language see Ierusalimschy (2016).

One big advantage of Lua is its small footprint; the complete Lua library takes 513kB of disk space. Lua provides an Application Programming Interface (API) for the C language. To be able to use Lua from Fortran we use Aotus[2] (Advanced Options and Tables in Universal Scripting); it is a library that provides a Fortran wrapper for

[1] `www.lua.org`

[2] `https://geb.sts.nt.uni-siegen.de/doxy/aotus/`

the Lua's C-API using Fortran's interoperability with C from Fortran 2003 standard. Aotus is developed by the University of Siegen.

We use Lua library together with Aotus library to process the input data for the framework. On top of Aotus we developed easy to use high level Fortran subroutines for the specific input in reservoir simulation. A scheme of these functions can be seen in Figure 3.9. The biggest advantage of using a scripting language for input processing is that it is flexible and powerful; it is possible to use conditionals, loops, functions within the input data. This leaves the door open for future extensibility.

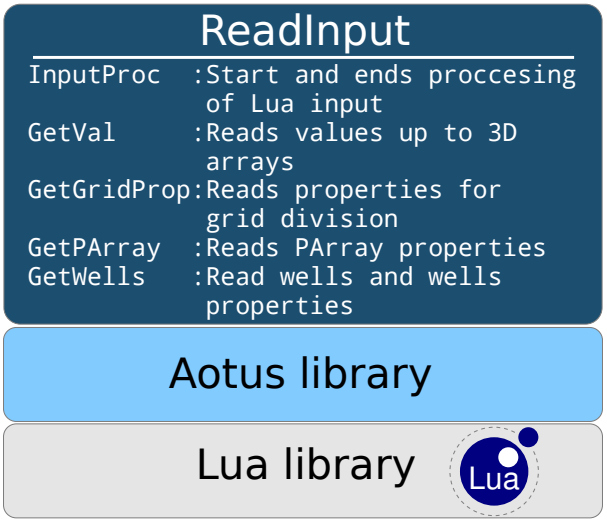


Figure 3.9: Functions developed on top of Aotus and Lua

An example of input for several types of data used in the framework in Lua is shown in Listing 3.3. Listing 3.4 shows how to read the Lua input with Fortran using the framework subroutines.

The input data processing is done by each processor. Initially processor 0 takes the input file and places it in a character array; then processor 0 broadcasts the input array to all processors. Each processor uses Lua and Aotus libraries and the subroutines provided by the framework to process the input data. It is important to note that the Lua input file does not contain explicitly data for PArrays besides when the data is a constant value (line 17 of Listing 3.3). For this reason, the Lua input file is expected to be small enough to not create a bottleneck in performance, nor create a big overhead when each processor processes it.

```lua
-- function creates a constant array
function const(n,val)
    tmp = {}
    for i = 1, n do
        tmp[i] = val
    end
    return tmp
end
--
INITIAL = {
    Final_time = 1000.0,
    COMPOUND = {"C1","C3","C6"},
    ...
    NX = 10,
    DX = const(10,25) -- creates {25,25,25,25,25,25,25,25,25,25}
    ...
    keyout   = {kind="const",value=1},
    Porosity = {kind="file",format="formatted",file="poro.inc"},
    XPerm    = {kind="file",format="unformatted",file="xperm.inc",units="md"},
    ...
    Wells = {
              [1]={name = "Injector1",
                   top  = {12.5,12.5,0.0},
                   ...},
              [2]={name = "Producer1",
                   top  = {237.5,237.5,0.0},
                   ...}
            }
}
```

Listing 3.3: Example of input data in Lua

```
use iso_fortran_env
use mpi_f08
use Core_mod, only : IniInputProc,FinInputProc,GetVal,GetGridProp,&
                     GetValLen,GetWellsProp,GridCart_type,PArray_type
real(kind=real64)             :: time_end
character(len=10),allocatable :: comp_name(:)
character(len=50),allocatable :: well_name(:)
integer(kind=int32)           :: ncomp, nx, ny, nz, nwell
real(kind=real64),allocatable :: dx(:), well_top(:,:)
type(GridCart_type)           :: grid
type(PArray_type)             :: ary_poro, ary_xperm
integer(kind=int32)           :: err = 0
character(len=100)            :: msg
...
call IniInputProc(err,emsg)
call OpenSection("INITIAL",err,msg)
call GetVal(vnam="Final_time",var=time_end,err=err,msg=msg,&
            internal_units="[day]",external_units="[day]")
call GetValLen("COMPOUND",ncomp,err,msg)
allocate(comp_name(ncomp))
call GetVal("COMPOUND",comp_name(1:ncomp),err,msg)
...
call GetVal("NX",nx,err,msg)
allocate(dx(nx))
call GetVal("DX",ddx,err,msg,internal_units="[ft]",external_units="[ft]")
...
grid = GridCart_type(nx,ny,nz,comm)
call GetGridProp("keyout",grid,"integer",err,msg)
call grid%Divide()
...
ary_poro  = PArray_type("porosity",[0,0,0],grid)
ary_xperm = PArray_type("xperm",[0,0,0],grid)
call ary_poro%Alloc(1.0_real64)
call ary_xperm%Alloc(1.0_real64)
call GetPArray(nam="Porosity",parray=ary_poro,err=err,msg=msg)
call GetPArray(nam="XPerm",parray=ary_xperm,err=err,msg=msg,&
               internal_units="[md]",external_units="[md]")
...
call GetValLen("Wells",nwell,err,msg)
allocate(well_name(nwell),well_top(3,nwell))
call GetWellsProp("name",well_name(1:nwell),err,msg)
call GetWellsProp(nam="top",var=well_top(1:3,1:nwell),err=err,msg=msg,&
                  internal_units="[ft]",external_units="[ft]")
...
call FinInputProc(err,emsg)
```

Listing 3.4: Example reading data from Listing 3.3 from Fortran

Input of PArray data is done by separate files referenced in Lua input. We expect these files to be considerable big, specially in multi-million cells reservoir models. Processing this input could potentially represent a bottleneck in performance of the simulator. To avoid this potential problem, we implemented two solutions. The first one is to read PArray data in unformatted binary files. This is the default way to input PArray data in the framework. Binary files use less disk space and are faster to read. The main problem with binary files is that are not readable by humans and not easy to create. To alleviate this problem the framework allows including PArray data as formatted text files. The framework takes the text file, converts it to binary and generates a new binary file and then the framework reads the binary file. The converted binary file could be used for future simulations. Caution needs to be taken on using text files for PArray data; the process of conversion is done by one processor; if the file is big enough to not to fit in the processor memory the conversion could fail. Line 18 of Listing 3.3 tells the simulator to use a formatted text file for the porosity PArray. Line 19 of Listing 3.3 tells the simulator to use a binary file for the X permeability PArray.

The second solution used to avoid performance problems while reading PArray data is to read the files in parallel. We use the parallel I/O capabilities of MPI (MPI-IO) to read the input data for PArrays in parallel. Each processor accesses simultaneously the file. Each processor reads only the portion of the file that contains the data for its part of the PArray. To be able to read files with MPI-IO the files must be in binary format; this is another reason for the framework to use binary format as default for PArray data. Figure 3.10 shows the process.

3.2.5 Visualization Output

To overcome the potential performance problem while writing PArray data for visualization, our framework provides two solutions:

- *S3graf format:* S3graf[1] is a post-processing software for reservoir simulation. This format uses binary files. In order to avoid performance bottlenecks, MPI-IO was employed whenever PArray data is written into files. Table 3.3 shows the files generated for visualization in S3graf format.

[1] `http://www.sciencesoft.com/products/s3graf/index.php`

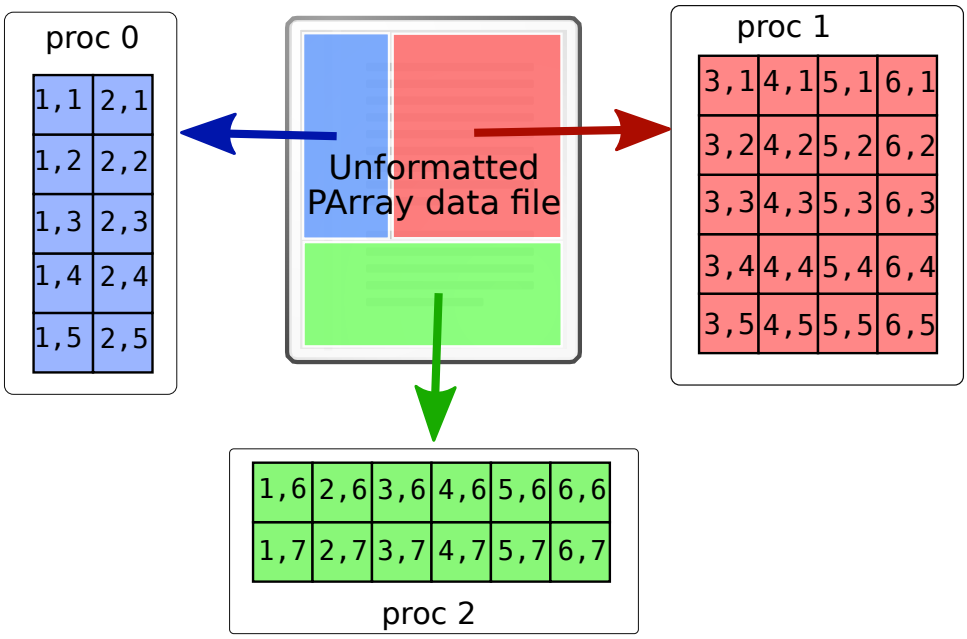


Figure 3.10: Framework employs MPI IO to read PArray input files

- *VTK format*: The Visualization toolkit (VTK)[1] is an open source system for 3D visualization. VTK files can be visualized in several visualization applications, such as VisIt[2], ParaView[3], and Mayavi[4] among others. We used Lib_VTK_IO[5], a Fortran library to write files in VTK format. VTK supports parallel files in which each processor writes its part of the data in a separate file; hence, there is no need for centralized output. Table 3.4 shows the files generated for visualization in VTK format. Figure 3.11 shows the generated file structure used in our framework.

[1] http://www.vtk.org/
[2] https://wci.llnl.gov/simulation/computer-codes/visit
[3] http://www.paraview.org/
[4] http://code.enthought.com/projects/mayavi/
[5] https://github.com/szaghi/VTKFortran

Table 3.3: Files generated for visualization in S3graf

File extension	Description
`*.S3ECH`	Grid definition
`*.S3PERM`	Static properties, for example porosity
`*.S3TAB`	Dynamic properties, for example pressure
`*.S3HIS`	Field and well history

Table 3.4: Files generated for visualization in VTK format

File	Description
`*_ini.pvtr`	Spatial distribution of static properties, it points to `*_ini_YYYY.vtr` files.
`*_dyn_XXXX.pvtr`	Spatial distribution of dynamic properties, `XXXX` represents the report number. It points to `*_dyn_YYYY_XXXX.vtr` files.
`*_ini_YYYY.vtr`	Static properties, `YYYY` is the processor id which wrote the file.
`*_dyn_YYYY_XXXX.vtr`	Dynamic properties for report `XXXX` , `YYYY` is the processor id which wrote the file.

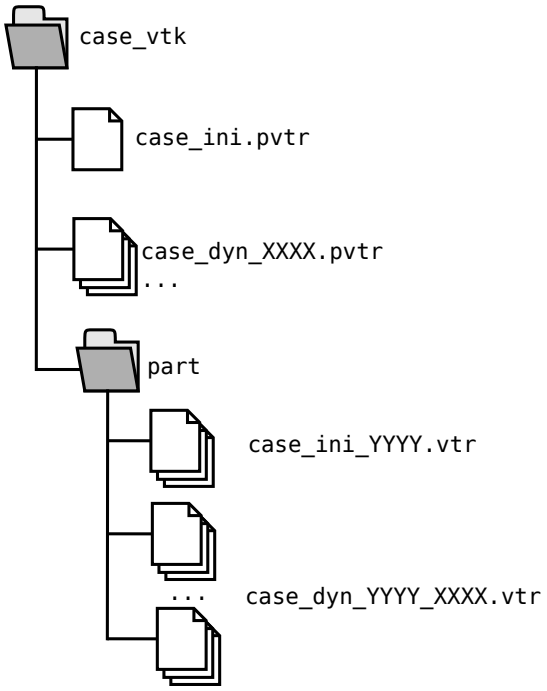


Figure 3.11: File structure of VTK output

3.3 Framework Design

The framework was designed as separate modules supported by external libraries. Each module can be extended without the need to modify the others. Figure 3.12 shows the framework structure. The external libraries used are.

- *MPI* to communicate among processors and to provide parallel input/output.
- *Zoltan* to perform domain decomposition.
- *Lua + Aotus* to process input data.
- *Lib_VTK_IO* to write output in VTK format.
- *PETSc* the Portable Extensible Toolkit for Scientific computation provides data structures and routines to solve linear and non-linear systems of equations and parallel environments (Balay et al. 2016). We use PETSc to solve systems of linear equations.

3.3.1 Core Module

This module contains the solutions discussed in Section 3.2. It encapsulates the parallel implementation and provides interfaces to the other framework modules to use the parallel tools. The core module has the parallel development focus. A developer interested in changing any aspect of the parallel solutions included in the framework (for example the domain decomposition algorithm) can focus his/her work in this module without the need to look at other parts of the framework.

3.3.2 Framework Module

This module defines the simulation work-flow; it controls the execution of the simulation. This module uses the tools provided by the core module, defines variables common to any type of simulation model, and provides subroutines for each step of the simulation process. The subroutines provided by this module serve as links to similar subroutines in the simulator module that contains model specific processes. The general framework's work-flow is shown in Figure 3.13.

3.3.3 Simulator Module

This module provides link subroutines that are called by the framework module. These subroutines handle the call to the simulator code. The simulator code is where the reservoir model specific subroutines are defined, for example flash calculation subroutines. The simulator module has the simulator development focus. A simulator developer can focus on modifying the simulator code without worrying on the other modules.

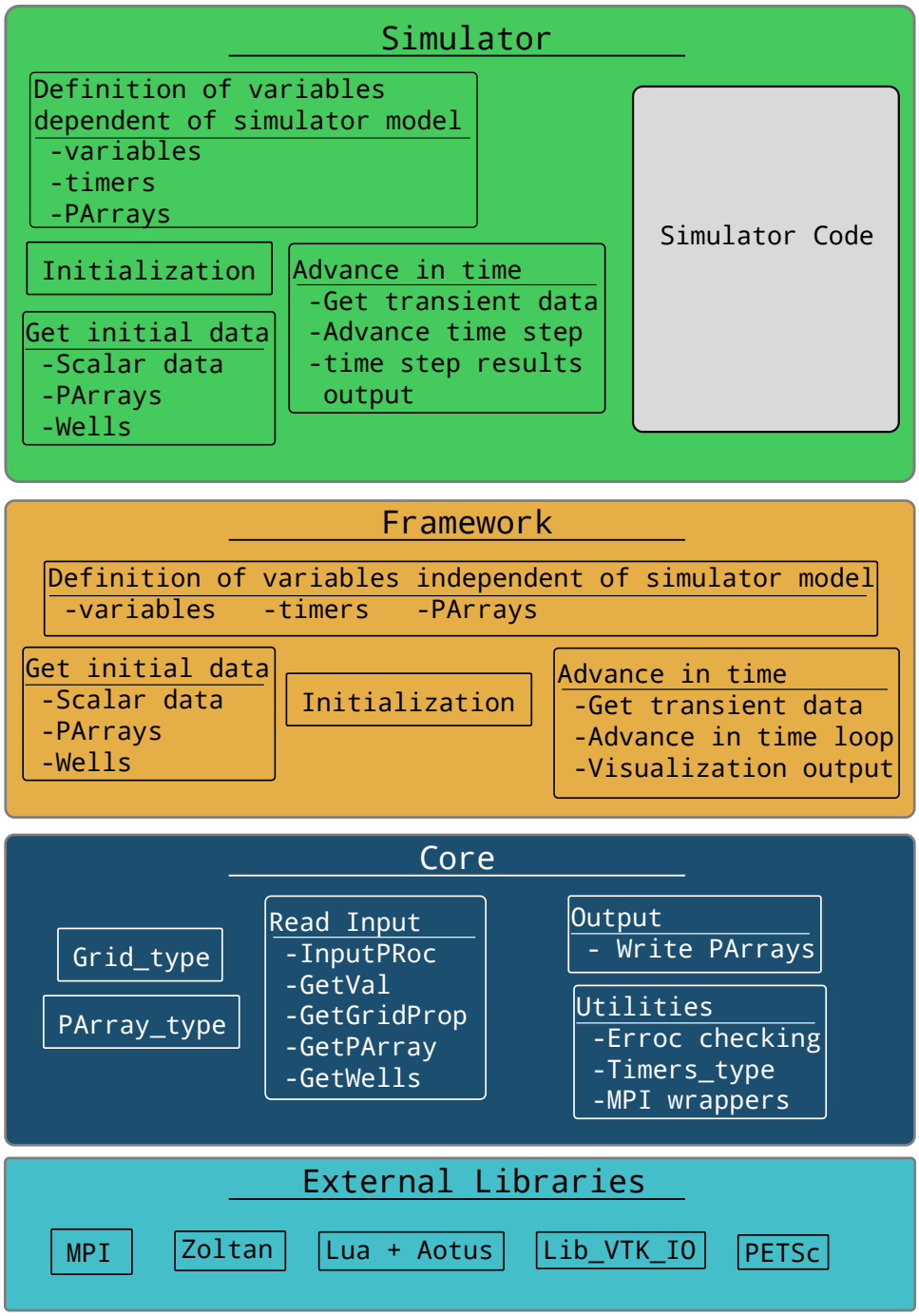


Figure 3.12: Detailed structure of framework

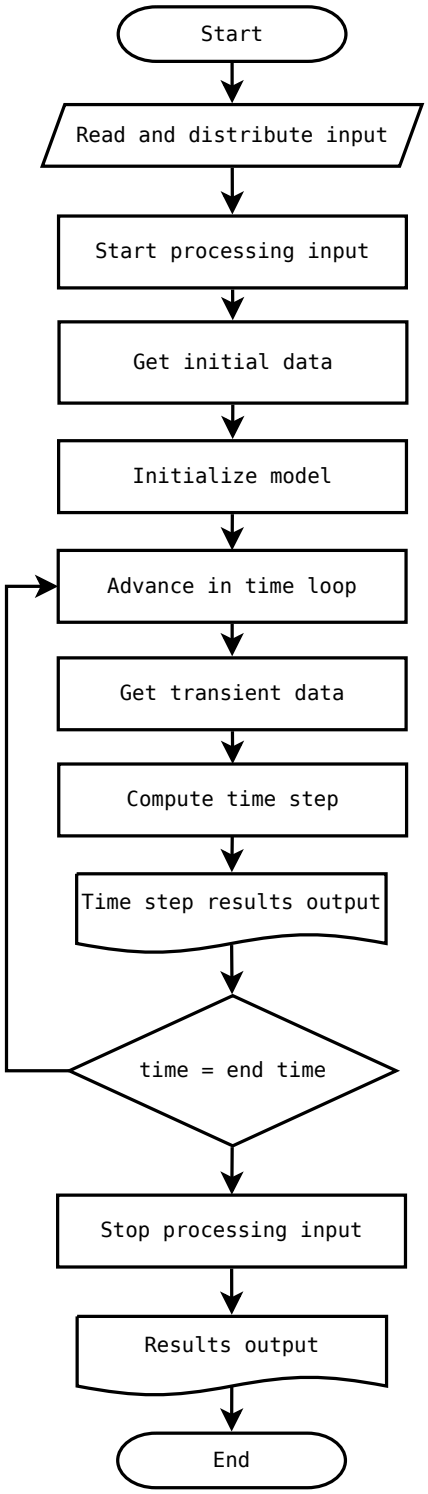


Figure 3.13: General framework's work-flow

3.4 Framework Capabilities

The framework presented in this work has the following capabilities:

- Input data processing using Lua scripting language provides easy, flexible, and expressible ways to give data to the framework that goes beyond keyword based input.
- Parallel reading of properties avoids possible bottlenecks for multi-million cell models.
- Cartesian grids with Parallel Arrays for distributed property management.
- RCB algorithm for domain decomposition that takes into account inactive cells creates regular domains distributed in an irregular manner.
- Multiple output formats for visualization (VTK, S3graf) optimized for parallel output.
- Modular framework allows future modification of existing capabilities or addition of new ones.
- Easy user interface to the parallel tools provided by the framework accelerates the development of new parallel reservoir simulators or the parallelization of existing ones.
- Division of focus in the development of parallel reservoir simulators into parallel development and reservoir simulator development.
- UTCOMP simulator adapted to the framework creates a new parallel reservoir simulator with all UTCOMP's features (see Section 4.1).

Chapter 4

Application of Framework to UTCOMP

In order to test the applicability of our framework, we created a new parallel reservoir simulator. The simulator subroutines were taken from UTCOMPP, a parallel version of the University of Texas compositional reservoir simulator (UTCOMP). In this chapter we present an overview of UTCOMP and UTCOMPP. Then we describe the changes made to the simulator subroutines from UTCOMPP in order to work with our new framework.

4.1 Overview of UTCOMP simulator

UTCOMP is The University of Texas compositional reservoir simulatior. UTCOMP was initially developed as a three-dimensional, isothermal, compositional simulator for miscible gas flood modeling (Chang 1990). During the 27 years since its initial development, UTCOMP has continuously added new capabilities and it has been evolving to the full featured reservoir simulator that today it is capable of modeling a variety of reservoir models. The main featrues of UTCOMP are:

- Three phase flash calculation
- Reduced method of flash calculation
- Four phase flow
- Higher order finite difference methods
- Fully physical dispersion tensor
- Vertical and horizontal wells
- Tracer flood
- Polymer flood

- Dilute surfactant with equilibrium and non-equilibrium mass transfer
- Gas-foam flood
- Asphaltene precipitation
- Black oil option
- CO_2 sequestration

Figure 4.1 shows the overall flowchart of UTCOMP. UTCOMP uses an IMPEC scheme, in which the pressure equation is solved implicitly using saturations and physical properties from the previous time step. And then, the material balance equations are solved explicitly to compute overall compositions. Then, phase compositions are calculated using pressure and overall composition already computed and flash calculations. For a detailed description of UTCOMP solution scheme and formulation see (Chang 1990).

4.1.1 The Mass Conservation Equation

The conservation of mass for component i in a multicomponent, multiphase mixture in a reservoir can be expressed by equation 4.1. This equation applies to every point in the reservoir.

$$\frac{\partial W_i}{\partial t} + \vec{\nabla} \cdot \vec{F}_i - R_i = 0 \tag{4.1}$$

where W_i is mass accumulation of component i, $\vec{F}_i$ is flux of component i, and R_i is source of component i (Lake et al. 1984).

Equation 4.1 can be expressed in terms of moles per unit bulk volume per unit time with adsorption neglected. Also, since hydrocarbon is not permitted in the aqueous phase and water is not permitted in hydrocarbon phases the accumulation term can be expressed as

$$W_i = \phi \sum_{j=1}^{N_p} \xi_j S_j x_{ij} = \begin{cases} \phi \sum_{j=2}^{N_p} \xi_j S_j x_{ij} & \text{for } i = 1, \ldots, N_c \\ \phi \xi_1 S_1 & \text{for } i = N_c + 1 \end{cases} \tag{4.2}$$

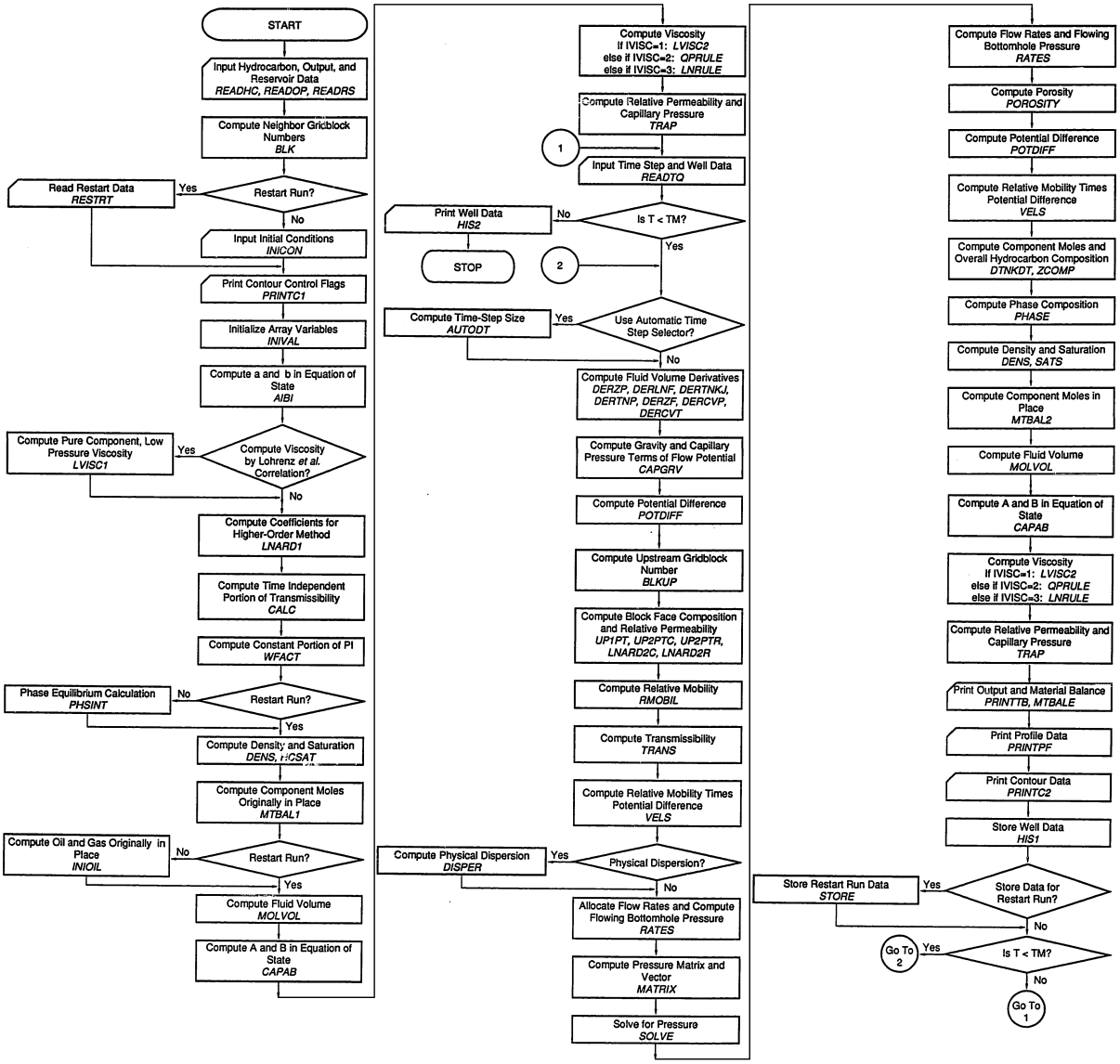


Figure 4.1: UTCOMP flowchart (Chang 1990)

where ϕ is porosity, ξ_j is the molar density of phase j, S_j is the saturation of phase j, and x_{ij} is the mole fraction of component i in phase j. There are four phases: 1 aqueous, 2 oleic, 3 gaseous, and 4 second non-aqueous phase. The hydrocarbon components are numbered from 1 to N_c and the water component is $N_c + 1$.

The flux of component i can be expressed as the sum of convective and dispersive fluxes:

$$\vec{F}_i = \sum_{j=1}^{N_p} \left(\xi_j x_{ij} \vec{u}_j - \phi S_j \xi_j \vec{\vec{K}}_{ij} \cdot \vec{\nabla} x_{ij} \right) \tag{4.3}$$

where $\vec{u}_j$ is the velocity of phase j and $\vec{\vec{K}}_{ij}$ is the dispersion tensor. The velocity of phase j can be expressed using the Dacy's law for multiphase fluid flow in porous media:

$$\vec{u}_j = -\vec{\vec{k}} \lambda_{rj} \left(\nabla P_j - \gamma_j \nabla D \right) \tag{4.4}$$

where $\vec{\vec{k}}$ is the permeability tensor, λ_{rj} is the relative mobility of phase j, P_j is the pressure of phase j, and γ_j is the specific gravity of phase j and D is depth. The relative mobility of phase j can be expressed as

$$\lambda_{rj} = \frac{k_{rj}}{\mu_j} \tag{4.5}$$

where k_{rj} is the relative permeability of phase j and μ_j is the viscosity of phase j.

The source terms are determined by the well conditions:

$$R_i = \frac{q_i}{V_b} \quad \text{for } i = 1, \ldots, N_c + 1 \tag{4.6}$$

where q_i is the molar flow rate of component i (positive for injection) and V_b is the bulk volume of a grid block.

Substituting equations 4.2, 4.3, 4.4, and 4.6 into equation 4.1 we get:

$$\frac{\partial}{\partial t} \left(\phi \sum_{j=1}^{N_p} \xi_j S_j x_{ij} \right) - \vec{\nabla} \cdot \sum_{j=1}^{N_p} \left[\xi_j x_{ij} \vec{\vec{k}} \lambda_{rj} \left(\nabla P_j - \gamma_j \nabla D \right) + \phi S_j \xi_j \vec{\vec{K}}_{ij} \cdot \vec{\nabla} x_{ij} \right] - \frac{q_i}{V_b} = 0 \tag{4.7}$$

$$\text{for } i = 1, \ldots, N_c + 1$$

Equation 4.7 is a group of partial differential equations that are solved along with other independent equations; see (Chang 1990) for further details.

4.1.2 The Pressure Equation

The pressure equation used in UTCOMP is derived based on the assumption that the pore volume V_p, which is a function of pressure P, is filled completely by the total fluid volume V_t, which is a function of pressure P and $\vec{N}$, the vector of the total number of moles of each component N_i (Chang 1990):

$$V_t(P, \vec{N}) = V_p(P) \tag{4.8}$$

Differentiating equation 4.8 with respect to time and using the chain rule:

$$\left(\frac{\partial V_t}{\partial P}\right)_{\vec{N}} \left(\frac{\partial P}{\partial t}\right) + \sum_{i=1}^{N_c+1} \left(\frac{\partial V_t}{\partial N_i}\right)_{P,N_{k(k\neq i)}} \left(\frac{\partial N_i}{\partial t}\right) = \left(\frac{dV_p}{dP}\right)\left(\frac{\partial P}{\partial t}\right) \tag{4.9}$$

Assuming the formation is slightly compressible, the pore volume can be approximated as

$$V_p = V_p^o \left[1 + c_f(P - P^o)\right] \tag{4.10}$$

where V_p^o is the pore volume at reference pressure P^o and c_f is the formation compressibility.

Writing the mass conservation equation (equation 4.7) as the net change of component i in moles:

$$\frac{\partial N_i}{\partial t} - V_b \vec{\nabla} \cdot \sum_{j=1}^{N_p} \left[\xi_j x_{ij} \vec{\vec{k}} \lambda_{rj} \left(\nabla P_j - \gamma_j \nabla D\right) + \phi S_j \xi_j \vec{\vec{K}}_{ij} \cdot \vec{\nabla} x_{ij}\right] - q_i = 0 \tag{4.11}$$

Selecting pressure of phase 2 as reference pressure, the pressure of other phases j are expressed as

$$P_j = P_2 + P_{c2j} \tag{4.12}$$

where P_{c2j} is the capillary pressure between phase 2 and phase j.

Substituting equations 4.10 and 4.11 into equation 4.9 we get the final form for the pressure equation

$$\begin{aligned}\left[V_p^o c_f - \left(\frac{\partial V_t}{\partial P}\right)_{\vec{N}}\right]\left(\frac{\partial P}{\partial t}\right) - V_b \sum_{i=1}^{N_c+1} \overline{V}_{ti}\vec{\nabla} \cdot \sum_{j=1}^{N_p} \vec{\vec{k}}\lambda_{rj}\xi_j x_{ij}\nabla P \\ = V_b \sum_{i=1}^{N_c+1} \overline{V}_{ti}\vec{\nabla} \cdot \sum_{j=1}^{N_p} \vec{\vec{k}}\lambda_{rj}\xi_j x_{ij}\left(\nabla P_{c2j} - \gamma_j \nabla D\right) \\ + V_b \sum_{i=1}^{N_c+1} \overline{V}_{ti}\vec{\nabla} \cdot \sum_{j=1}^{N_p} \phi\xi_j S_j \vec{\vec{K}}_{ij}\nabla x_{ij} + \sum_{i=1}^{N_c+1} \overline{V}_{ti} q_i\end{aligned} \tag{4.13}$$

where $\overline{V}_{ti}$ is the partial molar volume of component i.

4.2 Overview of UTCOMPP simulator

Ghasemi Doroh (2012) developed a parallel version of the UTCOMP simulator using the IPARS framework to handle the parallel part. UTCOMPP contains all physical and numerical features of UTCOMP. In order to make UTCOMP work with the IPARS framework Ghasemi Doroh (2012) performed the following modifications to UTCOMP subroutines:

- The subroutines can handle spatial variables input in i, j, and k indexes instead of NB (number of blocks) based indexing.
- The limits for loops through spatial variables can be defined by other subroutines.
- The loops through spatial variables take into account if the cells are active before performing calculations on those cells.
- The solution of system of linear of equations is handled by PETSc. PETSc can take advantage of the multiple processors used during a simulation run to solve the linear system of equations.

UTCOMPP has the following limitations; this limitations are because of the approach used to handle the parallel part of the simulator:

- Mixed programming language (C and Fortran) used in the parallel framework. This makes difficult to maintain and extend the simulator.
- Domain decomposition is performed only in one dimension (Y direction).
- Reservoir models can be up to 3.2 million cells.
- Keyword based input makes hard to implement complex input data.

4.3 Modifications Made to UTCOMPP's Simulation Subroutines in Order to Work with Our New Framework

Our new parallel reservoir simulator uses modified simulator subroutines from UTCOMPP. The modifications made in order to work with our new framework were:

- Update subroutines to use our framework's variables and modules instead of old Fortran `INCLUDE` files.
- Modify subroutines to use the approach used by our new framework. Use of PArrays and Grid types, and the new defined functions.
- Some subroutines were wrapped into Fortran modules.

It is worth noting that some UTCOMPP's simulation subroutines did not need any modification to be included in our new simulator.

As a result, by using the framework developed in this work we were able to take a limited parallel reservoir simulator and create a more versatile parallel reservoir simulator. This new parallel reservoir simulator inherits all the features of our framework (Section 3.4), and contains all the features of UTCOMP (Section 4.1). With this new simulator we were able to run reservoir models up to 102.4 million cells with 6 components and use up to 1024 processors (see Chapter 5).

Chapter 5

Case Studies

In this chapter we describe the cases used to verify usability and test the parallel performance of our new parallel reservoir simulator. We compare the results obtained with our new simulator against the results generated with UTCOMPP. The cases are divided in two groups. The first group are *Verification cases* which primary objective is to verify the simulation results and accuracy of the new reservoir simulator. The second group are *Performance cases* which main focus is to evaluate the parallel performance of our new simulator.

All cases presented in this chapter were run in Lonestar 5 supercomputer from the Texas Advanced Computing Center (TACC). Lonestar 5 has 1252 computing nodes each of them with 24 cores, a total of 30000 cores . Table 5.1 shows Lonestar 5 specifications for a compute node (Texas Advanced Computing Center 2017).

Table 5.1: Lonestar 5 compute node specifications

Processor socket	Xeon E5-2690 v3 @ 2.6 GHz
Cores per socket	12
Socket per node	2
RAM per node	64 GB

5.1 Verification Cases

5.1.1 Case 1 - CO_2 Flooding

In this case a gas mixture with 95% CO_2 is injected into the reservoir. The general description of the model is shown on Table 5.2. Component properties and compositions are shown in Table 5.3. Relative permeability data is shown in Table 5.4. The production well is controlled at a constant BHP of 1100.0 psi. The injector well is controlled at a constant BHP of 1250.0 psi. Figure 5.1 shows the gird and wells for the simulation model.

Table 5.2: Model description for case 1

Dimensions (ft)	Length Width Thickness	500 1000 20
Number of cells	800	(20x40x1)
Number of components		7
Max. number of phases		4
Porosity		0.25
Permeability (md)	X Y Z	250 250 10
Rock compressibility (psi^{-1})		5×10^{-5}
Water compressibility (psi^{-1})		3×10^{-6}
Initial water saturation		0.25
Irreducible water saturation		0.25
Reservoir temperature (°F)		105
Initial reservoir pressure (psi)		1100
Number of wells	2	1 Injector 1 Producer
Simulation time (days)		1000

Table 5.3: Component properties and compositions for case 1

	Properties					Composition (molar fraction)	
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
CO_2	547.56	1069.87	1.506	44.01	0.2250	0.0337	0.95
C_1	343.08	667.20	1.586	16.04	0.0080	0.0861	0.04999
C_2-C_3	619.57	652.56	2.902	37.20	0.1305	0.1503	0.000002
C_4-C_6	833.80	493.07	4.914	69.50	0.2404	0.1671	0.000002
C_7-C_{15}	1090.35	315.44	9.602	140.96	0.6177	0.3304	0.000002
C_{16}-C_{27}	1351.83	239.90	18.070	280.99	0.9566	0.1611	0.000002
C_{28}	1696.46	238.12	33.514	519.62	1.2683	0.0713	0.000002

Table 5.4: Relative permeability data for case 1

Model	Corey's model (Corey 1986; UTCOMP 2003)					
	Water	Gas	Oil		2^{nd} HC liquid	
Residual saturation	0.25	0.05	water-oil gas-oil	0.20 0.20	water-2^{nd} HC liq. gas-2^{nd} HC liq.	0.35 0.35
End point	0.21	0.35		0.7		0.35
Exponent	1.5	2.5	water-oil gas-oil	2.5 2.5	water-2^{nd} HC liq. gas-2^{nd} HC liq.	2.5 2.5

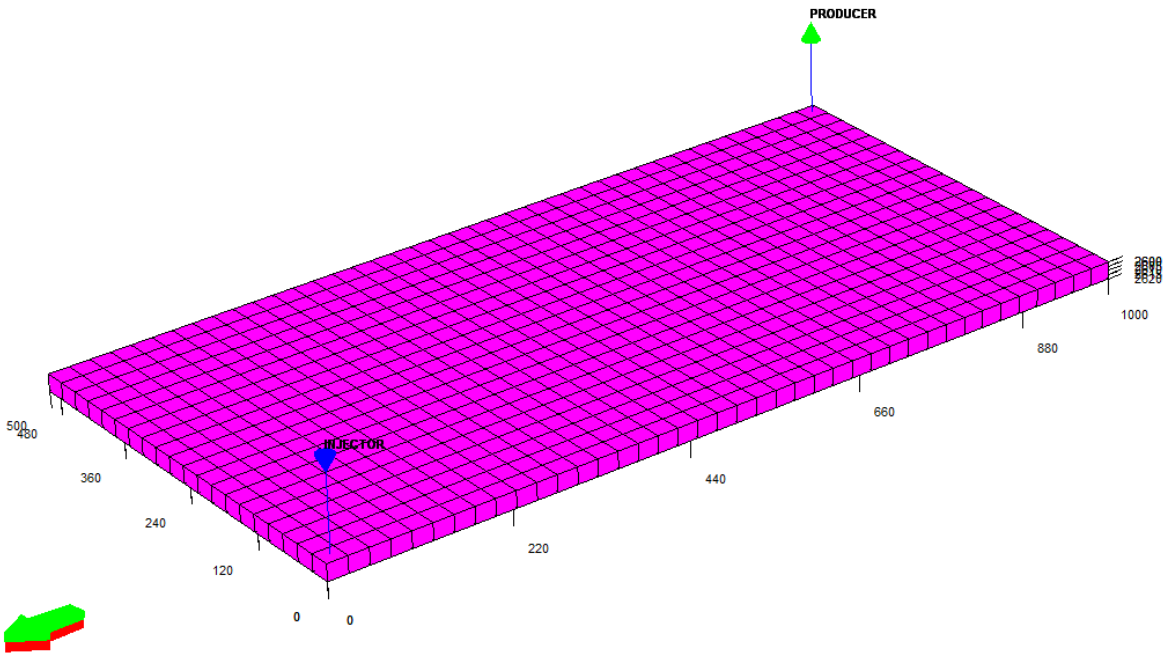


Figure 5.1: Grid and wells for verification case 1

We used this model to verify several features of UTCOMP. Table 5.5 shows the features tested and the keyword and values that activate such features in the simulator. The results of our new simulator are compared against UTCOMPP. Selected results are shown next.

Table 5.5: UTCOMP features tested in Case 1

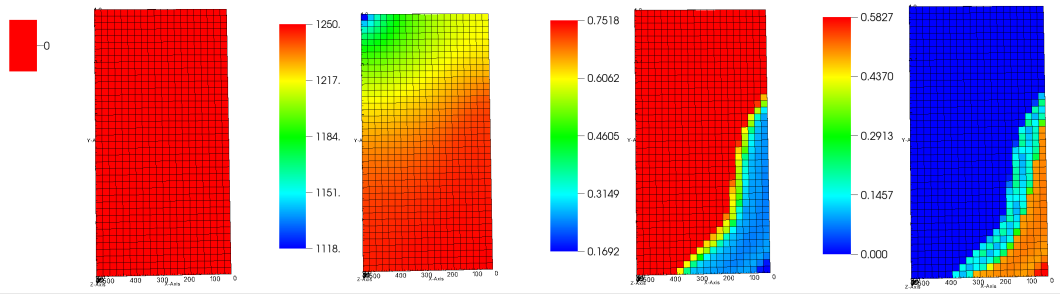
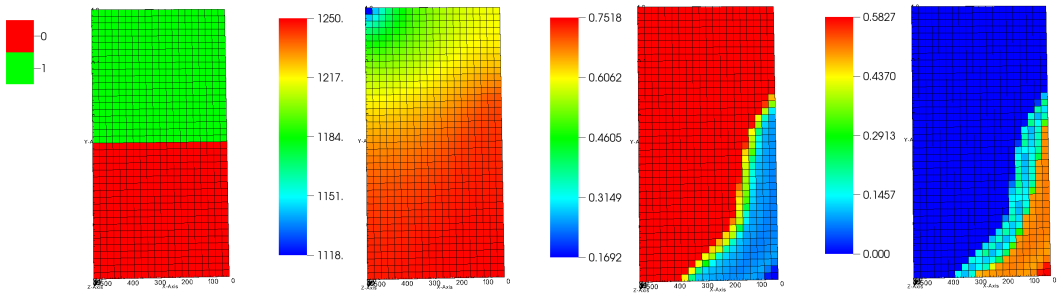
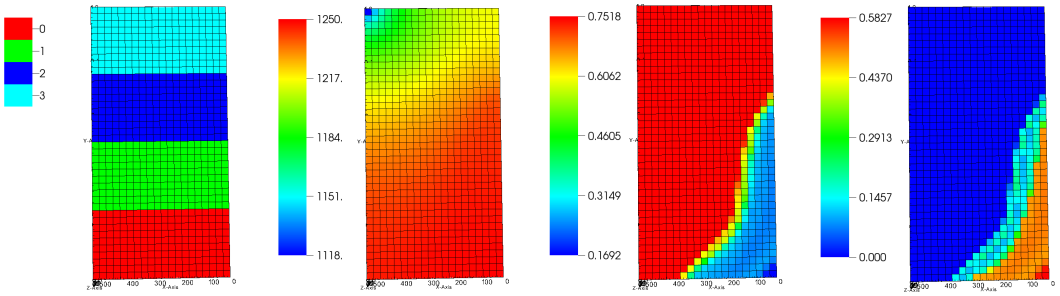
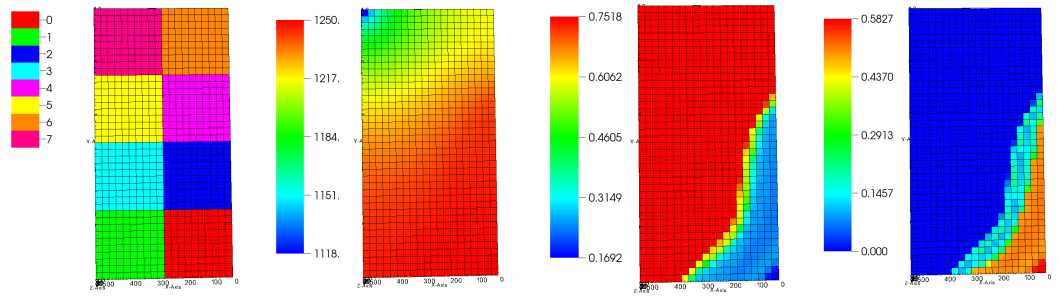
	Features	Keyword	Value
Base	Conventional flash, first order upwind scheme, Babu and Odeh well model	IFLASHTYPE IRFLA IRSA IUPSTW IWM	1 0 0 1 1
Flash type	Modified flash	IFLASHTYPE	2
	Reduced flash	IFLASHTYPE IRFLA IRSA	2 1 1
Higher order finite differences	Two point upstream weighted	IUPSTW	2
	Exponential upstream weighted third order	IUPSTW	3
	Total variation diminishing third order	IUPSTW	4
Well model	Peaceman well model	IWM	2

5.1.1.1 Base Model

For this model conventional flash, with one point upstream weighting method and the Babu and Odeh well model (Babu and Odeh 1989). The number of processors used were 1, 2, 4, and 8. Table 5.6 shows domain decomposition, pressure, oil saturation and gas saturation distribution at the end of the simulation when different number of processors are used. The results are consistent for any number of processors used.

Figures 5.2, 5.3, 5.4, and 5.5 show average reservoir pressure, surface oil production rate, and surface gas production rate and material balance error respectively. The results are compared with UTCOMPP. It can be observed from the Figures an excellent agreement on the results between UTCOMPP and our new simulator. Maximum material balance errors obtained were in the order of 7×10^{-14}.

Table 5.6: Results for case 1

Prcs	Domains	Pressure (psi)	S_o	S_g
1				
2				
4				
8				

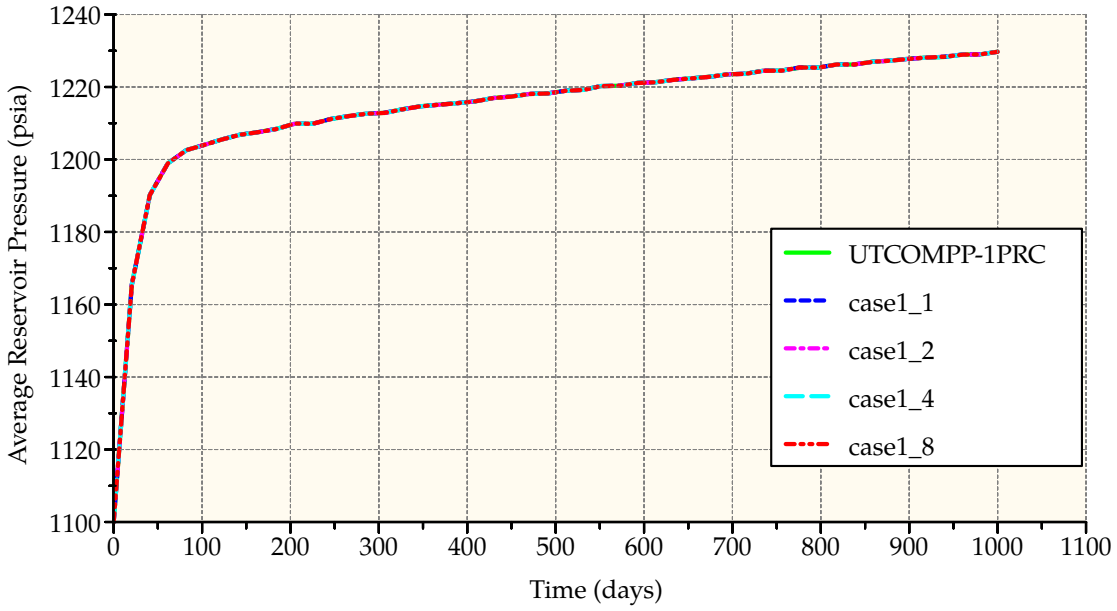


Figure 5.2: Average reservoir pressure, case 1 base

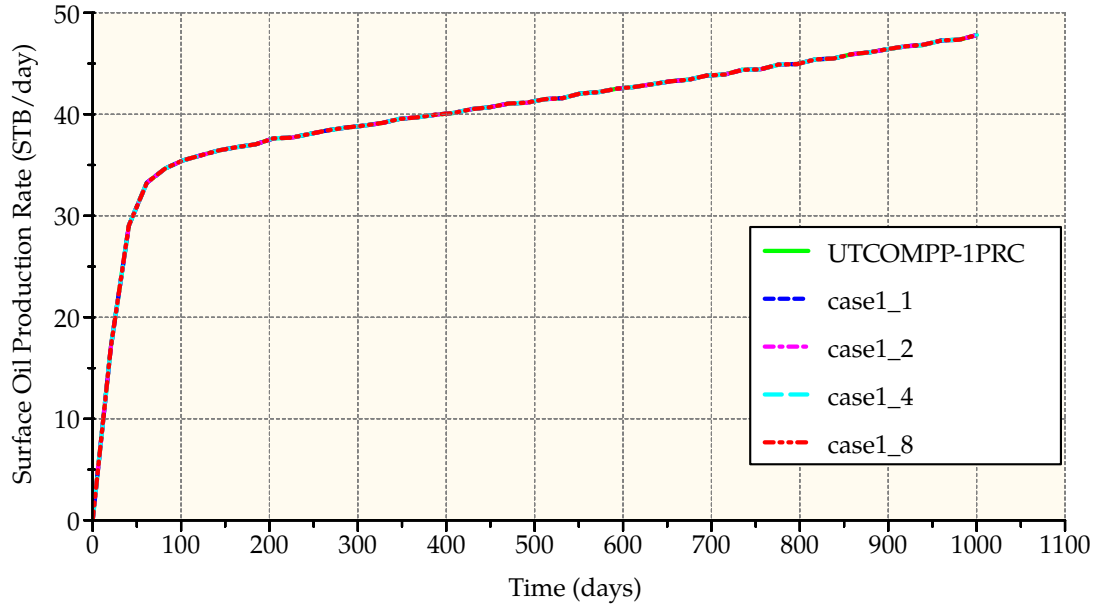


Figure 5.3: Surface oil production rate, case 1 base

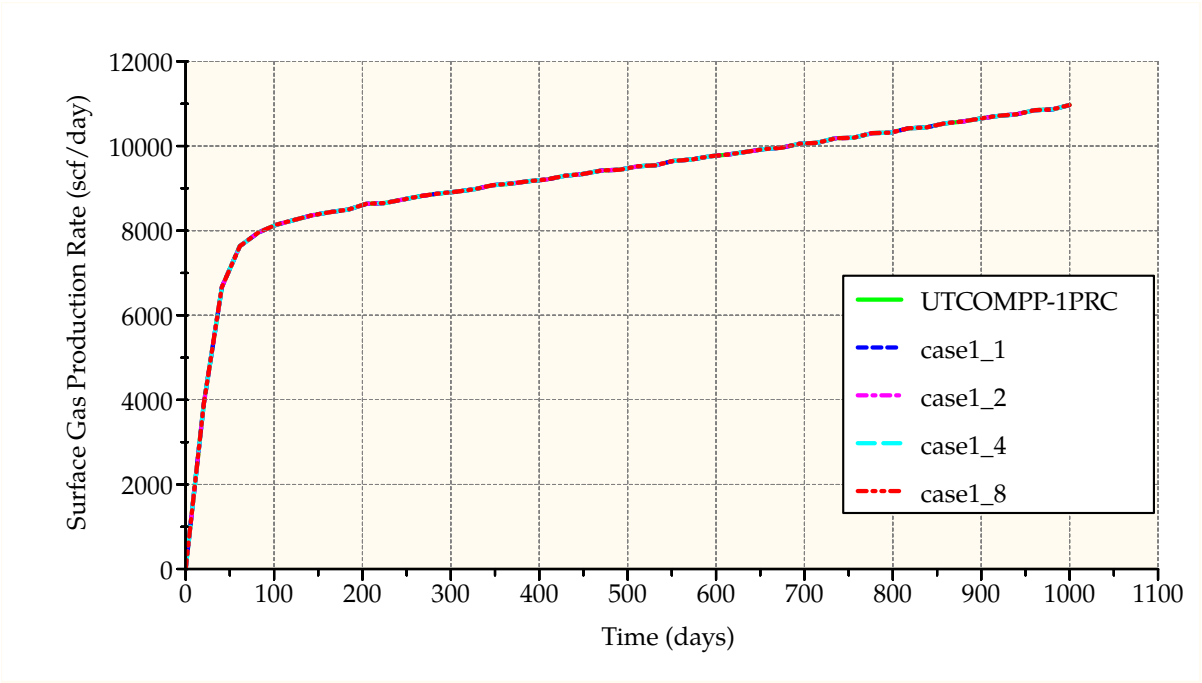


Figure 5.4: Surface gas production rate, case 1 base

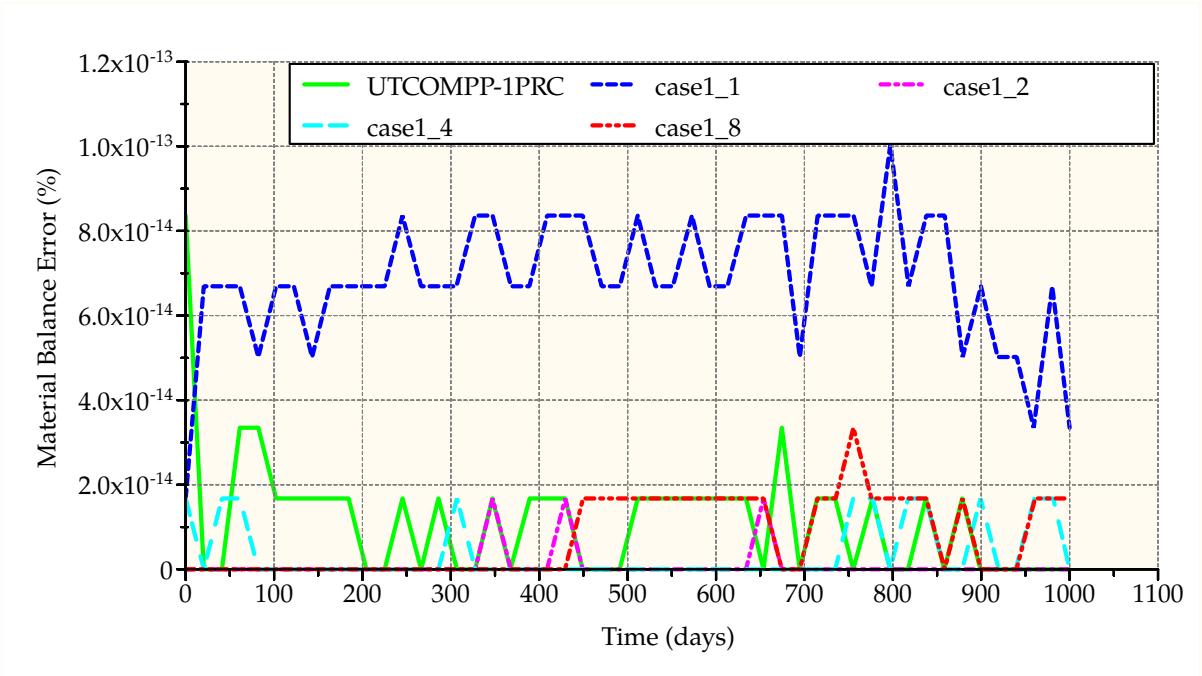


Figure 5.5: Material balance, case 1 base

Table 5.7 and Figure 5.6 show the CPU time taken with different number of processors used. The model does not scale well with increasing number of processors, this is because the model is too small. Above 4 processors the simulator takes more time in communication than doing calculations, this is expected for such a small model. Additionally, the small overhead created by the Lua processing (see Section 3.2.4) and more complex communication between processors than UTCOMPP generates a big impact in the CPU time in this model due to the small time used for computation. This explain the larger CPU times in our simulator compared with UTCOMPP in models of small size.

Table 5.7: CPU times for case 1

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	20.513	17.775
2	12.188	11.572
4	7.855	7.715
8	6.497	4.932

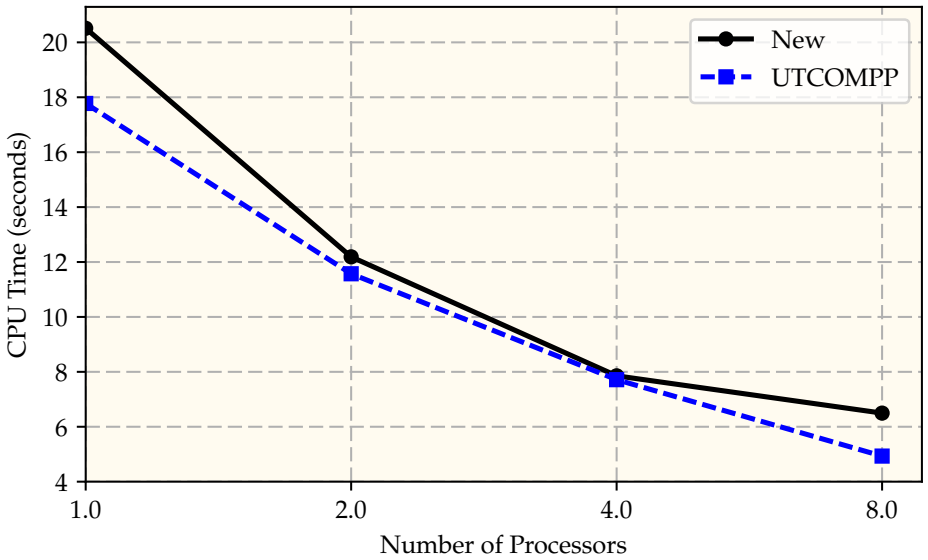


Figure 5.6: CPU time, case 1 base

5.1.1.2 Flash Calculation Options

Modified flash was implemented in UTCOMP by Okuno (2009). A comparison of the results between our new simulator and UTCOMPP is shown in Figures 5.7 and 5.8 for average reservoir pressure and surface oil production rate.

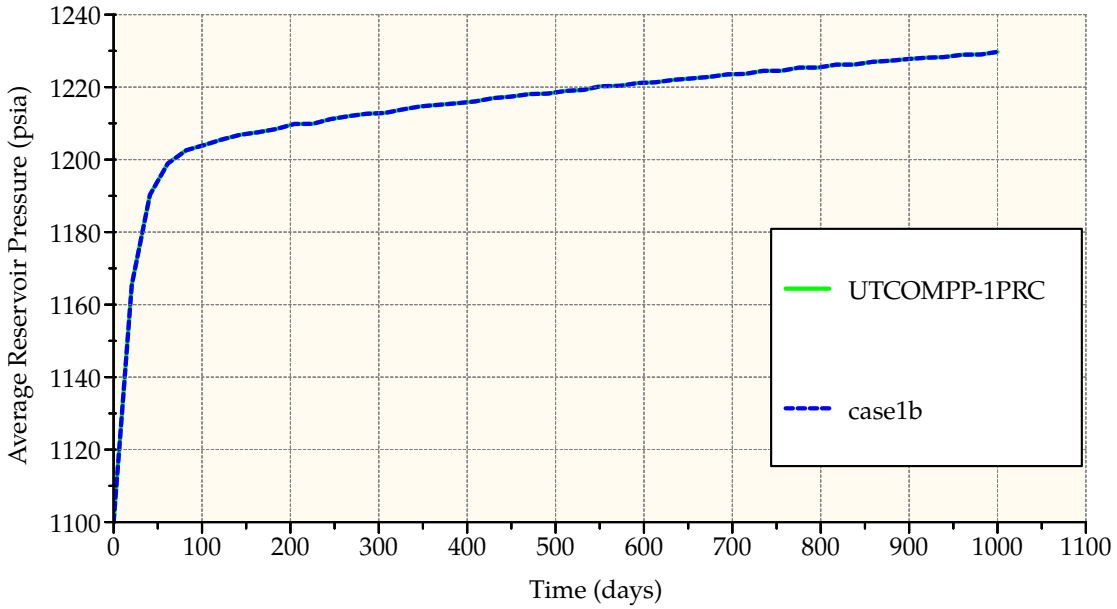


Figure 5.7: Average reservoir pressure, case 1 using modified flash calculations

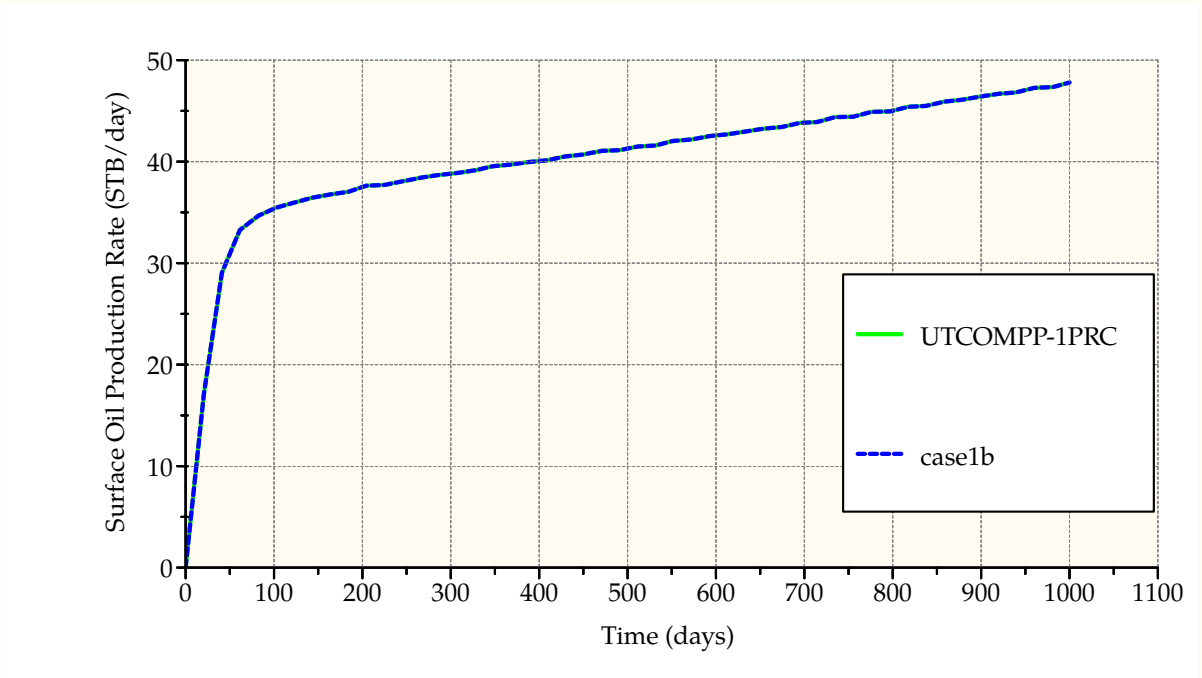


Figure 5.8: Surface oil production rate, case 1 using modified flash calculations

Reduced flash was implemented in UTCOMP by Okuno (2009). A comparison of the results between our new simulator and UTCOMPP is shown in Figures 5.9 and 5.10 for average reservoir pressure and surface oil production rate.

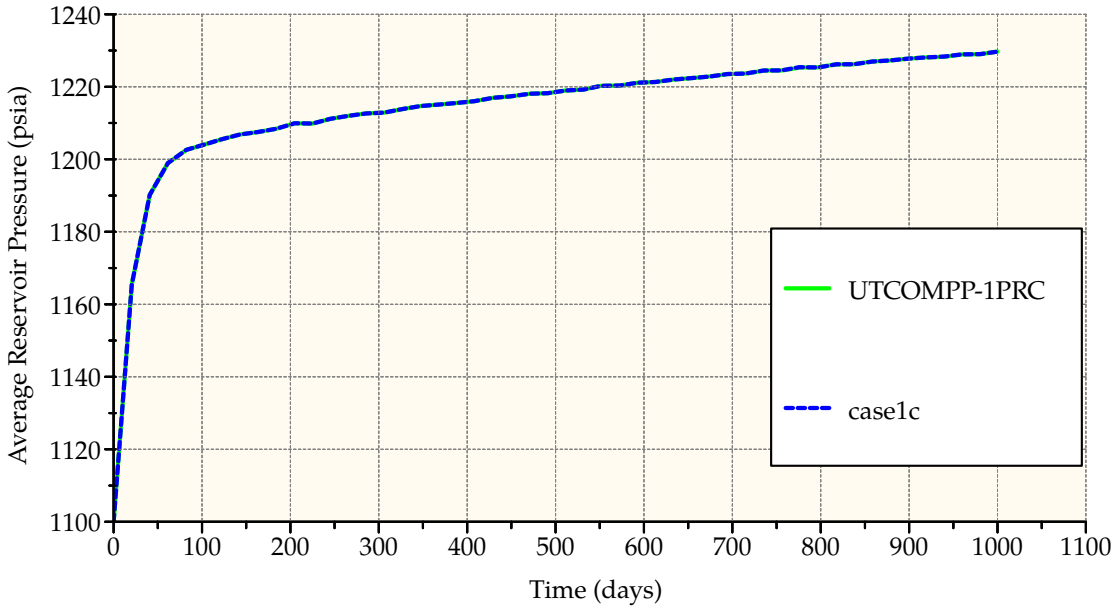


Figure 5.9: Average reservoir pressure, case 1 using reduced flash calculations

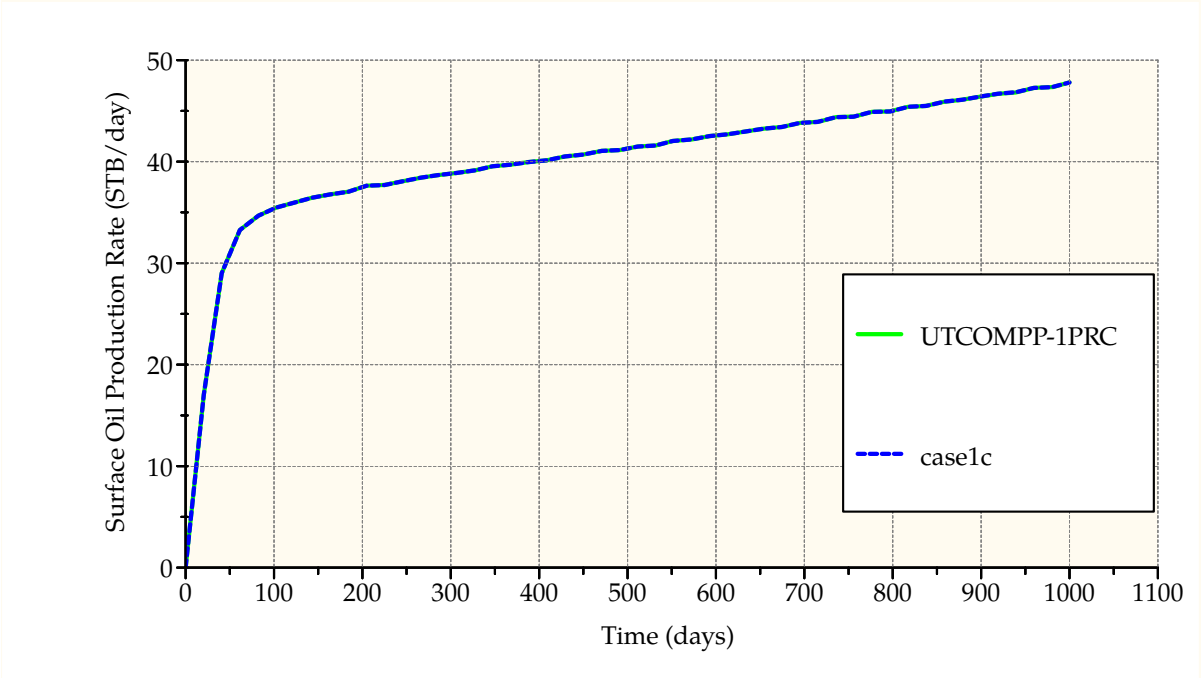


Figure 5.10: Surface oil production rate, case 1 using reduced flash calculations

5.1.1.3 Higher-Order Finite Differences

Higher order finite difference methods are implemented in UTCOMP (J. Liu 1993). Table 5.5 shows the options available and the keyword values used to activated them. These methods require more than one adjacent cell value in each direction, on the edge of each domain there is only one adjacent cell (ghost cell) provided from the neighboring domain. To overcome this problem, Ghasemi Doroh (2012) modified the simulation subroutines so that when a higher order method is used only the inner cells of a domain perform the calculation with the higher order method and the cells in the edge of the domain uses a first order approximation. This approximation in the domain edges causes no noticeable error.

Results using *Two point upstream weighted method* are shown in Figures 5.11 and 5.12.

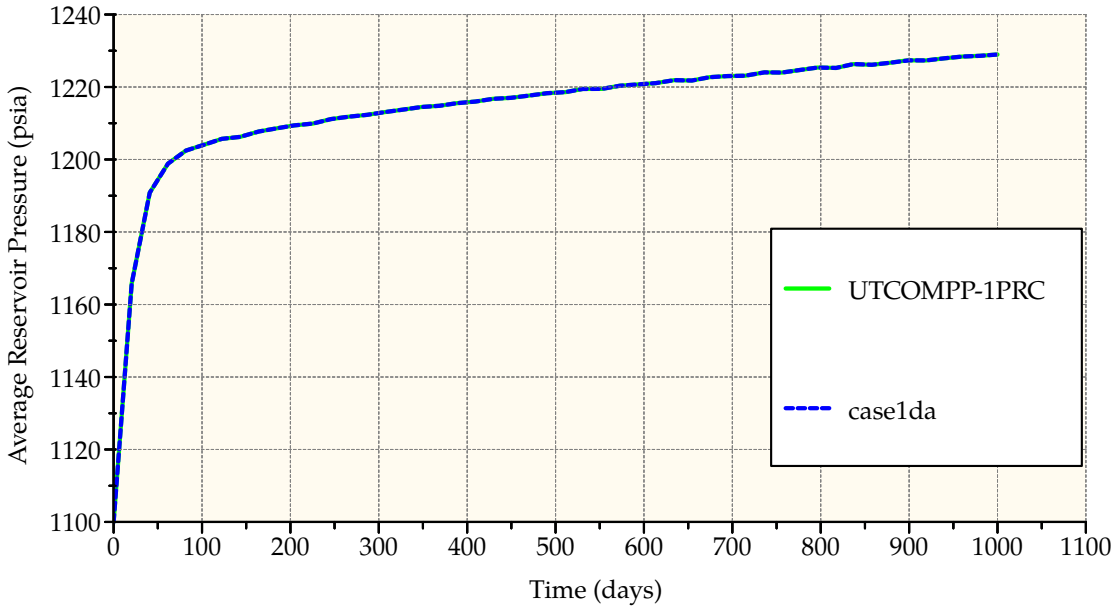


Figure 5.11: Average reservoir pressure, case 1 using two point upstream weighted method

Results using *Exponential upstream weighted third order method* are shown in Figures 5.13 and 5.14.

Results using *Total variation diminishing third order method* are shown in Figures 5.15 and 5.16.

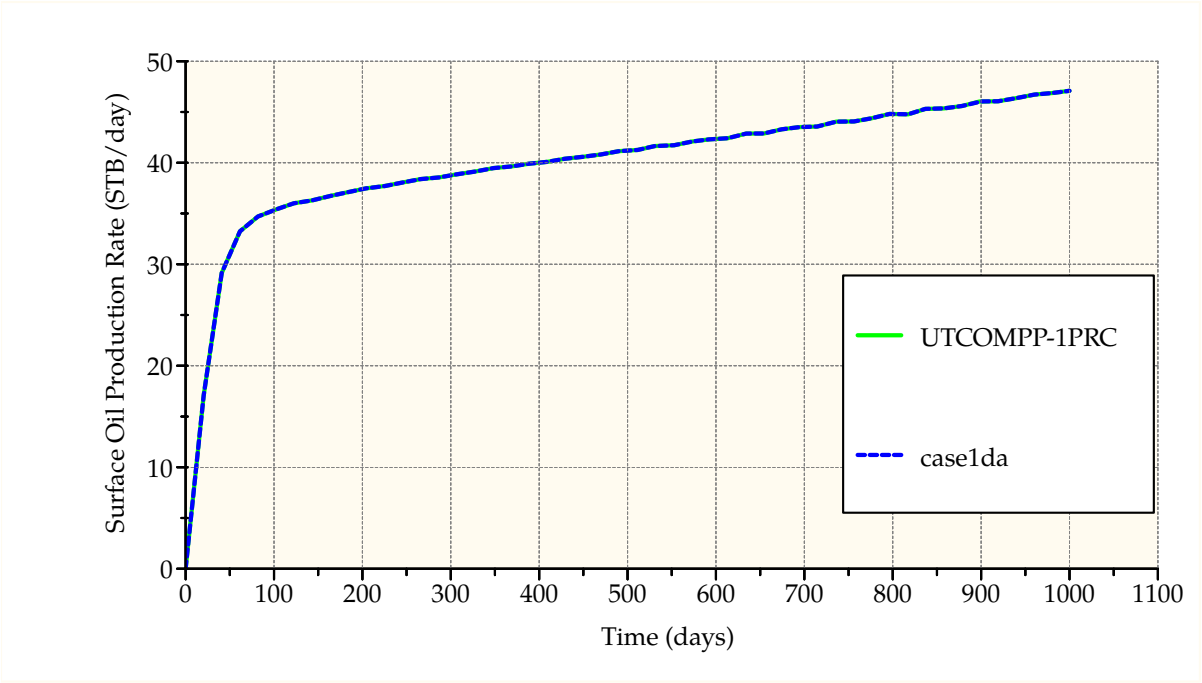


Figure 5.12: Surface oil production rate, case 1 using two point upstream weighted method

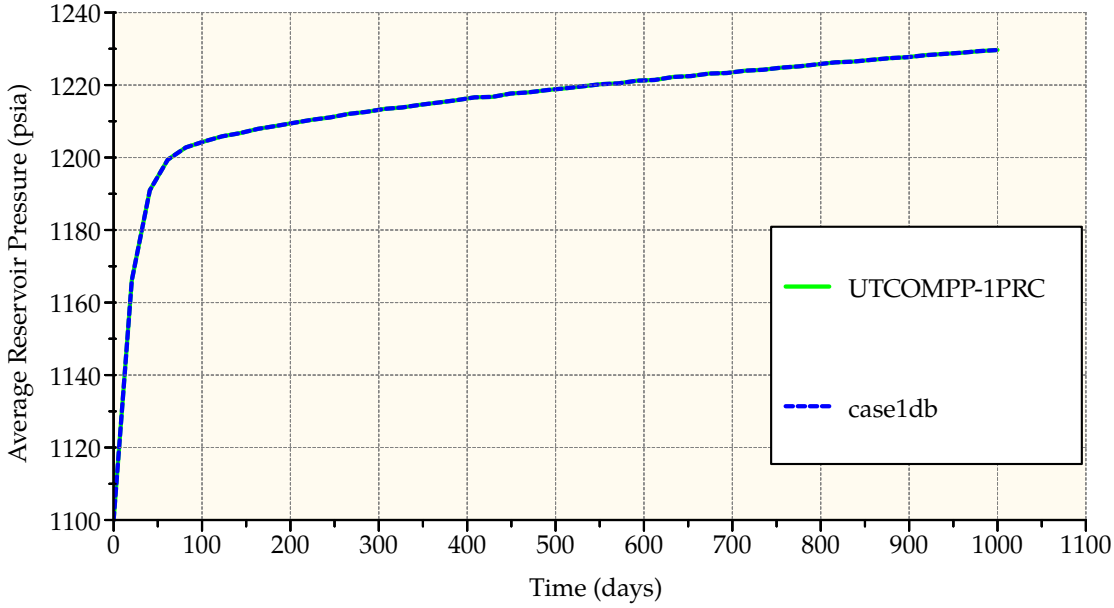


Figure 5.13: Average reservoir pressure, case 1 using exponential upstream weighted third order method

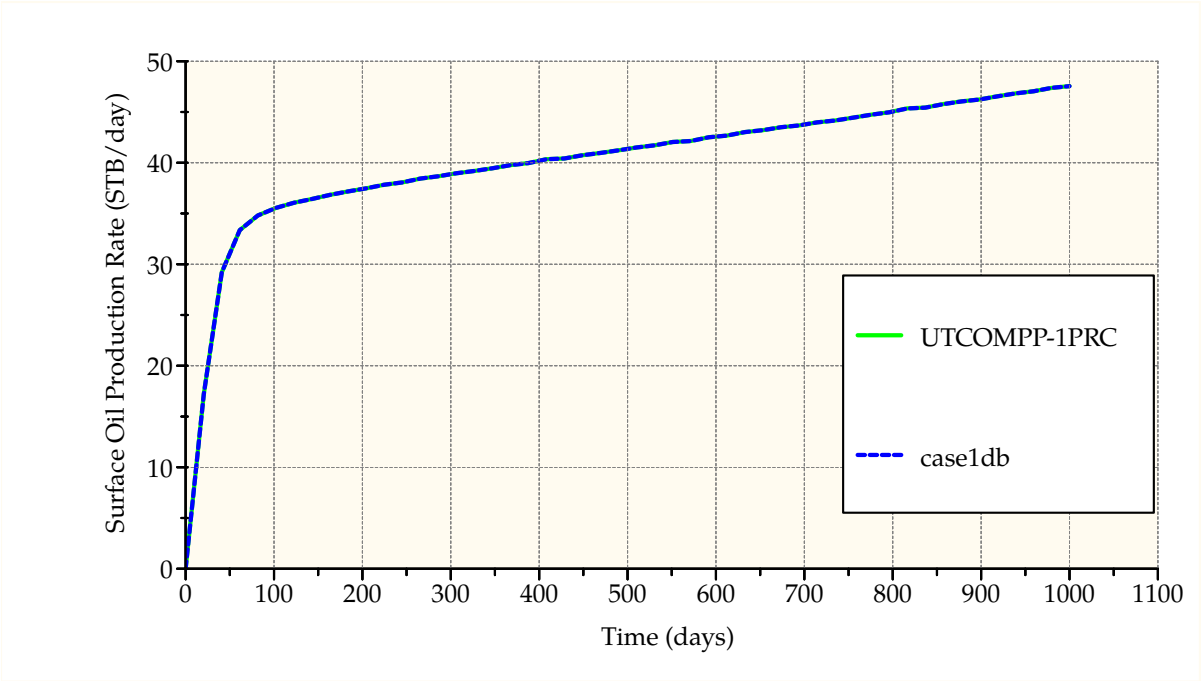


Figure 5.14: Surface oil production rate, case 1 using exponential upstream weighted third order method

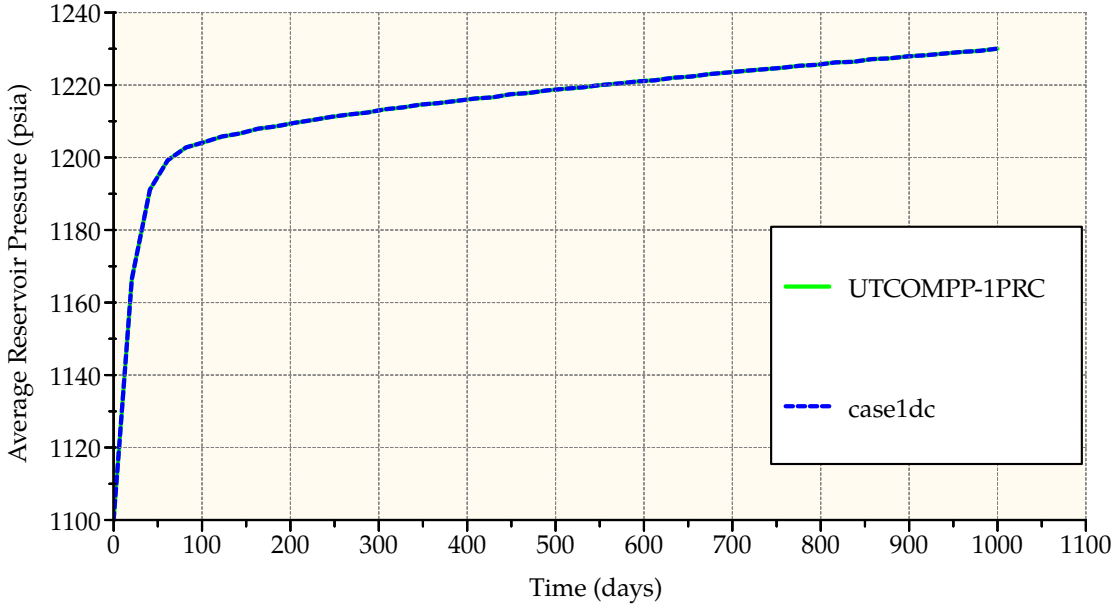


Figure 5.15: Average reservoir pressure, case 1 using total variation diminishing third order method

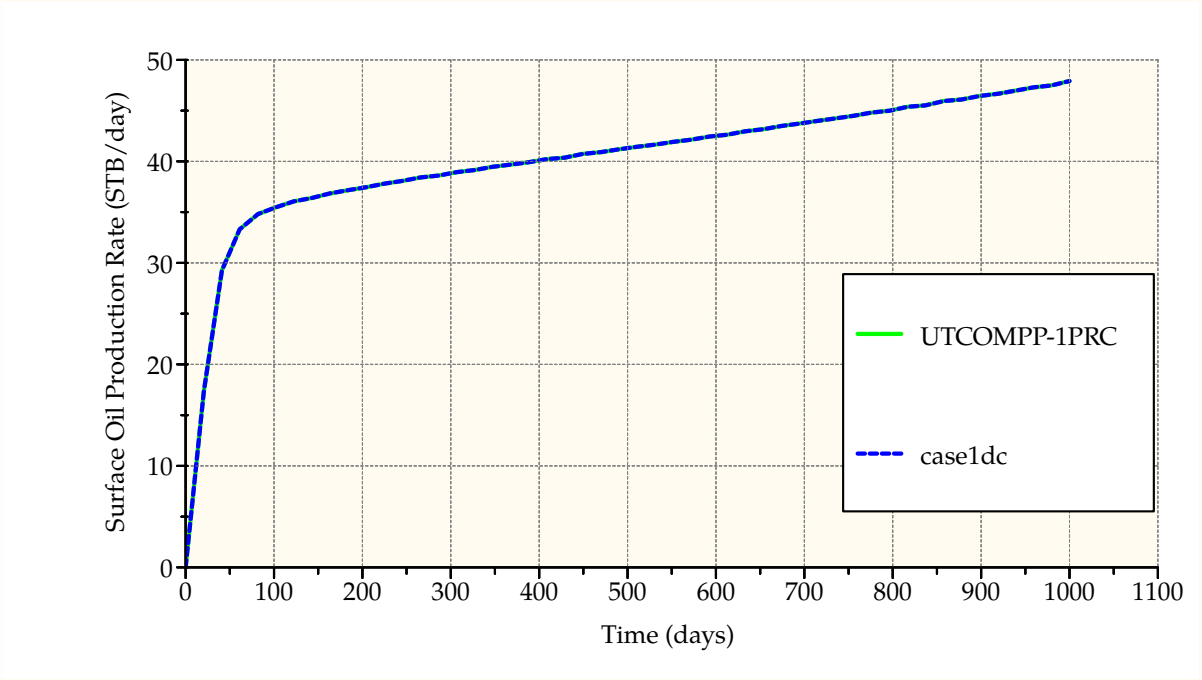


Figure 5.16: Surface oil production rate, case 1 using total variation diminishing third order method

5.1.1.4 Peaceman Well Model

Figure 5.17 and 5.18 show the results of case 1 using Peaceman well model.

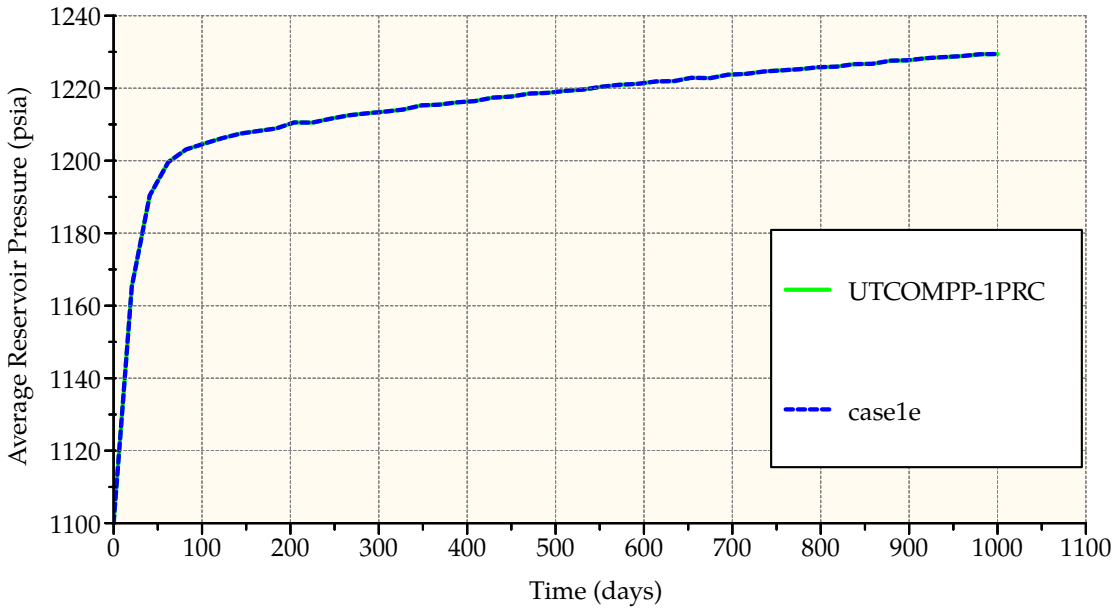


Figure 5.17: Average reservoir pressure, case 1 using Peaceman well model

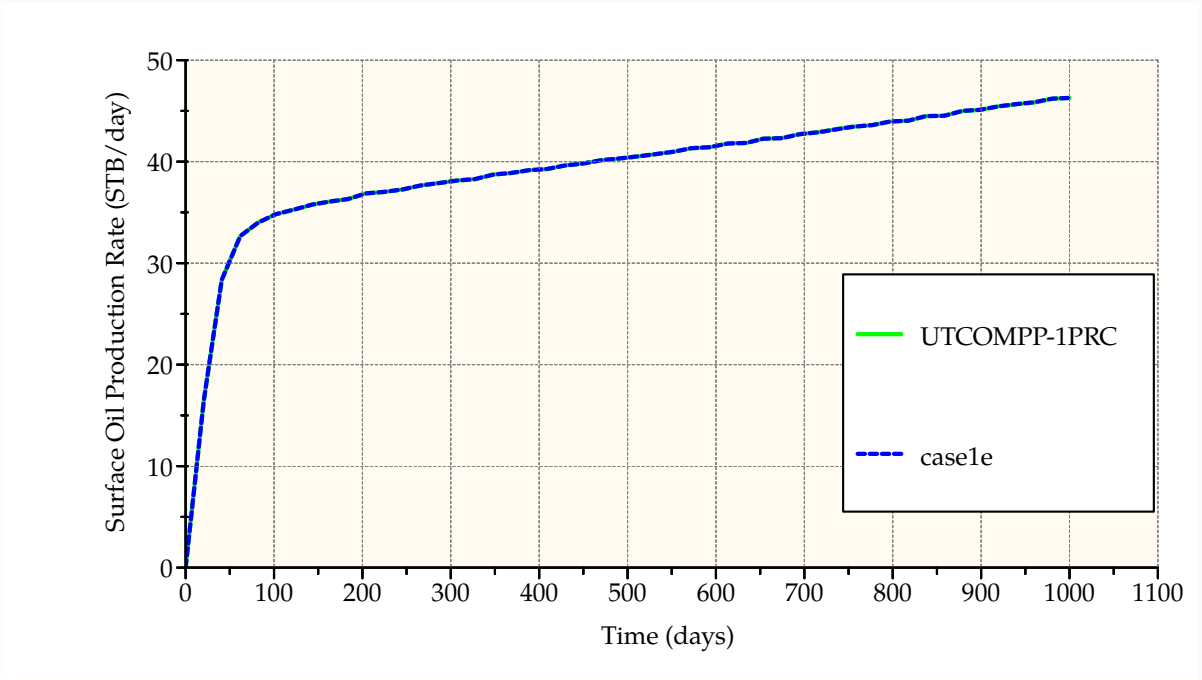


Figure 5.18: Surface oil production rate, case 1 using Peaceman well model

5.1.2 Case 2 - Gas Injection

This case models the injection of hydrocarbon gas into a reservoir. The general description of the model is shown in Table 5.8. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.9. Relative permeability data is shown in Table 5.10. The production well is controlled at constant BHP of 2500 psi. The injector well is controlled at a constant BHP of 3000 psi. Figure 5.19 shows the grid and wells for the simulation model.

Table 5.8: Model description for case 2

Dimensions (ft)	Length Width Thickness	3000 3000 250
Number of cells	8000	(40x40x5)
Number of components		6
Max. number of phases		3
Porosity		0.35
Permeability (md)	X Y Z	100 100 100
Rock compressibility (psi^{-1})		5×10^{-5}
Water compressibility (psi^{-1})		3×10^{-6}
Initial water saturation		0.2
Irreducible water saturation		0.2
Reservoir temperature (°F)		90
Initial reservoir pressure (psi)		2510
Number of wells	2	1 Injector 1 Producer
Simulation time (days)		1000

In this model up to 16 processors were used. Table 5.11 shows domains, pressure, oil saturation and gas saturation distribution at the end of the simulation when different number of processors are used. Figures 5.20, 5.21, and 5.22 show average reservoir pressure, surface oil production rate and gas production rate compared with UTCOMPP. All results are the same independently of the number of processors used, these verify the accuracy of our new reservoir simulator. Figure 5.23 shows the material balance error when different number of processors are used. Maximum material

Table 5.9: Component properties and compositions for case 2

	Properties					Composition (molar fraction)	
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
C_1	343.0	667.8	1.599	16.0	0.013	0.3	0.77
C_3	665.7	616.3	3.211	44.1	0.152	0.13	0.20
C_6	913.4	436.9	5.923	86.2	0.301	0.17	0.01
C_{10}	1111.8	304.0	10.087	142.3	0.488	0.2	0.01
C_{15}	1270.0	200.0	16.696	206.0	0.650	0.15	0.005
C_{20}	1380.0	162.0	21.484	282.0	0.850	0.05	0.005

Table 5.10: Relative permeability data for case 2

Model	Corey's model (Corey 1986; UTCOMP 2003)			
	Water	Gas	Oil	
Residual saturation	0.20	0.05	water-oil gas-oil	0.15 0.15
End point	0.40	0.85		0.75
Exponent	2.5	2.0	water-oil gas-oil	2.0 2.0

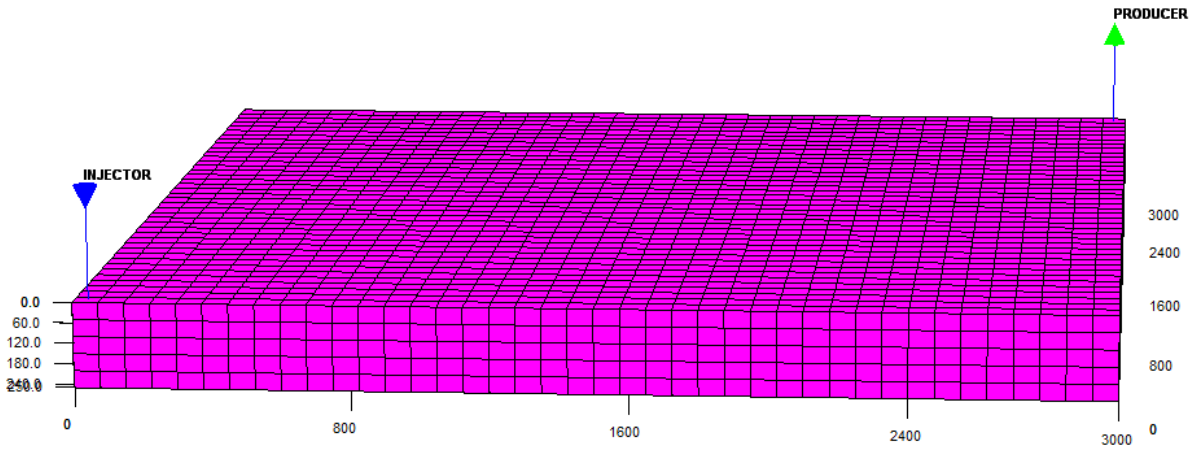


Figure 5.19: Grid and wells for verification case 2

balance error obtained were in the order of 4×10^{-13}.

Table 5.11: Results for case 2

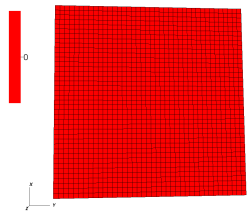
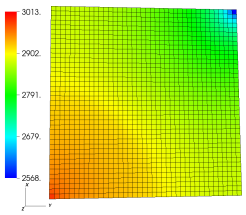
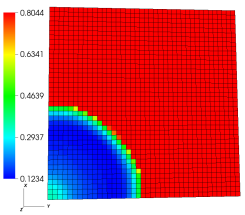
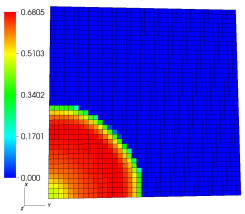
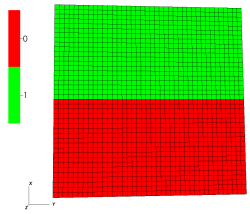
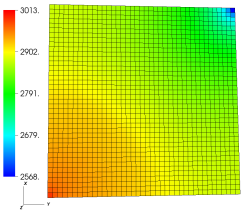
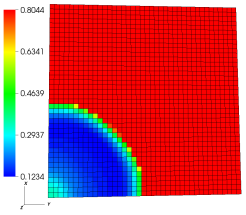
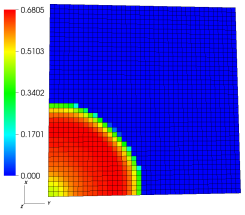
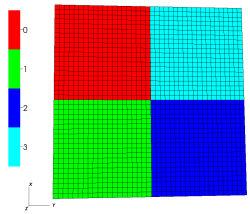
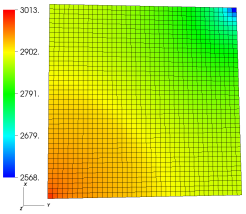
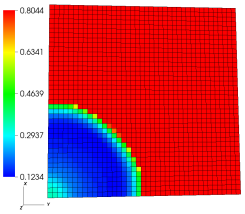
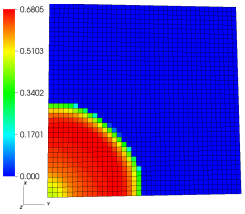
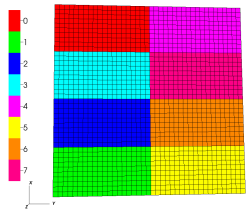
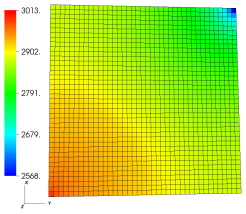
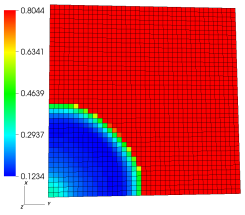
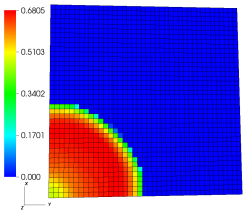
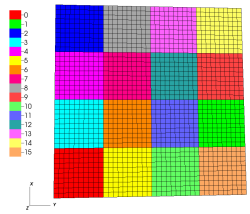
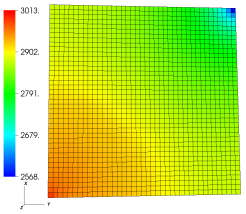
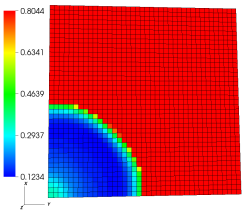
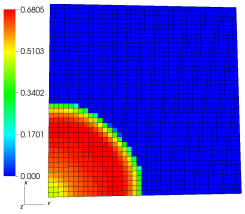
Prcs	Domains	Pressure (psi)	S_o	S_g
1				
2				
4				
8				
16				

Table 5.12 and Figure 5.24 show the CPU time when different number of processors are used. The model scales up well up to two processors. Using more than two processors in this model is inefficient because with more than two processors the time required to communicate among processors is considerably higher than the time required to do the computation. This behavior can be observed in Table 5.14 and Figure 5.25 in which the time to update ghost cells increases from two to four processors.

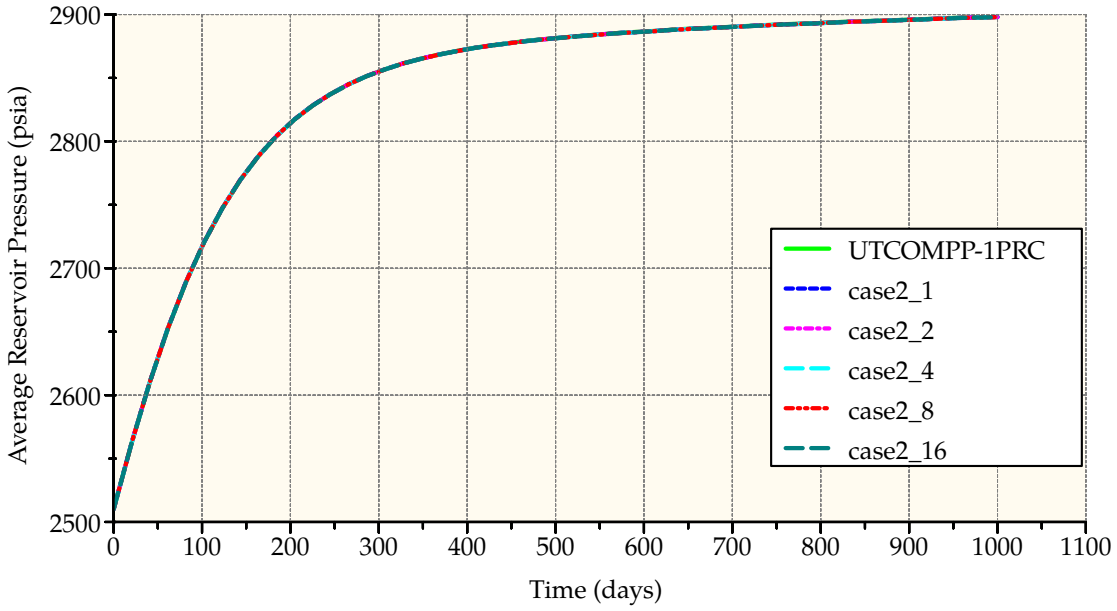


Figure 5.20: Average reservoir pressure, case 2

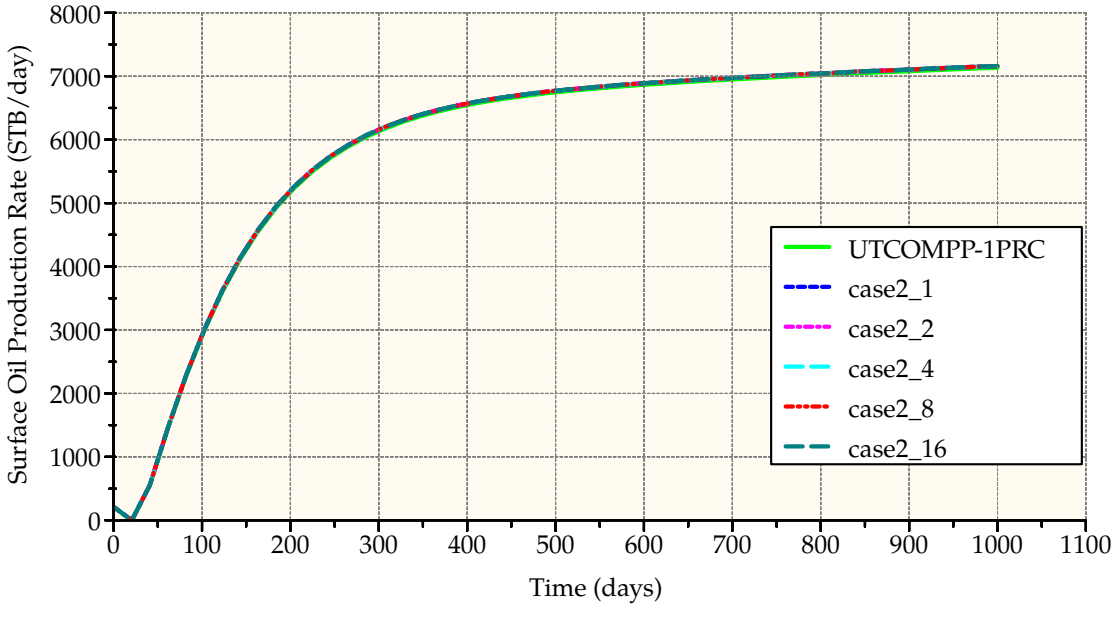


Figure 5.21: Surface oil production rate, case 2

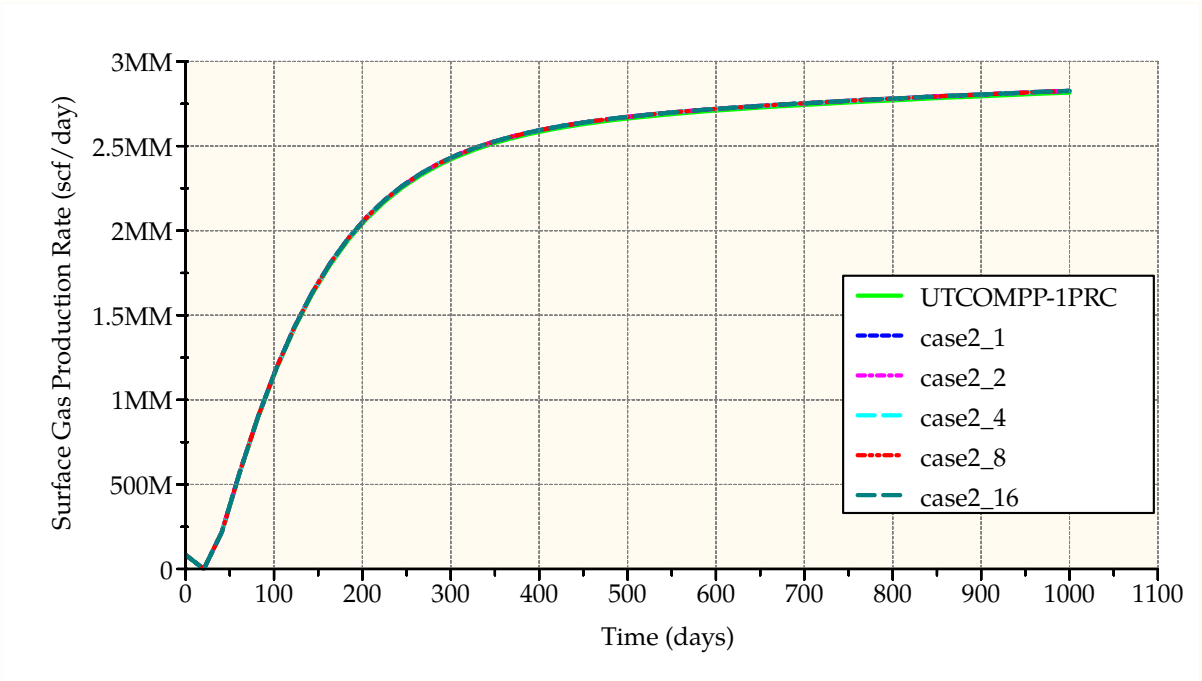


Figure 5.22: Surface gas production rate, case 2

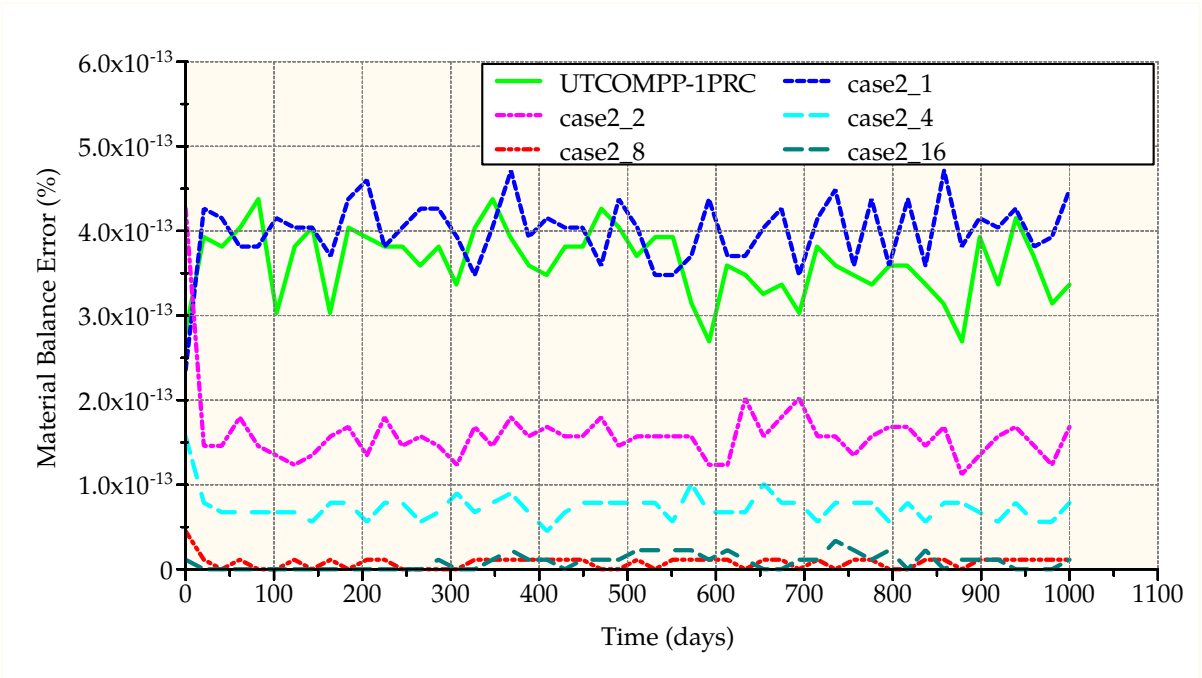


Figure 5.23: Material balance, case 2

Our new simulator measures the CPU time spent on specific sections of the simulator. Table 5.13 shows the list of the sections timed. Table 5.14 and Figure 5.25 shows the CPU time spent on specific sections of the simulator that used at least 5% of the total CPU time. If an specific section spend less than 5% of the total simulation time it is grouped in *Other* section.

Table 5.12: CPU times for case 2

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	101.532	97.868
2	66.007	64.01
4	49.597	42.788
8	39.825	27.123
16	30.718	16.532

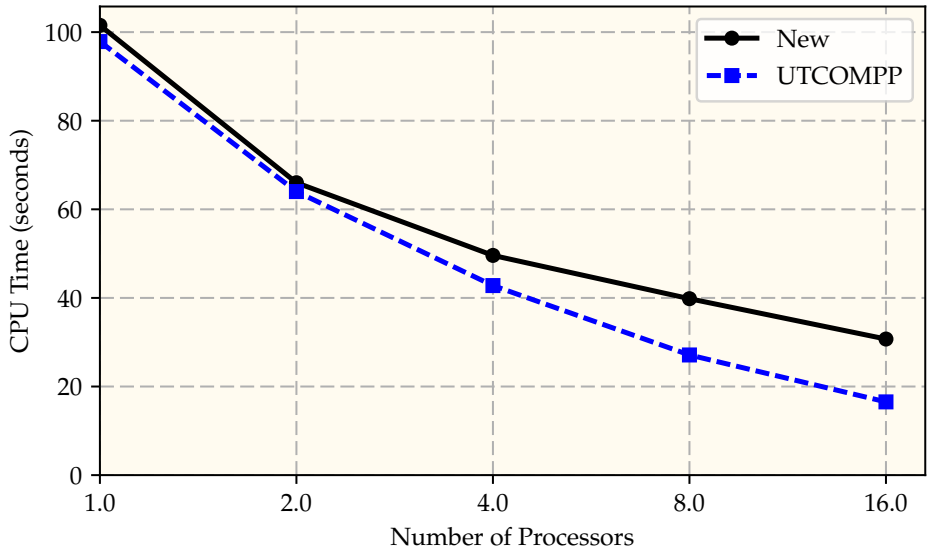


Figure 5.24: CPU time, case 2

Table 5.13: Sections timed in new simulator

Timer	Description
Total Execution Time	Time spent to complete simulation
Total Initialization Time	Time spent during initialization
Total Linear Solver Time	Time spent on linear solver
Total Well Management Time	Time spent managing the wells
Compute Next Time Step	Time spent computing the next time step
Matrix	Time spend setting up the matrix
Update Ghost Cells	Time spent on communication of ghost cells among processors
Compute Derivatives	Time spent computing derivatives
Physical Properties Before Solver	Time spent computing fluid properties before linear solver
Phase Composition Calculation	Time spent on flash calculations
Output File and Processors Transfer	Time spent on output
Monitor Printing	Time spent printing on monitor
Physical Properties After Solver	Time spent computing fluid properties after linear solver
Concentration Computations	Time spent computing fluid concentration

Table 5.14: Detailed CPU times in seconds for case 2

	Number of processors used				
	1	2	4	8	16
Update Ghost Cells	0.026	1.163	8.769	17.273	18.941
Physical Properties Before Solver	7.408	3.662	2.009	1.07	0.602
Phase Composition Calculation	70.276	35.18	17.728	9.36	4.745
Physical Properties After Solver	5.951	17.786	16.39	8.223	3.632
Other	17.844	8.301	4.807	3.928	2.763
Total Execution Time	101.532	66.007	49.597	39.825	30.718

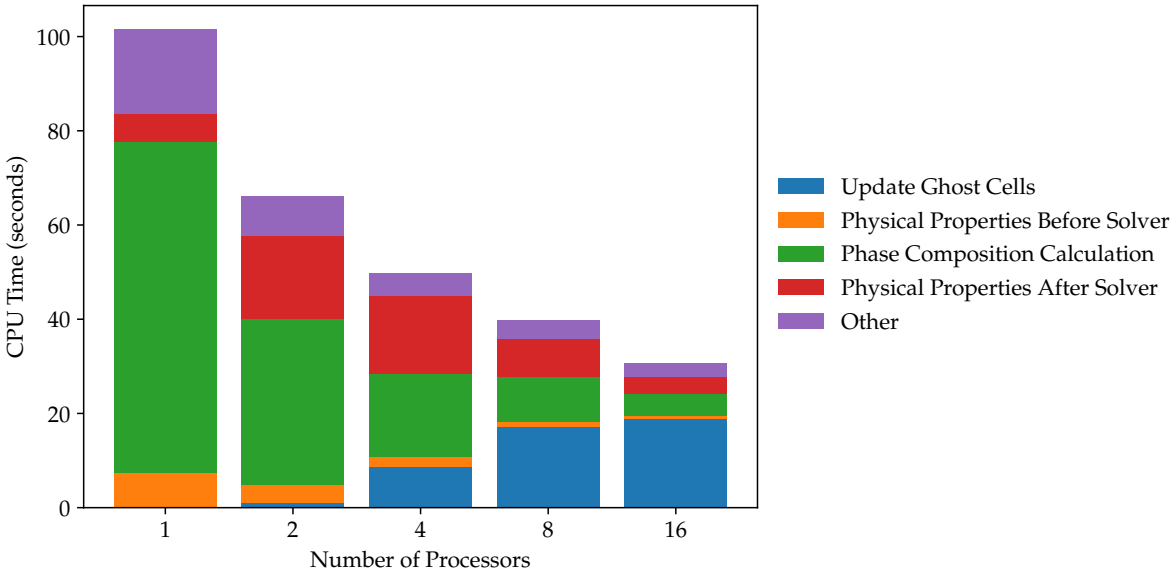


Figure 5.25: Detailed CPU times for case 2

5.1.3 Case 3 - CO_2 and Gas Injection

This case models the injection of a mix of CO_2 and hydrocarbon gas into a reservoir. The model is described in Table 5.15. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.16. Relative permeability data is shown in Table 5.17. The model contains three horizontal injector wells and two vertical producer wells. The injector wells are operated at constant BHP of 4800 psi. The producer wells are operated at constant BHP of 3500 psi. Figure 5.26 shows the grid and wells for the simulation model.

Table 5.15: Model description for case 3

Dimensions (ft)	Length	6200
	Width	6400
	Thickness	10
Number of cells	992	(31x32x1)
Number of components		6
Max. number of phases		3
Porosity		0.3
Permeability (md)	X	90
	Y	90
	Z	90
Rock compressibility (psi^{-1})		4×10^{-6}
Water compressibility (psi^{-1})		3.3×10^{-6}
Initial water saturation		0.3
Irreducible water saturation		0.3
Reservoir temperature (°F)		335
Initial reservoir pressure (psi)		4500
Number of wells	5	3 Injector 2 Producer
Simulation time (days)		800

In this model up to 8 processors were used. Table 5.18 shows domains, pressure, oil saturation and gas saturation distribution at the end of the simulation when different number of processors were used. Results comparison of our new simulator against UTCOMPP for average reservoir pressure, surface oil production rate and gas production rate are shown in Figures 5.27, 5.28, and 5.29, respectively. Figure 5.30 shows the material balance error when different number of processors are used. All

Table 5.16: Component properties and compositions for case 3

Component	Properties					Composition (molar fraction)	
	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
CO_2	547.56	1070.16	1.504	44.01	0.225	0.0618	0.5
C_1	343.08	667.38	1.589	16.04	0.008	0.1098	0.4998
C_2-C_3	594.95	672.38	2.717	34.33	0.1191	0.079	0.00005
C_4-C_6	816.91	513.03	4.783	67.13	0.2257	0.126	0.00005
PS1	1090.84	382.35	7.855	122.63	0.3056	0.319973	0.00005
PS2	1565.39	218.59	16.963	294.67	0.7879	0.303427	0.00005

Table 5.17: Relative permeability data for case 3

Model	Corey's model with trapping (UTCOMP 2003; G. Pope et al. 1998)			
	Water	Gas	Oil	
Residual saturation	0.30	0.35	water-oil	0.30
			gas-oil	0.30
End point	0.50	0.30		0.25
Exponent	1.0	4.0	water-oil	2.00
			gas-oil	2.96

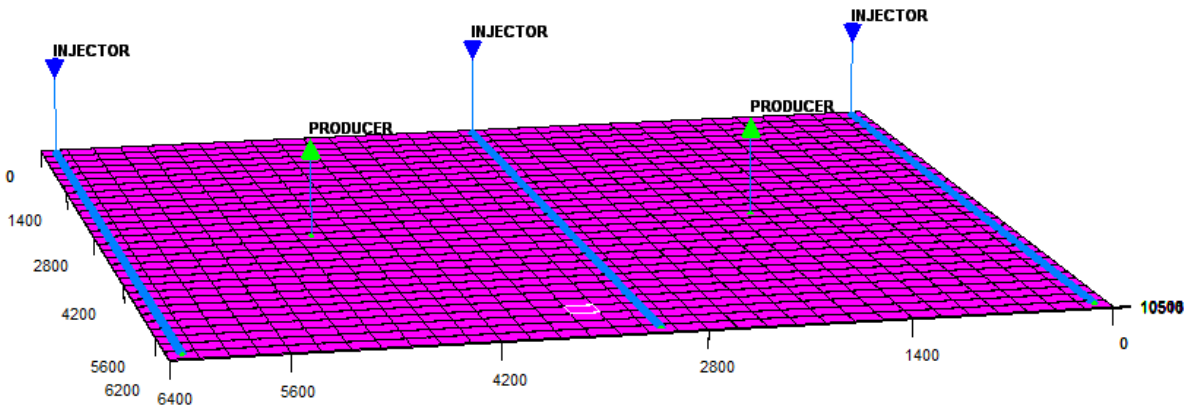


Figure 5.26: Grid and wells for verification case 3

material balance error values obtained were lower than 6×10^{-14}.

Table 5.18: Results for case 3

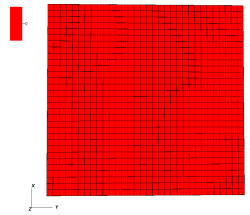
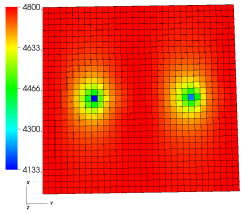
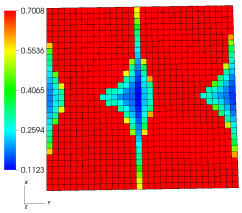
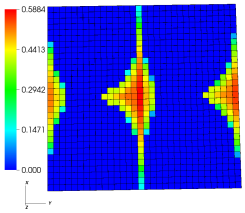
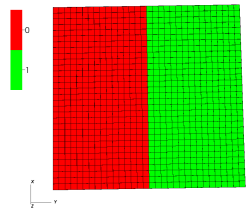
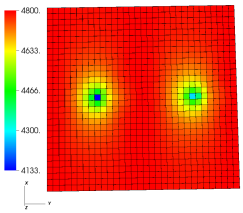
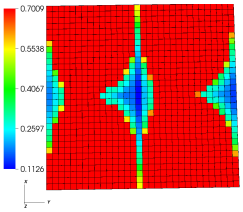
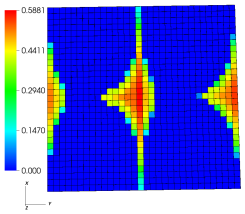
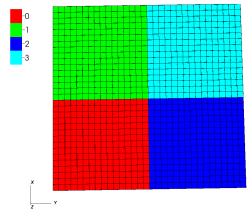
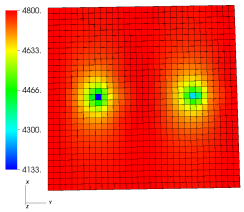
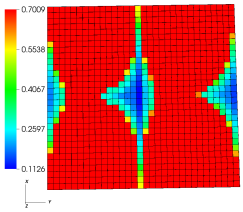
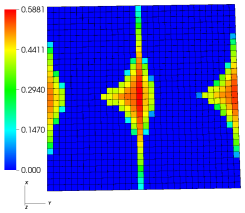
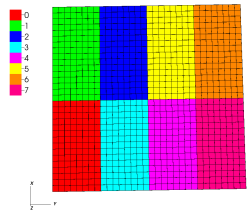
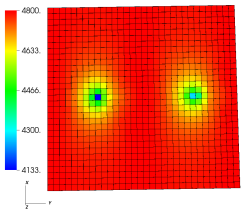
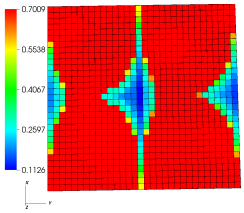
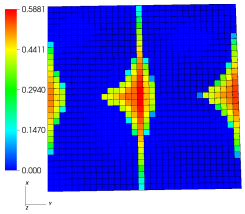
Prcs	Domains	Pressure (psi)	S_o	S_g
1				
2				
4				
8				

Table 5.19 and Figure 5.31 show the CPU time when different number of processors are used. Phase composition calculation takes the largest CPU time. This case is not affected by communication between processors. However, initial processing of the data in our new simulator is the part that creates an additional overhead in CPU time and because of this the CPU time of our new simulator is slightly higher than UTCOMPP. The detailed CPU time can be observed in Table 5.20 and Figure 5.32.

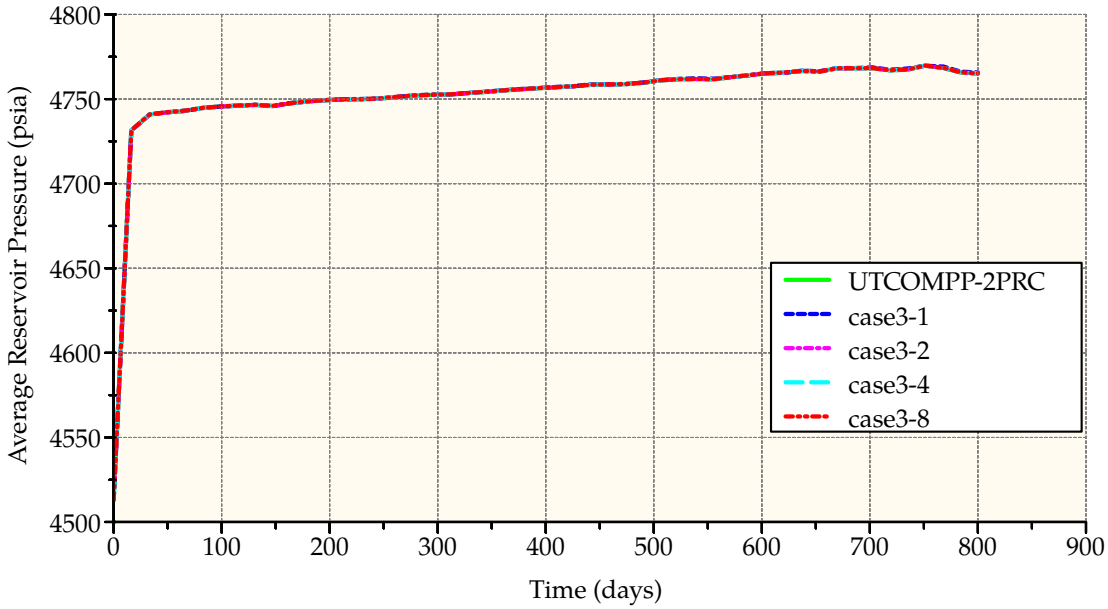


Figure 5.27: Average reservoir pressure, case 3

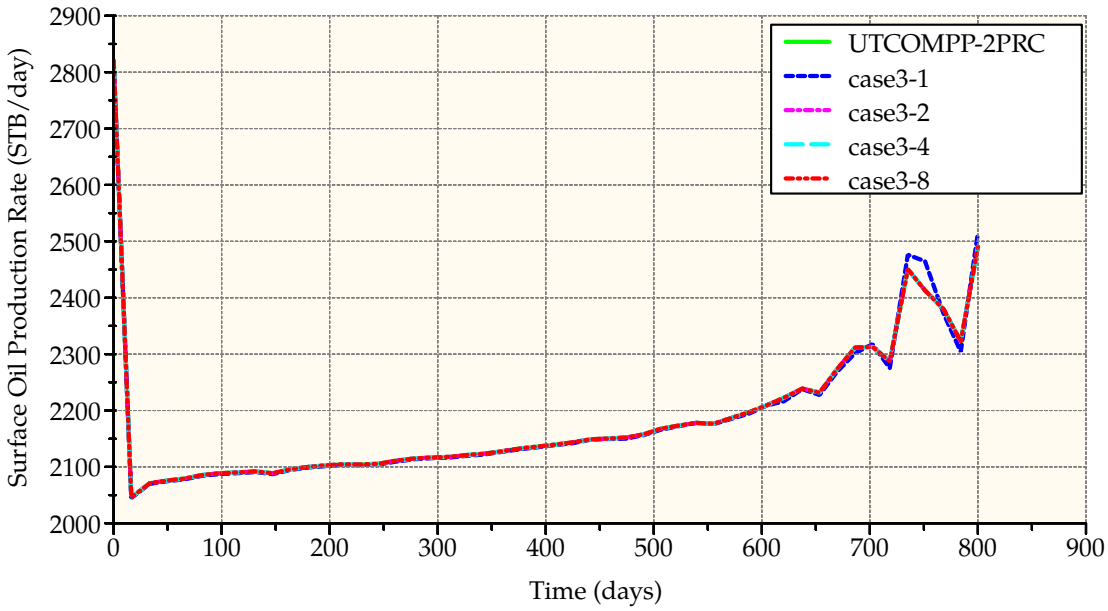


Figure 5.28: Surface oil production rate, case 3

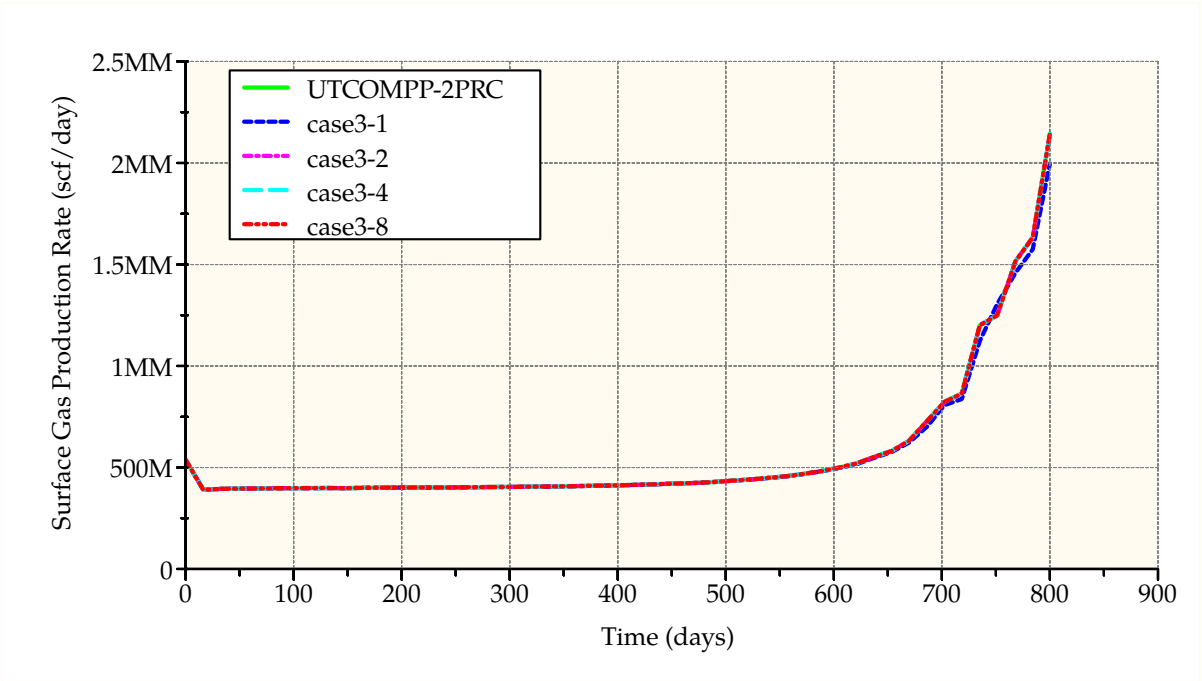


Figure 5.29: Surface gas production rate, case 3

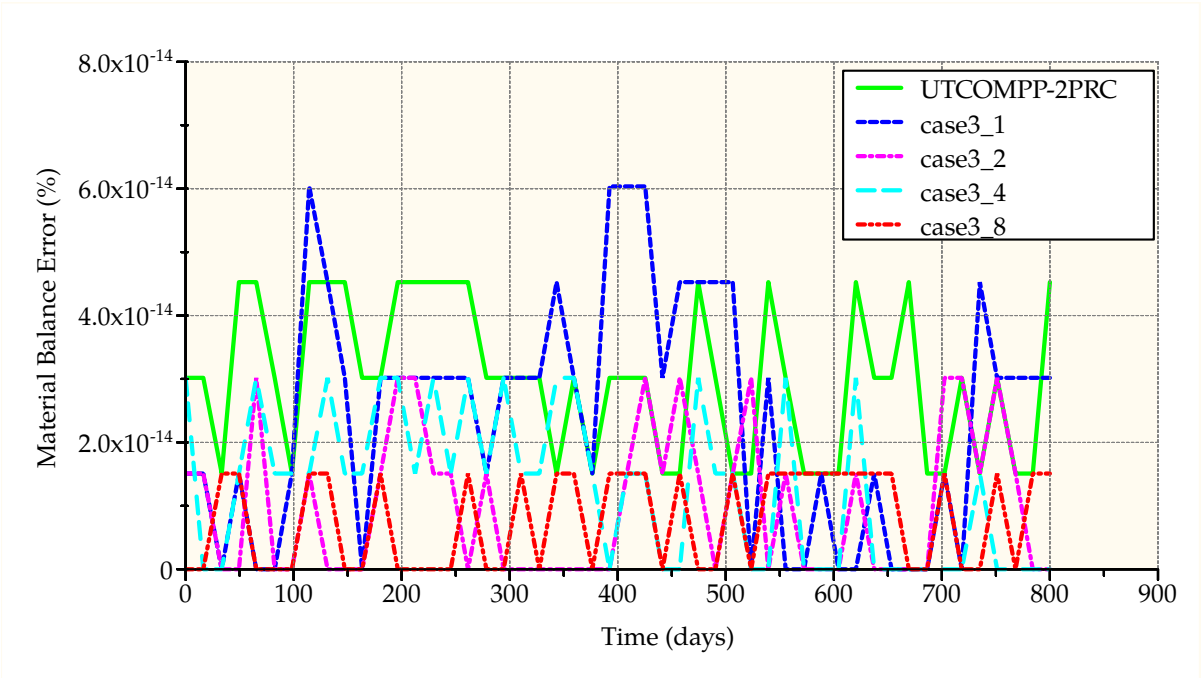


Figure 5.30: Material balance, case 3

Table 5.19: CPU times for case 3

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	8.211	8.47
2	5.375	4.148
4	3.02	2.439
8	2.456	1.729

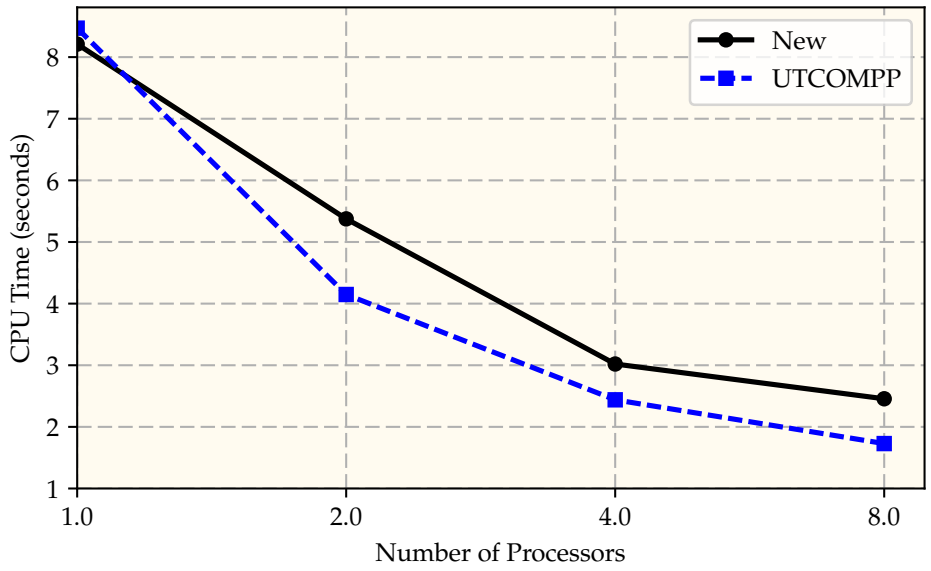


Figure 5.31: CPU time, case 3

Table 5.20: Detailed CPU times in seconds for case 3

	Number of processors used			
	1	2	4	8
Total Initialization Time	0.94	0.953	0.513	0.578
Total Linear Solver Time	0.565	0.529	0.212	0.204
Update Ghost Cells	0.008	0.164	0.241	0.337
Phase Composition Calculation	4.591	2.308	1.161	0.617
Output File and Processors Transfer	0.085	0.147	0.099	0.136
Physical Properties After Solver	0.938	0.61	0.396	0.285
Other	1.037	0.631	0.361	0.263
Total Execution Time	8.211	5.375	3.02	2.456

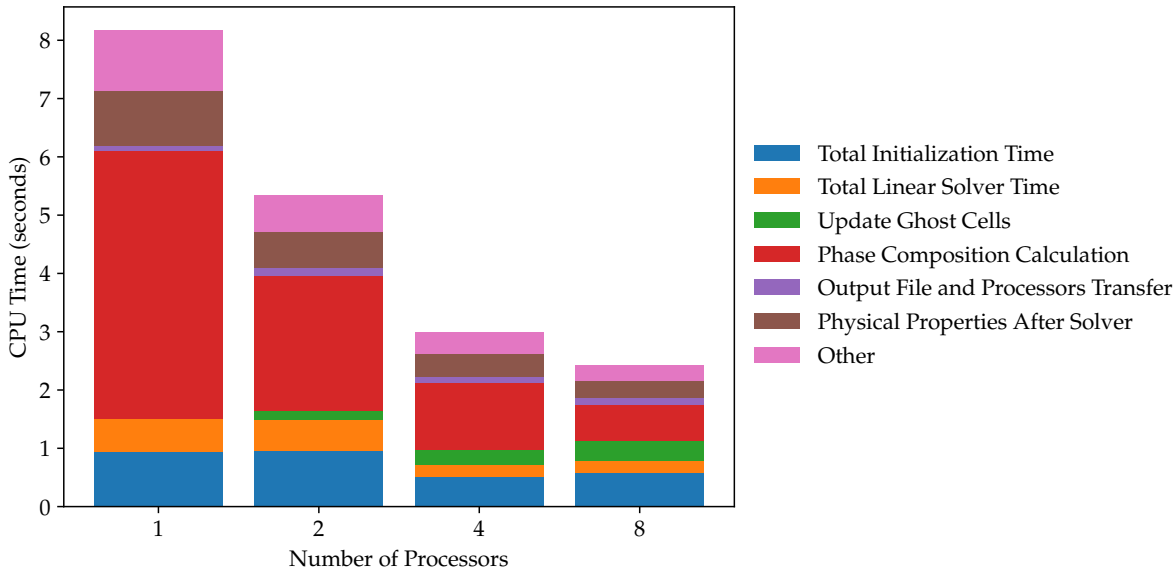


Figure 5.32: Detailed CPU times for case 3

5.1.4 Case 4 - Asphalthene Precipitation

Asphaltene precipitation was implemented in UTCOMP by Qin (1998) and later by Darabi (2014). This case verifies the implementation of the asphaltene precipitation model in our new simulator. This model is Case 5 from Qin (1998). The model is described in Table 5.21.Component properties and compositions are shown in Table 5.22. Relative permeability data is shown in Table 5.23. The model contains one water injector well and one producer well. The injector well is shut in the first year of simulation and then operated at constant injection rate of 300 barrels per day of water. The producer well is operated at constant BHP of 1500 psi for the first year and then at a constant BHP of 1000 psi. Figure 5.33 shows the grid and wells for the simulation model.

Table 5.21: Model description for case 4

Dimensions (ft)	Length	560
	Width	560
	Thickness	100
Number of cells	147	(7x7x3)
Number of components		6
Max. number of phases		3
Porosity		0.25
Permeability (md)	X	800
	Y	800
	Z	800
Rock compressibility (psi^{-1})		5×10^{-6}
Water compressibility (psi^{-1})		3.3×10^{-6}
Initial water saturation		0.3
Irreducible water saturation		0.2
Reservoir temperature (°F)		212
Initial reservoir pressure (psi)		5200
Number of wells	2	1 Injector 1 Producer
Simulation time (days)		1460

Table 5.22: Component properties and compositions for case 4

	Properties					Composition (molar fraction)
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial
CO_2	547.56	1070.09	1.5071	44.010	0.22500	0.0246
C_1-C_2	360.61	668.51	1.6431	17.417	0.015127	0.4041
C_3-C_5	732.89	573.15	3.8098	53.516	0.179313	0.0755
C_6-C_{19}	1135.31	291.41	13.7197	164.423	0.655007	0.2719
C_{20}-C_{30}	1419.29	175.41	29.0330	340.927	1.064023	0.1064
C_{31+}	1682.93	143.17	56.5486	665.624	1.371778	0.0775
Asphaltene						0.04

Table 5.23: Relative permeability data for case 4

Model	Stone's model II used in the SPE 5th comparative study (Stone 1973; Killough and Kossack 1987)			
	Water	Gas	Oil	
Residual saturation	0.20	0.05	water-oil gas-oil	0.30 0.15
End point	0.4089	0.39		1.00
Exponent	3.0	3.0	water-oil gas-oil	2.0000 2.1952

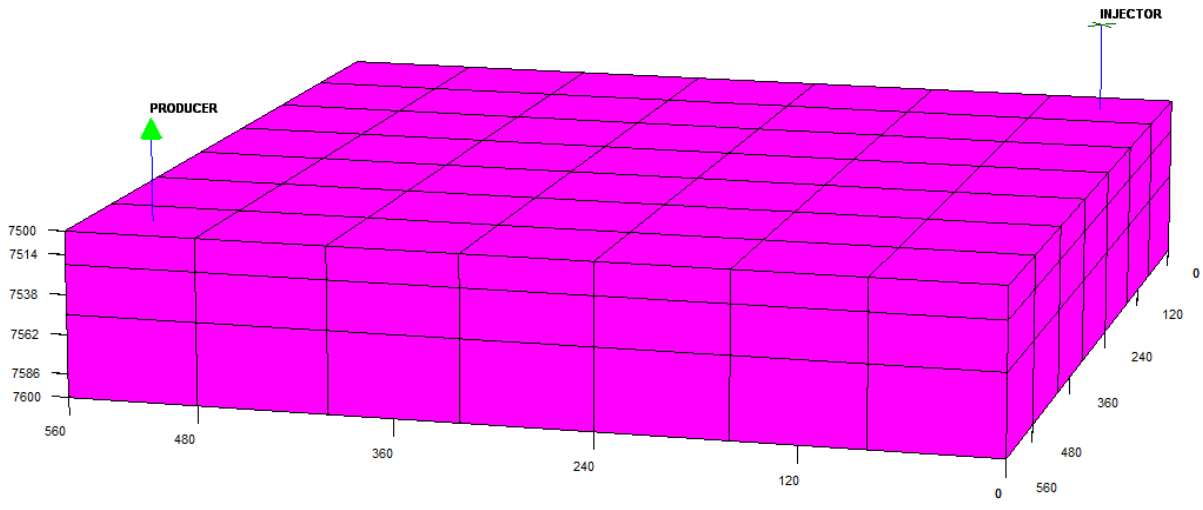


Figure 5.33: Grid and wells for verification case 3

In this model up to 4 processors were used. Table 5.24 shows domains and pressure distribution at the end of the simulation when different number of processors were used. Table 5.25 shows oil , gas and water saturation distribution at the end of the simulation when different number of processors were used. Figures 5.34, 5.35, and 5.36 5.37 show average reservoir pressure, surface oil production rate and gas production rate compared with UTCOMPP. Figure 5.38 shows the material balance error when different number of processors are used.

Table 5.24: Results for case 4. Domain decomposition and pressure distribution

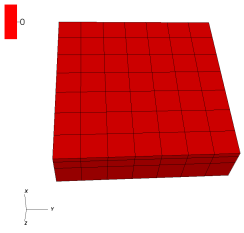
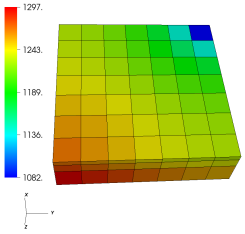
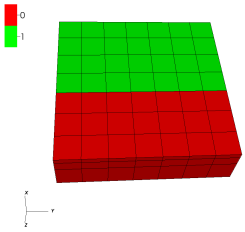
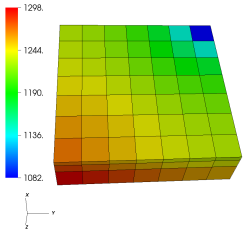
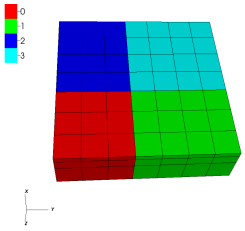
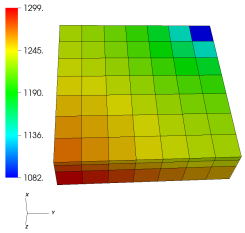
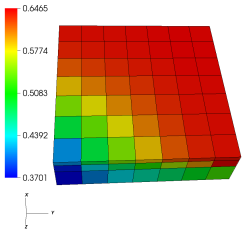
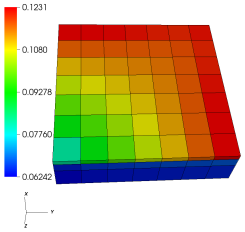
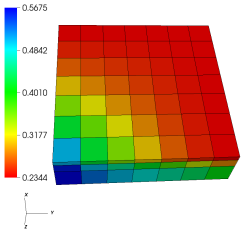
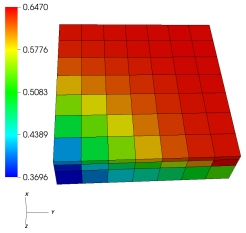
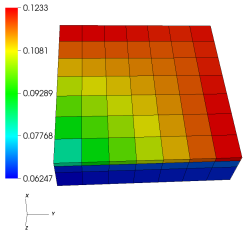
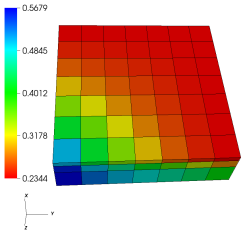
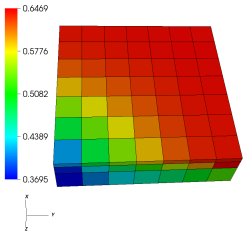
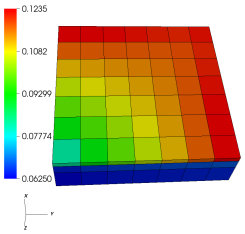
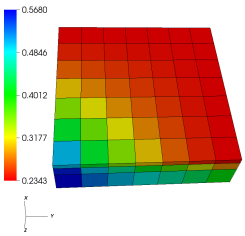
Prcs	Domains	Pressure (psi)
1		
2		
4		

Table 5.25: Results for case 4. Saturation distribution

Prcs	S_o	S_g	S_w
1			
2			
4			

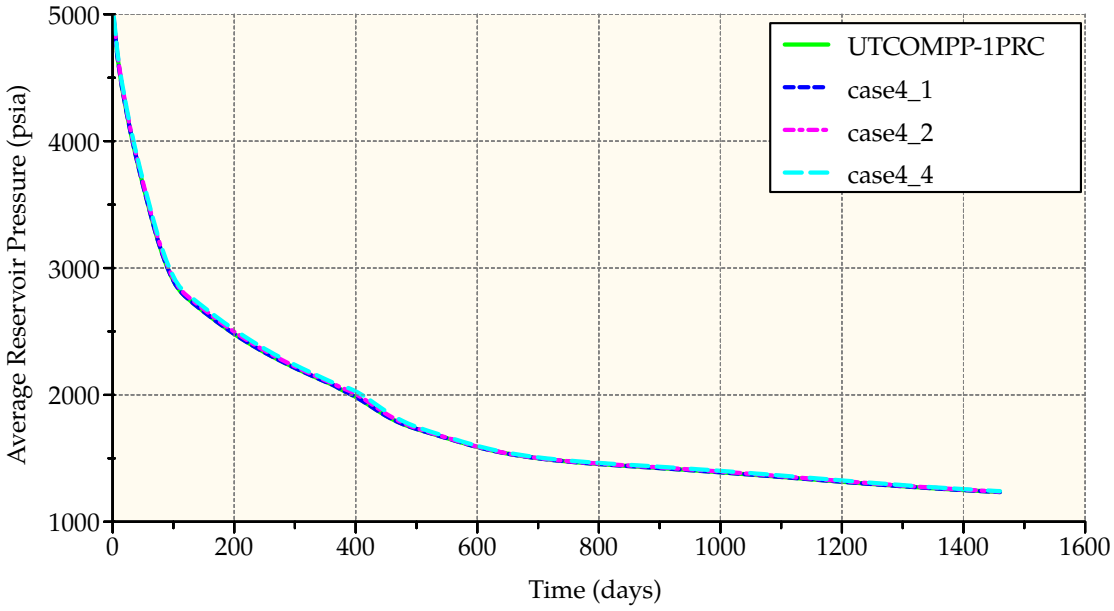


Figure 5.34: Average reservoir pressure, case 4

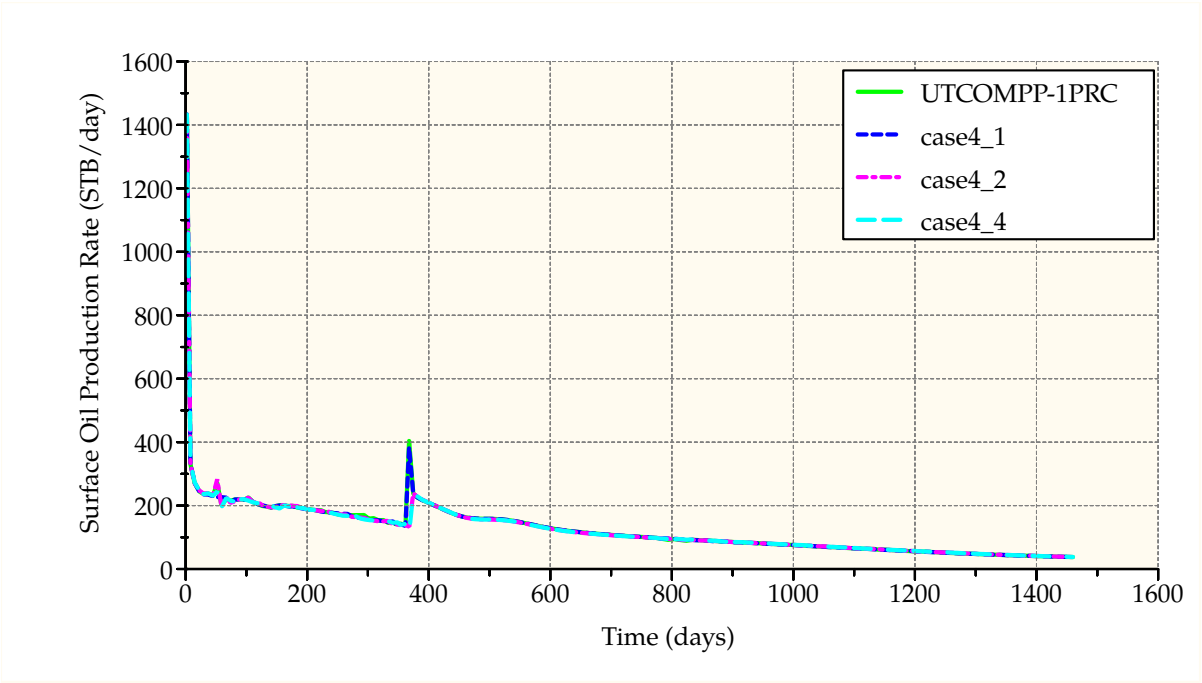


Figure 5.35: Surface oil production rate, case 4

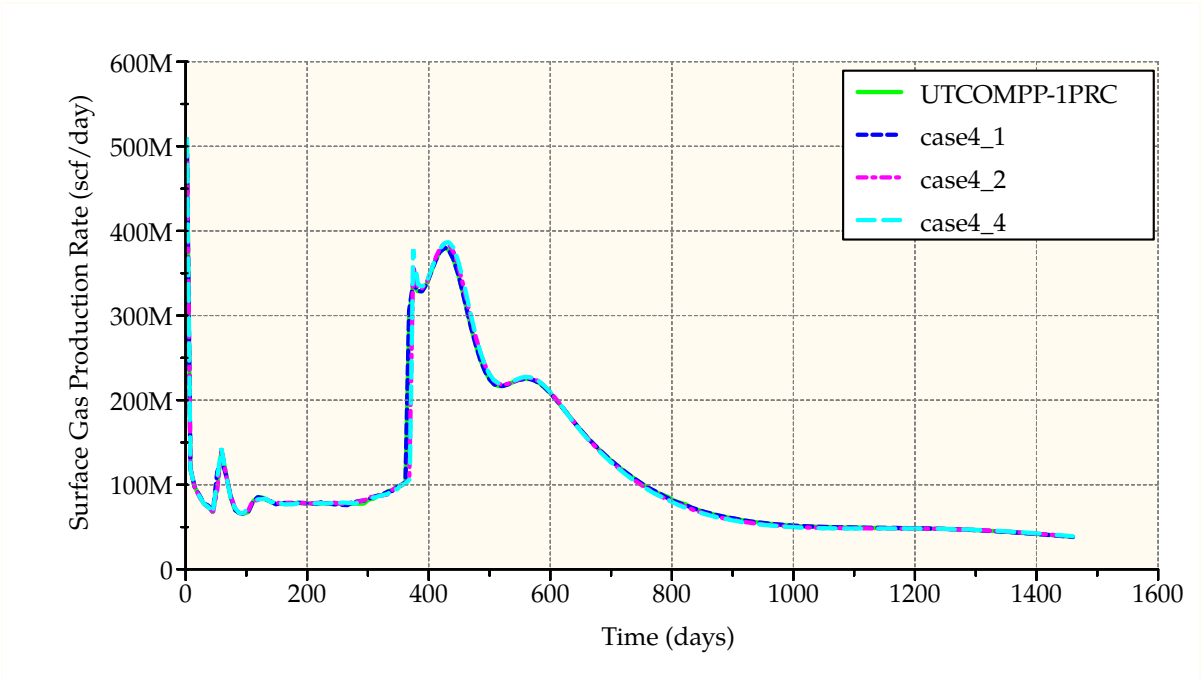


Figure 5.36: Surface gas production rate, case 4

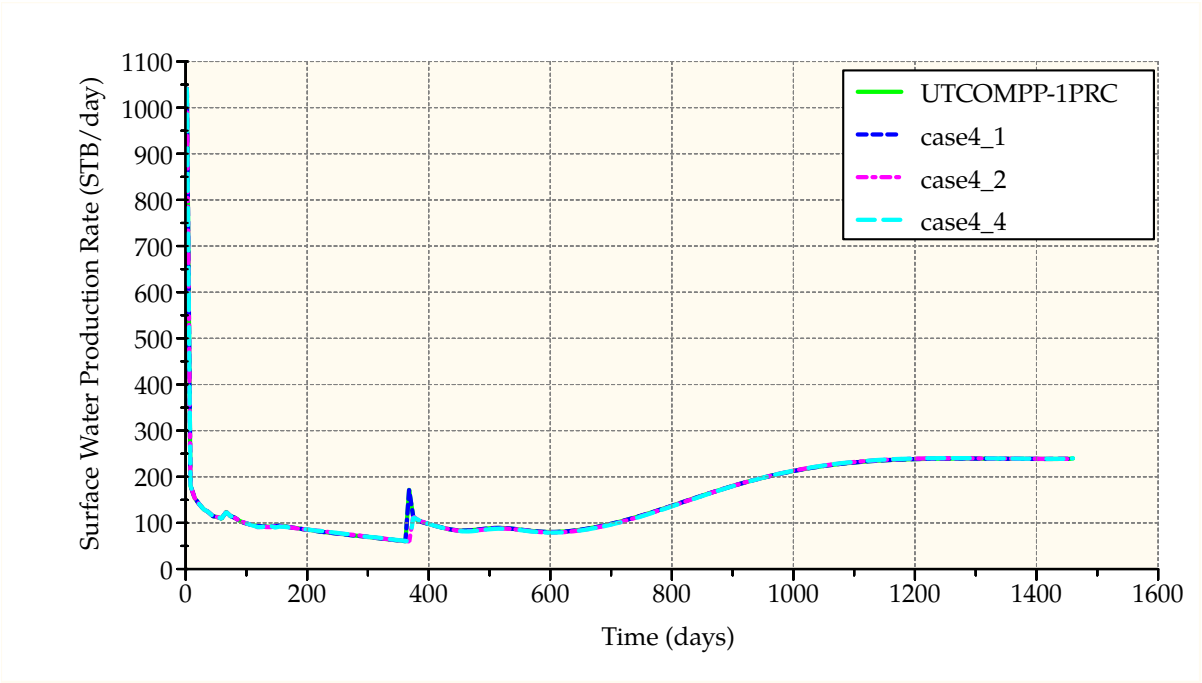


Figure 5.37: Surface water production rate, case 4

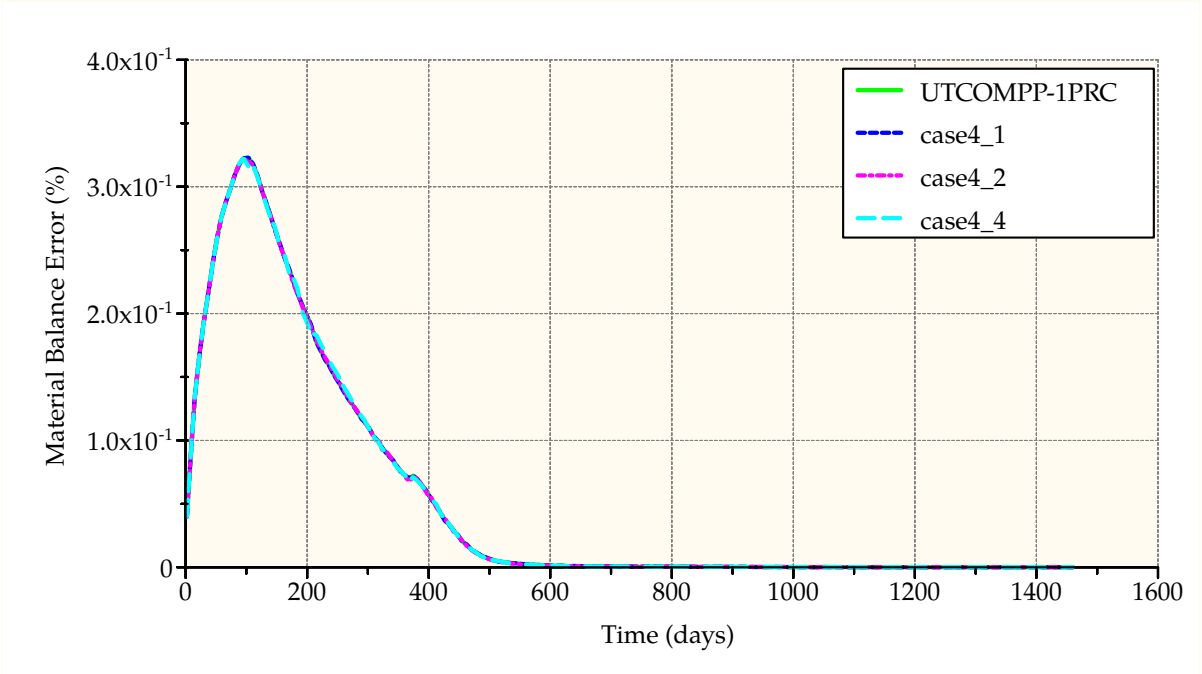


Figure 5.38: Material balance, case 4

Table 5.26 and Figure 5.39 show the CPU time when different number of processors are used. Our new simulator has larger CPU times than UTCOMPP because of the overhead in initial processing of input data. This overhead is noticeable in this case because the model is very small and computation and communication times are very small.

Table 5.26: CPU times for case 4

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	5.911	5.391
2	3.952	3.333
4	2.793	2.127

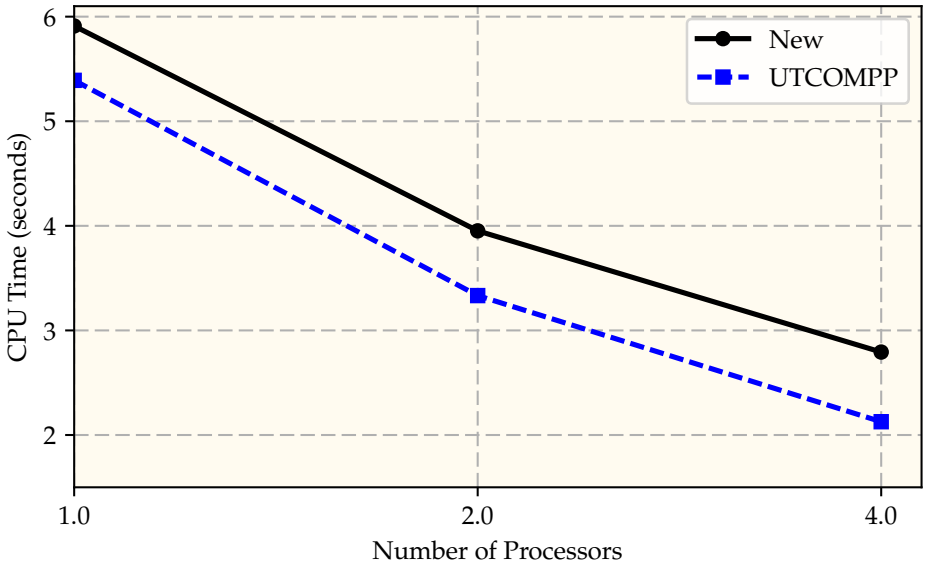


Figure 5.39: CPU time, case 4

5.2 Performance Cases

5.2.1 Case 5 - WAG Heterogeneous

This case simulates a water alternate gas injection process in a heterogeneous reservoir. The model is described in Table 5.27. Figures 5.40 and 5.41 show porosity and permeability distribution. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.28. Relative permeability data is shown in Table 5.29. The model contains one injector and one producer well. Figure 5.42 shows the grid and wells for the simulation model. Table 5.30 shows the well control schedule through the simulation.

Table 5.27: Model description for case 5

Dimensions (ft)	Length Width Thickness	3500 3500 100
Number of cells	200000	(200x200x5)
Number of components		6
Max. number of phases		3
Porosity		Figure 5.40
Permeability (md)	X Y Z	Figure 5.41 Figure 5.41 10
Rock compressibility (psi^{-1})		5×10^{-6}
Water compressibility (psi^{-1})		3.3×10^{-6}
Initial water saturation		0.2
Irreducible water saturation		0.2
Reservoir temperature (°F)		160
Initial reservoir pressure (psi)		3000
Number of wells	2	1 Injector 1 Producer
Simulation time (days)		2410

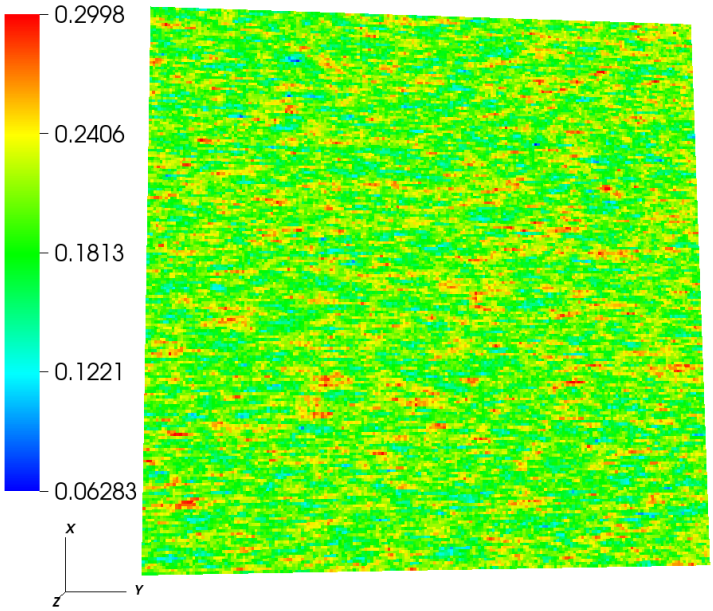


Figure 5.40: Porosity distribution for case 5

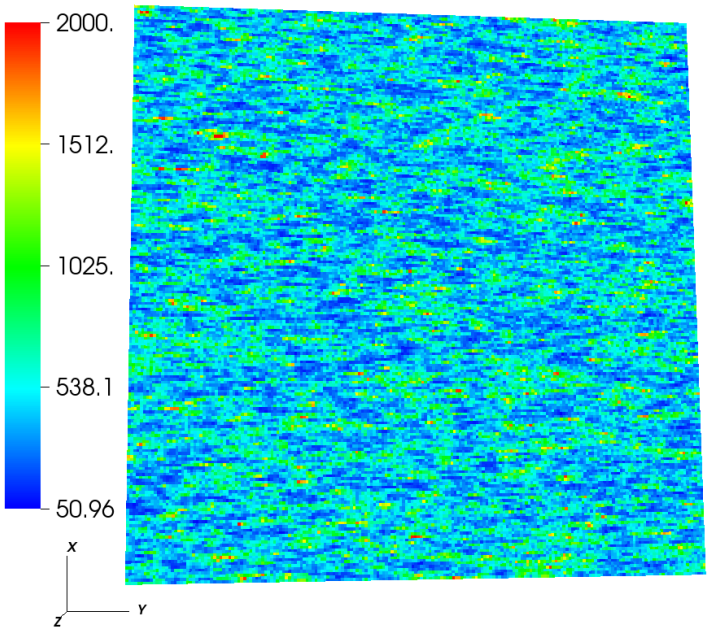


Figure 5.41: Permeability distribution for case 5 (md)

Table 5.28: Component properties and compositions for case 5

	Properties					Composition (molar fraction)	
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
C_1	343.0	667.784	1.6	16.040	0.013	0.3	0.77
C_3	665.7	616.348	3.21	44.100	0.1524	0.13	0.2
C_6	913.4	436.911	5.92	86.180	0.3007	0.17	0.01
C_{10}	1111.8	304.059	10.08	142.290	0.4885	0.2	0.01
C_{15}	1270.0	200.012	16.69	206.0	0.6500	0.15	0.005
C_{20}	1380.0	161.949	21.49	282.0	0.8500	0.05	0.005

Table 5.29: Relative permeability data for case 5

Model	Corey's model (Corey 1986; UTCOMP 2003)			
	Water	Gas	Oil	
Residual saturation	0.20	0.00	water-oil gas-oil	0.00001 0.15
End point	1.00	1.00		1.00
Exponent	2.0	4.0	water-oil gas-oil	4.0 3.0

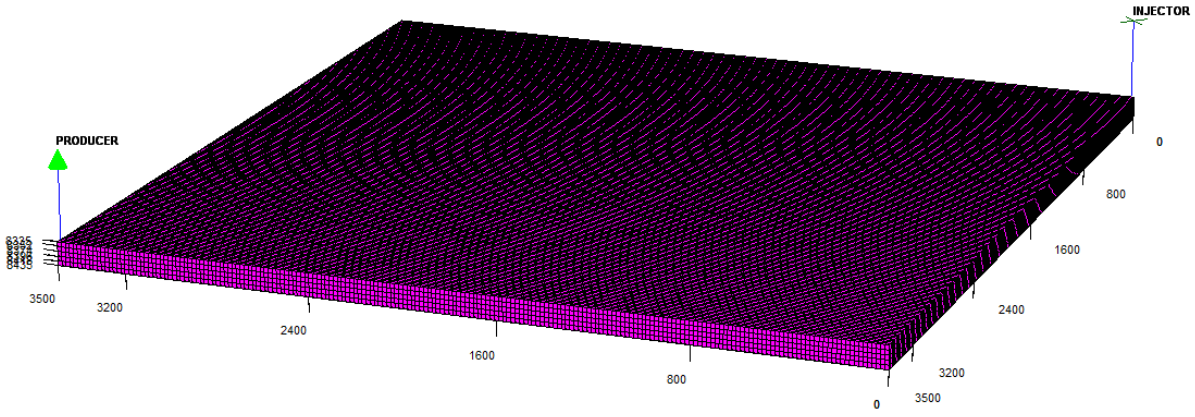


Figure 5.42: Grid and wells for case 5

Table 5.30: Well schedule for case 5

Sim. days	Producers Control	Injectors Control	Comments
0 - 200	BHP 2500 psi	closed	Primary production
200 - 2010	BHP 2500 psi	Water Inj. @ BHP 3300psi	1st WAG cycle
2010 - 2110	BHP 2500 psi	Gas Inj. @ BHP 3300psi	
2110 - 2210	BHP 2500 psi	Water Inj. @ BHP 3300psi	2nd WAG cycle
2210 - 2310	BHP 2500 psi	Gas Inj. @ BHP 3300psi	
2310 - 2410	BHP 2500 psi	Water Inj. @ BHP 3300psi	3rd WAG cycle

Up to 128 processors were used in this model. Table 5.31 shows domains and pressure distribution at the end of the simulation when different number of processors were used. Table 5.32 shows oil , gas and water saturation distribution at the end of the simulation when different number of processors were used. Results comparison of our new simulator against UTCOMPP for average reservoir pressure, surface oil production rate, gas production rate and water production rate are shown in Figures 5.43, 5.44, 5.45, and 5.46, respectively. Figure 5.47 shows the material balance error when different number of processors are used. Maximum material balance error values obtained were in the order of 1.0×10^{-12}.

Table 5.31: Results for case 5. Domain decomposition and pressure distribution

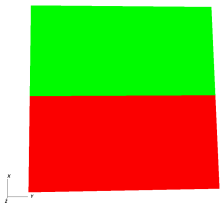
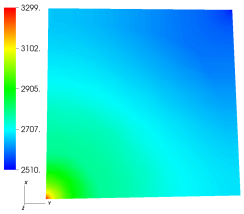
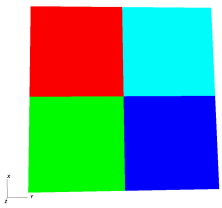
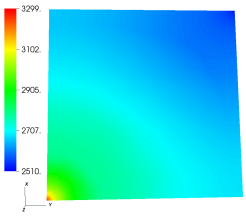
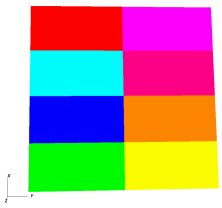
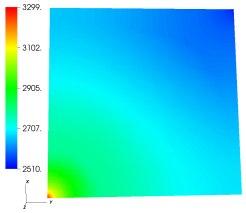
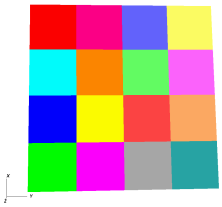
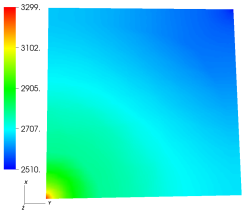
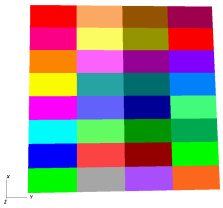
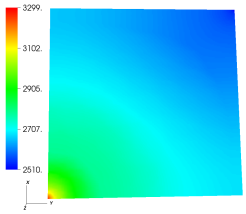
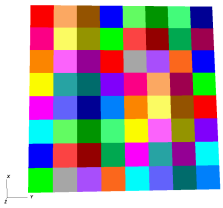
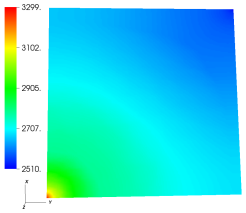
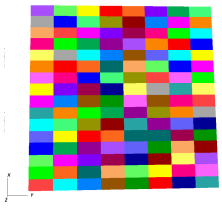
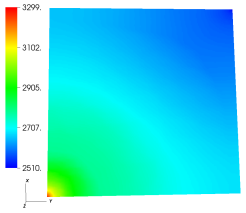
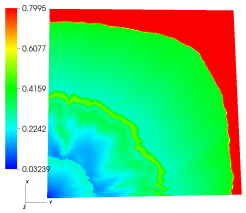
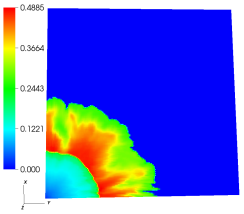
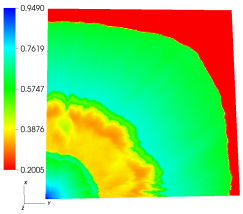
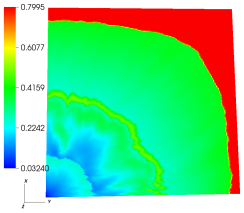
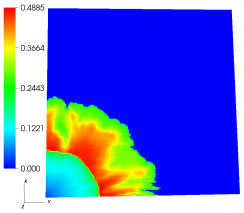
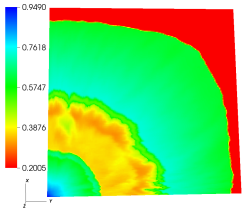
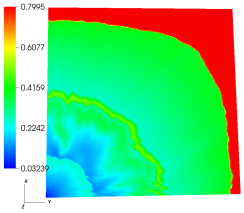
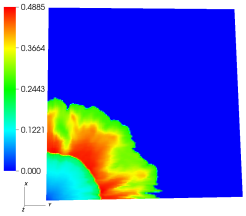
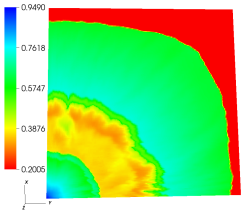
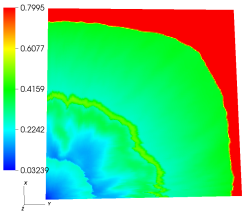
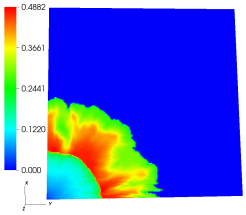
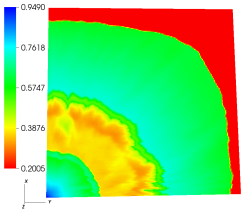
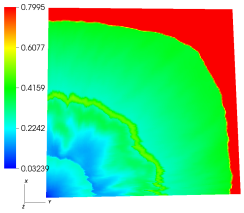
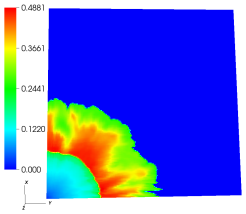
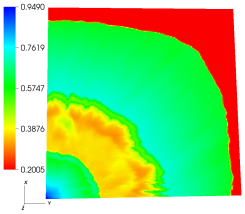
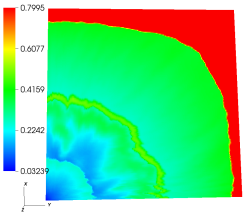
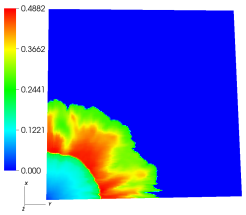
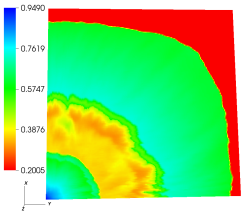
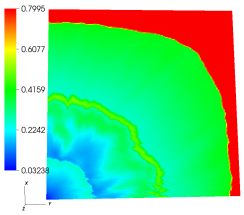
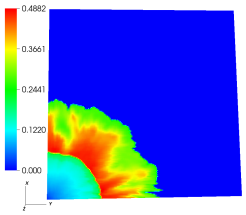
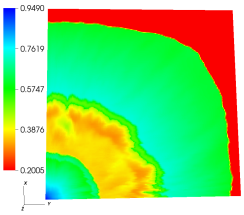
Prcs	Domains	Pressure (psi)
2		
4		
8		
16		
32		
64		
128		

Table 5.32: Results for case 5. Saturation distribution

Prcs	S_o	S_g	S_w
2			
4			
8			
16			
32			
64			
128			

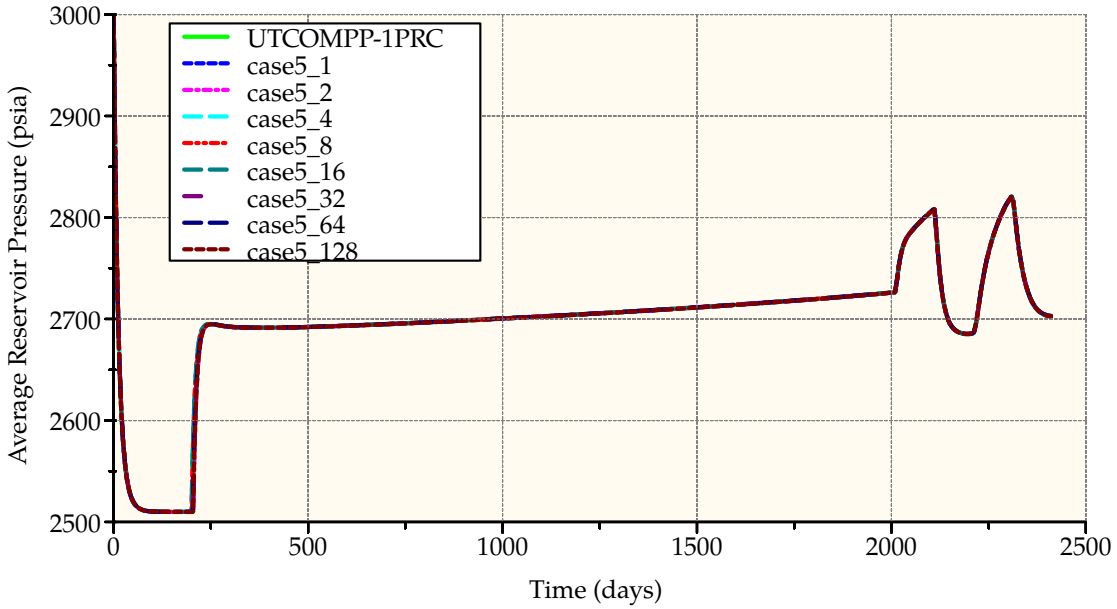


Figure 5.43: Average reservoir pressure, case 5

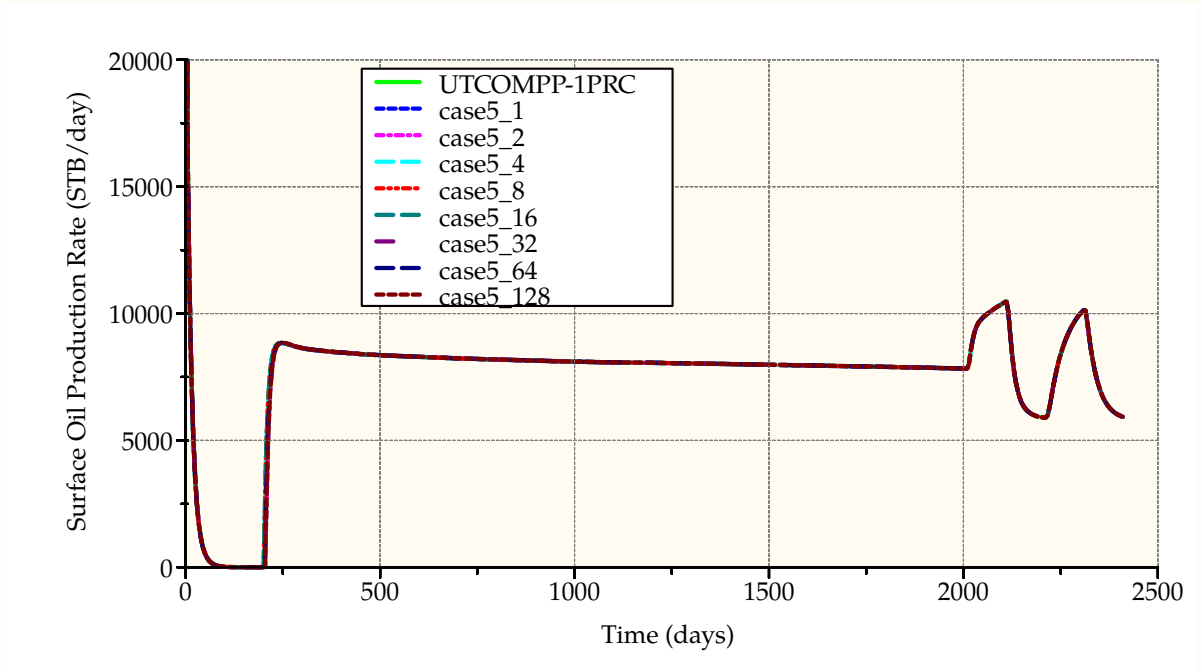


Figure 5.44: Surface oil production rate, case 5

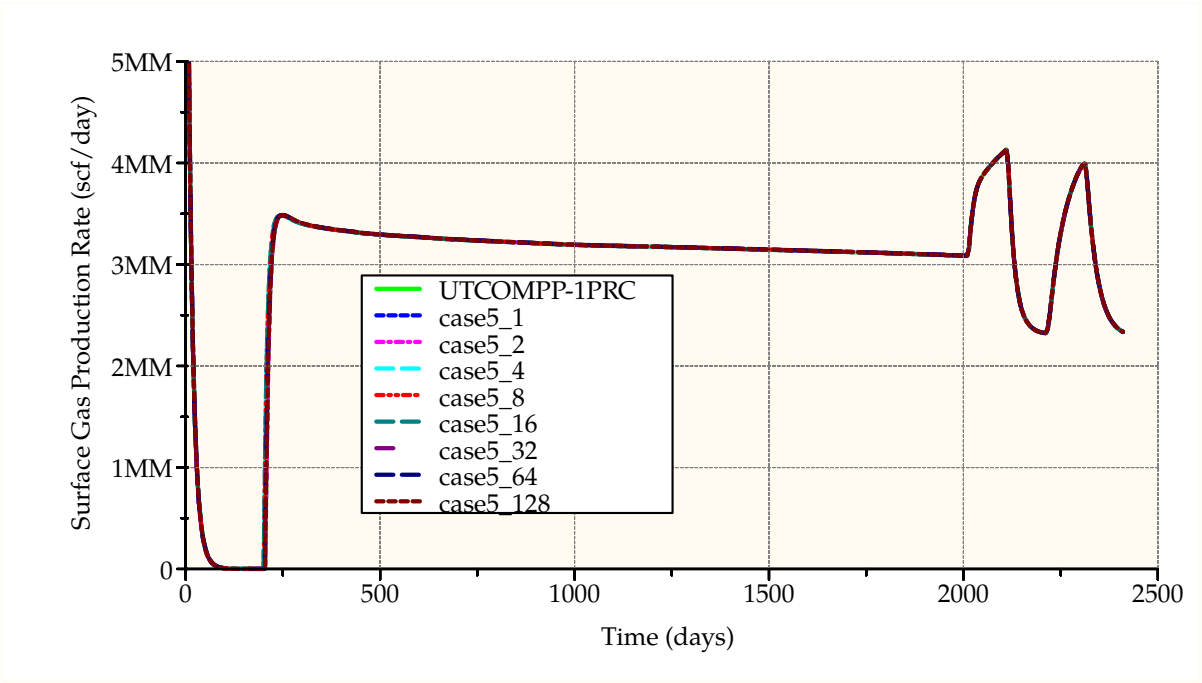


Figure 5.45: Surface gas production rate, case 5

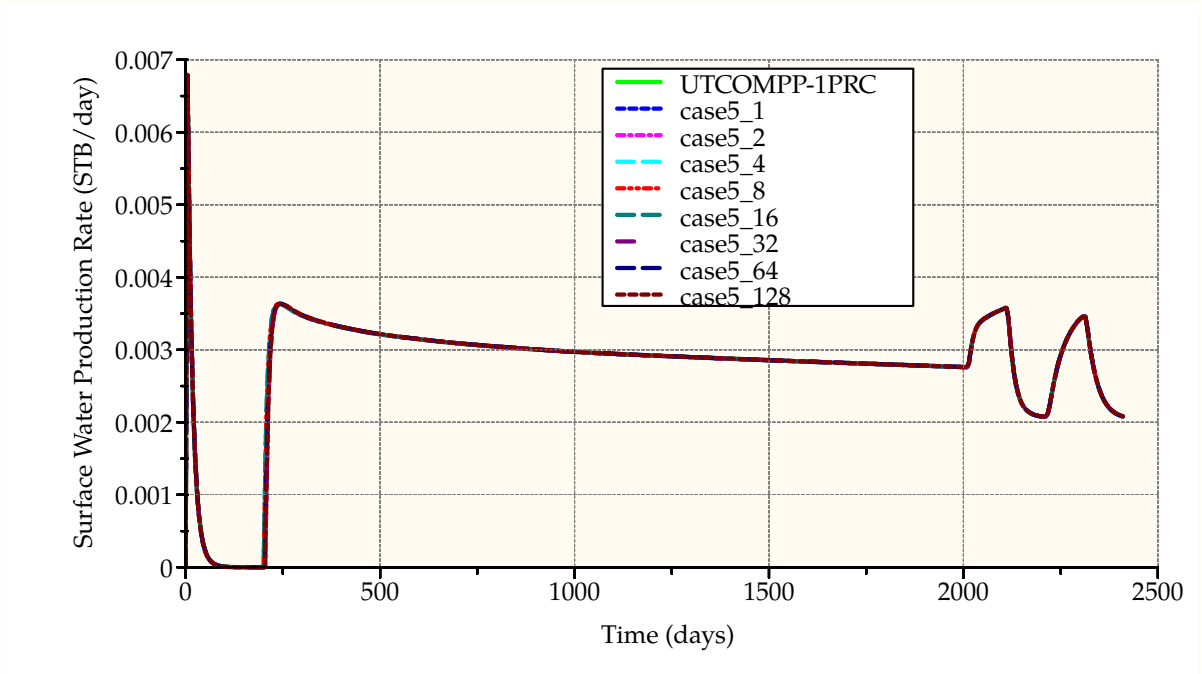


Figure 5.46: Surface water production rate, case 5

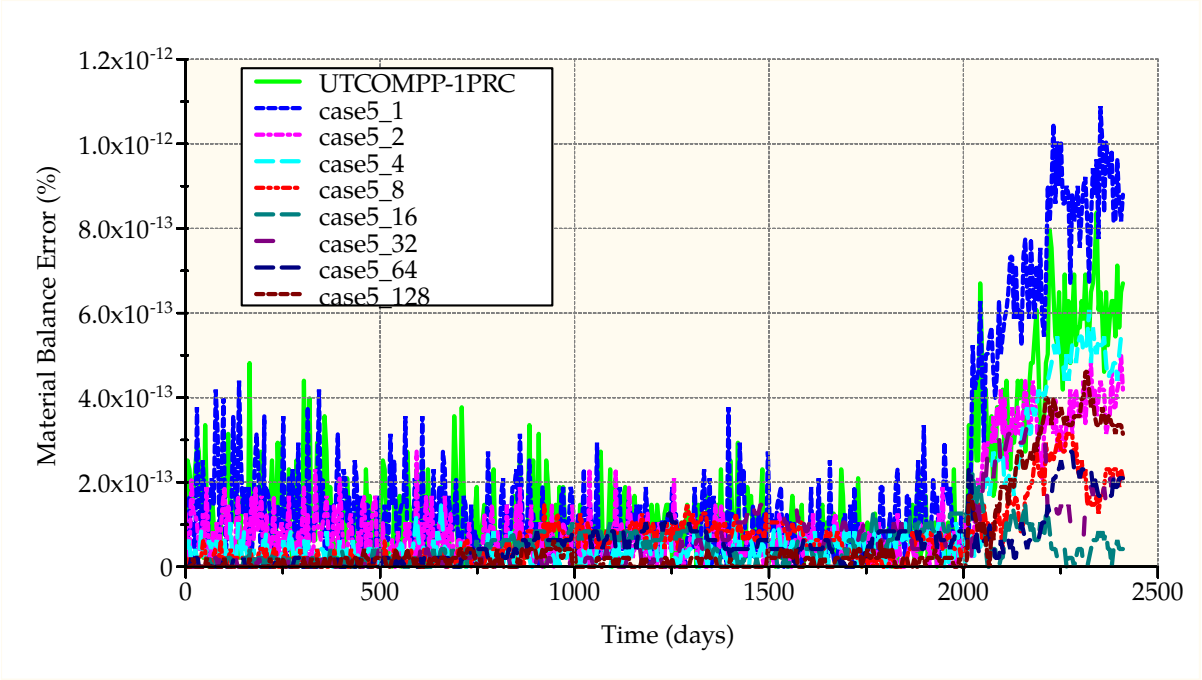


Figure 5.47: Material balance, case 5

Table 5.33 and Figure 5.48 show the CPU time when different number of processors are used. Above 4 processors our new simulator spends more CPU time than UTCOMPP, this is also observed in a lower speedup at above 4 processors. Table 5.34 and Figure 5.49 show the speedup obtained with different number of processors used. The detailed CPU time for our new simulator can be observed in Table 5.35 and Figure 5.50, for comparison, the detailed CPU time for UTCOMPP is shown in Table 5.36 and Figure 5.51. Figure 5.52 shows the percentage of total CPU time spent for our new simulator and Figure 5.53 for UTCOMPP. It can be observed from the detailed CPU time that the source of increase of CPU time in our new simulator comes from the communication between processors (Figure 5.54).

Table 5.33: CPU times for case 5

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	165563	166743
2	93030.9	92706
4	58040.1	52780.9
8	39698.4	30949.9
16	28214.3	16936.7
32	16597.2	9103.22
64	10083	5362.61
128	5954.42	3224.01

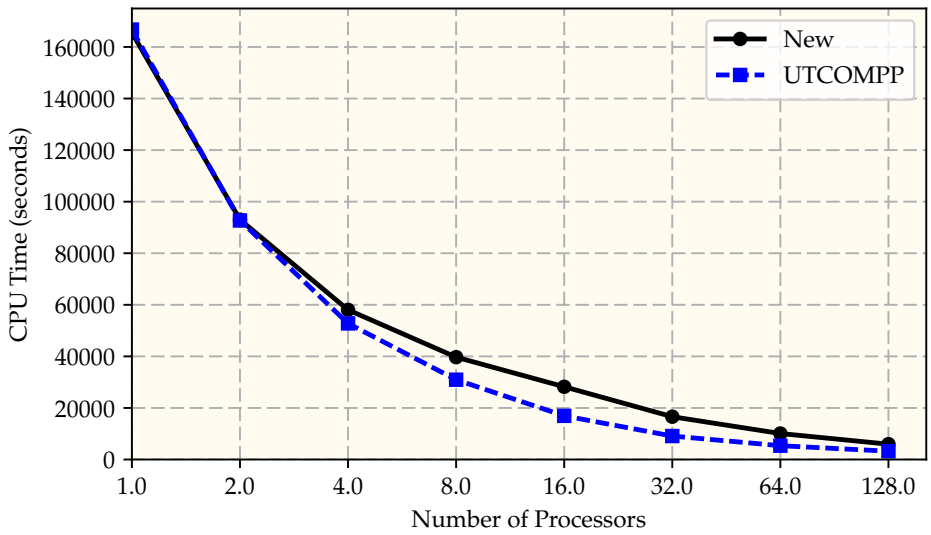


Figure 5.48: CPU time, case 5

Table 5.34: Speedup for case 5

	Speedup	
# processors	New simulator	UTCOMPP
1	1	1
2	1.77965	1.79862
4	2.85256	3.15915
8	4.17051	5.38751
16	5.86803	9.84505
32	9.97534	18.3169
64	16.42	31.0936
128	27.805	51.7191

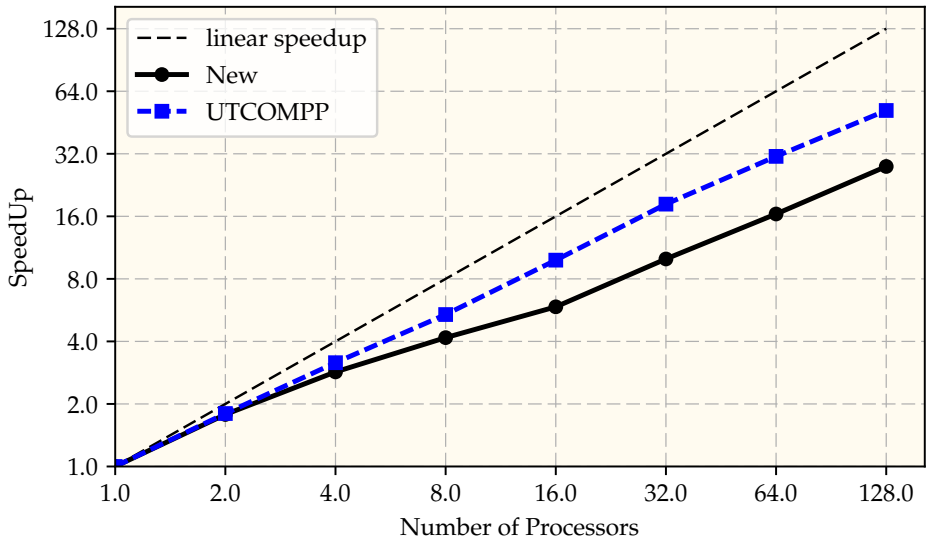


Figure 5.49: Speedup, case 5

Table 5.35: Detailed CPU times in seconds for case 5

	Number of processors used							
	1	2	4	8	16	32	64	128
Total Linear Solver Time	11355.9	6018.75	3210.39	1795.11	973.91	573.192	445.573	425.532
Update Ghost Cells	2.284	717.839	5962.82	12698.6	13833.7	8965.41	4855.12	1979
Compute Derivatives	9588.75	4090.35	2125.29	1218.4	854.962	353.859	226.034	86.855
Physical Properties Before Solver	13143.7	6419	3299.82	1366.12	1074.76	624.941	288.887	116.114
Phase Composition Calculation	110363	55180.1	27693.2	14250.9	7430.2	3761.93	1927.46	972.516
Physical Properties After Solver	10379.8	15401.7	13064.9	6808.91	3197.58	1403.57	622.936	302.405
Other	10726.7	5182.64	2682.58	1511.91	843.597	414.44	249.948	160.662
Total Execution Time	165563	93030.9	58040.1	39698.4	28214.3	16597.2	10083	5954.42

Table 5.36: Detailed CPU times in seconds for UTCOMPP, case 5

	Number of processors used							
	1	2	4	8	16	32	64	128
Total Linear Solver Time	11554.7	5995.14	3215.97	1770.06	1010.43	613.143	582.858	575.942
Update Ghost Region	0.737	324.011	8378.73	7153.2	4647.88	1888.51	820.052	445.007
Compute Derivatives	9389.74	4077.09	2101.68	1190.96	857.403	387.061	226.682	90.793
Physical Properties Before Solver	13091.5	6379.25	2366.96	1754.77	949.206	483.425	244.558	100.591
Phase Composition Calculation	111594	55368.5	26928.4	14712.3	7277.67	3701.86	1908.66	958.28
Physical Properties After Solver	10352.7	15333.2	7066.89	2873.49	1281.1	535.111	266.956	132.561
Other	10759	5174.99	2692.09	1475.83	832.176	435.623	304.032	292.334
Total Time	166743	92706	52780.9	30949.9	16936.7	9103.22	5362.61	3224.01

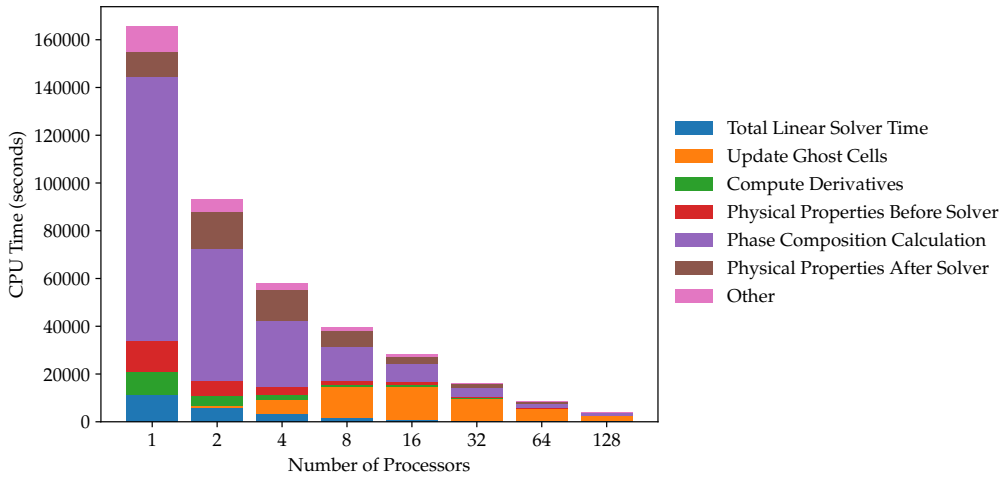


Figure 5.50: Detailed CPU times for case 5

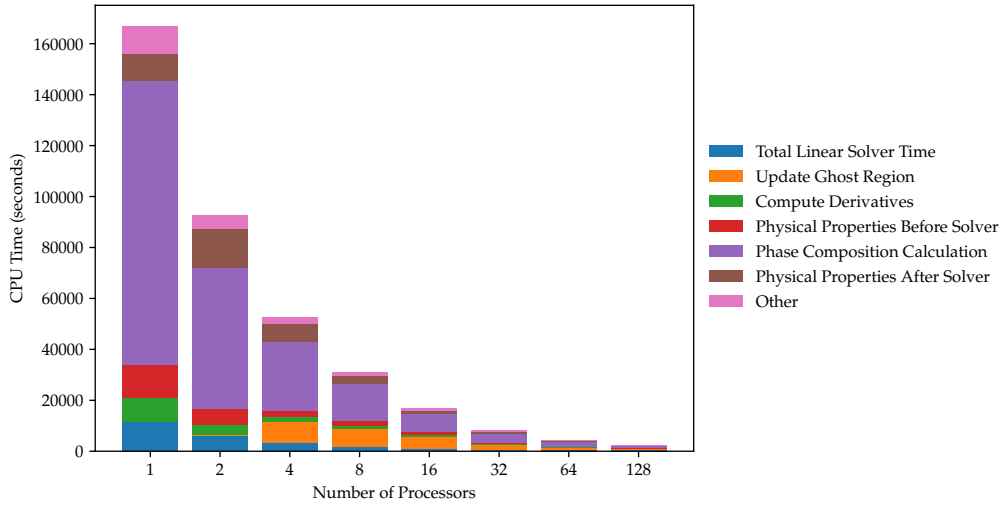


Figure 5.51: Detailed CPU times for UTCOMPP, case 5

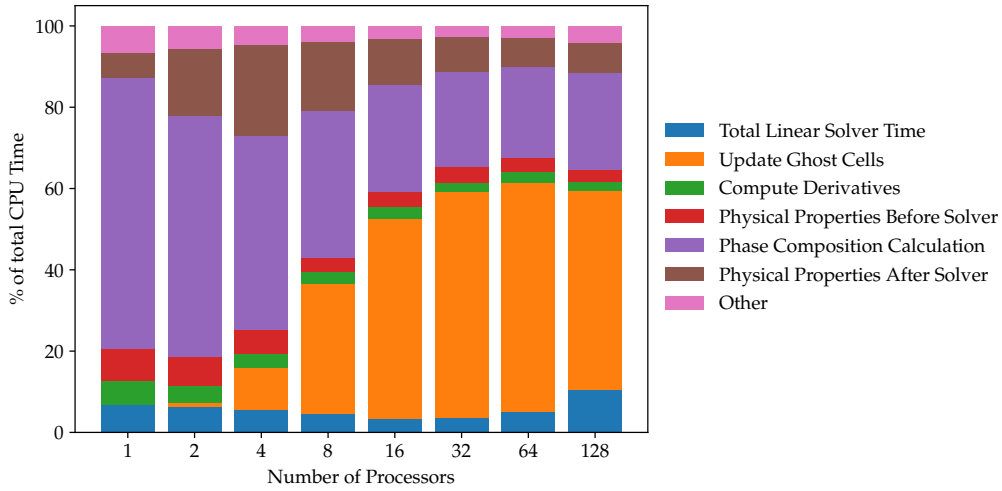


Figure 5.52: Percentage of total CPU time for timers for case 5

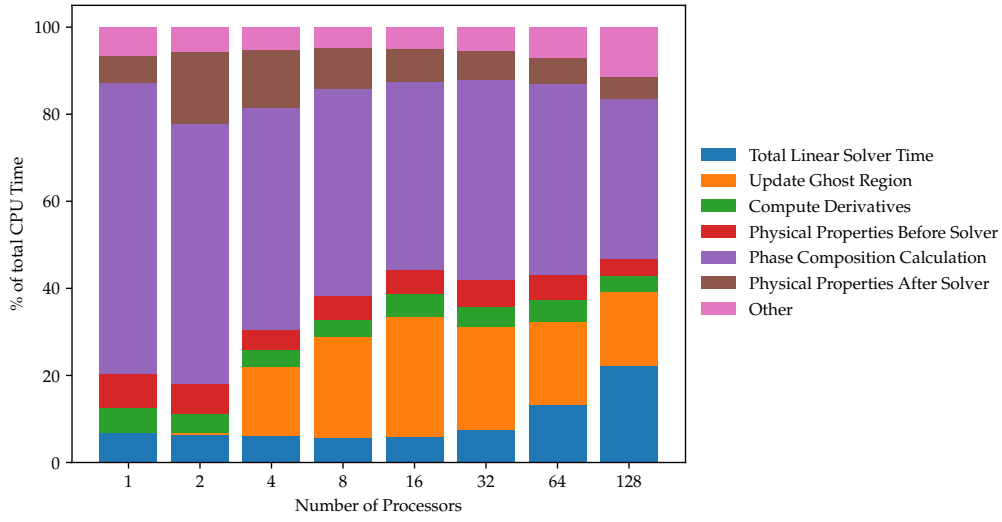


Figure 5.53: Percentage of total CPU time for timers for UTCOMPP, case 5

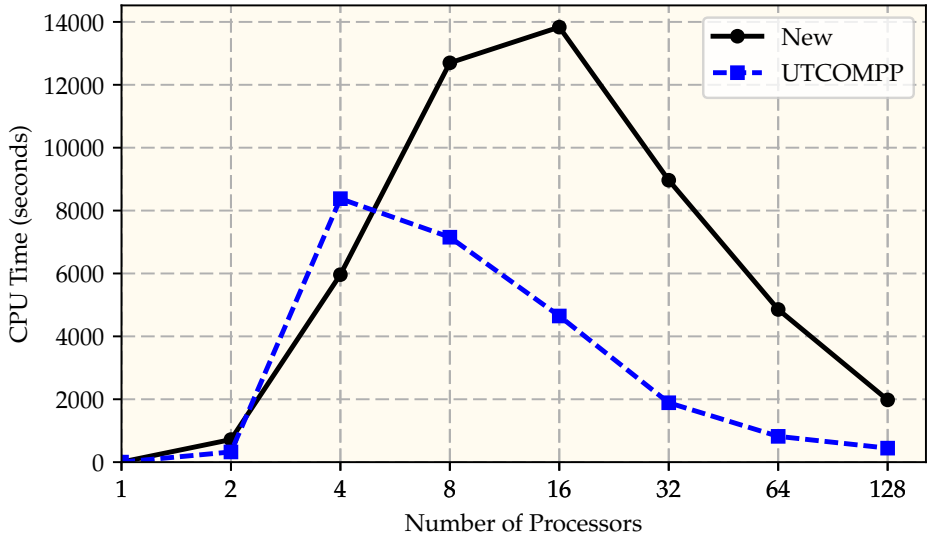


Figure 5.54: CPU time spent on inter processor communication for case 5

5.2.2 Case 6 - WAG Homogeneous

This case simulates a water alternate gas injection process in a homogeneous reservoir. The model is described in Table 5.37. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.38. Relative permeability data is shown in Table 5.39. The model contains one injector and four producer wells in a inverted 5-spot pattern. Figure 5.55 shows the grid and wells for the simulator model. Table 5.40 shows the well control schedule through the simulation.

Table 5.37: Model description for case 6

Dimensions (ft)	Length Width Thickness	4500 4800 50
Number of cells	240000	(150x160x10)
Number of components		6
Max. number of phases		3
Porosity		0.3
Permeability (md)	X Y Z	90 90 90
Rock compressibility (psi^{-1})		4×10^{-6}
Water compressibility (psi^{-1})		3.3×10^{-6}
Initial water saturation		0.3
Irreducible water saturation		0.3
Reservoir temperature (°F)		90
Initial reservoir pressure (psi)		4800
Number of wells	5	1 Injector 4 Producer
Simulation time (days)		4500

Table 5.38: Component properties and compositions for case 6

	Properties					Composition (molar fraction)	
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
CO_2	547.56	1070.16	1.504	44.01	0.225	0.002	0.0018
C_1	343.08	667.38	1.589	16.04	0.008	0.4	0.943
C_2-C_3	594.95	672.38	2.717	34.33	0.1191	0.25	0.039
C_4-C_6	816.91	513.03	4.783	67.13	0.2257	0.25	0.016
PS1	1090.84	382.35	7.855	122.63	0.3056	0.05	0.0001
PS2	1565.39	218.59	16.963	294.67	0.7879	0.048	0.0001

Table 5.39: Relative permeability data for case 6

Model	Corey's model with trapping (UTCOMP 2003; G. Pope et al. 1998)			
	Water	Gas	Oil	
Residual saturation	0.30	0.05	water-oil gas-oil	0.30 0.10
End point	0.50	0.3		0.25
Exponent	1.0	4.0	water-oil gas-oil	2.00 2.93

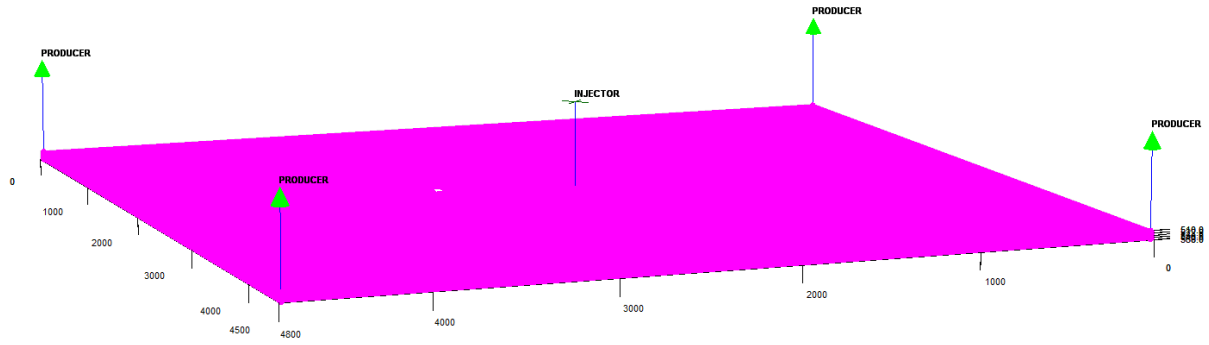


Figure 5.55: Grid and wells for case 6

Table 5.40: Well schedule for case 6

Sim. days	Producers Control	Injectors Control	Comments
0 - 100	BHP 4700 psi	closed	Primary production
100 - 200	BHP 4600 psi	closed	
200 - 365	BHP 4500 psi	closed	
365 - 1460	BHP 4500 psi	Water rate 1500STB/day BHP limit 6000psi	1st WAG cycle
1460 - 1825	BHP 4500 psi	Gas rate 5000scf/day BHP limit 6000psi	
1825 - 2200	BHP 4500 psi	Water rate 1500STB/day BHP limit 6000psi	2nd WAG cycle
2200 - 2400	BHP 4500 psi	Gas rate 5000scf/day BHP limit 6000psi	
2400 - 2800	BHP 4500 psi	Water rate 1500STB/day BHP limit 6000psi	3rd WAG cycle
2800 - 3000	BHP 4500 psi	Gas rate 5000scf/day BHP limit 6000psi	
3000 - 3350	BHP 4500 psi	Water rate 1500STB/day BHP limit 6000psi	4th WAG cycle
3350 - 3500	BHP 4500 psi	Gas rate 5000scf/day BHP limit 6000psi	
3500 - 4000	BHP 4500 psi	Water rate 1500STB/day BHP limit 6000psi	5th WAG cycle
4000 - 4250	BHP 4500 psi	Gas rate 5000scf/day BHP limit 6000psi	
4250 - 4500	BHP 4500 psi	Water rate 1500STB/day BHP limit 6000psi	6th WAG cycle

Up to 64 processors were used in this model. There was not possible to obtain results for one processors because Lonestar 5 queue system impose an 48 hours limit for the duration of any simulation job. When using only one processor, case 6 requires more than 48 hours to finish and hence it was killed before finishing. Table 5.41 shows domains and pressure distribution at the end of the simulation when different number of processors were used. Table 5.42 shows oil , gas and water saturation distribution at the end of the simulation when different number of processors were used. Figures 5.56, 5.57, 5.58 and 5.59 show average reservoir pressure, surface oil production rate, surface gas production rate and surface water production rate compared with UTCOMPP. Figure 5.60 shows the material balance error when different number of processors are used. All material balance error values obtained were lower than 1.5×10^{-11}.

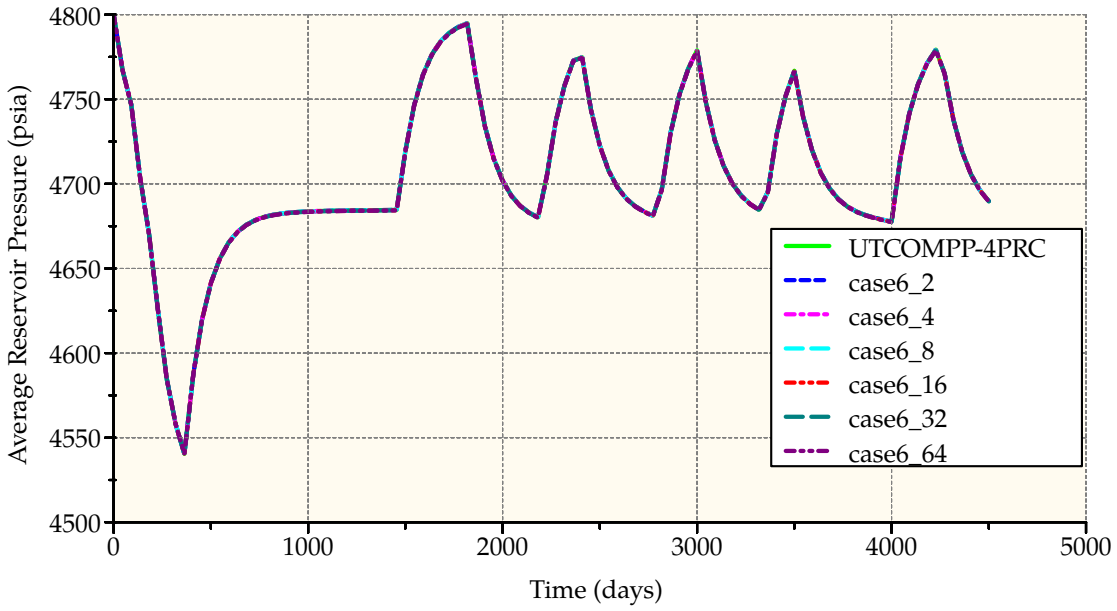


Figure 5.56: Average reservoir pressure, case 6

Table 5.41: Results for case 6. Domain decomposition and pressure distribution

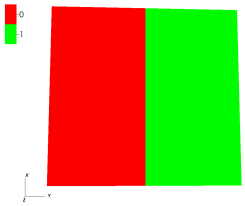
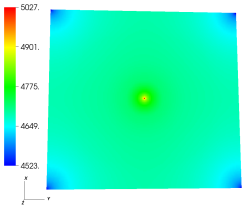
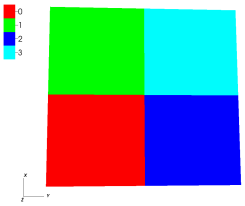
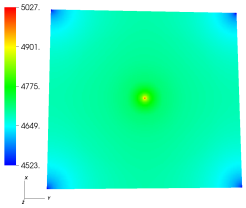
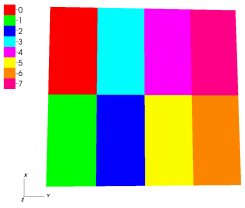
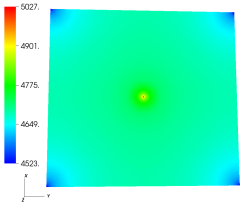
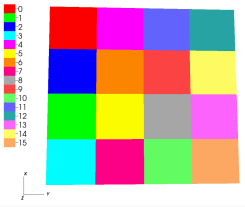
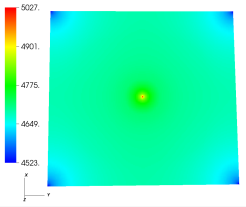
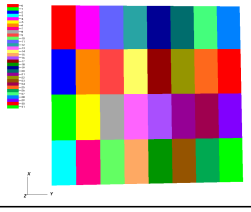
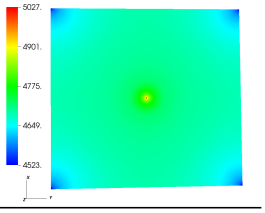
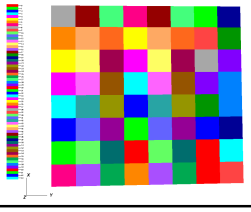
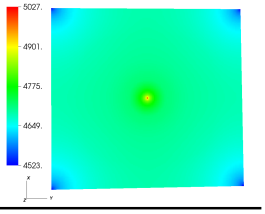
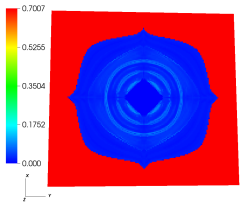
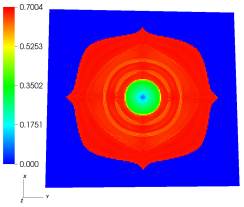
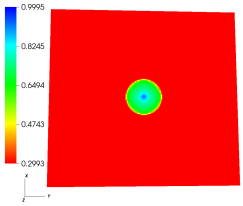
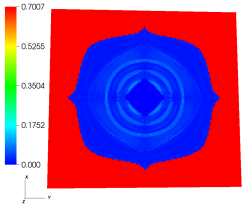
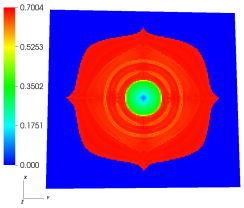
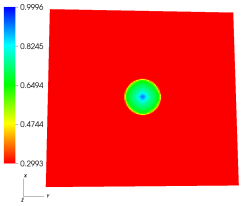
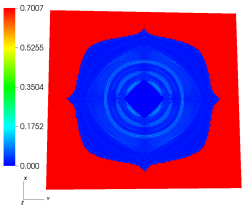
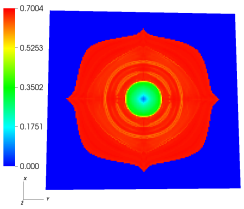
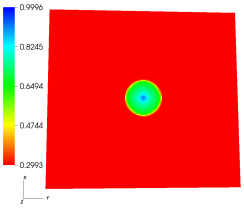
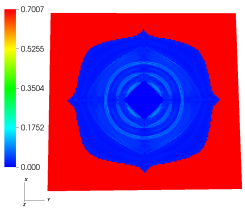
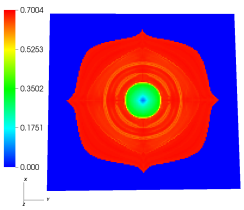
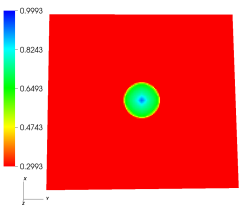
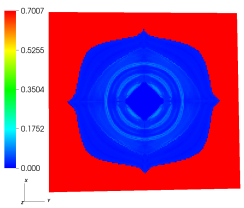
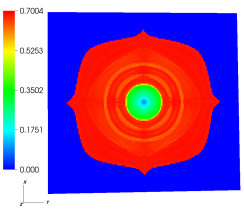
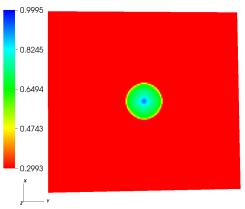
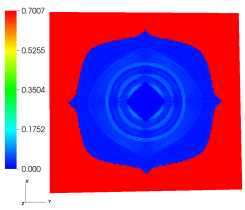
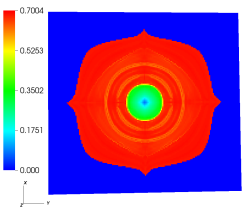
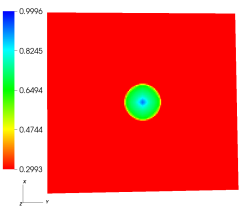
Prcs	Domains	Pressure (psi)
2		
4		
8		
16		
32		
64		

Table 5.42: Results for case 6. Saturation distribution

Prcs	S_o	S_g	S_w
2			
4			
8			
16			
32			
64			

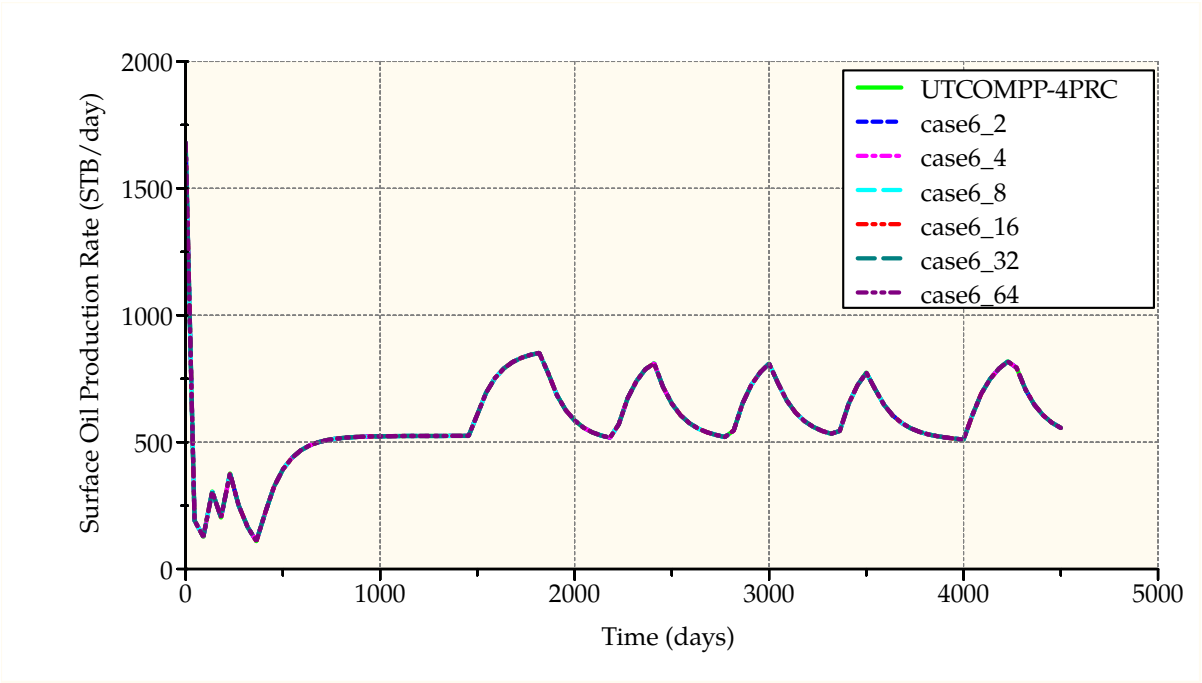


Figure 5.57: Surface oil production rate, case 6

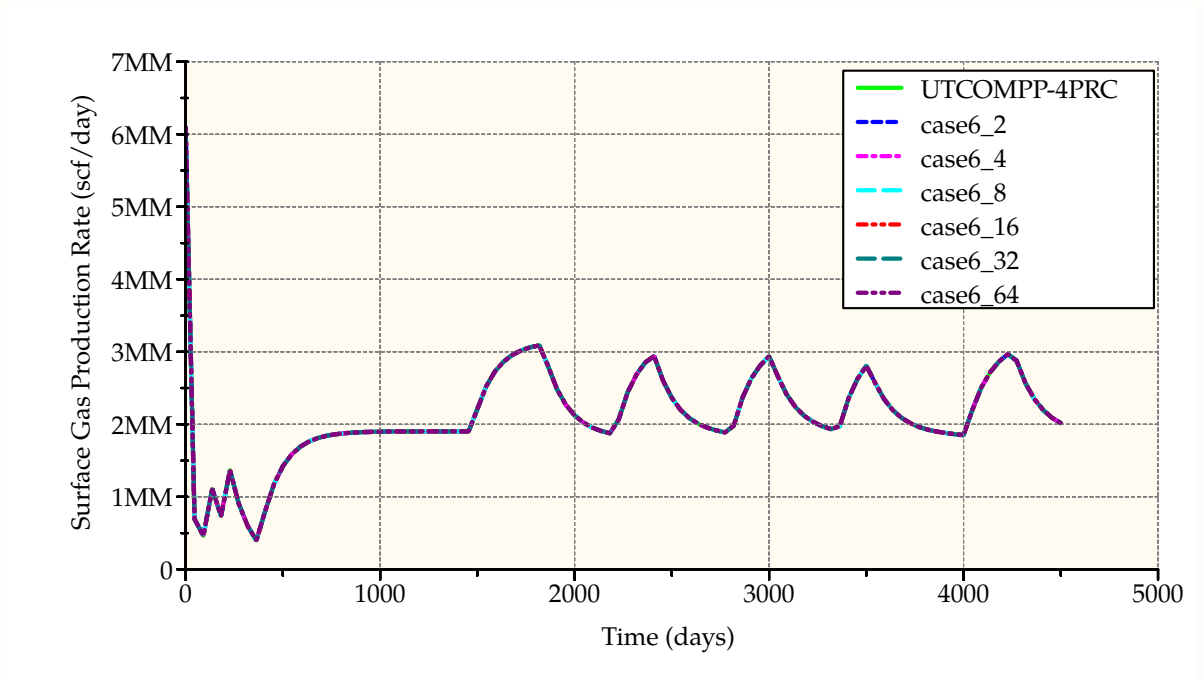


Figure 5.58: Surface gas production rate, case 6

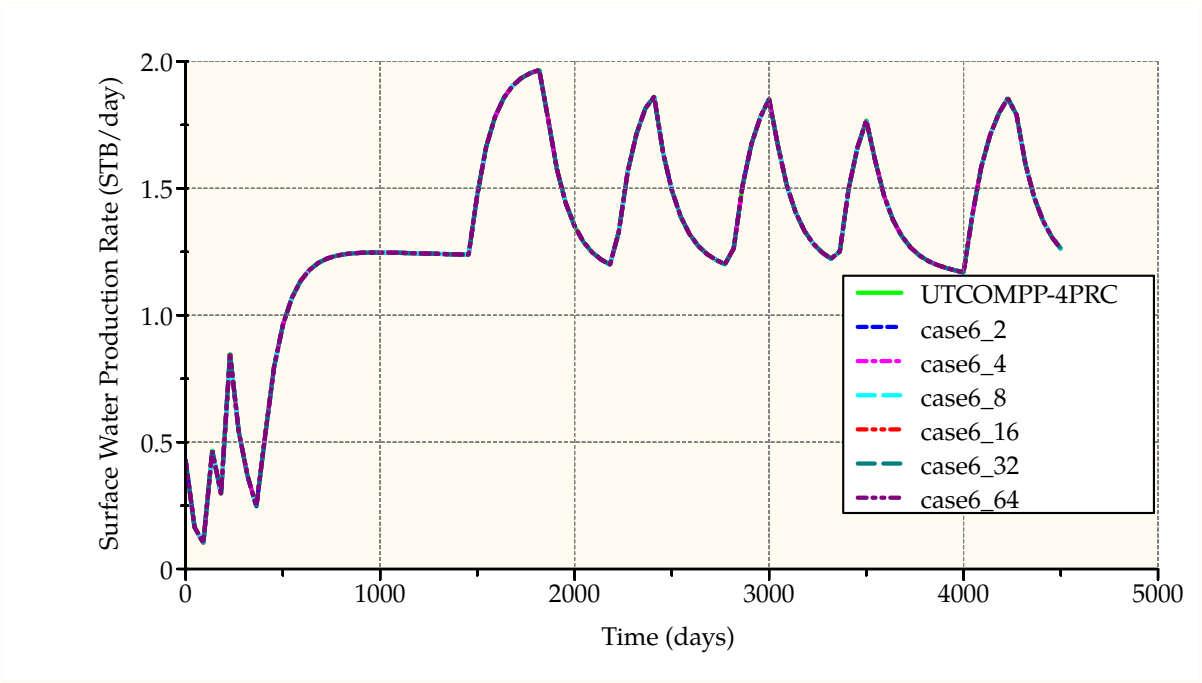


Figure 5.59: Surface water production rate, case 6

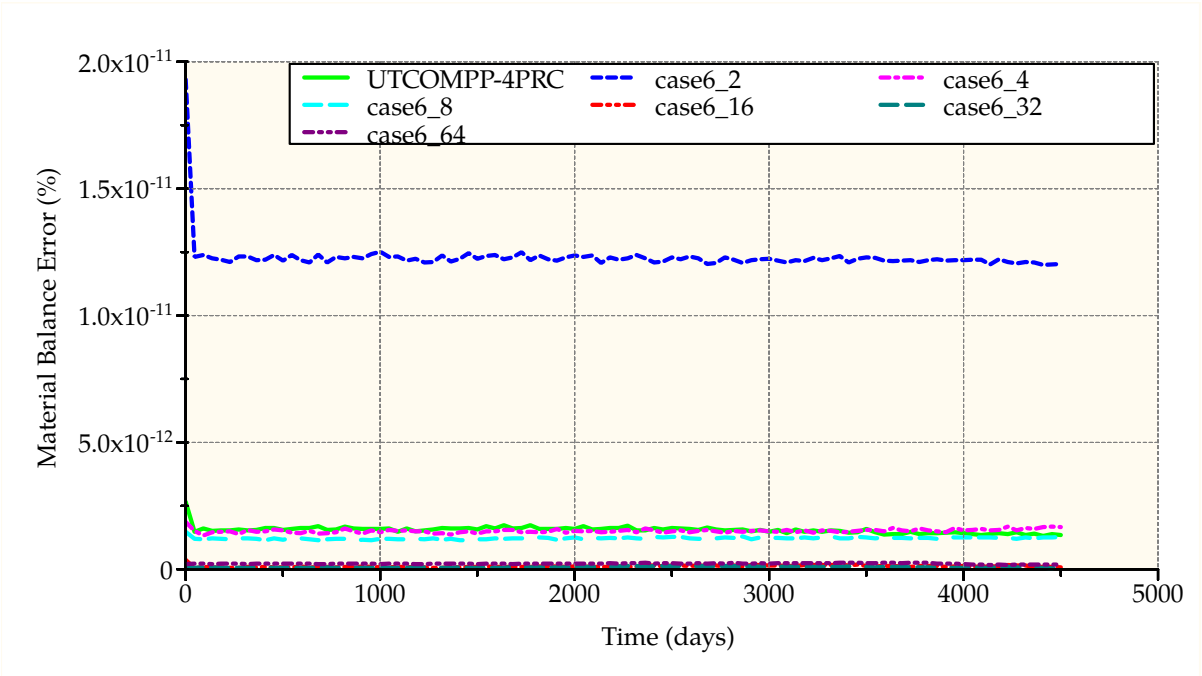


Figure 5.60: Material balance, case 6

Table 5.43 and Figure 5.61 show the CPU time when different number of processors are used. CPU times between our new simulator and UTCOMPP are almost the same. With 8 processors or less our new simulator is faster. With 16 processors or more UTCOMPP is slightly faster. Our new simulator is capable of the same parallel performance as UTCOMPP employing a more complex domain decomposition algorithm and communication pattern. The reference to compute speedup was the time obtained with two processors because it was not possible to obtain the CPU time for this model with one processor. Speedup was assumed for two processors as 2. Table 5.44 and Figure 5.62 show the speedup obtained with different number of processors used. With up to 4 processors the speedup obtained was nearly linear and it gets deviated from linear speedup with higher number of processors. This is because of the increase in communication with more processors.

Table 5.43: CPU times for case 6

	CPU time (s)	
# processors	New simulator	UTCOMPP
2	117036	117242
4	60072.6	63270.3
8	34086.7	35265.4
16	20639.4	19504.7
32	10726.1	10309.6
64	6157.18	5893.73

Table 5.44: Speedup for case 6

	Speedup	
# processors	New simulator	UTCOMPP
2	2	2
4	3.89648	3.70607
8	6.86695	6.64913
16	11.341	12.0219
32	21.8226	22.7443
64	38.016	39.7853

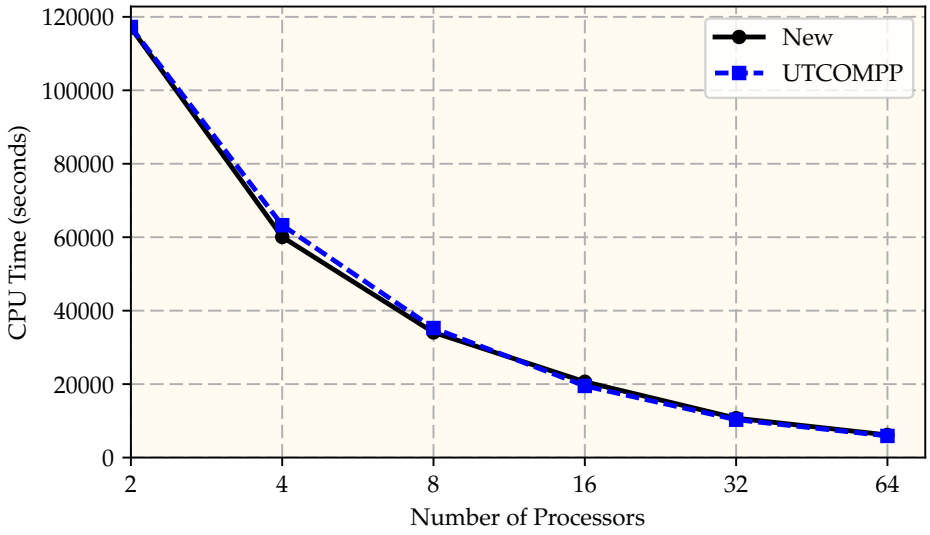


Figure 5.61: CPU time, case 6

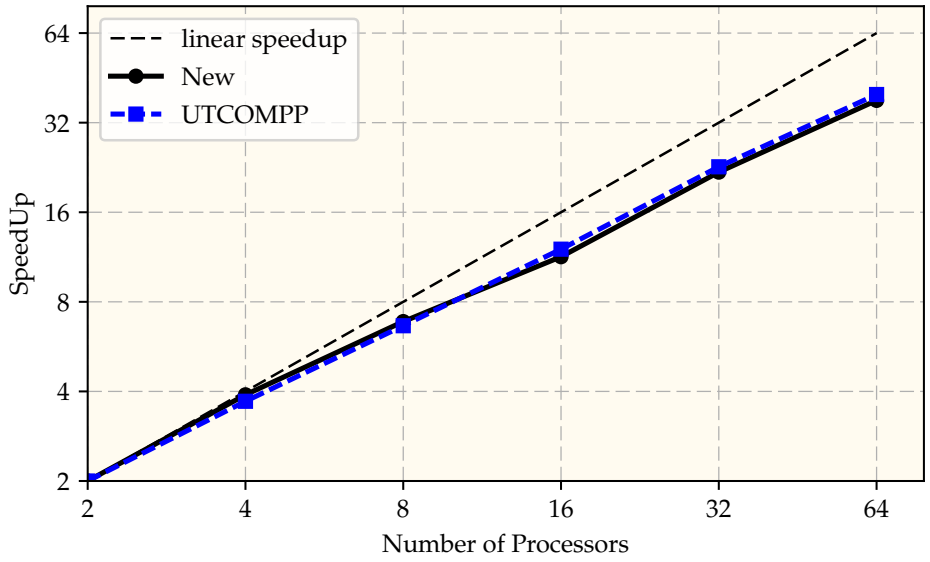


Figure 5.62: Speedup, case 6

Table 5.45: Detailed CPU times in seconds for case 6

	Number of processors used					
	2	4	8	16	32	64
Total Linear Solver Time	4495.24	2377.12	1300.03	746.713	432.628	354.121
Update Ghost Cells	532.443	891.252	993.021	1427.34	955.164	733.912
Compute Derivatives	6335.69	3230.03	1845.85	1388.37	599.75	345.029
Physical Properties Before Solver	6917.49	3543.14	2000.85	1330.92	581.96	318.631
Phase Composition Calculation	77272.1	38809.5	20629	10394.6	5180.88	2610.95
Physical Properties After Solver	13716.5	7199.84	5092.85	3945.72	2276.63	1377.07
Other	7764.81	4022.54	2217.01	1330.74	653.499	388.098
Total Execution Time	117036	60072.6	34086.7	20639.4	10726.1	6157.18

The detailed CPU time can be observed in Table 5.45 and Figure 5.63. Figure 5.64 shows the percentage of total CPU time spent on each simulation section. The increase of CPU time taken for update of ghost cells with increasing number of processors indicates the higher requirement in communication among processors. The decrease of biggest portion of the simulation used for computation, the phase composition calculation, with increasing number of processors indicate a good parallel load balancing between processors.

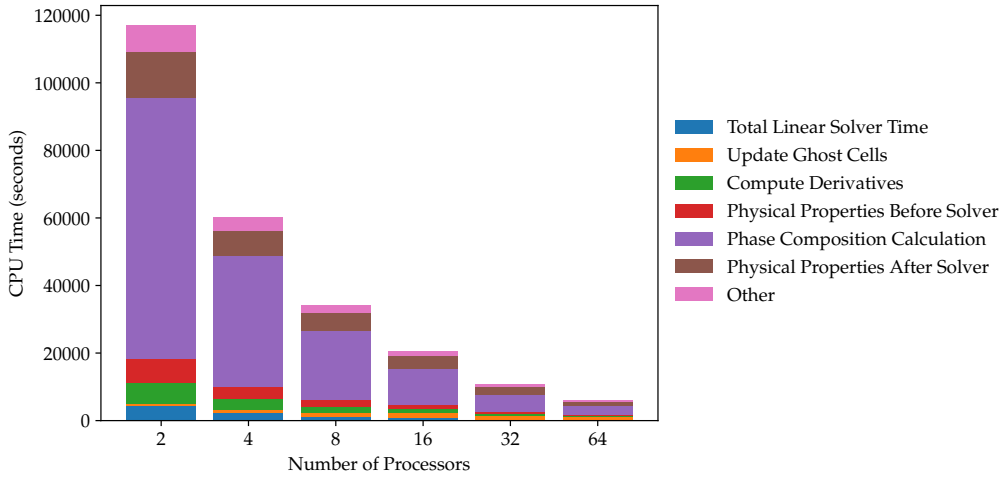


Figure 5.63: Detailed CPU times for case 6

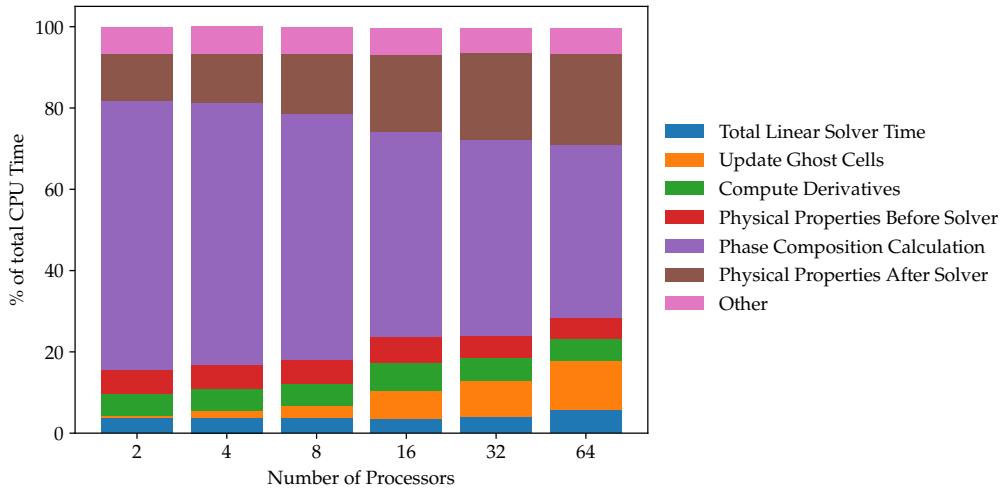


Figure 5.64: Percentage of total CPU time for timers for case 6

5.2.2.1 Effect of Hardware on Speedup

This case can be used as test of the impact of hardware update on speedup. This case is the same as Case Study 8 from Ghasemi Doroh (2012). The supercomputer used by Ghasemi Doroh (2012) to run the simulations was Lonestar 4. A comparison between Lonestar 5 and Lonestar 4 compute nodes is shown in Table 5.46. There is a trend towards increase of core number per processor rather than increase on core speed. Also, the rate at which core count increases is faster than the increase in RAM memory speed and memory channels.

Table 5.46: Lonestar 4 and Lonestar 5 compute node comparison

	Lonestar 4	Lonestar 5
Processor socket	Xeon 5680	Xeon E5-2690 v3
Processor speed	3.33 GHz	2.6 GHz
Cores per socket	6	12
Cache	12 MB	30 MB
Memory channels per socket	3	4
Sockets per node	2	2
Cores per node	12	24
Memory channels per node	6	8
Memory channels to core ratio	1:2	1:3
RAM per node	24 GB	64 GB
RAM speed	DDR3 @ 1333 MHz	DDR4 @ 2133 MHz

Table 5.47 and Figure 5.65 show the CPU time for UTCOMPP reported by Ghasemi Doroh (2012) and the CPU time obtained in this work using Lonestar 5. In all runs the CPU times obtained with Lonestar 5 were smaller even when the maximum CPU speed of Lonestar 4 cores is faster than the cores in Lonestar 5. This is because in our simulations the processor speed is not used at the maximum and it is bounded by the speed of RAM. Our simulations are limited by the speed of RAM because each domain cannot fit in the cache of the processor and the information needs to be retrieved from RAM. The increase in RAM speed from Lonestar 4 to Lonestar 5 explain the decrease in CPU time in our simulations in two processors. When the simulation is faster in two processors, communication between processors became more relevant for parallel performance. Hence, the computed speedup is decreased. The difference in CPU time at two processors makes the speedup computed in our work more deviated

from ideal than the speedup reported by Ghasemi Doroh (2012) despite the CPU times being lower in our work. Table 5.48 and Figure 5.66 show the speedup obtained by Ghasemi Doroh (2012) and the speedup obtained in this work for case 6.

Table 5.47: CPU times, case 6 using UTCOMPP in Lonestar 4 (Ghasemi Doroh 2012) and Lonestar 5

	CPU time (s)	
# processors	Lonestar 4	Lonestar 5
2	159245	117242
4	80557	63270.3
8	40764	35265.4
16	20515	19504.7
32	10895	10309.6

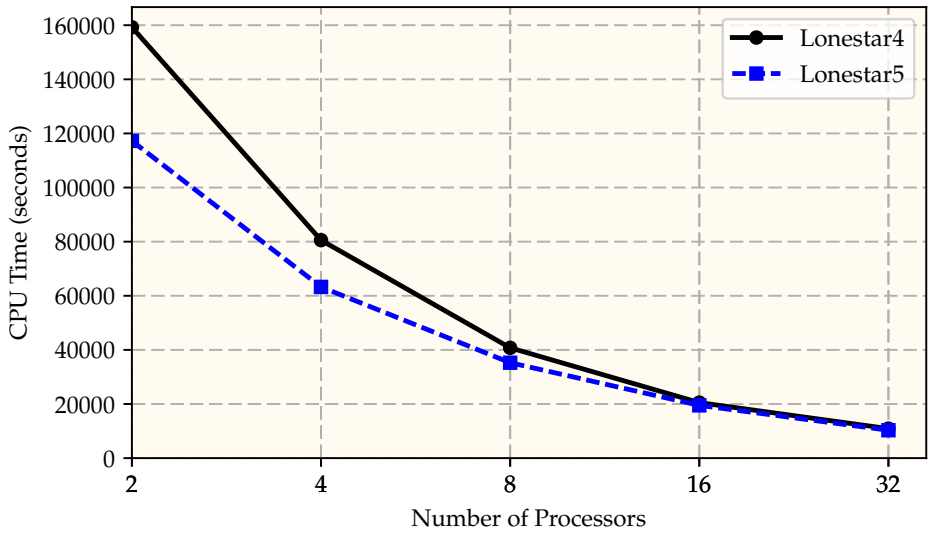


Figure 5.65: CPU time for UTCOMPP on Lonestar 4 and Lonestar 5, case 6

Table 5.48: Speedup, case 6 using UTCOMPP in Lonestar 4 (Ghasemi Doroh 2012) and Lonestar 5

	Speedup	
# processors	Lonestar 4	Lonestar 5
2	2	2
4	3.9536	3.70607
8	7.8130	6.64913
16	15.5247	12.0219
32	29.2326	22.7443

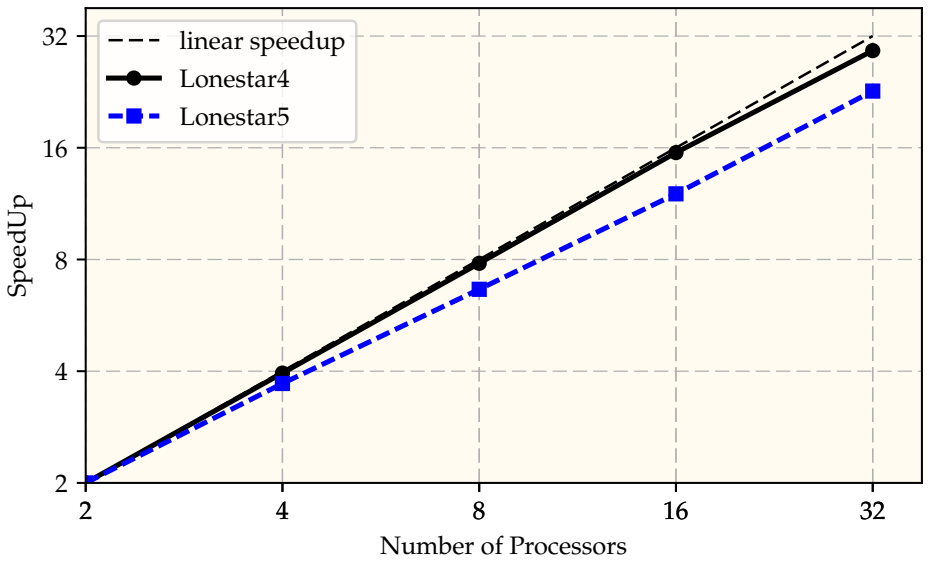


Figure 5.66: Speedup for UTCOMPP on Lonestar 4 and Lonestar 5, case 6

5.2.3 Case 7 - Simultaneous Water-Gas Injection

This case simulates simultaneous injection of water and gas into a heterogeneous reservoir. The model is described in Table 5.49. Figures 5.67 and 5.68 show porosity and permeability distribution. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.50. Relative permeability data is shown in Table 5.51. The model contains 13 injector and 12 producer wells. Figure 5.69 shows the grid and wells for the simulator model. Table 5.52 shows the well control schedule through the simulation.

Table 5.49: Model description for case 7

Dimensions (ft)	Length Width Thickness	5600 5600 100
Number of cells	800000	(400x400x5)
Number of components		6
Max. number of phases		3
Porosity		Figure 5.67
Permeability (md)	X Y Z	Figure 5.68 Figure 5.68 10
Rock compressibility (psi^{-1})		5×10^{-5}
Water compressibility (psi^{-1})		3×10^{-6}
Initial water saturation		0.17
Irreducible water saturation		0.17
Reservoir temperature (°F)		250
Initial reservoir pressure (psi)		4000
Number of wells	25	13 Injector 12 Producer
Simulation time (days)		1095

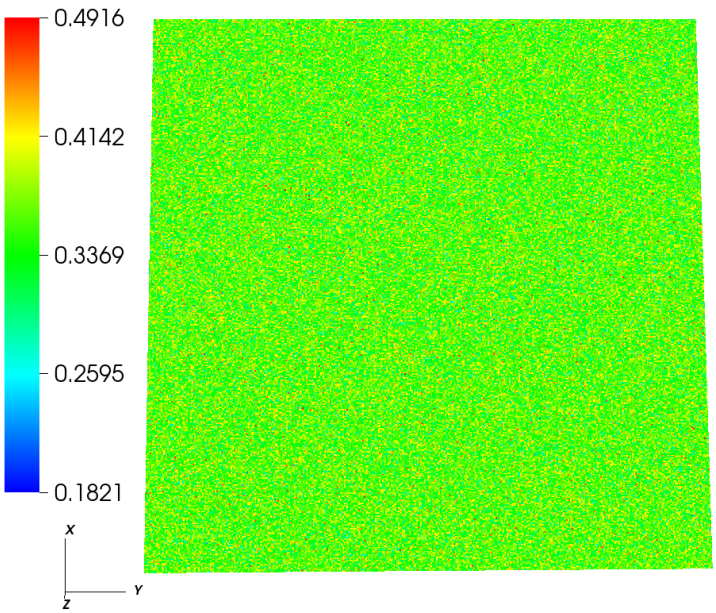


Figure 5.67: Porosity distribution for case 7

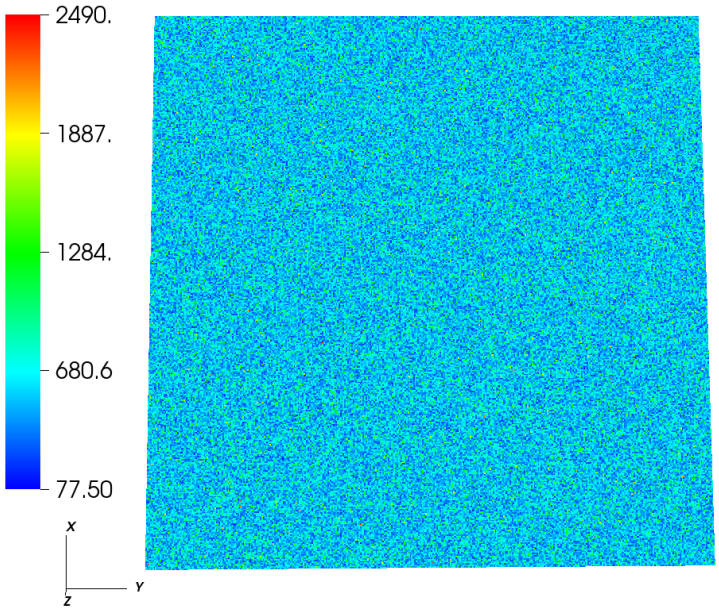


Figure 5.68: Permeability distribution for case 7 (md)

Table 5.50: Component properties and compositions for case 7

	Properties					Composition (molar fraction)	
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
C_1	343.0	667.8	1.599	16.0	0.013	0.5	0.85
C_3	665.7	616.3	3.211	44.1	0.152	0.03	0.1
C_6	913.4	436.9	5.923	86.2	0.301	0.07	0.03
C_{10}	1111.8	304.0	10.087	142.3	0.488	0.2	0.0199
C_{15}	1270.0	200.0	16.696	206.0	0.650	0.15	0.00005
C_{20}	1380.0	162.0	21.484	282.0	0.850	0.05	0.00005

Table 5.51: Relative permeability data for case 7

Model	Corey's model (Corey 1986; UTCOMP 2003)			
	Water	Gas	Oil	
Residual saturation	0.17	0.05	water-oil	0.17
			gas-oil	0.10
End point	0.40	0.85		0.75
Exponent	2.5	2.0	water-oil	2.00
			gas-oil	2.00

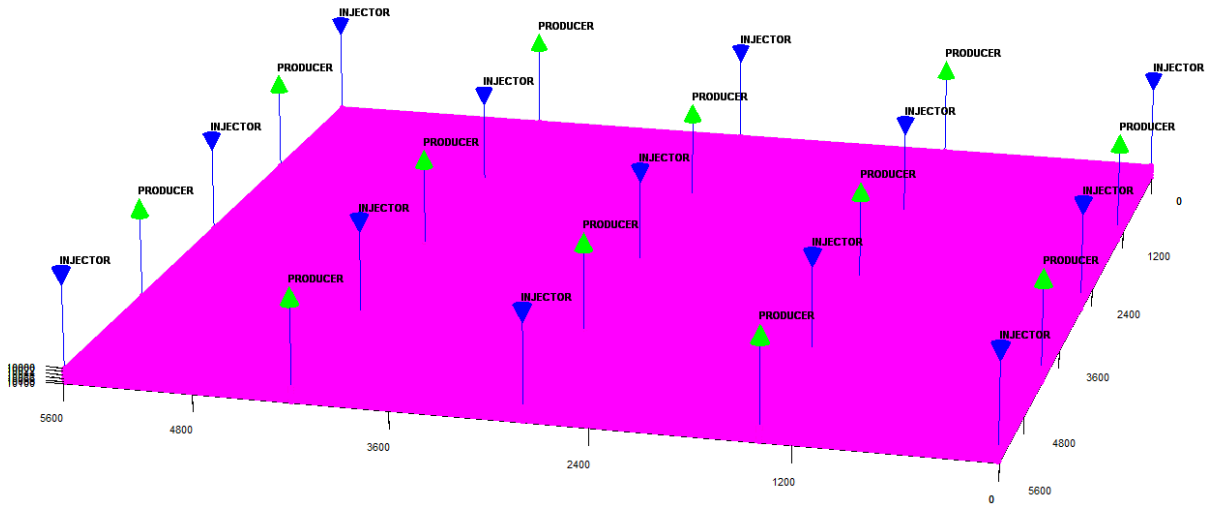


Figure 5.69: Grid and wells for case 7

Table 5.52: Well schedule for case 7

Sim. days	Producers Control	Injectors Control
0 - 365	BHP 3700 psi	Water rate 7500STB/day BHP limit 6000psi
365 - 730	BHP 3700 psi	75 mole% water and 25 mole% gas Inj. BHP 4300psi
730 - 1095	BHP 3700 psi	Water rate 7500STB/day BHP limit 6000psi

Up to 128 processors were used in this model. There was not possible to obtain results for one and two processors because Lonestar 5 queue system impose an 48 hours limit for the duration of any simulation job. UTCOMPP was not able to run with 128 processors. Table 5.53 shows domains and pressure distribution at the end of the simulation when different number of processors were used. Table 5.54 shows oil , gas and water saturation distribution at the end of the simulation when different number of processors were used. Figures 5.70, 5.71, 5.72 and 5.73 show average reservoir pressure, surface oil production rate, surface gas production rate and surface water production rate compared with UTCOMPP. Figure 5.74 shows the material balance error when different number of processors are used. Maximum material balance error values obtained were in the order of 4.0×10^{-13}.

Table 5.53: Results for case 7. Domain decomposition and pressure distribution

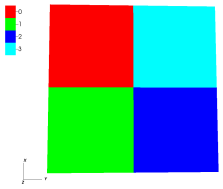
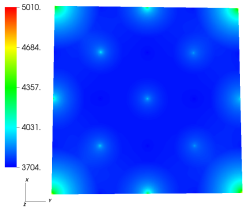
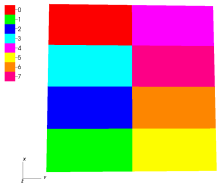
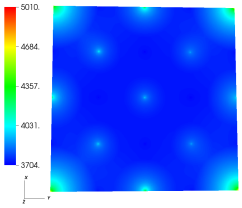
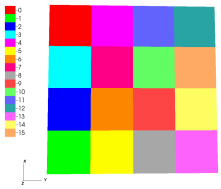
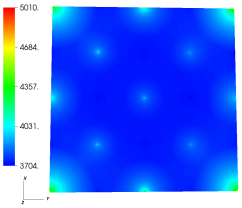
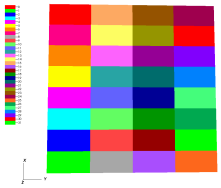
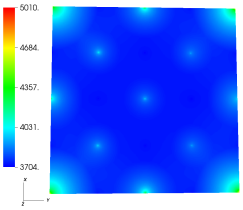
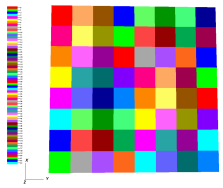
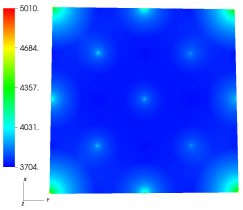
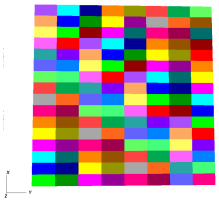
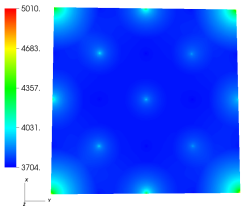
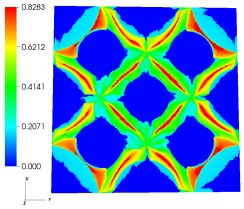
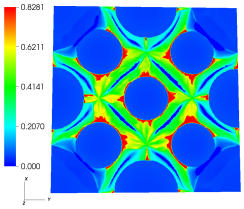
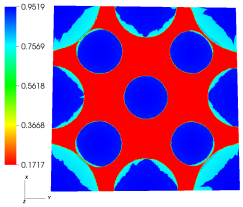
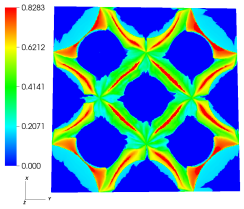
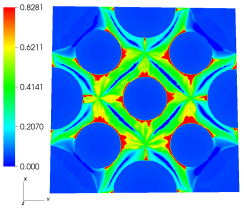
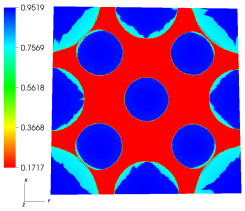
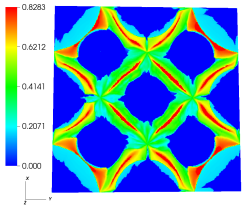
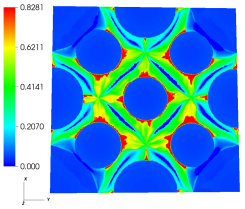
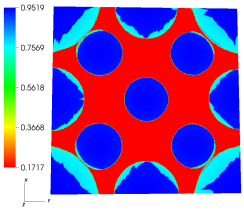
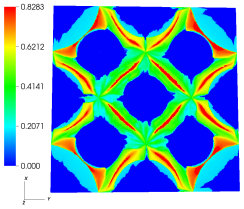
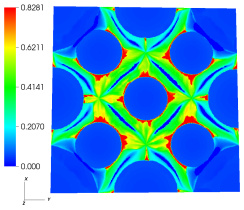
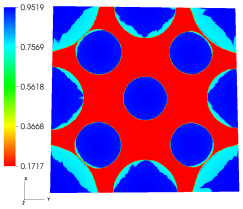
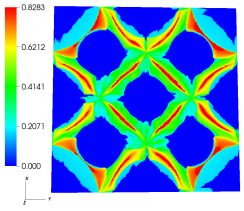
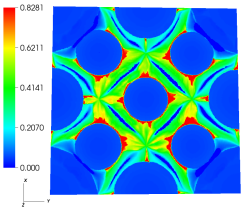
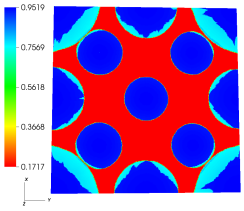
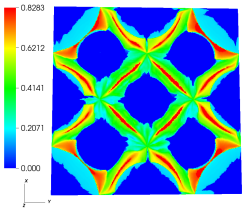
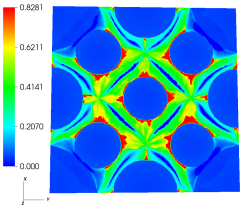
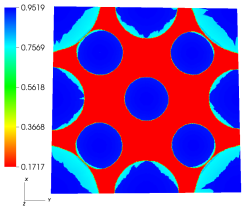
Prcs	Domains	Pressure (psi)
4		
8		
16		
32		
64		
128		

Table 5.54: Results for case 7. Saturation distribution

Prcs	S_o	S_g	S_w
4			
8			
16			
32			
64			
128			

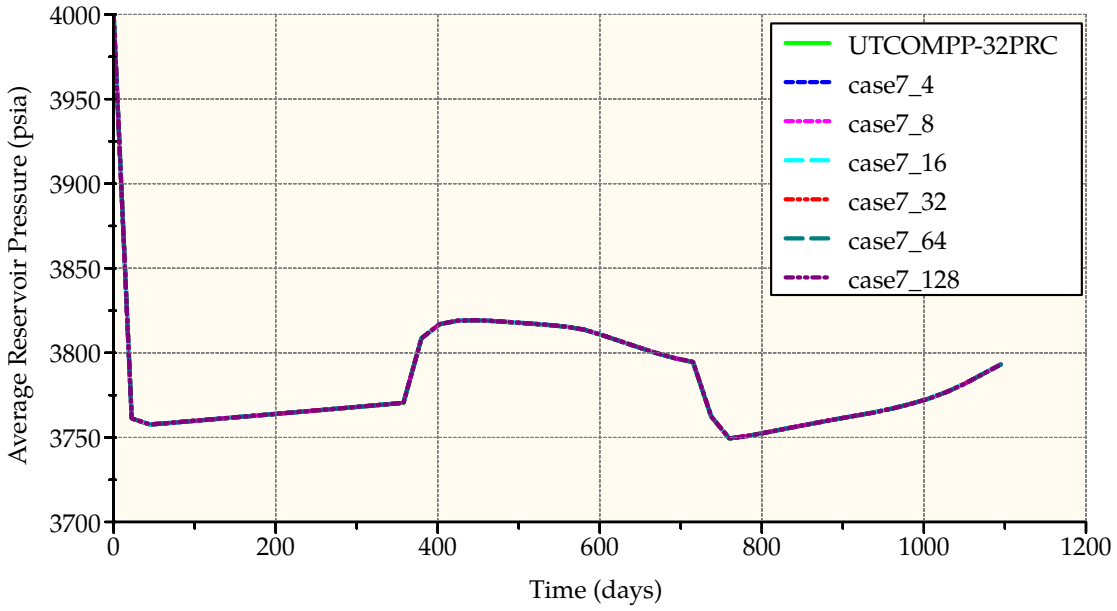


Figure 5.70: Average reservoir pressure, case 7

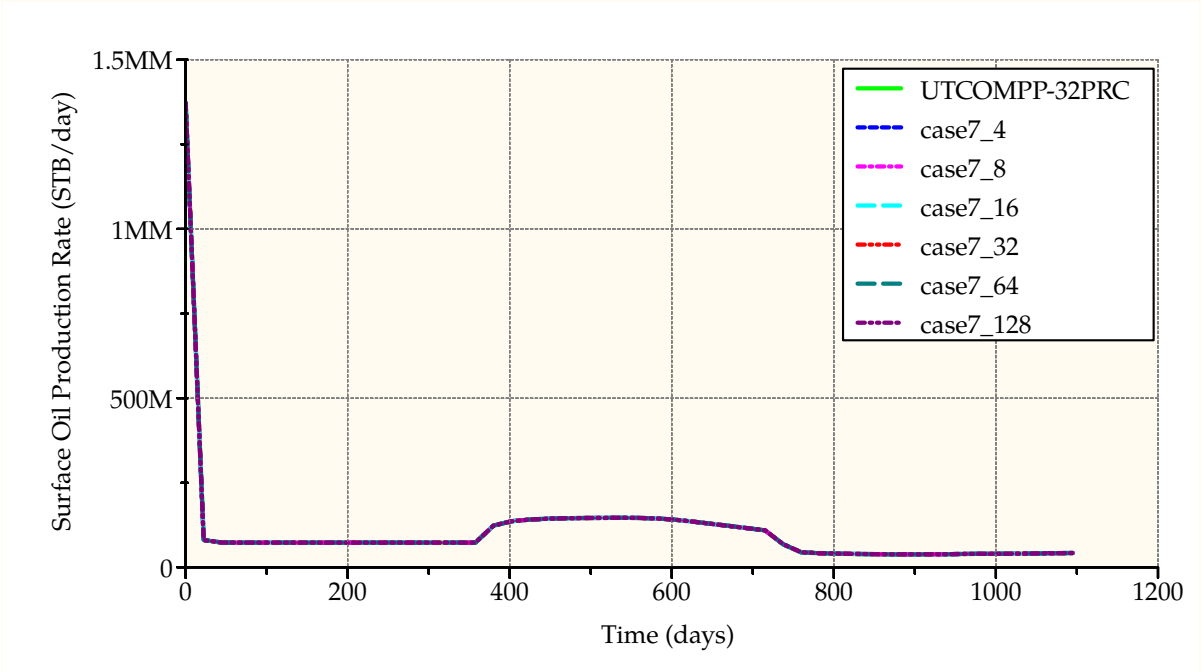


Figure 5.71: Surface oil production rate, case 7

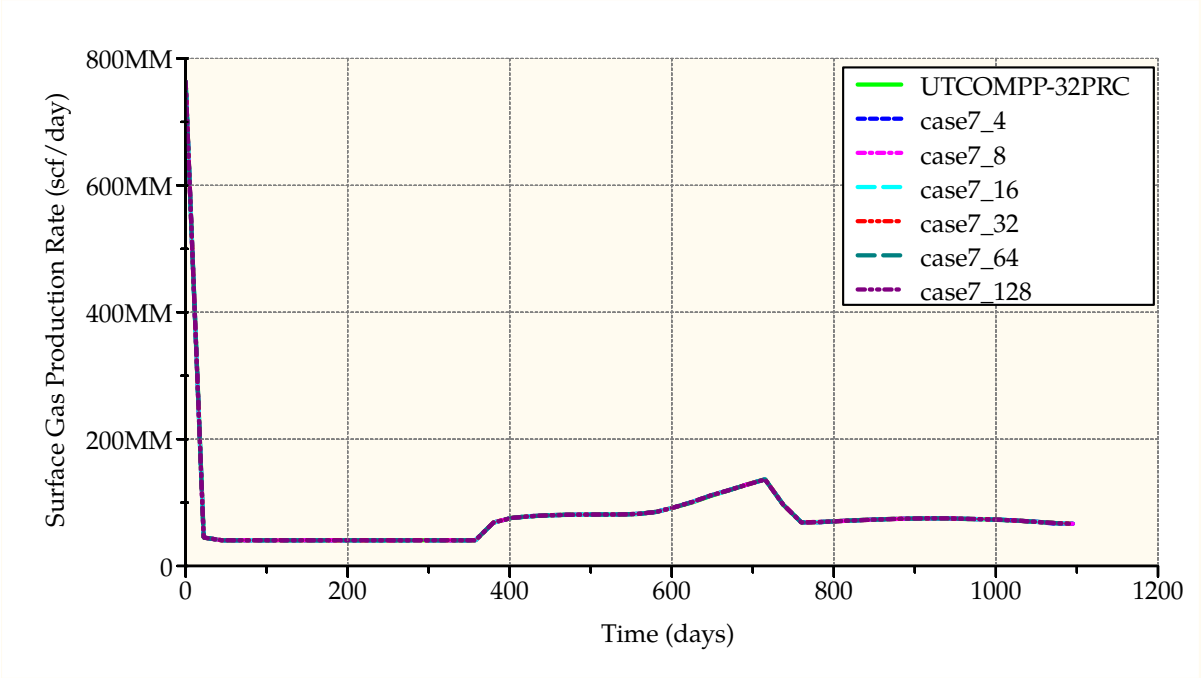


Figure 5.72: Surface gas production rate, case 7

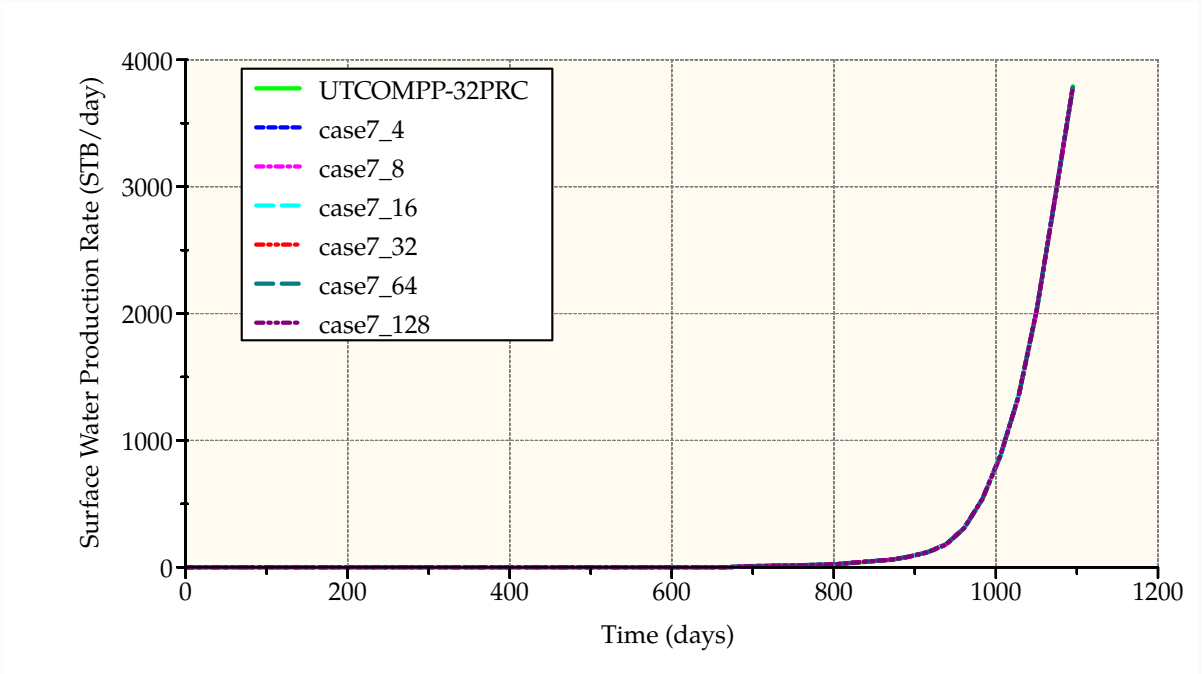


Figure 5.73: Surface water production rate, case 7

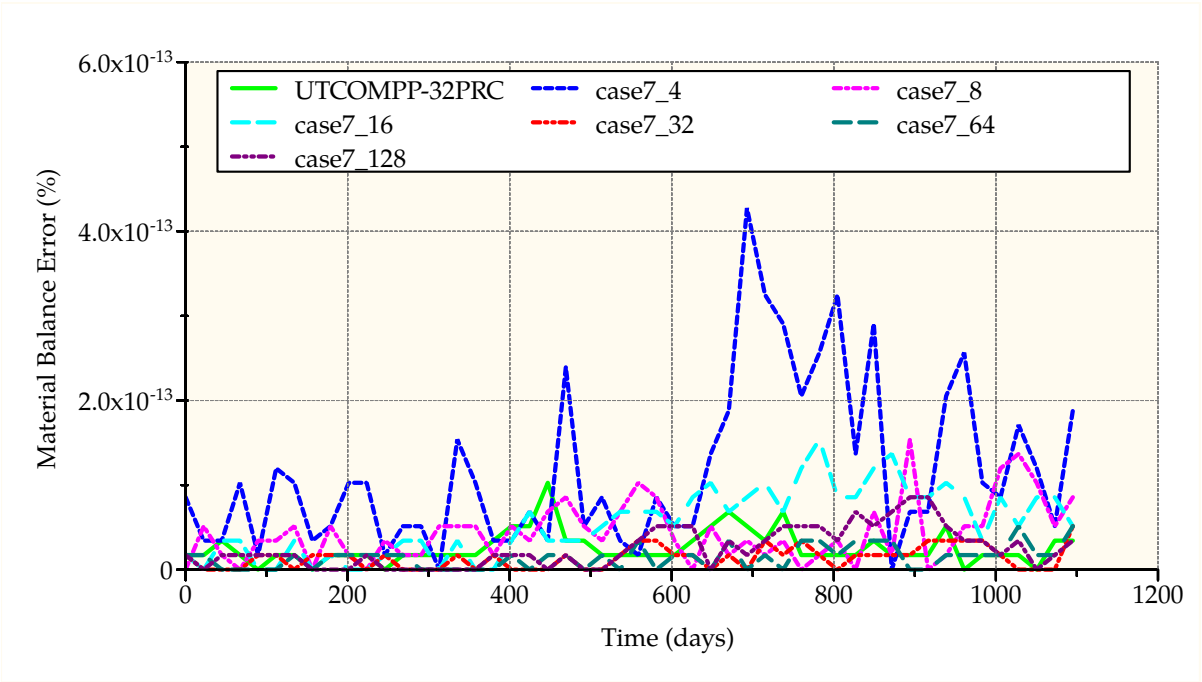


Figure 5.74: Material balance, case 7

Table 5.55 and Figure 5.75 show the CPU time when different number of processors are used. Table 5.56 and Figure 5.76 show the speedup obtained with different number of processors used. Our new simulator is faster for all the number of processors used than UTCOMPP and our new simulator obtains a speedup closer to linear speedup than UTCOMPP. Additionally, our new simulator is capable of running on more processors than UTCOMPP. The detailed CPU time can be observed in Table 5.57 and Figure 5.77. Figure 5.78 shows the percentage of total CPU time spent on each simulation section.

Table 5.55: CPU times for case 7

	CPU time (s)	
# processors	New simulator	UTCOMPP
4	121721	153873
8	66919.1	82717.7
16	36960.7	50193.4
32	17993	28324.4
64	11424.4	17957.8
128	6771.28	–

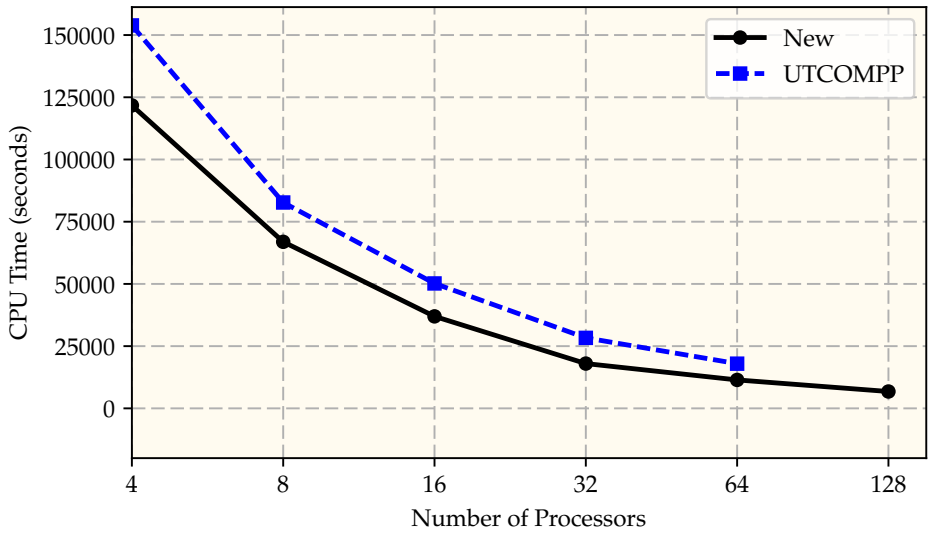


Figure 5.75: CPU time, case 7

Table 5.56: Speedup for case 7

	Speedup	
# processors	New simulator	UTCOMPP
4	4	4
8	7.27568	7.44088
16	13.173	12.2624
32	27.0594	21.7301
64	42.6176	34.2745
128	71.904	–

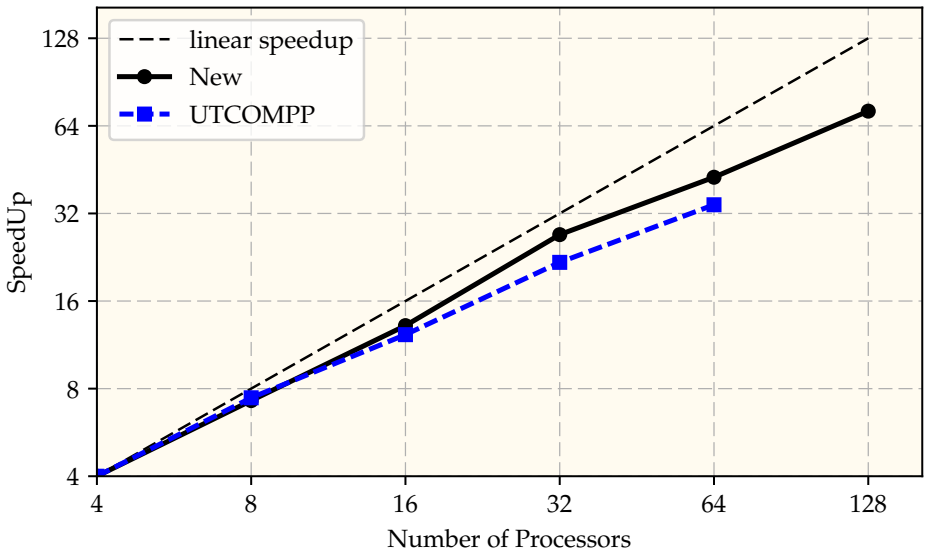


Figure 5.76: Speedup, case 7

Table 5.57: Detailed CPU times in seconds for case 7

	Number of processors used					
	4	8	16	32	64	128
Update Ghost Cells	895.868	789.172	1070.36	1064.83	1272.41	1197.53
Compute Derivatives	5548.73	3254.05	2742.87	1083.37	529.966	227.866
Physical Properties Before Solver	5945.31	3053.26	1899.89	720.037	497.955	261.543
Phase Composition Calculation	94481.7	49930.6	25145	12145.8	6196.88	3160.56
Physical Properties After Solver	5482.95	4944.76	2832.07	1492.22	2099.63	1396.23
Other	9104.32	4932.34	3262.46	1471.41	791.713	410.698
Total Execution Time	121721	66919.1	36960.7	17993	11424.4	6771.28

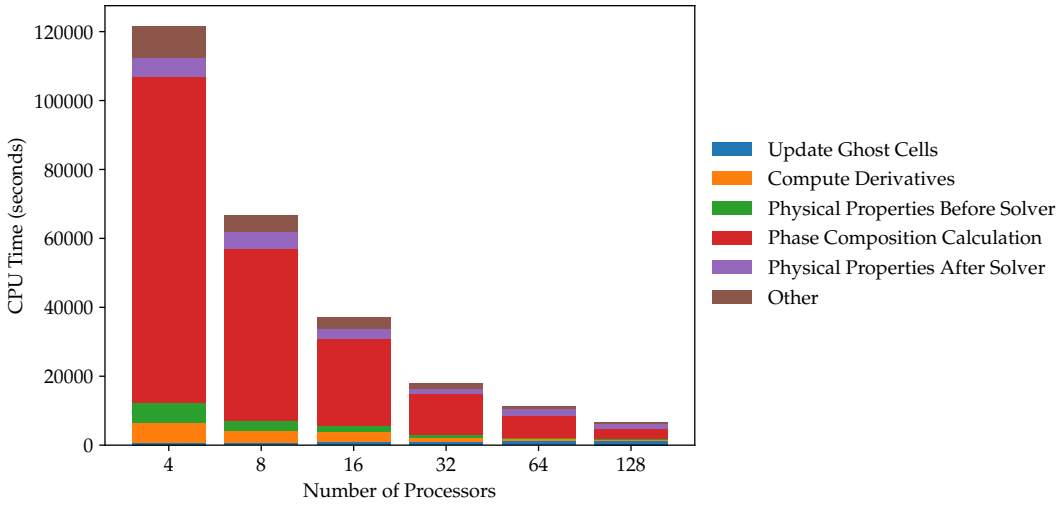


Figure 5.77: Detailed CPU times for case 7

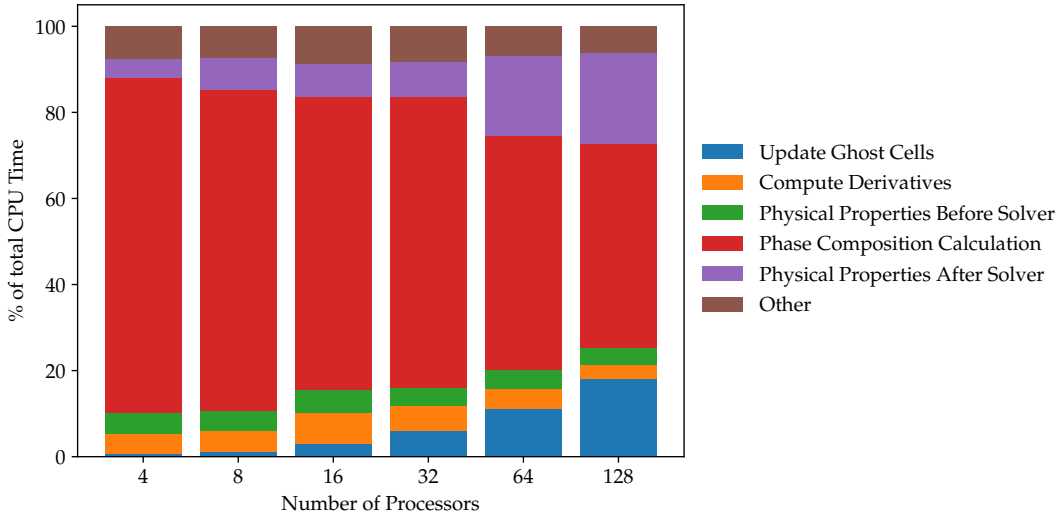


Figure 5.78: Percentage of total CPU time for timers for case 7

5.2.4 Case 8 - Waterflooding

This case simulates a waterflooding process in a heterogeneous reservoir. The model is described in Table 5.58. Figures 5.79 and 5.80 show porosity and permeability distribution. The model contains 1,697,680 active cells (Figure 5.81). Inactive cells are not considered for computation. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.59. Relative permeability data is shown in Table 5.60. The model contains 17 injector and 13 producer wells. Figure 5.82 shows the grid and wells for the simulator model. The production wells are controlled at constant BHP of 3050 psi. The injector wells are controlled at a constant water injection rate of 1000 STB/day with a BHP limit of 5000 psi.

Table 5.58: Model description for case 8

Dimensions (ft)	Length Width Thickness	6000 4800 200
Number of cells	Total: 2,560,000 Active: 1,697,680	(400x320x20)
Number of components		6
Max. number of phases		3
Porosity		Figure 5.79
Permeability (md)	X Y Z	Figure 5.80 Figure 5.80 10
Rock compressibility (psi^{-1})		5×10^{-5}
Water compressibility (psi^{-1})		3×10^{-6}
Initial water saturation		0.17
Irreducible water saturation		0.17
Reservoir temperature (°F)		90
Initial reservoir pressure (psi)		3100
Number of wells	30	17 Injector 13 Producer
Simulation time (days)		5000

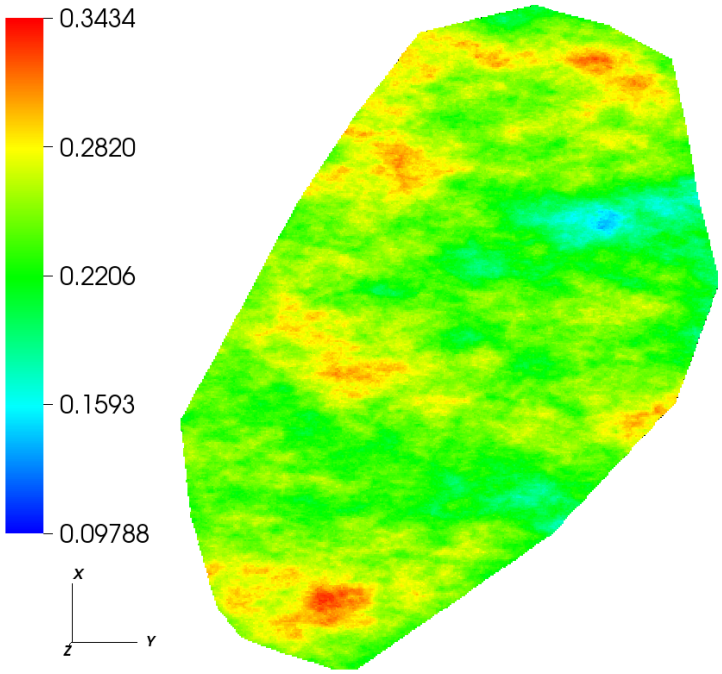


Figure 5.79: Porosity distribution for case 8

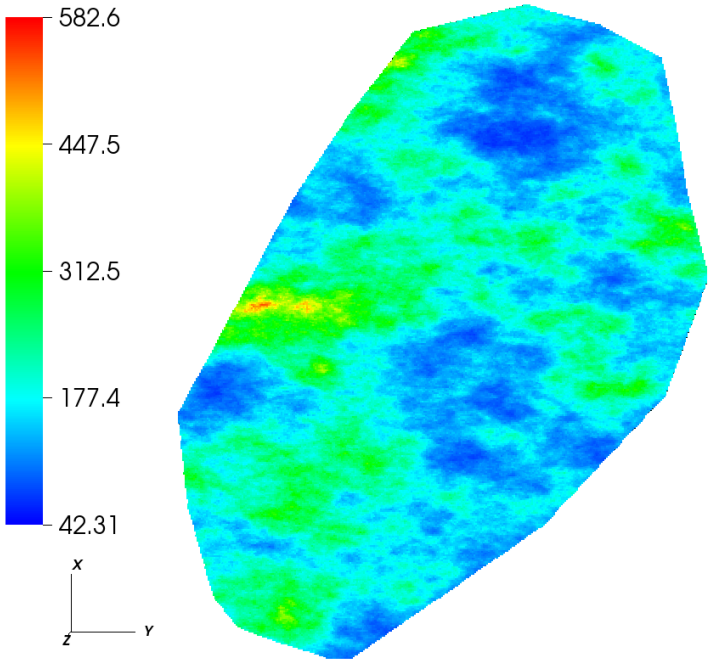


Figure 5.80: Permeability distribution for case 8 (md)

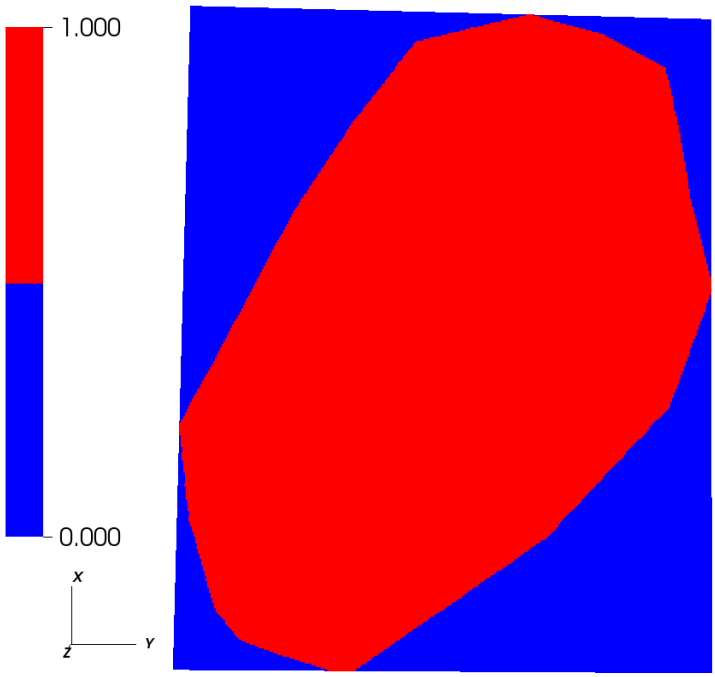


Figure 5.81: Active cells for case 8

Table 5.59: Component properties and compositions for case 8

	Properties					Composition (molar fraction)
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial
C_1	343.0	667.8	1.599	16.0	0.013	0.5
C_3	665.7	616.3	3.211	44.1	0.152	0.03
C_6	913.4	436.9	5.923	86.2	0.301	0.07
C_{10}	1111.8	304.0	10.087	142.3	0.488	0.2
C_{15}	1270.0	200.0	16.696	206.0	0.650	0.15
C_{20}	1380.0	162.0	21.484	282.0	0.850	0.05

Table 5.60: Relative permeability data for case 8

Model	Corey's model (Corey 1986; UTCOMP 2003)			
	Water	Gas	Oil	
Residual saturation	0.17	0.17	water-oil gas-oil	0.17 0.17
End point	0.40	0.85		0.75
Exponent	2.5	2.0	water-oil gas-oil	2.0 2.0

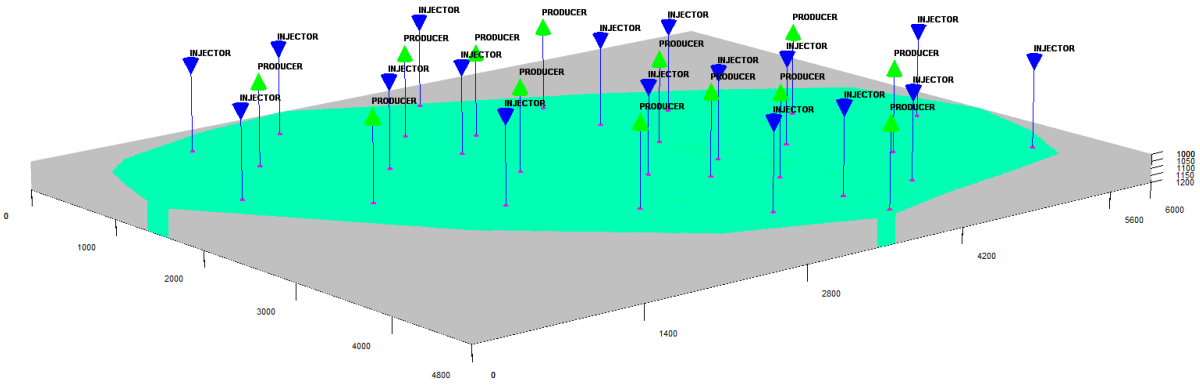


Figure 5.82: Grid and wells for case 8

Up to 256 processors were used in this model. UTCOMPP was not able to run with 256 processors. Table 5.61 shows domains, pressure distribution and oil and water saturation distribution at the end of the simulation when different number of processors were used. It can be observed that the domains created by RCB algorithm for this case are irregular and balance the computational load giving to few domains the inactive portions of the model and performing a more refined division in the active part of the model.

Figures 5.83, 5.84, 5.85 and 5.86 show average reservoir pressure, surface oil production rate, gas production rate and water production rate compared with UTCOMPP. Average reservoir pressure between our new simulator and UTCOMPP is less than 4 psi and the other results match perfectly. Figure 5.87 shows the material balance error when different number of processors are used. All material balance error values obtained were lower than 1.5×10^{-12}.

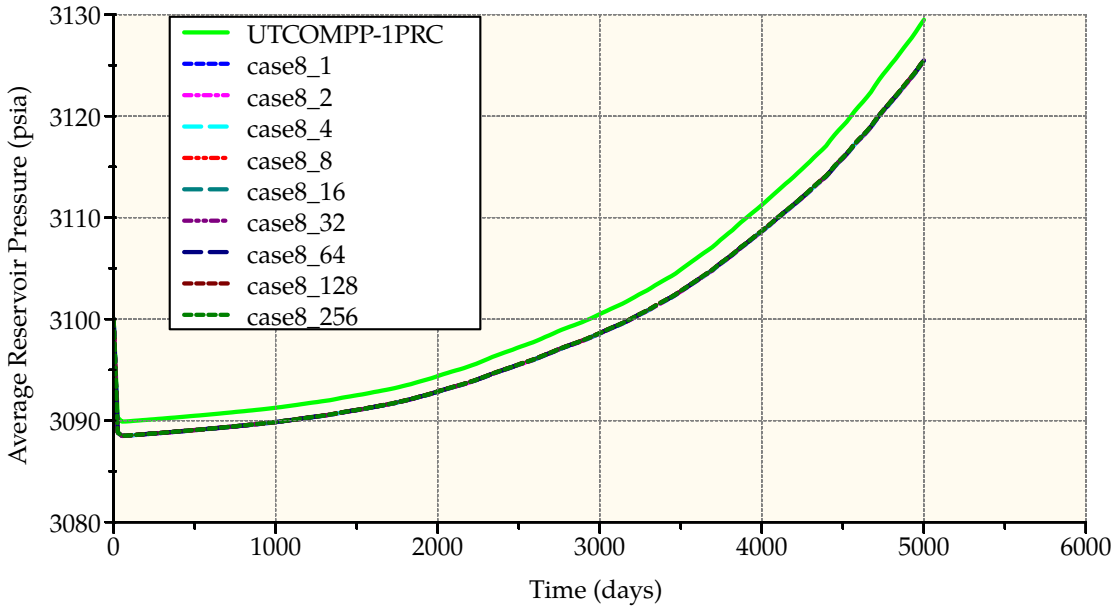
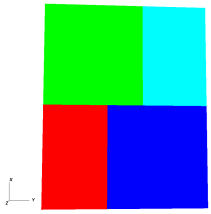
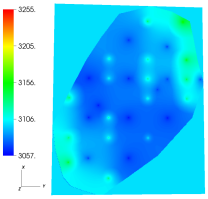
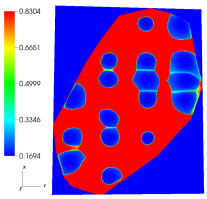
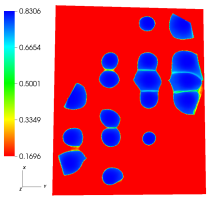
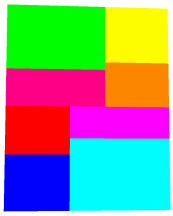
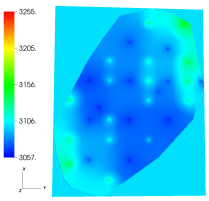
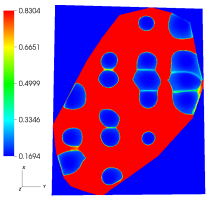
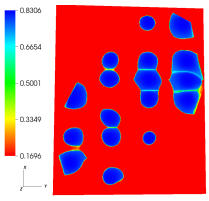
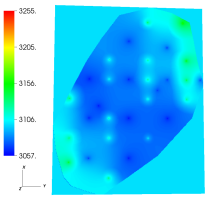
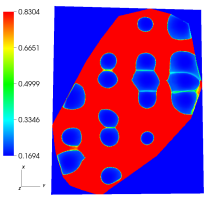
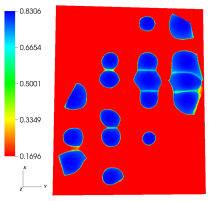
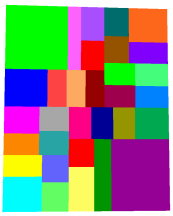
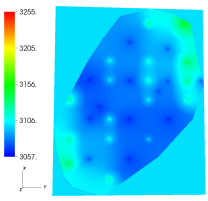
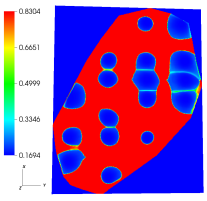
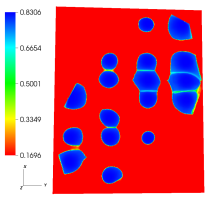
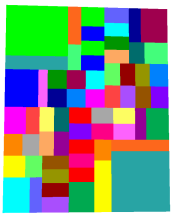
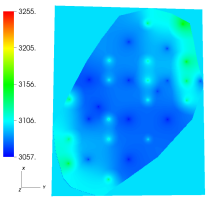
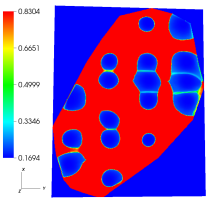
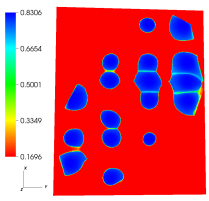
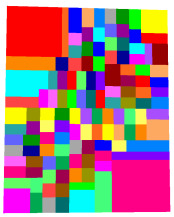
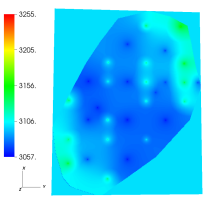
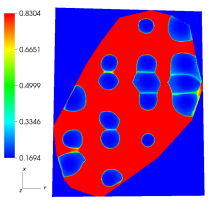
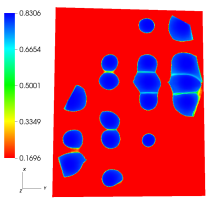
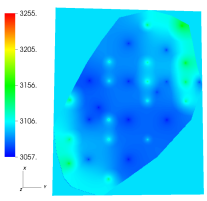
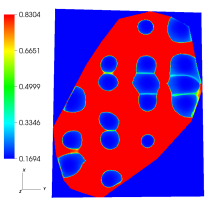
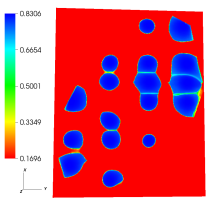


Figure 5.83: Average reservoir pressure, case 8

Table 5.61: Results for case 8

Prcs	Domains	Pressure (psi)	S_o	S_w
4				
8				
16				
32				
64				
128				
256				

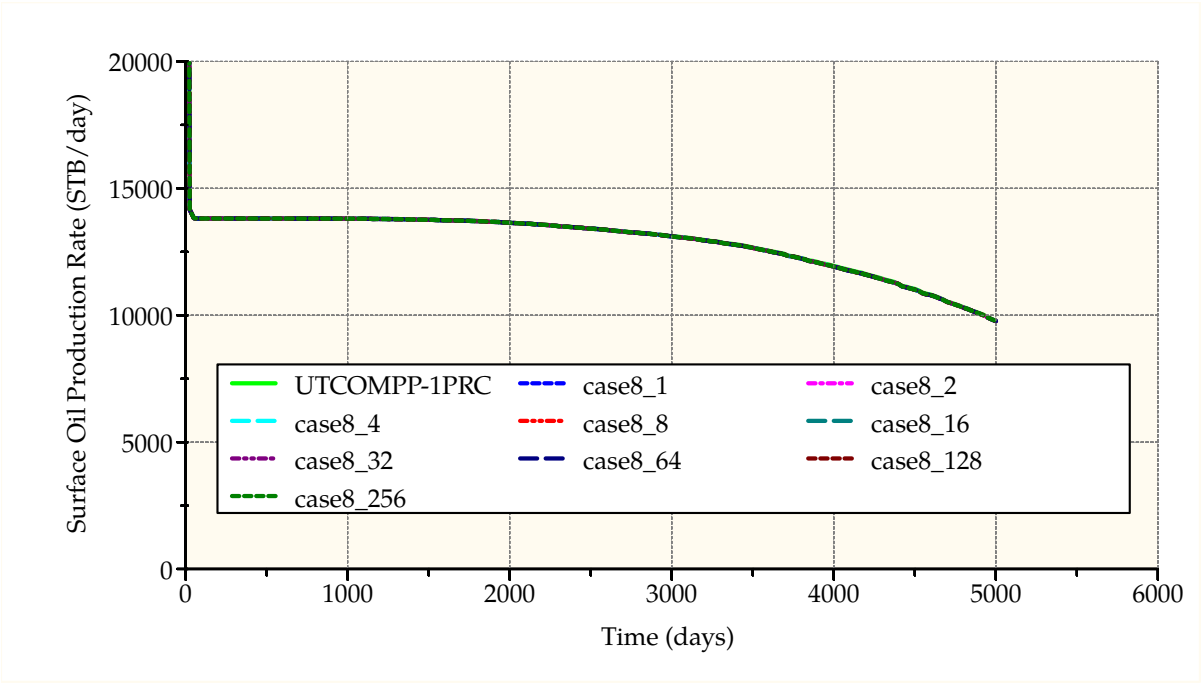


Figure 5.84: Surface oil production rate, case 8

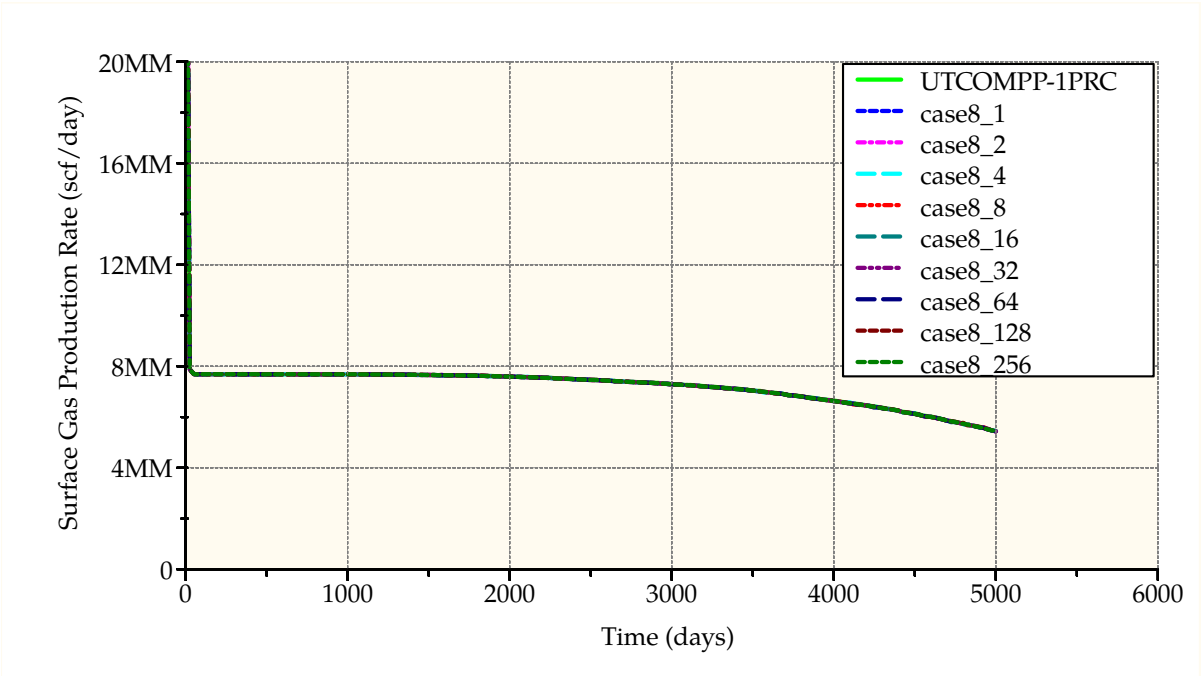


Figure 5.85: Surface gas production rate, case 8

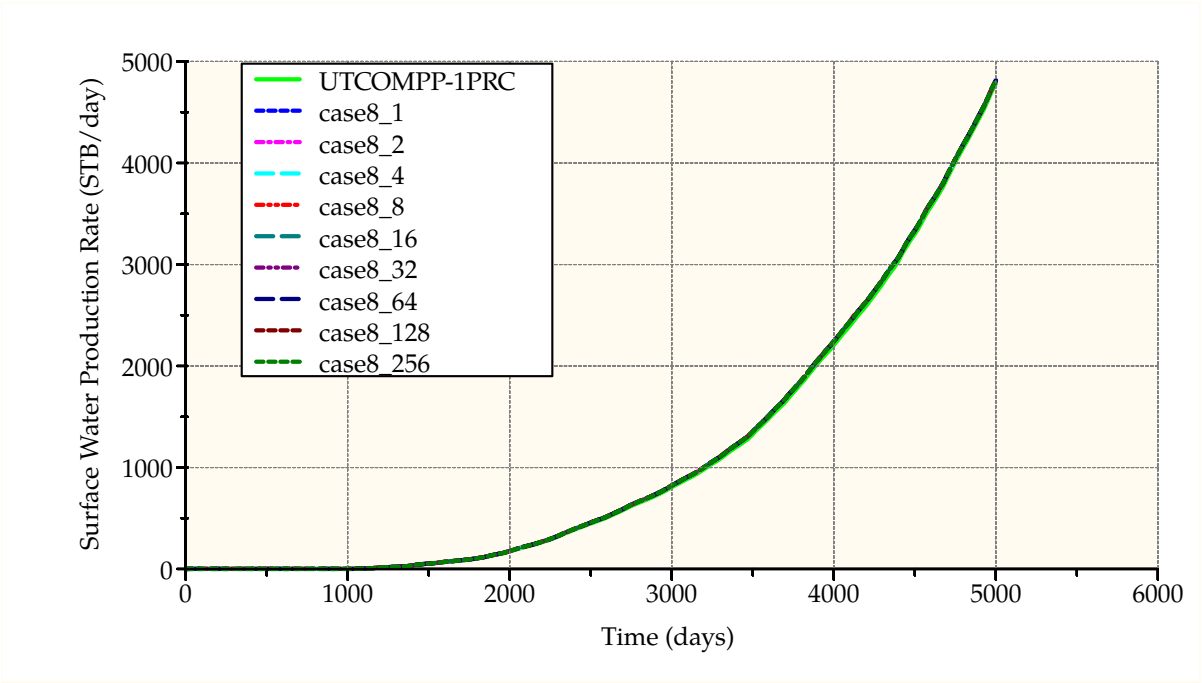


Figure 5.86: Surface water production rate, case 8

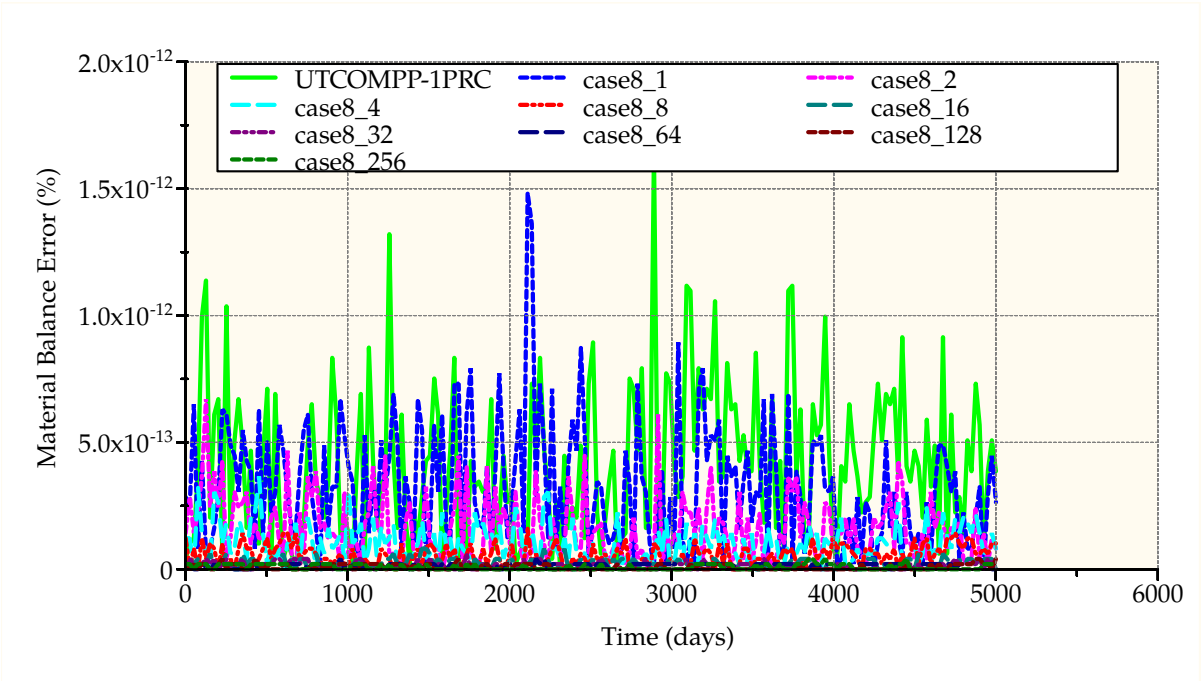


Figure 5.87: Material balance, case 8

Table 5.62 and Figure 5.88 show the CPU time when different number of processors are used. Table 5.63 and Figure 5.89 show the speedup obtained with different number of processors used. Our new simulator had a parallel performance equal or better than UTCOMPP for all number of processors used. Additionally, our new simulator was able to run in more number of processors than UTCOMPP. The detailed CPU time can be observed in Table 5.64 and Figure 5.90. Figure 5.91 shows the percentage of total CPU time spent on each simulation section. An increase in the percentage of total CPU time spent on communication with more processors used can be observed.

Table 5.62: CPU times for case 8

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	116184	125166
2	61039.5	60831.2
4	34707.8	37223
8	18576.5	22181.6
16	11483.7	12496.7
32	5604.76	6493.47
64	3532.63	3967.18
128	2073.61	2055.21
256	1399.92	–

Table 5.63: Speedup for case 8

	Speedup	
# processors	New simulator	UTCOMPP
1	1	1
2	1.90342	2.0576
4	3.34748	3.36261
8	6.25435	5.64279
16	10.1172	10.016
32	20.7294	19.2757
64	32.8887	31.5504
128	56.0297	60.902
256	82.993	–

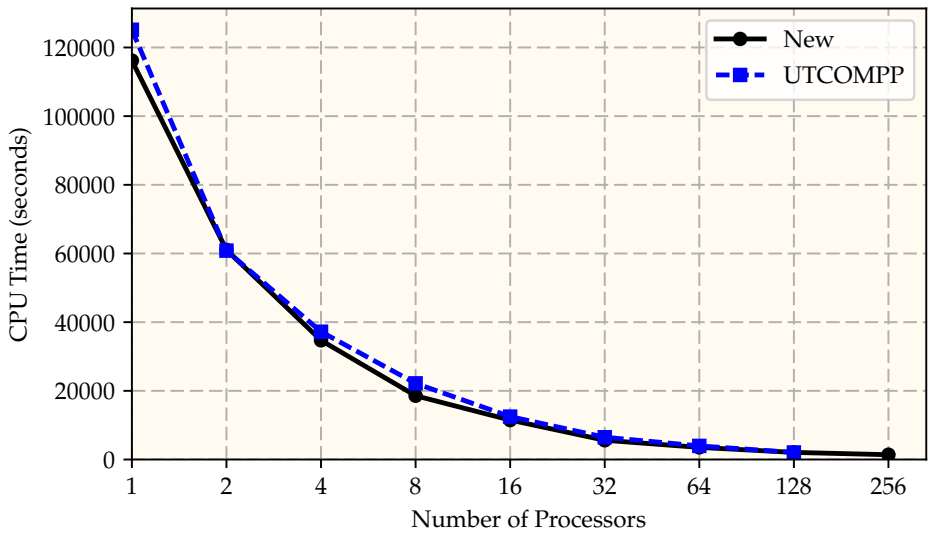


Figure 5.88: CPU time, case 8

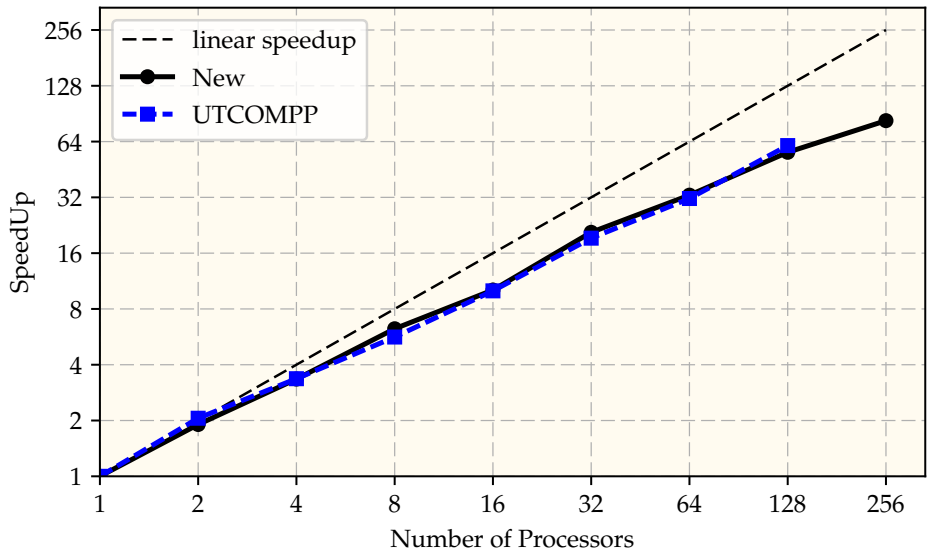


Figure 5.89: Speedup, case 8

Table 5.64: Detailed CPU times in seconds for case 8

	Number of processors used								
	1	2	4	8	16	32	64	128	256
Total Initialization Time	49.869	462.956	124.449	49.816	19.53	8.814	6.955	110.556	112.589
Total Linear Solver Time	11394.3	6372.98	3904.52	2023.71	1633.86	762.047	567.732	321.649	221.481
Matrix	4537.96	2211.73	1284.46	721.889	587.734	196.342	70.232	29.884	14.628
Update Ghost Cells	0.197	350.592	1339.38	955.073	841.047	626.274	571.784	453.35	396.823
Compute Derivatives	5532.45	2751.89	1570.19	951.53	880.223	349.622	250.738	73.824	28.007
Physical Properties Before Solver	10010.3	6008.34	3424.15	1833.65	1053.32	469.798	249.59	117.996	60.215
Phase Composition Calculation	71771.2	36602.7	18856.1	9885.24	4959.71	2430.16	1329.21	653.614	331.769
Physical Properties After Solver	7544.46	3703.09	2398.52	1254.89	719.909	372.427	206.706	105.758	60.045
Concentration Computations	4526.35	2222.85	1337.79	809.39	666.368	339.412	231.93	173.655	141.8
Other	815.819	373.859	209.804	111.578	72.618	36.908	22.627	15.697	11.899
Total Execution Time	116184	61039.5	34707.8	18576.5	11483.7	5604.76	3532.63	2073.61	1399.92

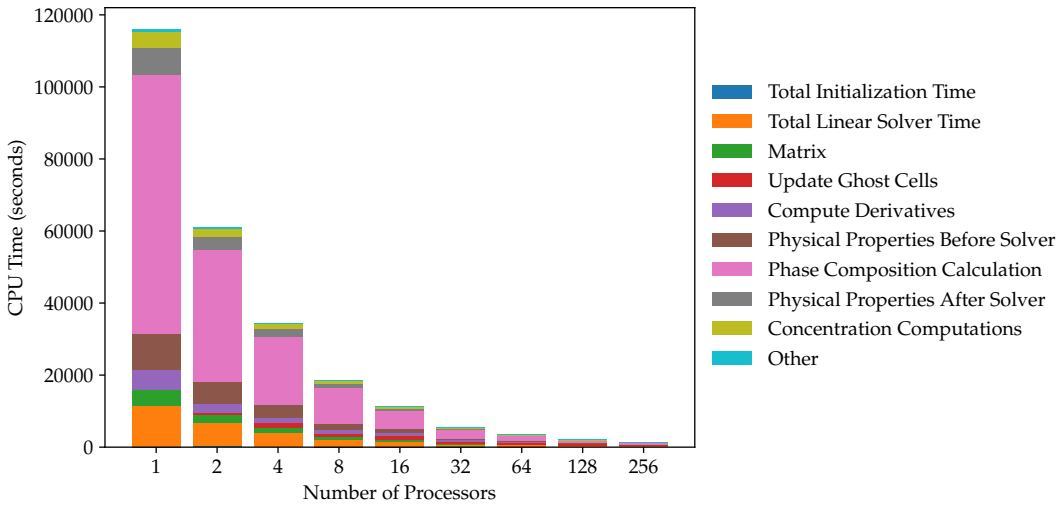


Figure 5.90: Detailed CPU times for case 8

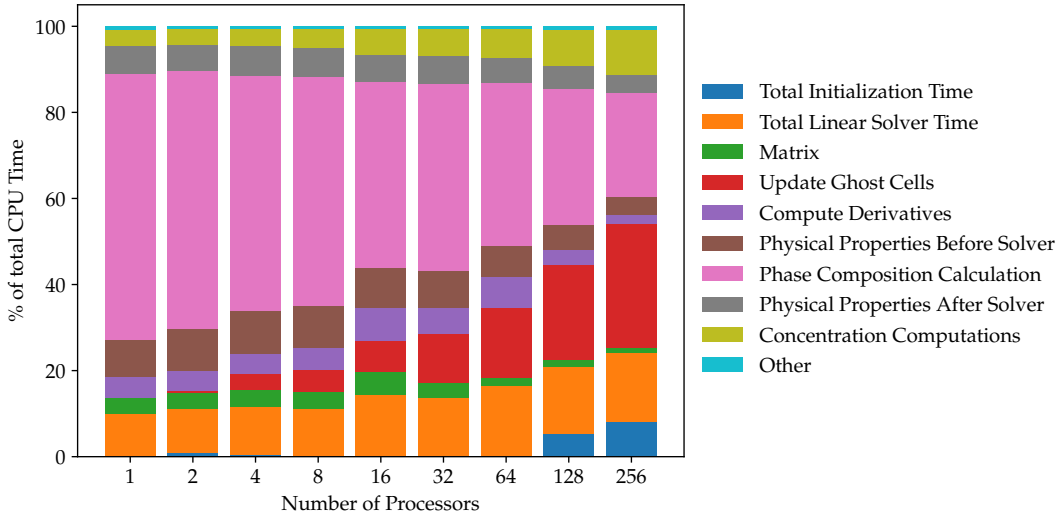


Figure 5.91: Percentage of total CPU time for timers for case 8

5.2.5 Case 9 - Water and Gas Injection Weak Scalability Test

This case simulates the injection of water and gas into a homogeneous reservoir. The model is described in Table 5.65. The model uses 6 hydrocarbon components. Component properties and compositions are shown in Table 5.66. Relative permeability data is shown in Table 5.67. The model contains 5 injector and 4 producer wells. Table 5.68 shows the well control schedule through the simulation.

This is a weak scalability test in which the size of the model increases as the number of processors increase. Table 5.69 shows the model size for each number of processors used. The objective of this test is to evaluate the capability of the simulator to handle very big models and to determine the capacity, or the size of the model that the simulator is capable to run in a specific hardware.

Table 5.65: Model description for case 9

Dimensions (ft)	Length	Table 5.69
	Width	Table 5.69
	Thickness	Table 5.69
Number of cells		Table 5.69
Number of components		6
Max. number of phases		3
Porosity		0.3
Permeability (md)	X	90
	Y	90
	Z	10
Rock compressibility (psi^{-1})		5×10^{-5}
Water compressibility (psi^{-1})		3×10^{-6}
Initial water saturation		0.17
Irreducible water saturation		0.17
Reservoir temperature (°F)		250
Initial reservoir pressure (psi)		4000
Number of wells	9	5 Injector 4 Producer
Simulation time (days)		1000

Table 5.66: Component properties and compositions for case 9

	Properties					Composition (molar fraction)	
Component	T_c (°R)	P_c (psia)	V_c ($\frac{ft^3}{Lb\text{-}mol}$)	MW	Acentric factor	Initial	Injected
C_1	343.0	667.8	1.599	16.0	0.013	0.5	0.85
C_3	665.7	616.3	3.211	44.1	0.152	0.03	0.1
C_6	913.4	436.9	5.923	86.2	0.301	0.07	0.03
C_{10}	1111.8	304.0	10.087	142.3	0.488	0.2	0.0199
C_{15}	1270.0	200.0	16.696	206.0	0.650	0.15	0.00005
C_{20}	1380.0	162.0	21.484	282.0	0.850	0.05	0.00005

Table 5.67: Relative permeability data for case 9

Model	Corey's model (Corey 1986; UTCOMP 2003)			
	Water	Gas	Oil	
Residual saturation	0.17	0.05	water-oil	0.17
			gas-oil	0.10
End point	0.40	0.85		0.75
Exponent	2.5	2.0	water-oil	2.00
			gas-oil	2.00

Table 5.68: Well schedule for case 9

Sim. days	Producers Control	Injectors Control
0 - 365	BHP 3500 psi	Water rate 10000STB/day BHP limit 6000psi
365 - 500	BHP 3500 psi	Gas Inj. @ BHP 4500psi
500 - 1000	BHP 3500 psi	Water rate 5000STB/day BHP limit 6000psi

Table 5.69: Model size

	Grid size		Model dimensions (ft)		
# processors	# of cells (Million)	(NXxNYxNZ)	Length	Width	Thickness
1	0.2	(200x200x5)	2800	2800	100
2	0.4	(400x200x5)	5600	2800	100
4	0.8	(400x400x5)	5600	5600	100
8	1.6	(800x400x5)	11200	5600	100
16	3.2	(800x800x5)	11200	11200	100
32	6.4	(1600x800x5)	22400	11200	100
64	12.8	(1600x1600x5)	22400	22400	100
128	25.6	(3200x1600x5)	44800	22400	100
256	51.2	(3200x3200x5)	44800	44800	100

Up to 256 processors were used in this model. UTCOMPP was not able to run models bigger than 3.2 million cells on 16 processors. Table 5.70 and Figure 5.92 show the CPU time taken with different number of processors used. In a weak scalability test the computational load for each processor is kept constant. Ideally, the CPU time taken for any number of processors should be the same as one processor. Table 5.71 and Figure 5.93 show the ratio of CPU time to the CPU time with one processor. Our new simulator outperforms UTCOMPP in CPU time, CPU time ratio and capacity. Our new simulator is faster for all number of processors used than UTCOMPP and it was capable to handle all model sizes up to 51.2 millions on 256 processors. The detailed CPU time can be observed in Table 5.72 and Figure 5.94. Figure 5.95 shows the percentage of total CPU time spent on each simulation section.

Table 5.70: CPU times for case 9

	CPU time (s)	
# processors	New simulator	UTCOMPP
1	6477.59	8564.66
2	6757.46	9135.96
4	7527.65	10279.6
8	8451.22	13412.3
16	11580.2	17072.7
32	12610.8	–
64	15080.6	–
128	15935.1	–
256	19002.7	–

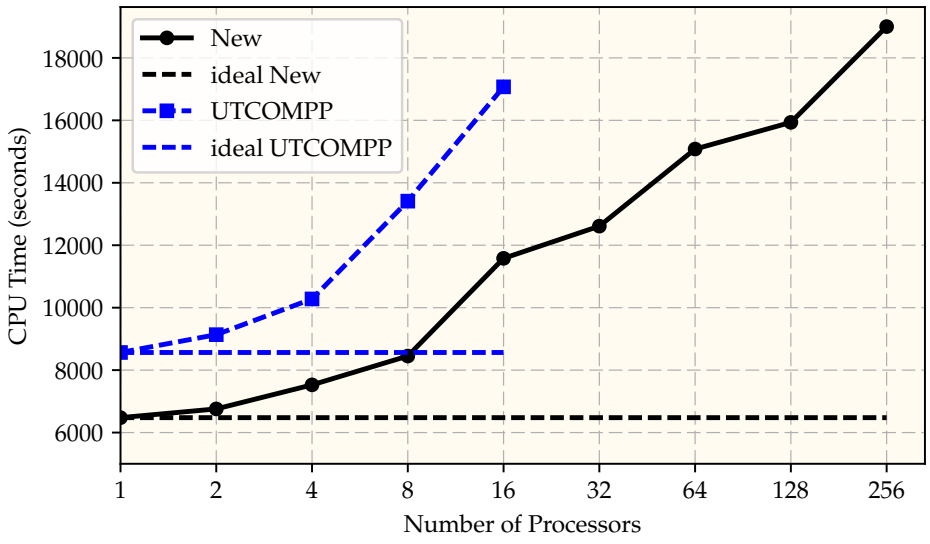


Figure 5.92: CPU time, case 9

Table 5.71: CPU time ratio case 9

	CPU time / CPU time @ 1 prc	
# processors	New simulator	UTCOMPP
1	1	1
2	1.043	1.066
4	1.162	1.200
8	1.304	1.566
16	1.787	1.993
32	1.946	–
64	2.328	–
128	2.460	–
256	2.933	–

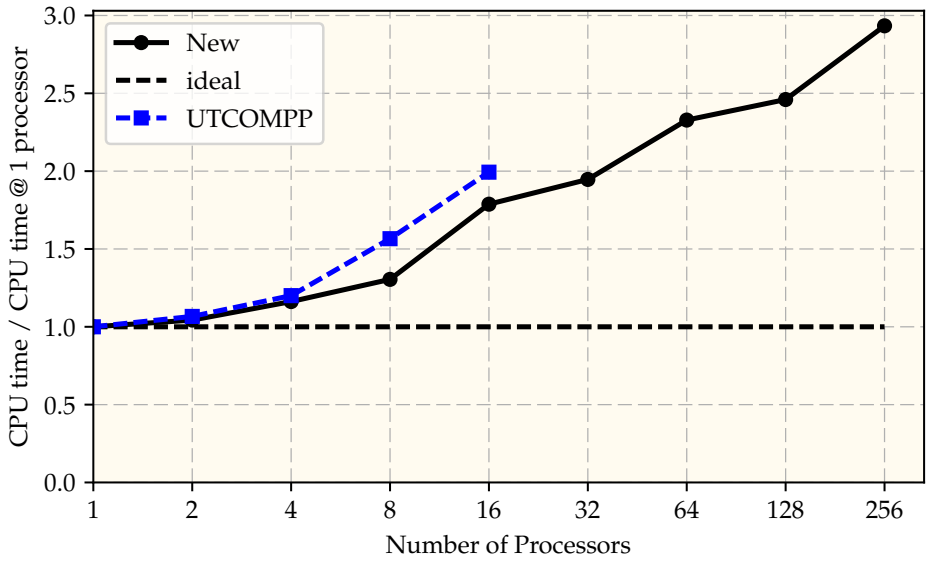


Figure 5.93: CPU time ratio, case 9

Table 5.72: Detailed CPU times in seconds for case 9

	Number of processors used								
	1	2	4	8	16	32	64	128	256
Total Initialization Time	5.65	16.193	17.307	21.632	21.812	23.141	24.768	594.169	1993.66
Total Linear Solver Time	574.317	747.751	930.265	1028.49	1729.93	1725.53	2198.11	2123.06	2344.43
Matrix	167.587	170.262	214.569	359.966	697.472	786.781	1067.33	1082.55	1154.05
Update Ghost Cells	0.083	26.469	165.521	92.378	361.352	323.457	392.386	716.286	1406.47
Compute Derivatives	342.665	341.992	447.346	573.528	996.403	1174.91	1677.03	1652.99	1754.84
Physical Properties Before Solver	487.022	490.905	544.029	608.163	938.868	1048.24	1306.67	1199.27	1335.73
Phase Composition Calculation	4282.24	4334.72	4362.59	4878.44	5213.15	5715.33	5898.95	6012.03	5926.42
Physical Properties After Solver	388.3	396.089	495.267	545.485	921.075	1044.97	1368.39	1405.29	1516.8
Concentration Computations	185.428	188.13	229.884	287.358	582.614	666.885	902.752	957.062	1051.49
Other	43.924	44.686	50.353	54.69	76.395	86.819	124.309	130.396	151.433
Total Execution Time	6477.59	6757.46	7527.65	8451.22	11580.2	12610.8	15080.6	15935.1	19002.7

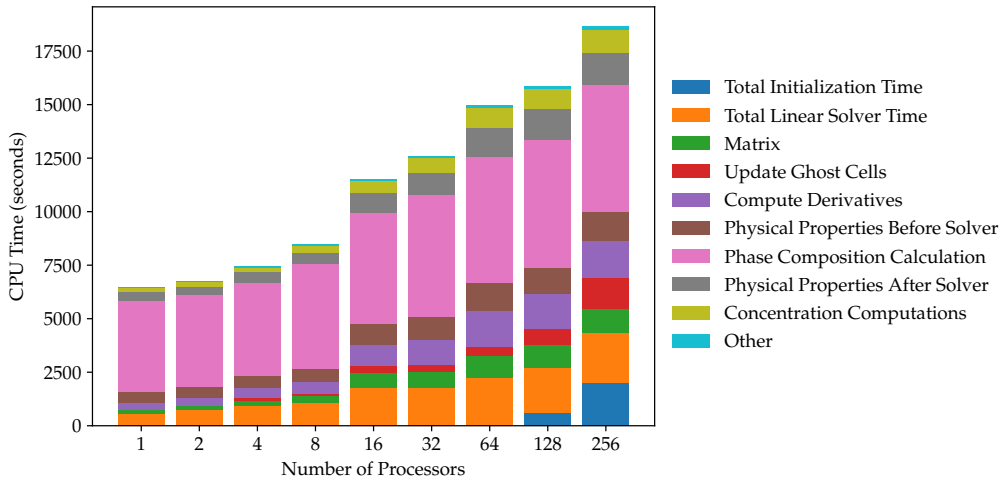


Figure 5.94: Detailed CPU times for case 9

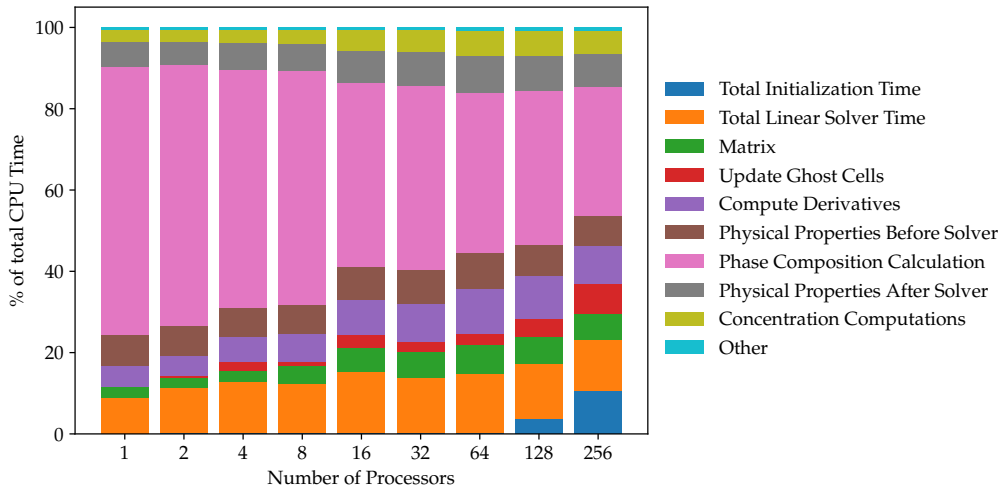


Figure 5.95: Percentage of total CPU time for timers for case 9

5.2.6 Case 10 - Effect of Number of Cores per Node used

The objective of this case is to evaluate the impact of multicore architecture into the parallel performance of our new simulator. Multicore processors impose additional variables that need to be considered when evaluating parallel performance on a parallel reservoir simulator. In a multicore architecture, cache, RAM memory and channels to access RAM are shared among all cores. The trend to increase performance in new multicore processors is to increase the number of cores. However, cache, RAM speed and number of memory channels does not increase at the same rate as number of cores does. This case evaluates the effect of the number of cores per node used on the parallel performance of our new simulator. Table 5.73 shows the specifications of the compute nodes of Lonestar 5. For any node we have up to 24 cores available but only 8 memory channels. Our models does not fit on the 30 MB cache, hence we need to access the RAM to store our model during the simulation.

Table 5.73: Lonestar 5 compute node specification

Processor socket	Xeon E5-2690 v3
Processor speed	2.6 GHz
Cores per socket	12
Cache	30 MB
Memory channels per socket	4
Sockets per node	2
Cores per node	24
Memory channels per node	8
Memory channels to core ratio	1:3
RAM per node	64 GB
RAM speed	DDR4 @ 2133 MHz

The model used in this case is the same model described in Section 5.2.5. Simulation time was reduced to 60 days. Production wells are controlled with constant BHP at 3500 psi. Injector wells are controlled with constant water rate of 10000 STB/day and a BHP limit of 6000psi.

The simulation case was run from 1 to 128 processors with different number of cores per node, from 1 to 16. As shown in Table 5.74, the column All cores available indicates that for any number of processors, the minimum number of nodes was used.

For example, when 64 processors are needed, 3 compute nodes are used. Hence, two compute nodes will be using 21 processors and one will be using 22.

Table 5.74: Number of compute nodes used in each run, case 10

	Cores per node used						
	1	2	4	8	16	All cores available	
# processors	# Nodes used					Cores/Node	# Nodes used
1	1					1	1
2	2	1				2	1
4	4	2	1			4	1
8	8	4	2	1		8	1
16	16	8	4	2	1	16	1
32	32	16	8	4	2	16	2
64	64	32	16	8	4	64/3	3
128	128	64	32	16	8	64/3	6

Table 5.75 and Figure 5.96 show the CPU times obtained for each simulation run. Figure 5.97 Shows each result normalized with the CPU time obtained from 1 processor.

Table 5.75: CPU time (seconds) case 10

	Cores per node used					
# processors	1	2	4	8	16	All available
1	1954.92					1954.92
2	2048.24	1942.88				1942.88
4	2031.58	1913.35	2088.15			2088.15
8	2020.85	2264.4	1975.01	2200.03		2200.03
16	2005.83	1895.18	1962.57	2203.75	2706.24	2706.24
32	1987.66	2274.52	1991.54	2171.26	2694.97	2694.97
64	2030.21	2263.35	2027.55	2194.73	2746.07	3385.5
128	2501.66	2388	2531.42	2732.02	3337.79	3964.78

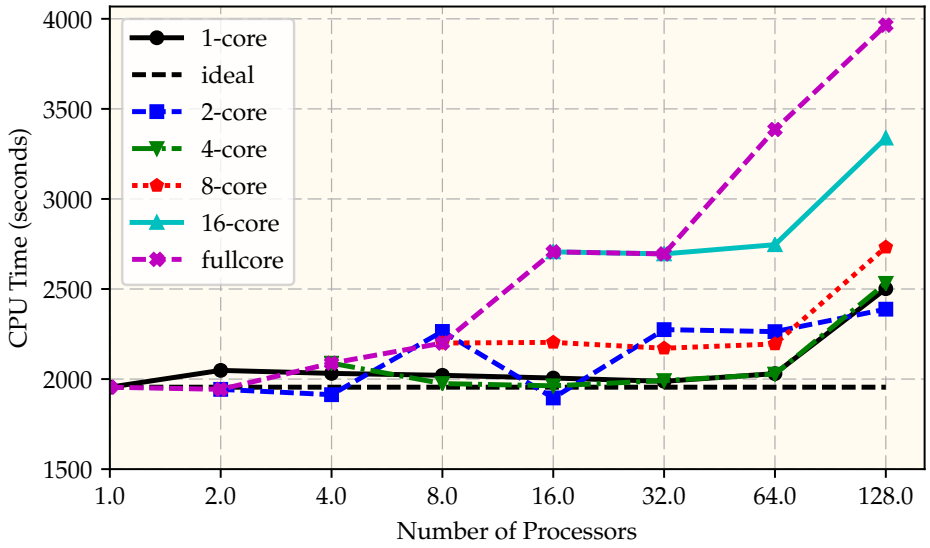


Figure 5.96: CPU time, case 10

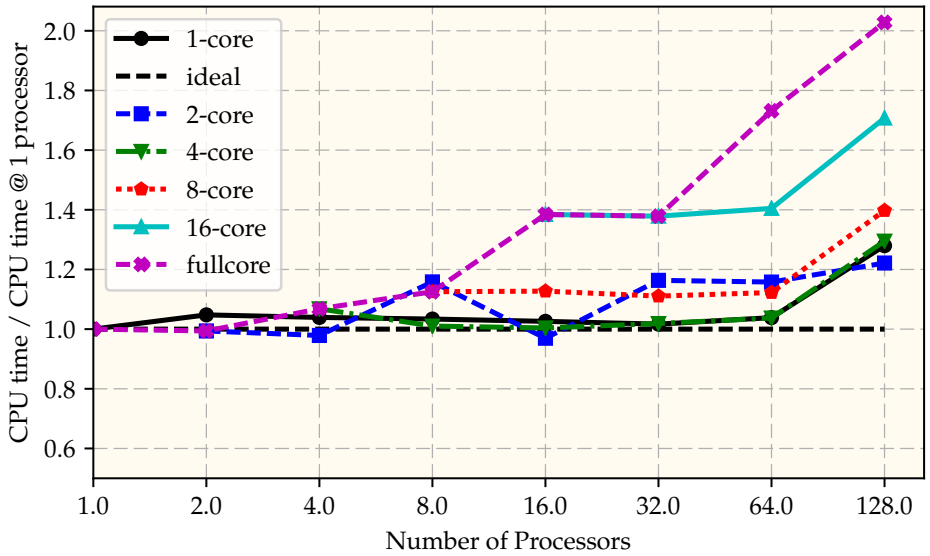


Figure 5.97: CPU time ratio, case 10

Table 5.76 shows the memory channels to core ratio when different cores per node were used. From the results of this case we can observe that the number of memory channels available per core has a big impact on the parallel performance of our new simulator. When there is more than one memory channel available per core, that is 1, 2, 4, and 8 cores per node, then the CPU time behaves very close to ideal. No more than 20% of increase in CPU time is obtained. The CPU times with 1 and 4 cores per node were the closest to ideal behavior.

On the other hand, when there is less than one memory channel available per core, that is when 16 and 64/3 (two nodes using 21 processors and one using 22 processors) cores per node are used, the parallel performance of our new simulator is decreased. The memory channels became a performance bottleneck because many cores are trying to access RAM through it at the same time.

Table 5.76: Number of number of channels available per core vs. cores per node used

Cores per node	Memory channels to core ratio
1	8:1
2	4:1
4	2:1
8	1:1
16	1:2
64/3	3:8

5.2.7 Case 11 - Waterflooding Weak Scalability Test

The objective of this case is to evaluate the performance of our new simulator under higher number of processors and bigger models than the previous cases. The model used in this case is the same model described in Section 5.2.5. Simulation time was reduced to 30 days. Production wells are controlled with constant BHP at 3500 psi. Injector wells are controlled with constant water rate of 10000 STB/day and a BHP limit of 6000 psi.

Table 5.77 shows the model size for each number of processors used. Up to 512 processors and up to 102.4 million grid cells were used.

Table 5.77: Model size, case 11

	Grid size		Model dimensions (ft)		
# processors	# of cells (Million)	(NXxNYxNZ)	Length	Width	Thickness
1	0.2	(200x200x5)	2800	2800	100
2	0.4	(400x200x5)	5600	2800	100
4	0.8	(400x400x5)	5600	5600	100
8	1.6	(800x400x5)	11200	5600	100
16	3.2	(800x800x5)	11200	11200	100
32	6.4	(1600x800x5)	22400	11200	100
64	12.8	(1600x1600x5)	22400	22400	100
128	25.6	(3200x1600x5)	44800	22400	100
256	51.2	(3200x3200x5)	44800	44800	100
512	102.4	(6400x3200x5)	89600	44800	100

Table 5.78: CPU times for case 11

# processors	CPU time (s)
1	1581.56
2	1580.04
4	1620.84
8	1808.72
16	2223.09
32	2297.32
64	2697.2
128	3308.59
256	4919.48
512	10301.3

CPU time is shown in Table 5.78 and Figure 5.98. CPU time to CPU time at 1 processor is shown in Table 5.79 and Figure 5.99.

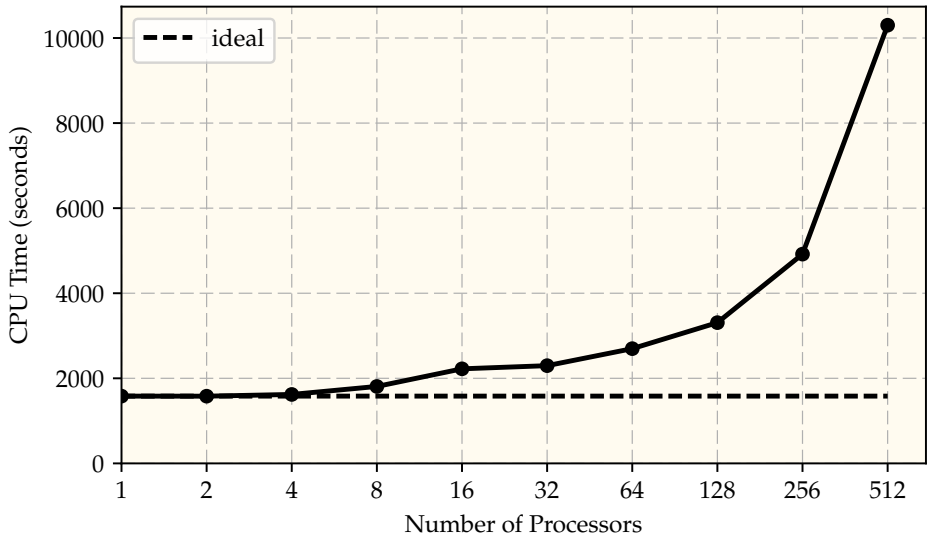


Figure 5.98: CPU time, case 11

Table 5.79: CPU time ratio case 11

# processors	CPU time / CPU time @ 1 prc
1	1
2	0.999
4	1.024
8	1.143
16	1.405
32	1.452
64	1.705
128	2.091
256	3.110
512	6.513

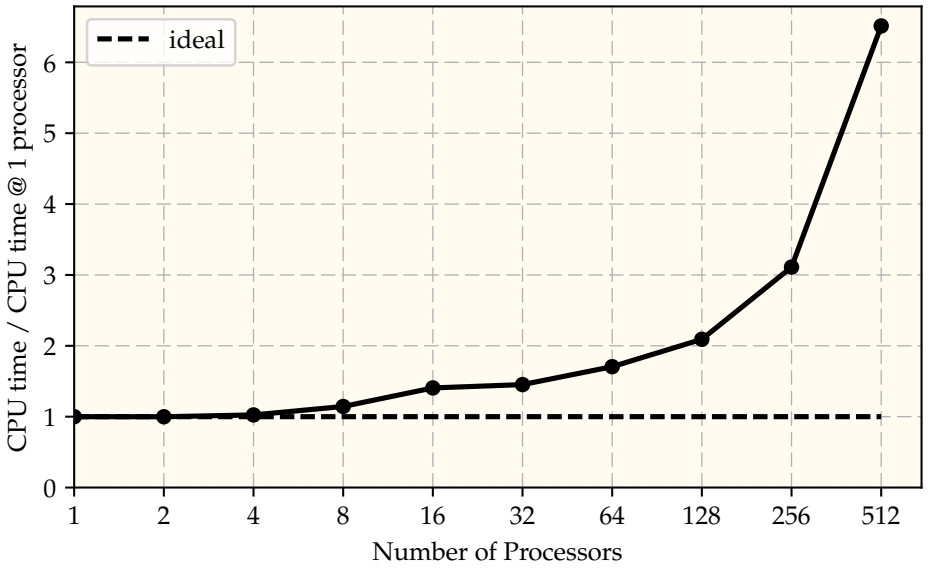


Figure 5.99: CPU time ratio, case 11

Table 5.80: Detailed CPU times in seconds for case 11

	Number of processors used				
	1	2	4	8	16
Total Initialization Time	5.604	16.146	17.049	20.706	21.735
Total Linear Solver Time	99.769	119.77	132.691	143.628	207.891
Matrix	43.074	41.958	49.968	81.624	147.742
Compute Derivatives	88.2	84.591	99.465	130.834	230.418
Physical Properties Before Solver	126.292	121.9	124.66	138.497	194.426
Phase Composition Calculation	1060.65	1031.16	1019.09	1072.09	1072.03
Physical Properties After Solver	98.864	96.59	103.674	120.367	188.753
Concentration Computations	47.574	46.981	50.122	65.237	117.618
Other	11.403	19.881	23.82	35.145	42.047
Total Execution Time	1581.56	1580.04	1620.84	1808.72	2223.09
	Number of processors used				
	32	64	128	256	512
Total Initialization Time	23.49	25.957	595.514	2059.66	7307.56
Total Linear Solver Time	208.155	271.836	256.463	262.452	260.146
Matrix	147.771	195.17	194.631	212.133	210.101
Compute Derivatives	227.414	295.116	294.468	319.63	328.121
Physical Properties Before Solver	190.816	239.751	221.586	248.623	230.974
Phase Composition Calculation	1078.45	1124.67	1142.08	1161.15	1161.53
Physical Properties After Solver	194.582	261.182	266.718	290.428	294.078
Concentration Computations	125.149	165.031	166.546	186.106	189.115
Other	100.702	112.667	160.423	161.507	235.787
Total Execution Time	2297.32	2697.2	3308.59	4919.48	10301.3

The detailed CPU time can be observed in Table 5.80 and Figure 5.100. Figure 5.101 shows the percentage of total CPU time spent on each simulation section. It can be observed that above 128 processors the initialization time, that is the domain decomposition algorithm abruptly increases the CPU time required to operate. Domain decomposition time takes more than 70% of the total CPU time when 512 processors are used. If we keep increasing the number of processors and the size of the model then the time required to divide the reservoir becomes prohibitively expensive. The others part of the simulation are close to constant.

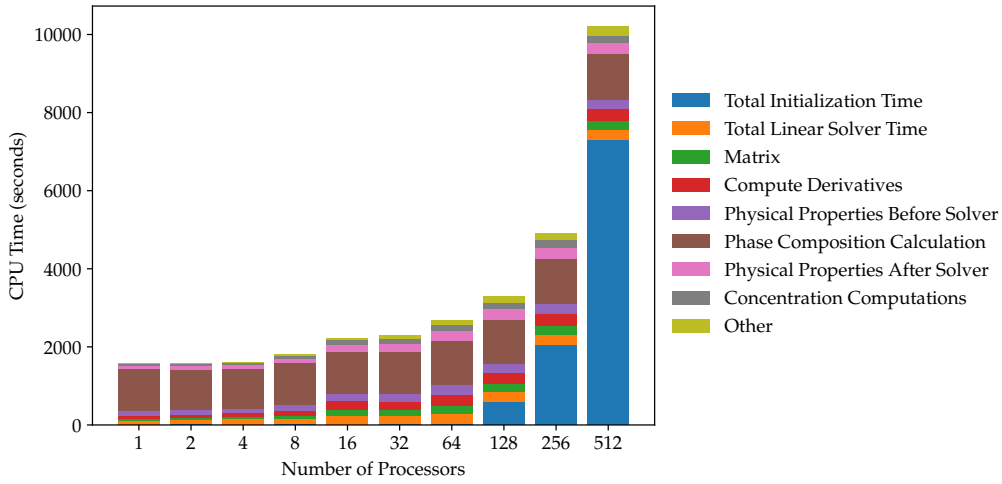


Figure 5.100: Detailed CPU times for case 11

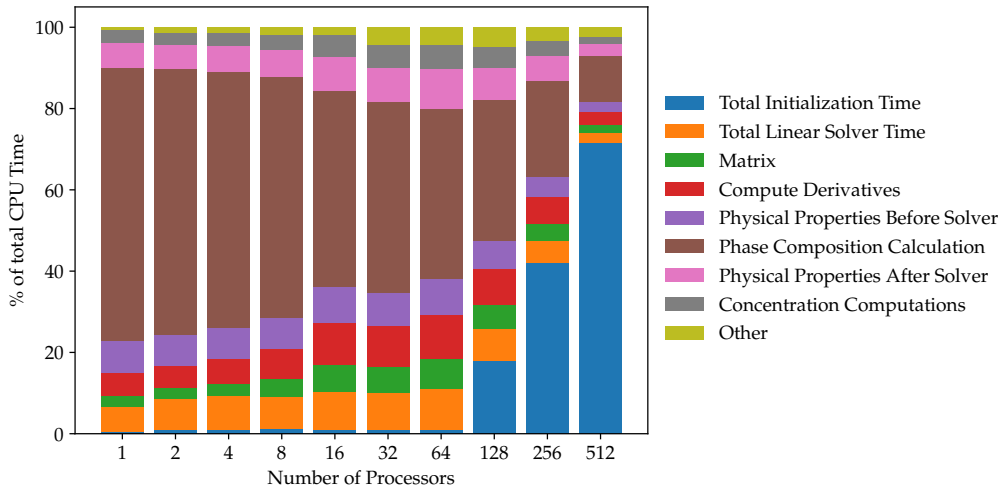


Figure 5.101: Percentage of total CPU time for timers for case 11

5.2.8 Case 12 - Waterflooding Strong Scalability Test

The model used in this case is the same model described in Section 5.2.5. Grid size is 3.2 million cells (800x800x5). Simulation time was reduced to 60 days. Production wells are controlled with constant BHP at 3500 psi. Injector wells are controlled with constant water rate of 10000 STB/day and a BHP limit of 6000psi. Up to 1024 processors were used. The model was not able to run in one processor. CPU time is shown in Table 5.81 and Figure 5.102. Speedup is shown in Table 5.82 and Figure 5.103. The detailed CPU time can be observed in Table 5.83 and Figure 5.104. The percentage of total CPU time spent on each simulation section is shown in Figure 5.105.

Table 5.81: CPU times for case 12

# processors	CPU time (s)
2	13142.9
4	6526.77
8	3609.7
16	2293.5
32	1109.9
64	646.022
128	288.561
256	156.569
512	636.239
1024	4267.22

Table 5.82: Speedup for case 12

# processors	Speedup
2	2
4	4.02737
8	7.28197
16	11.4609
32	23.6829
64	40.6886
128	91.0924
256	167.886
512	41.3142
1024	6.15991

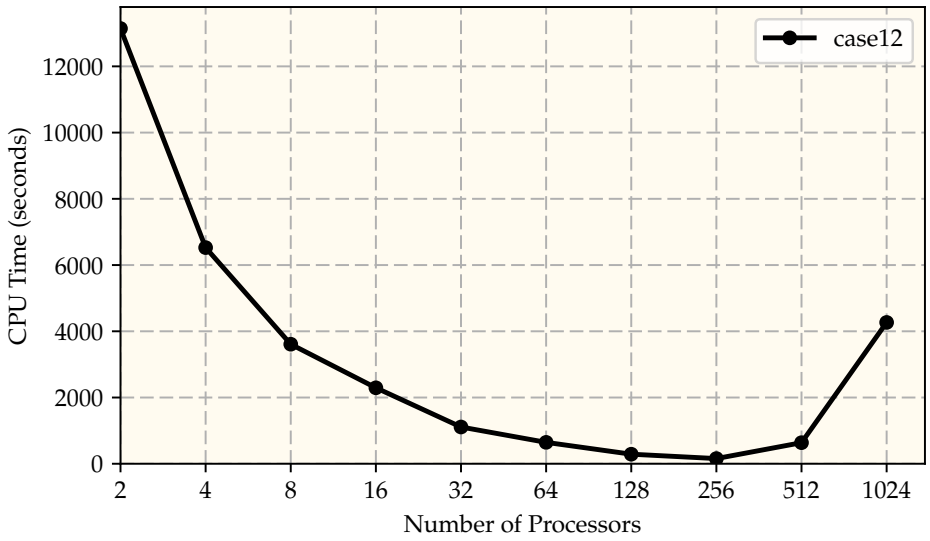


Figure 5.102: CPU time, case 12

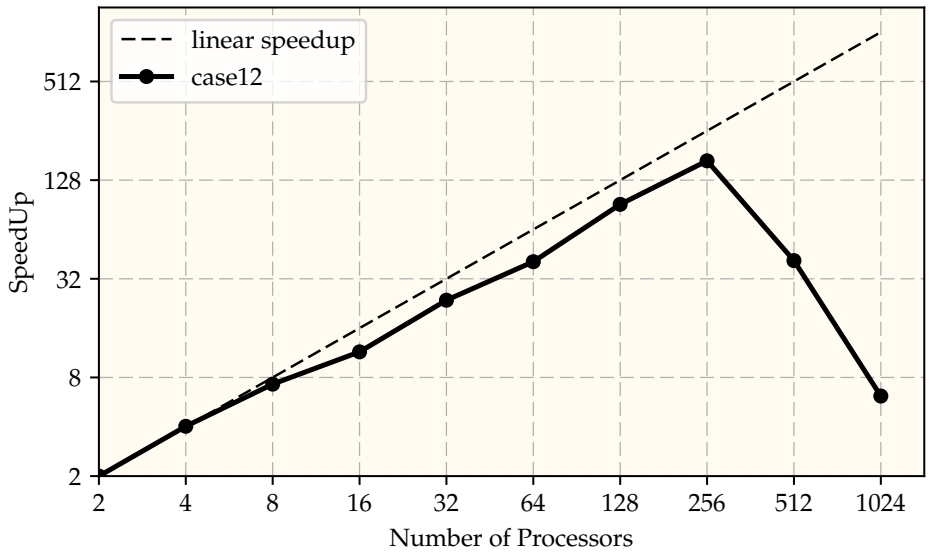


Figure 5.103: Speedup, case 12

Table 5.83: Detailed CPU times in seconds for case 12

	Number of processors used				
	2	4	8	16	32
Total Initialization Time	721.867	190.809	69.907	21.842	8.138
Total Linear Solver Time	847.874	484.656	288.501	229.319	105.152
Matrix	476.572	250.126	166.884	145.526	69.295
Update Ghost Cells	31.052	29.226	36.567	82.146	48.412
Compute Derivatives	697.314	358.929	233.976	211.925	110.556
Physical Properties Before Solver	995.681	504.196	280.359	197.29	88.765
Phase Composition Calculation	8042.09	4013.15	2125.1	1072.61	535.515
Physical Properties After Solver	771.692	406.329	241.537	187.037	82.832
Concentration Computations	428.044	228.038	138.788	123.589	53.55
Other	128.287	59.212	26.933	16.174	7.217
Total Execution Time	13142.9	6526.77	3609.7	2293.5	1109.9
	Number of processors used				
	64	128	256	512	1024
Total Initialization Time	14.705	11.223	10.093	548.578	4185.51
Total Linear Solver Time	67.031	30.328	15.279	10.381	15.608
Matrix	37.634	7.821	3.616	1.677	0.852
Update Ghost Cells	32.328	15.208	11.175	12.481	17.759
Compute Derivatives	73.758	31.702	11.718	4.763	2.281
Physical Properties Before Solver	50.752	20.211	11.855	6.196	2.818
Phase Composition Calculation	281.403	136.727	72.897	36.835	18.573
Physical Properties After Solver	47.261	22.427	11.158	5.784	3.795
Concentration Computations	29.981	10.189	5.017	2.727	2.62
Other	4.191	2.273	2.185	1.959	3.012
Total Execution Time	646.022	288.561	156.569	636.239	4267.22

The model scales up very well up to 256 processors achieving speedup of 167.88. Above that there is a abrupt drop on performance. The reason for the poor performance is the Initialization section. As in Section 5.2.7 the domain decomposition algorithm does not scale up well with many processors, in this case above 256. Domain decomposition takes 86.2% of total CPU time for 512 processors and 98.08% of total CPU time for 1024 processors.

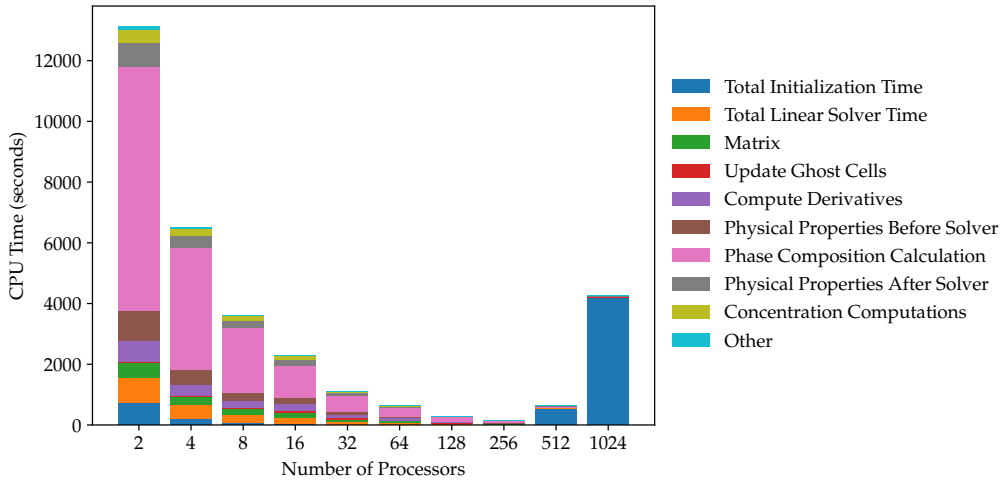


Figure 5.104: Detailed CPU times for case 12

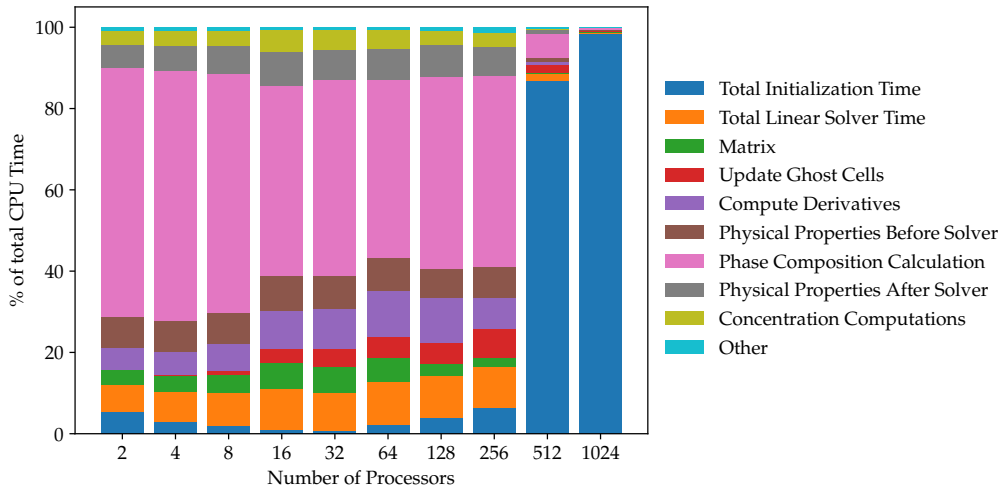


Figure 5.105: Percentage of total CPU time for timers for case 12

Chapter 6

Summary, Conclusions and Recommendations

6.1 Summary

- A framework was developed to aid the development of parallel reservoir simulators.
- The framework was designed to be modular, maintainable, extensible and compatible with code already developed.
- The whole framework was developed in Fortran. Using only one programming language decreases the complexity at the time of maintenance and extension.
- MPI was used for inter processor communication and input/output of spatial related data.
- Recursive Coordinate Bisection algorithm was used for domain decomposition.
- A custom algorithm based on lists of ghost cells was developed in order to identify neighbor processors and to define communication patterns.
- Parallel distributed arrays (PArrays) module was developed as an easy to used solution to the problem of properties distributed among processors.
- Lua scripting language was embedded in the framework to provide input processing.
- The framework has the capability to create output for visualization in S3graf and VTK format.
- The UTCOMP simulator was adapted to the framework to create a new parallel reservoir simulator. The new parallel reservoir simulator has all the features of UTCOMP.

- Several simulation cases were used to verify the results of our new parallel reservoir simulator.
- Several simulation cases were used to evaluate the parallel performance of our new parallel reservoir simulator.

6.2 Conclusions

- Parallel computing applied to reservoir simulation is capable to reduce overall CPU time. However, the creation of parallel reservoir simulators is more complex than non-parallel reservoir simulators.
- The framework developed in this work provides the necessary tools to reduce the complexity associated with parallel programming during the development of parallel reservoir simulators.
- MPI is a widespread standard. It has many implementations and it is very portable. MPI provides with the necessary tools to efficiently communicate data between processors.
- The use of MPI for parallel input and output of spatial properties avoids the bottlenecks related with sequential input and output.
- To the extend of our knowledge. This is the first time a reservoir simulator uses Lua scripting language as input processing. This allows greater flexibility compared with traditional keyword based input processing.
- The new parallel reservoir simulator developed in this work has a more complex input processing, domain decomposition, and communication pattern than UTCOMPP. Although, Our new simulator was able to have a very similar parallel performance than UTCOMPP in strong scalability tests. Additionally, our new simulator had a much more better performance that UTCOMPP in weak scalability tests.
- Our new parallel simulator was able to run simulation models up to 102.4 million cells and up to 1024 processors. This capacity is far higher than UTCOMPP.

- Recursive Coordinate Bisection algorithm does require only cells I J K coordinate information to perform domain decomposition in 2D and possibly in 3D. This algorithm requires less memory and input data than graph based domain decomposition algorithms. The resulting domains are regular and they can be distributed irregularly.

- RCB algorithm does a very well domain distribution when the simulation model has a lot of inactive grid cells. The resulting distribution of domains assigns few processors to the parts of the model with many inactive grid cells.

- The performance of the RCB algorithm was greatly reduced when 512 or more processors were used or when the size of the model is bigger than 51 million cells. The domain decomposition algorithm used in a parallel reservoir simulator could impact negatively in the parallel performance of the simulator or limit its capability.

- Parallel reservoir simulation, under the actual hardware, is limited by the speed of access of RAM and not by the processor speed. This is because the size of simulation models commonly used in parallel reservoir simulation are big enough to not to fit in cache memory and the data needs to be stored in RAM and accessed through memory channels by the cores. The speed of accessing RAM is slower than the typical speed of processors, hence, RAM speed is the limiting factor.

- When hardware is updated, the biggest improvement in CPU time is obtained with one processor. This is usually because the increase in RAM speed. The improvement in more than one processor is not as big because there are other factors involved such as number of cores per socket or number of memory channels per socket that can decrease the improvement. Additionally, when the speed of processing is higher, the communication between processors became more important and have a bigger impact on CPU time in newer hardware than older hardware. For these reasons. The computed speedup in newer multicore machines is lower than the speedup obtained with older multicore machines.

- In multi core machines, the number of memory channels available to each core used during simulation has a great impact on parallel performance. In order not to create a bottleneck in memory access at least one memory channel per used core is needed.

- During parallel tests. It is advisable to maintain a constant number of cores per node. By doing this we maintain the number of memory channels available to each used core constant and avoid the possibility of include a variable bottleneck at different number of processors.

6.3 Recommendations for Future Work

- Try an hybrid programming approach for parallel communication. The hybrid approach consists of using OpenMP for shared memory parallelization within a compute node and MPI for communication between compute nodes.

- Modify the framework to use of accelerators like Many Integrated Cores (MIC) architecture or General Purpose Graphical Processing Units (GPGPU). Additional research should be performed on which part of the framework provides the greatest benefit of being executed by the accelerators.

- Memory optimization of UTCOMP subroutines should be performed. This allows bigger models to be run on serial and parallel machines without the need of increasing RAM.

- Implement corner point and unstructured grids on the framework.

- Add capability to the framework of handling spatial properties using one index. This could greatly increase the framework's applicability because several reservoir simulators use single index properties.

- Further research is needed to identify efficient and scalable domain decomposition algorithms for parallel reservoir simulation. These algorithms should be implemented in the framework.

Bibliography

Abate, J., Wang, P., and Sepehrnoori, K., "Parallel Compositional Reservoir Simulation on Clusters of PCs," in: *International Journal of High Performance Computing Applications* 15.1 (2001), pp. 13–21, URL: http://hpc.sagepub.com/content/15/1/13.abstract.

Amdahl, G. M., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), Atlantic City, New Jersey: ACM, 1967, pp. 483–485, URL: http://doi.acm.org/10.1145/1465482.1465560.

Atan, S., Kazemi, H., and Caldwell, D. H., "Efficient Parallel Computing Using Multiscale Multimesh Reservoir Simulation," in: *SPE-103101-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2006.

Babu, D. and Odeh, A. S., "Productivity of a Horizontal Well (Includes Associated Papers 20306, 20307, 20394, 20403, 20799, 21307, 21610, 21611, 21623, 21624, 25295, 25408, 26262, 26281, 31025, and 31035)," in: *SPE-18298-PA* (Nov. 1, 1989).

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., *PETSc Web Page*, 2016, URL: http://www.mcs.anl.gov/petsc.

Barua, J. and Horne, R., "Improving the Performance of Parallel (and Serial) Reservoir Simulators," in: *SPE-18408-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1989.

Beckner, B., Haugen, K. B., Maliassov, S., Dyadechko, V., and Wiegand, K. D., "General Parallel Reservoir Simulation," in: *SPE-177532-MS*, SPE: Society of Petroleum Engineers, Nov. 9, 2015.

Berger, M. J. and Bokhari, S. H., "A Partitioning Strategy for Nonuniform Problems on Multiprocessors," in: *IEEE Transactions on Computers* C-36.5 (May 1987), pp. 570–580.

Blaise, B., "Introduction to Parallel Computing," in: *Lawrence Livermore National* UCRL-MI-133316 (EC3500 2015), URL: https://computing.llnl.gov/tutorials/parallel_comp/.

— "OpenMP," in: *Lawrence Livermore National* UCRL-MI-133316 (EC3507 2015), URL: https://computing.llnl.gov/tutorials/openMP/.

Briens, F. J., Wu, C. H., Gazdag, J., and Wang, H. H., "Sequential Staging of Tasks: A New Approach to Parallel Computations," in: *SPE-21088-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1990.

Ceze, L. H., "Shared-Memory Multiprocessors," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 1810–1812, URL: http://dx.doi.org/10.1007/978-0-387-09766-4.

Chang, Y.-B., "Development and Application of an Equation of State Compositional Simulator," Ph.D. Dissertation, 1990, 551 pp., URL: http://search.proquest.com/docview/303893086.

Chapman, B., Jost, G., and Van Der Pas, R., *Using OpenMP: Portable Shared Memory Parallel Programming*, vol. 10, MIT press, 2008.

Chapman, B. and LaGrone, J., "OpenMP," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 1365–1371, URL: http://dx.doi.org/10.1007/978-0-387-09766-4.

Cheshire, I. and Bowen, G., "Parallelization in Reservoir Simulation," in: *SPE-23657-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1992.

Chien, M. and Northrup, E., "Vectorization and Parallel Processing of Local Grid Refinement and Adaptive Implicit Schemes in a General Purpose Reservoir Simulator," in: *SPE-25258-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1993.

Chien, M., Wasserman, M., Yardumian, H., Chung, E., Nguyen, T., and Larson, J., "The Use of Vectorization and Parallel Processing for Reservoir Simulation," in: *SPE-16025-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1987.

Corey, A. T., *Mechanics of Immiscible Fluids in Porous Media*, 2nd, Water Resources Publications, 1986.

Darabi, H., "Development of a Non-Isothermal Compositional Reservoir Simulator to Model Asphaltene Precipitation, Flocculation, and Deposition and Remediation," Ph.D. Dissertation, The University of Texas at Austin, 2014, URL: http://hdl.handle.net/2152/24810.

DeBaun, D., Byer, T., Childs, P., Chen, J., Saaf, F., Wells, M., Liu, J., Cao, H., Pianelo, L., Tilakraj, V., Crumpton, P., Walsh, D., Yardumian, H., Zorzynski, R., Lim, K.-T., Schrader, M., Zapata, V., Nolen, J., and Tchelepi, H., "An Extensible Architecture for Next Generation Scalable Parallel Reservoir Simulation," in: *SPE-93274-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2005.

Delshad, M., Han, C., Sepehrnoori, K., and Najafabadi, N. F., "Development of a Three Phase, Fully Implicit, Parallel Chemical Flood Simulator," in: *SPE-119002-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2009.

Devine, K., Boman, E., Heaphy, R., Hendrickson, B., and Vaughan, C., "Zoltan Data Management Services for Parallel Dynamic Applications," in: *Computing in Science Engineering* 4.2 (Mar. 2002), pp. 90–96.

Dogru, A. H., Fung, L. S. K., and Sindi, M. O., "Multi-Paradigm Parallel Acceleration for Reservoir Simulation," in: *SPE-163591-MS*, SPE: Society of Petroleum Engineers, Feb. 18, 2013.

Fazelipour, W., Pope, G. A., and Sepehrnoori, K., "Development of a Fully Implicit, Parallel, EOS Compositional Simulator to Model Asphaltene Precipitation in Petroleum Reservoirs," in: *SPE-120203-STU*, SPE: Society of Petroleum Engineers, Jan. 1, 2008.

Flynn, M., "Flynn's Taxonomy," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 689–697, URL: http://dx.doi.org/10.1007/978-0-387-09766-4.

— "Some Computer Organizations and Their Effectiveness," in: *IEEE Transactions on Computers* C-21.9 (Sept. 1972), pp. 948–960.

Fung, L. S. K. and Du, S., "Parallel-Simulator Framework for Multipermeability Modeling With Discrete Fractures for Unconventional and Tight Gas Reservoirs," in: *SPE-179728-PA* (Aug. 1, 2016).

Gebali, F., *Algorithms and Parallel Computing*, Hoboken, UNITED STATES: Wiley, 2011.

Ghasemi Doroh, M., "Development and Application of Parallel Compositional Reservoir Simulator," Master's Thesis, University of Texas at Austin, 2012.

Ghori, S., Wang, C., Lim, M., Pope, G., Sepehrnoori, K., and Wheeler, M. F., "Compositional Reservoir Simulation on CM-5 and KSR-1 Parallel Machines," in: *SPE-29140-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1995.

Gropp, W., "MPI (Message Passing Interface)," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 1184–1190, URL: http://dx.doi.org/10.1007/978-0-387-09766-4.

Gropp, W., Hoefler, T., Thakur, R., and Lusk, E., *Using Advanced MPI: Modern Features of the Message-Passing Interface*, Scientific and Engineering Computation, MIT Press, 2014.

Gropp, W., Lusk, E., and Skjellum, A., *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 3rd, Scientific and Engineering Computation, Cambridge, MA: MIT Press, 2014.

Guan, W., Qiao, C., Zhang, H., Zhang, C.-S., Zhi, M., Zhu, Z., Zheng, Z., Ye, W., Zhang, Y., Hu, X., Li, Z., Feng, C., Xu, Y., and Xu, J., "On Robust and Efficient Parallel Reservoir Simulation on Tianhe-2," in: *SPE-175602-MS*, SPE: Society of Petroleum Engineers, Sept. 14, 2015.

Gustafson, J. L., "Amdahl's Law," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 53–60, URL: http://dx.doi.org/10.1007/978-0-387-09766-4.

— "Fixed Time, Tiered Memory, and Superlinear Speedup," in: *Proceedings of the Fifth Distributed Memory Computing Conference, 1990.* Vol. 2, Apr. 1990, pp. 1255–1260.

— "Reevaluating Amdahl's Law," in: *Commun. ACM* 31.5 (May 1988), pp. 532–533, URL: http://doi.acm.org/10.1145/42411.42415.

Han, C., Delshad, M., Sepehrnoori, K., and Pope, G. A., "A Fully Implicit, Parallel, Compositional Chemical Flooding Simulator," in: *SPE-97217-PA* (Sept. 1, 2007).

Ierusalimschy, R., *Programming in Lua*, 4th, Rio de Janeiro: Roberto Ierusalimschy, 2016.

Ierusalimschy, R., de Figueiredo, L. H., and Celes, W., "The Evolution of Lua," in: *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, HOPL III, San Diego, California: ACM, 2007, pp. 2–1–2–26, URL: http://doi.acm.org/10.1145/1238844.1238846.

Ierusalimschy, R., de Figueiredo, L. H., and Filho, W. C., "Lua—An Extensible Extension Language," in: *Software: Practice and Experience* 26.6 (1996), pp. 635–652, URL: http://dx.doi.org/10.1002/(SICI)1097-024X(199606)26:6%3C635::AID-SPE26%3E3.0.CO;2-P.

John, A., Han, C., Delshad, M., Pope, G., and Sepehrnoori, K., "A New Generation Chemical Flooding Simulator," in: *SPE-89436-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2004.

Kårstad, T., Henriquez, A., and Korsell, K., "Parallelization of a Reservoir Simulator," in: *Supercomputing: 1st International Conference Athens, Greece, June 8–12, 1987 Proceedings*, ed. by E. N. Houstis, T. S. Papatheodorou, and C. D. Polychronopoulos, Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 863–872, URL: http://dx.doi.org/10.1007/3-540-18991-2.

Khait, M. L., "Efficient Parallel Reservoir Simulation Using Multicore Architectures," in: *SPE-119107-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2009.

Killough, J. E. and Kossack, C. A., "Fifth Comparative Solution Project: Evaluation of Miscible Flood Simulators," in: *SPE-16000-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1987.

Killough, J. E. and Bhogeswara, R., "Simulation of Compositional Reservoir Phenomena on a Distributed-Memory Parallel Computer," in: *SPE-21208-PA* (Nov. 1, 1991).

Killough, J. E. and Wheeler, M. F., "Parallel Iterative Linear Equation Solvers: An Investigation of Domain Decomposition Algorithms for Reservoir Simulation," in: *SPE-16021-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1987.

Kremer, U. and Ramé, M., "Compositional Oil Reservoir Simulation in Fortran D: A Feasibility Study On Intel iPsc/860," in: *The International Journal of Supercomputer Applications and High Performance Computing* 8.2 (1994), pp. 119–128, URL: http://dx.doi.org/10.1177/10943420940080020 4.

Lake, L., Carey, G., Pope, G., and Sepehrnoori, K., "Isothermal, Multiphase, Multicomponent Fluid Flow in Permeable Media," in: *In Situ; (United States)* 8:1 (Jan. 1984).

Liu, H., Wang, K., Chen, Z., Jordan, K. E., Luo, J., and Deng, H., "A Parallel Framework for Reservoir Simulators on Distributed-Memory Supercomputers," in: *SPE-176045-MS*, SPE: Society of Petroleum Engineers, Oct. 20, 2015.

Liu, J., "High-Resolution Methods for Enhanced Oil Recovery Simulation," Ph.D. Dissertation, The University of Texas at Austin, 1993.

Liu, W., Cao, J., Mezzatesta, A., and Zhu, P., "Parallel Reservoir Simulation on Shared and Distributed Memory System," in: *SPE-64797-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2000.

Löf, H. T., Gerritsen, M. G., and Thiele, M. R., "Parallel Streamline Simulation," in: *SPE-113543-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2008.

Maliassov, S., Beckner, B., and Dyadechko, V., "Parallel Reservoir Simulation Using a Specific Software Framework," in: *SPE-163653-MS*, SPE: Society of Petroleum Engineers, Feb. 18, 2013.

Malony, A. D., "Metrics," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 1124–1130, URL: http://dx.doi.org/10.1007/978-0-387-09766-4.

Mayer, D., "Application of Reservoir Simulation Models to a New Parallel Computing System," in: *SPE-19121-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1989.

Meijerink, J., Van Daalen, D., Hoogerbrugge, P., and Zeestraten, R., "Towards a More Effective Parallel Reservoir Simulator," in: *SPE-21212-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1991.

Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 3.1*, 2015, URL: http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

Metcalf, M., Reid, J., and Cohen, M., *Modern Fortran Explained*, 4th, New York, NY, USA: Oxford University Press, Inc., 2011.

Michielse, P., "Parallel Multigrid Using PVM," in: *Applied Numerical Mathematics* 19.1 (1995), pp. 63–69, URL: http://www.sciencedirect.com/science/article/pii/016892749500018P.

Naimi-Tajdar, R., Han, C., Sepehrnoori, K., Arbogast, T. J., and Miller, M. A., "A Fully Implicit, Compositional, Parallel Simulator for IOR Processes in Fractured Reservoirs," in: *SPE-100079-PA* (Sept. 1, 2007).

Okuno, R., "Modeling of Multiphase Behavior for Gas Flooding Simulation," Ph.D. Dissertation, The University of Texas at Austin, 2009, 370 pp., URL: http://hdl.handle.net/2152/10585.

OpenMP Architecture Review Board, *OpenMP Application Programming Interface Version 4.5*, 2015, URL: http://www.openmp.org.

Pan, F., Sepehrnoori, K., and Chin, L., "Development of a Coupled Geomechanics Model for a Parallel Compositional Reservoir Simulator," in: *SPE-109867-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2007.

Parashar, M., Wheeler, J. A., Pope, G., Wang, K., and Wang, P., "A New Generation EOS Compositional Reservoir Simulator: Part II - Framework and Multiprocessing," in: *SPE-37977-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1997.

Pope, G., Wu, W., Narayanaswamy, G., Delshad, M., Sharma, M., and Wang, P., "Modeling Relative Permeability Effects in Gas-Condensate Reservoirs," in: *SPE-49266-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1998.

Qin, X., "Modeling Asphaltene Precipitation and Implementation of Group Contributuon Equation of State Into UTCOMP," Master's Thesis, The University of Texas at Austin, 1998.

Rame, M. and Delshad, M., "A Compositional Reservoir Simulator on Distributed Memory Parallel Computers," in: *SPE-29103-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1995.

Rutledge, J., Jones, D., Chen, W., and Chung, E., "The Use of a Massively Parallel SIMD Computer for Reservoir Simulation," in: *SPE-21213-PA* (July 1, 1992).

Scott, S., Wainwright, R., Raghavan, R., and Demuth, H., "Application of Parallel (MIMD) Computers to Reservoir Simulation," in: *SPE-16020-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1987.

Sherman, A., "A Hybrid Approach to Parallel Compositions Reservoir Simulation," in: *OTC-6829-MS*, OTC: Offshore Technology Conference, Jan. 1, 1992.

Shuhong, W., Baohua, W., Qiaoyun, L., Xu, J., Chensong, Z., and Chunsheng, F., "Cost-Effective Parallel Reservoir Simulation on Shared Memory," in: *SPE-182367-MS*, SPE: Society of Petroleum Engineers, Oct. 25, 2016.

Snir, M., "Distributed-Memory Multiprocessor," in: *Encyclopedia of Parallel Computing*, ed. by D. Padua, Boston, MA: Springer US, 2011, pp. 574–578, URL: `http://dx.doi.org/10.1007/978-0-387-09766-4`.

Sterling, T. L., Salmon, J., Becker, D. J., and Savarese, D. F., *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters*, MIT press, 1999.

Stone, H., "Estimation of Three-Phase Relative Permeability And Residual Oil Data," in: *PETSOC-73-04-06* (Oct. 1, 1973).

Tarman, M., Wang, K., Killough, J. E., and Sepehrnoori, K., "Automatic Decomposition for Parallel Reservoir Simulation," in: *SPE-141716-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2011.

Texas Advanced Computing Center, *Lonestar 5 User Guide*, 2017, URL: `https://portal.tacc.utexas.edu/user-guides/lonestar5`.

UTCOMP, *Technical Documentation for UTCOMP 3.8*, The University of Texas at Austin, 2003.

Van Daalen, D., Hoogerbrugge, P., Meijerink, J., and Zeestraten, R., "The Parallelisation of Bosim, Shell's Black/Volatile Oil Reservoir Simulator," in: *ECMOR I-1st European Conference on the Mathematics of Oil Recovery*, 1989.

Wallis, J., Foster, J., and Kendall, R., "A New Parallel Iterative Linear Solution Method for Large-Scale Reservoir Simulation," in: *SPE-21209-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1991.

Wang, K., Liu, H., Luo, J., and Chen, Z., "A Multi-Continuum Multi-Phase Parallel Simulator for Large-Scale Conventional and Unconventional Reservoirs," in: *Journal of Natural Gas Science and Engineering* 33 (2016), pp. 483–496, URL: http://www.sciencedirect.com/science/article/pii/S1875510016303547.

Wang, P., Balay, S., Sepehrnoori, K., Wheeler, J. A., Abate, J., Smith, B., and Pope, G., "A Fully Implicit Parallel EOS Compositional Simulator for Large Scale Reservoir Simulation.," in: *SPE-51885-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1999.

Wang, P., Yotov, I., Wheeler, M. F., Arbogast, T., Dawson, C., Parashar, M., and Sepehrnoori, K., "A New Generation EOS Compositional Reservoir Simulator: Part I - Formulation and Discretization," in: *SPE-37979-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 1997.

Wang, Y. and Killough, J. E., "A New Approach to Load Balance for Parallel/Compositional Simulation Based on Reservoir-Model Overdecomposition," in: *SPE-163585-PA* (Apr. 1, 2014).

Wheeler, J. A. and Smith, R. A., "Reservoir Simulation on a Hypercube," in: *SPE-19804-PA* (Nov. 1, 1990).

Willmore, F., "Introduction to Parallel Computing," Texas Advanced Computing Center, The University of Texas at Austin, 2013.

Yu, S., Liu, H., Chen, Z. J., Hsieh, B., and Shao, L., "GPU-Based Parallel Reservoir Simulation for Large-Scale Simulation Problems," in: *SPE-152271-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2012.

Yuan, C., Delshad, M., and Wheeler, M. F., "Parallel Simulations of Commercial-Scale Polymer Floods," in: *SPE-132441-MS*, SPE: Society of Petroleum Engineers, Jan. 1, 2010.

Zhong, H., Liu, H., Cui, T., Wang, K., Yang, B., Yang, M., and Chen, Z., "A Parallel Thermal Reservoir Simulator on Distributed-Memory Supercomputers," in: *SPE-182379-MS*, SPE: Society of Petroleum Engineers, Oct. 25, 2016.