2009

Realtime reservoir characterization and beyond: cyber-infrastructure tools and technologies

Yaakoub Youssef El-Khamra
Louisiana State University and Agricultural and Mechanical College, yelkhamra@gmail.com

REALTIME RESERVOIR CHARACTERIZATION AND BEYOND: CYBER-INFRASTRUCTURE TOOLS AND TECHNOLOGIES

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Petroleum Engineering

in

The Department of Petroleum Engineering

by
Yaakoub Y. El Khamra
B.E. American University of Beirut, 2002
December 2009

Dedication

To my parents.

Acknowledgments

This thesis would not have been possible without the encouragement and support of my advisor, Professor Chris White, who challenged me to be a better engineer. It is a pleasure to thank Professors Mayank Tyagi, Richard Hughes and Shantenu Jha for teaching me more about engineering and science than I would have ever thought possible. This work was motivated and partly funded the Ubiquitous Computing and Monitoring System for Discovery and Management of Energy Resources (UCoMS project, Department of Energy award No. DE-FG02- 04ER46136). I would also like to acknowledge the support of my friends and colleagues at the Texas Advanced Computing Center; in particular Prof. Kent Milfeld, Dr. Carlos Rosales and Mr. Bob Garza for his patience while reviewing my manuscript . It is a pleasure to thank my friends and former colleagues at the Center for Computation & Technology CCT, the developers of some of the infrastructure I used in this work, as well as the faculty and personnel of the Craft and Hawkins Department of Petroleum Engineering for putting up with me through all of those years.

Table of Contents

List of Figures

Abstract

The advent of the digital oil field and rapidly decreasing cost of computing creates opportunities as well as challenges in simulation based reservoir studies, in particular, real-time reservoir characterization and optimization. One challenge our efforts are directed toward is the use of real-time production data to perform live reservoir characterization using high throughput, high performance computing environments. To that end we developed the required tools of parallel reservoir simulator, parallel ensemble Kalman filter and a scalable workflow manager.

When using this collection of tools, a reservoir modeler is able to perform large scale reservoir management studies in short periods of time. This includes studies with thousands of models that are individually complex and large, involving millions of degrees of freedom. Using parallel processing, we are able to solve these models much faster than we otherwise would on a single, serial machine. This motivated the development of a fast parallel reservoir simulator. Furthermore, distributing those simulations across resources leads to a smaller total time to completion by making use of distributed processing. This allows the development of a scalable high throughput workflow manager. Finally, with thousands of models, each with millions of degrees of freedom, we end up with a superfluity of model parameters. This translates directly to billions of degrees of freedom in the reservoir study. To be able to use the ensemble Kalman filter on these models, we needed to develop a parallel implementation of the ensemble Kalman filter.

This thesis discusses the enabling tools and technologies developed to address a specific problem: how to accurately characterize reservoirs, using large numbers of complex detailed models. For these characterization studies to be helpful in making production decisions, the time to solution must be feasible. To that end, our work is focused on developing and extending these tools, and optimizing their performance.

Nomenclature

$\boldsymbol{K}$	Permeability tensor
$\Delta S_{\max}$	The maximum permissible rate of change in saturation over the entire domain
Δt	Time step
Δt_p	Time step for the pressure solve
Δt_S	Time step for the saturation update
δ	Kronecker delta
γ_l	Specific gravity of phase l
$\lambda_l = \frac{kk_{rl}}{\mu_l B_l}$	Transmissibility for phase l
μ_l	Viscosity for phase l
ϕ	Porosity of the porous media
ϕ^0	Reference porosity for rock compressibility calculation
Ψ_k	ensemble of augmented state vectors at time step k
ρ_l	Density of phase l
$\tilde{J}$	Well index
ε_k	Observation errors at time step k
A_1, A_2, A_3	The cross sectional areas of the grid blocks in the x_1, x_2 and x_3 directions
A_k	State transition model which is applied to the previous state x_{k1}

B_l	Formation volume factor for phase l
c_R	Rock compressibility factor
$d_{obs,k}$	Observations at time step k
G	Gravitational constant
$g_{k,i}$	Simulated data from the model state $x_{k,i}$ at timestep k for ensemble member i
H	Measurement operator, to extracts the simulated data from the state vector y^f
h_1, h_2, h_3	Distance between grid block centers in the x_1, x_2 and x_3 directions
k_{11}, k_{22}, k_{33}	Principle directional components of the permeability tensor
k_{rl}	Relative permeability for phase l
N_e	Number of ensemble members
p^0	Reference pressure for rock compressibility calculation
P_{cog}	Capillary pressure for the oil–gas contact
P_{cow}	Capillary pressure for the oil–water contact
p_l	Pressure for phase l
q_l	Production/injection rate for phase l
R_e	Effective well radius
R_{SO}	Gas oil solubility
S_l	Saturation for phase l
t	Time
$T_{l,i}$	Transmissibility for phase l in direction i
w_k	Model errors at time step K
x_1, x_2, x_3	The three Cartesian axes
x_k	State vector at time step k

$x_{k,i}$	Vector containing variables for rock properties and flow system in every grid block for ensemble member i
BQP	Batch Queue Prediction
EnKF	Ensemble Kalman Filter
EnOpt	Ensemble optimization filter
EnRML	Ensemble Randomized Maximum Likelihood filter
IMPES	Implicit Pressure Implicit Saturation
SAGA	Simple API for Grid Applications

Chapter 1

Introduction

1.1 Motivation

Given a reservoir where the quantities of oil, gas and water, the locations and transmissivities of faults, porosity and permeability and the multiphase flow properties such as relative permeability and capillary pressure at all locations are known, it is conceptually possible to develop a mathematical model of the reservoir such that the outcome of any action can be predicted. In other words, if all the model variables are known, the outcome can be predicted (within tolerances of errors: formulaic, algorithmic, numeric), typically with the use of a reservoir simulator that solves partial differential equations numerically. This is called the *forward problem*.

In real life, we rarely have full, accurate knowledge of the reservoir. Direct observations are limited to the actual well locations that are sometimes thousands of feet apart. Indirect observations are typically made at the surface, either at the well head (production rates and pressures) or at distributed locations (e.g., seismic). In the *inverse problem*, the observations are used to determine the variables that describe the system. Our efforts are directed at solving the inverse problem using ensemble Kalman filters, thus obtaining a reliable reservoir characterization, and updating the reservoir characterization using real-time direct and indirect observations. Furthermore, with the new reservoir characterization, production parameters can be modified to increase performance. This is the *Closed Loop* (Chen 2007; Jansen et al. 2009; Chen, Oliver, and Zhang 2008; Wang, Li, and Reynolds 2007) scenario.

We developed a fast parallel reservoir simulator, a parallel ensemble Kalman filter and a high throughput workflow manager with sensor data integration. The parallel reservoir simulator allows us to run large (over 33 million grid points with **Defiant**) simulations in short periods of time. The parallel ensemble Kalman filter allows us to

Task	Cactus Tool Suite	PETSc Tool Suite
Reservoir Simulator	BlackOil	Defiant
EnKF: Gets data from ensemble of simulations, computes Kalman gain, updates model, moves updated parameters and state back to simulator	Blanc	Valiant
Workflow Controller: allocates space, queues jobs, evolves simulations, applies EnKF	Lazarus	Lazarus/Relentless
Sensor Communication	Rouge	Built-in Relentless
Primary File Format	HDF5 through Bleu	HDF5, NetCDF, PETSc native

Figure 1.1: Tools developed. These tools are used in reservoir characterization and the "Closed Loop" scenario

perform data assimilation of large numbers of ensembles of problems of that size. Finally, the workflow manager runs the simulations in parallel, distributed across many resources and optimized for a reduced total time to completion. Using these tools together (figure 1.1) we can run real time history matching and reservoir forecast studies involving large numbers of detailed, complex models.

We will start by covering the basic tools and technologies developed that will enable us to perform real-time reservoir characterization. The basic building blocks include a reservoir simulator, an ensemble Kalman filter and a workflow-manager. These tools are built on top of various computational frameworks, file format libraries, grid infrastructure libraries and tools.

1.2 The Reservoir Simulators: BlackOil and Defiant

Reservoir simulators are typically used to answer questions such as where to place production and injection wells, where to perforate the wells, how much oil will be produced and at what time is the reservoir depleted. Other uses for reservoir simulators include calculating the fraction of water produced (water cut) and how it varies over time, gas-oil ratios, productive capacities and well treatment/workover possibilities.

Black oil models (sometimes referred to as β-models) are reservoir models that typically contain three distinct phases: oleic, aqueous and vapor, and three components: oil, water and gas. Water and oil are assumed to be immiscible and do not exchange mass or change phase. Gas is assumed to be soluble in oil but usually not in water (Aziz and Settari 1979; Chen 2007). Furthermore, in black oil models, fluids are assumed to be at a constant temperature and in thermodynamic equilibrium throughout the reservoir (Aziz and Settari 1979; Chen 2007; Carlson 2003; Fanchi 2005).

We first developed a parallel reservoir simulator using the **Cactus Code** high performance scientific computing framework. The **BlackOil** reservoir simulator is a basic reservoir simulator that implements the implicit pressure explicit saturation (IMPES) formulation for two phases (oil and water). The focus in **BlackOil** is on simplicity, performance, scalability and portability rather than abstraction.

The second generation parallel reservoir simulator, **Defiant** (El-Khamra 2009a), is a **PETSc** (Balay et al. 2001) based reservoir simulator that is both a library of tools for developing reservoir simulators and a fully functional simulator. The **Defiant** platform is as a general purpose reservoir simulation platform that is abstract and extensible. Using **Defiant**, several reservoir simulators can be built with simple function calls; this includes IMPES 2 and 3 phase simulator and a fully implicit 2 and 3 phase Newton-Raphson solver.

1.3 The EnKF: Blanc and Valiant

Ensemble Kalman filters (EnKF) methods, Monte Carlo implementations of the Kalman filter method, are widely used in science and engineering (Evensen 2006; Kalman 2005). EnKF methods, first introduced by Evensen (Evensen 2006), are recursive filters that can be used to handle large, noisy data; the data can be the results and parameters of ensembles of models that are sent through the Kalman filter to obtain updated versions of the state, model, and observation vectors.

Running the EnKF involves two steps (Evensen 2006; Oliver and Liu 2008): the forecast where we run the forward model (or reservoir simulator) on an ensemble of initial models (that are generated consistently with prior knowledge) and subsequently assimilating actual reservoir production data.

Using abstract linear algebra tools from the **PETSc** (Balay et al. 2001) libraries, we implemented a parallel ensemble Kalman filter method for **BlackOil** inside the **Blanc** thorn and for **Defiant** in **Valiant** (El-Khamra 2009d). As is the case with **Defiant**, **Valiant** is implemented as both a library of routines for loading state vectors and observations from **Defiant** in parallel, observational error generation and utility routines as well as a parallel implementation of the ensemble Kalman filter. This allows us to modularly extend **Valiant** to include ensemble randomized maximum likelihood filters (EnRML; Oliver and Liu 2008) to handle stiff, non-linear data as well as ensemble optimization for production parameter optimization. **Valiant** also includes a single processor prototype EnRML code that is a precursor to yet–to–be–developed parallel EnRML code.

1.4 The Workflow Manager: Lazarus and Relentless

Lazarus (El-Khamra and Jha 2009; Jha et al. 2007; Lazarus3) is a small, portable framework written in the Python scripting language that handles job-launching and data migration. **Lazarus** uses the **BigJob** (Luckow et al. 2009) abstraction to launch reservoir simulations, the Simple API for Grid Applications: **SAGA** (Goodale and Shantenu 2005) file adapter Python bindings to move files from one machine to another and BQP (Brevik, Nurmi, and Wolski 2006) data from the BQP command line tool to retrieve the optimal location and number of big jobs required to satisfy the computation power required. Lazarus also has a self–healing ability that allows it to run a user–defined test on the data to make sure it is not corrupt, missing, or otherwise defective. If a simulation is found to be defective it is resubmitted to be run again. This self–healing or "resurrection" ability overcomes some of the errors encountered when running large numbers of jobs such as exceeding wall clock time, running out of disk space (resubmit on a different machine) and of course node hardware failure. **Lazarus** cannot resolve simulator failure caused by numerical problems or incorrect input data to the simulator. It will abort

the entire workflow when simulations fail repeatedly and flag these simulations for user inspection. This prevents wasting service units with faulty simulations.

Lazarus is under constant development and has recently been moved to a new workflow management platform: **Relentless** (El-Khamra 2009b). **Relentless**, is a scalable workflow manager, that can handle EnRML and EnOpt scenarios and is capable of launching forecast studies after the history matching is complete, including geothermal, gas shale and CO2 sequestration studies. It can also poll a sensor data spool (for example: a simple MySQL database; AB MySQL 1995) for the availability of new sensor data and perform data assimilation after its retrieval.

Chapter 2

The Reservoir Simulators

2.1 Applications of Reservoir Simulation

Correctly applied, reservoir simulators are powerful tools. Using reservoir simulators as predictive tools is now a standard in the oil industry (Mattax 1990; Carlson 2003). Reservoir simulators can be used to obtain performance predictions of formations under different operating conditions. Operating conditions and selected development strategies directly affect production, thereby revenue. Typically, hydrocarbon recovery projects involve capital investments. Since reservoirs are large complex systems with irregular geometries that are deep underneath the surface, it is not uncommon to have sporadic, uncertain, limited information about their structure and behavior (Holstein 2007). These uncertainties contribute to a risk factor associated with the investment made in recovering the hydrocarbon. Therefore reservoir simulators can be used as risk assessment tools to predict the behavior of the reservoir and optimize its performance (Fanchi 2005). Simulators can also be used in reservoir characterization by solving the inverse problem, before moving forward with the forecast. This improves our knowledge of the reservoir and reduces the uncertainty in its future behavior (Oliver and Liu 2008). Other uses for reservoir simulators (with extensions) include their use in CO2 sequestration studies, underground water movement simulations and geothermal energy.

2.2 BlackOil Reservoir Simulator

2.2.1 The CactusCode Framework

As mentioned earlier, the **BlackOil** reservoir simulator is based on the Cactus Code (Seidel 1999) framework. Cactus Code is a framework for high performance scientific computing designed for scientists and engineers to develop and run codes for solving complex problems. Developing code for high performance parallel machines has

many challenges including scalability, efficiency (for computation, communication and input/output), portability and flexibility. Frameworks such as Cactus allow scientists and engineers to develop modules which can then be used together with modules written by other researchers to solve complex computational problems. The framework provides tools ranging from basic computational building blocks to complete toolkits that can be used to solve complex problems in astrophysics, computational fluid dynamics or other disciplines. Tools developed in the Cactus Code framework run on a wide range of architectures including desktop PCs, supercomputers and computational Grids. Cactus and its associated toolkits are publicly available for download from the Cactus Code website (Seidel 1999).

From an architectural standpoint, the Cactus Code framework consists of a central part (the "flesh") and code modules ("thorns"). The flesh serves as a module manager, scheduling the routines of the thorns and passing data between thorns. Thorns perform tasks ranging from setting up the computational grid (e.g., the CartGrid3D thorn), decomposing the computational grid for parallel processing (e.g., the PUGH thorn), providing boundary conditions (e.g., the Boundary thorn) and so on.

While all the actual code in the **BlackOil** reservoir simulator lives in a set of thorns in Cactus, **BlackOilEvolve** and **IDBlackOil**, the routines that solve the linear system of equations in parallel are called from the PETSc (Balay et al. 2001) library. This gives **BlackOil** access to a large variety of Krylov subspace solvers (Saad 2003; Balay et al. 2001) and preconditioners that are otherwise unavailable natively in the Cactus Code framework.

Since the most evolved drivers in the Cactus Code are those of the structured Cartesian mesh, it was convenient to implement **BlackOil** using a finite difference approximation (Patankar 1980). More specifically, a second order accurate central difference scheme in space (for constant Δ_x), first order in time.

2.2.2 Numerical Discretization: Finite Difference

Finite difference methods (Patankar 1980) are numerical methods that solve partial differential equations. A full treatment of both theory and applications of finite differences can be found in numerical analysis references and numerical solution of differential equations (e.g., Forsythe and Wasow 1960, Lapipus and Seinfeld 1971, Hilderbrand 1968).

When solving partial differential equations, we seek to find a function (or some discrete approximation thereof) that satisfies certain constraints. These constraints can be relationships between its various derivatives on a given region of space or time, and/or behavior at the boundaries of the domain (Kreyszig 1998). The methods approximate the solution derivatives with approximate discretized difference quotients (Chapra and Canale 2000).

To approximate the first derivative which is, by definition:

$$f'(\alpha) = \lim_{h \to 0} \frac{f(\alpha + h) - f(\alpha)}{h} \tag{2.1}$$

we use a truncated Taylor polynomial that takes the form:

$$f'(\alpha) \approx \lim_{h \to 0} \frac{f(\alpha + h) - f(\alpha)}{h} \tag{2.2}$$

Equation 2.2 is a first order approximation of the first derivative, as the higher order terms of the Taylor polynomial are truncated. It is also a forward differencing scheme that uses the function f evaluated at $\alpha + h$. Using the same Taylor polynomial expansion, we can find second order accurate approximations to the first derivative:

$$f'(\alpha) \approx \lim_{h \to 0} \frac{f(\alpha + h) - f(\alpha - h)}{2h} \tag{2.3}$$

Equation 2.3 is a centered difference discretization as we use the function f evaluated at $\alpha + h$ and $\alpha - h$ but not at α.

The second order accurate, centered, second derivative of function f evaluated at α is given by:

$$f''(\alpha) \approx \lim_{h \to 0} \frac{f(\alpha + h) - 2f(\alpha) + f(\alpha - h)}{h^2} \tag{2.4}$$

This is the approximation we will use for discretizing the spatial second derivative of the pressure terms in the partial differential equations that govern the flow through the reservoir.

2.2.3 BlackOil Implementation: Discretization

The black oil (or β) model, used in both **BlackOil** and **Defiant**, assumes there are at most three distinct phases: oil, water and gas (Aziz and Settari 1979). We make the following assumptions:

- Oil has intermediate wettability
- Gas is the non–wetting phase
- Water and oil are immiscible and do not exchange mass or change phase
- Gas is soluble in oil but not in water
- All fluids are in constant temperature and in thermodynamic equilibrium

- Darcy's law applies

With the assumptions made so far, the governing equations for water, oil and gas are as follows:

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{\phi S_w}{B_w}\right) = & \\ & \frac{\partial}{\partial x_1}\left(T_{w1}\left(\frac{\partial P_w}{\partial x_1} - \gamma_w \frac{\partial z}{\partial x_1}\right)\right) \\ & + \frac{\partial}{\partial x_2}\left(T_{w2}\left(\frac{\partial P_w}{\partial x_2} - \gamma_w \frac{\partial z}{\partial x_2}\right)\right) \\ & + \frac{\partial}{\partial x_2}\left(T_{w3}\left(\frac{\partial P_w}{\partial x_3} - \gamma_w \frac{\partial z}{\partial x_3}\right)\right) \\ & + \tilde{q}_{Ws}\end{aligned} \tag{2.5}$$

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{\phi S_o}{B_o}\right) = & \\ & \frac{\partial}{\partial x_1}\left(T_{o1}\left(\frac{\partial P_o}{\partial x_1} - \gamma_o \frac{\partial z}{\partial x_1}\right)\right) \\ & + \frac{\partial}{\partial x_2}\left(T_{o2}\left(\frac{\partial P_o}{\partial x_2} - \gamma_o \frac{\partial z}{\partial x_2}\right)\right) \\ & + \frac{\partial}{\partial x_2}\left(T_{o3}\left(\frac{\partial P_o}{\partial x_3} - \gamma_o \frac{\partial z}{\partial x_3}\right)\right) \\ & + \tilde{q}_{Os}\end{aligned} \tag{2.6}$$

and

$$\frac{\partial}{\partial t}\left[\phi\left(\frac{S_g}{B_g}+\frac{R_{so}S_o}{B_o}\right)\right] = \frac{\partial}{\partial x_1}\left(T_{g1}\left(\frac{\partial P_g}{\partial x_1}-\gamma_g\frac{\partial z}{\partial x_1}\right)\right) + \frac{\partial}{\partial x_2}\left(T_{g2}\left(\frac{\partial P_g}{\partial x_2}-\gamma_g\frac{\partial z}{\partial x_2}\right)\right) + \frac{\partial}{\partial x_2}\left(T_{g3}\left(\frac{\partial P_g}{\partial x_3}-\gamma_g\frac{\partial z}{\partial x_3}\right)\right) + \frac{\partial}{\partial x_1}\left(R_{so}T_{o1}\left(\frac{\partial P_o}{\partial x_1}-\gamma_o\frac{\partial z}{\partial x_1}\right)\right) + \frac{\partial}{\partial x_2}\left(R_{so}T_{o2}\left(\frac{\partial P_o}{\partial x_2}-\gamma_o\frac{\partial z}{\partial x_2}\right)\right) + \frac{\partial}{\partial x_2}\left(R_{so}T_{o3}\left(\frac{\partial P_o}{\partial x_3}-\gamma_o\frac{\partial z}{\partial x_3}\right)\right) + \tilde{q}_{Gs} \tag{2.7}$$

the reservoir rates and standard rates (e.g., $\tilde{q}_{Ws}$) are related by:

$$\tilde{q}_{Ws} = \frac{q_{Ws}}{B_w} \tag{2.8}$$

$$\tilde{q}_{Os} = \frac{q_{Os}}{B_o} \tag{2.9}$$

$$\tilde{q}_{Gs} = \frac{q_{Gs}}{B_g} + \frac{q_{Os}R_{so}}{B_o} \tag{2.10}$$

the effective transmissibility of phase α is:

$$T_\alpha = \frac{k_{r\alpha}K}{\mu_\alpha B_\alpha} \tag{2.11}$$

Since **BlackOil** considers only two-phases, we restrict ourselves to working with equations 2.5 and 2.6. Also, in the **BlackOil** implementation of the two-phase IMPES formulation we follow the nomenclature and methodology in Aziz and Settari (Aziz and Settari 1979) very closely; however we solve for water pressure and saturation (S_w and P_w) as opposed to oil pressure and saturation (S_o and P_o). Therefore, the reduced set

of differential equations will have the following form:

$$\sum_i \frac{\partial}{\partial x_i}\left[\lambda_w\left(\frac{\partial p_w}{\partial x_i} - \gamma_w \frac{\partial z}{\partial x_i}\right)\right] = \frac{\partial}{\partial t}\left(\phi \frac{S_w}{B_w}\right) + q_w \tag{2.12}$$

$$\sum_i \frac{\partial}{\partial x_i}\left[\lambda_o\left(\frac{\partial p_o}{\partial x_i} - \gamma_o \frac{\partial z}{\partial x_i}\right)\right] = \frac{\partial}{\partial t}\left(\phi \frac{S_o}{B_o}\right) + q_o \tag{2.13}$$

The oil saturation is related to the water saturation through the saturation constraint: $S_w + S_o = 1$ and the oil pressure is related to the water pressure through the capillary pressure constraint: $P_{cow}(S_w) = P_o - P_w$, where the capillary pressure P_{cow} is a function of water saturation.

$$\begin{aligned}\frac{C_{2o}}{C_{1w}}\left[\Delta T_w\left(\Delta P^{n+1} - \gamma_w \Delta z\right)\right] + \left[\Delta T_o\left(\Delta P^{n+1} + \Delta P^n_{cow} - \gamma_o \Delta z\right)\right] \\ = \frac{C_{2o}}{C_{1w}} C_{1p} \Delta_t P^n + \frac{C_{2o}}{C_{1w}} C_{1p} Q_w + C_{2p} \Delta_t P^n + Q_o\end{aligned} \tag{2.14}$$

where

$$C_{1w} = \frac{V}{\Delta t}(\phi / B_w)^{n+1} \tag{2.15}$$

$$C_{2o} = \frac{V}{\Delta t}(\phi / B_o)^{n+1} \tag{2.16}$$

$$C_{1p} = \frac{V}{\Delta t}[(S_w \phi)^n b'_w + S^n_w b^{n+1}_w \phi'] \tag{2.17}$$

$$C_{2p} = \frac{V}{\Delta t}[(S_o \phi)^n b'_o + S^n_w b^{n+1}_o \phi'] \tag{2.18}$$

Since there are two equations relating the saturations of oil and water (equations 2.12 and 2.13) to their pressures, we can algebraically manipulate these equations to "cancel out" the saturations and obtain a single equation in the oil and water phase pressures; the oil pressure is subsequently eliminated using the oil-water capillary pressure. The resulting equation is equation 2.14.

This equation can be discretized using finite difference methods and solved implicitly for the water pressure with an iterative solver and preconditioner from the **PETSc** library. The saturation can be calculated by substituting the newly calculated pressure

back into the discretized form of:

$$\sum_i \frac{\partial}{\partial x_i}\left[\lambda_w\left(\frac{\partial p_w}{\partial x_i} - \gamma_w \frac{\partial z}{\partial x_i}\right)\right] = \frac{\partial}{\partial t}\left(\phi \frac{S_w}{B_w}\right) + q_w \tag{2.19}$$

Other physical/fluid property variables depend on the reservoir model parameters such as porosity (affected by pressure through rock compressibility), density/volume factor (affected by pressure), relative permeability (affected by saturation). Initial values for these variables are set in **IDBlackOil** and their updated values are calculated at every IMPES iteration.

2.2.4 Capability

The **BlackOil** reservoir simulator can handle regular, Cartesian–gridded, heterogeneous reservoirs. Currently it supports only two phases (oil and water). It can also handle multiply connected wells semi–implicitly and uses the directional, anisotropic Peaceman (Peaceman 1977) well model. Since **BlackOil** is based on Cactus it cannot easily make use of advanced **PETSc** features such as numerical Jacobian calculation. This makes implementing fully implicit Newton–Raphson methods a difficult, major undertaking.

An important feature of the **BlackOil** reservoir simulator is its extensibility. Using the Cactus Code computational framework allows us to organize the code base into modules which we can swap, add or modify with ease. A good example of this is the **Streamline** thorn, a streamline simulator based on **BlackOil** that shares most of the code base of **BlackOil** but adds a streamline based saturation updating routine.

Figure 2.1 is a simple visualization using MatLAB of the streamlines traced across a quarter of a five–point pattern reservoir simulated using **BlackOil**. The streamlines are traced using the velocity fields, computed from the pressure field. The water saturation (a nonlinear transform of time–of–flight, related via the slope of the fractional flow curve; Datta-Gupta and King 2007) is computed using functional analysis inside the **Streamline** thorn in Cactus.

2.2.5 Performance

In terms of performance, the **BlackOil** reservoir simulator has been designed to be a fast reservoir simulator, reflected by the optimized code, run–time switching of solvers and preconditioners, and optimized use of PETSc Mat, Vec and KSP functions to minimize memory copy overhead. However, since we use Cactus Code drivers for parallelization (which are incompatible with PETSc vectors) we are forced to make a direct copy (in most cases across processors) to and from Cactus arrays to PETSc arrays. This can get very costly, especially when it has to be done at every IMPES iteration.

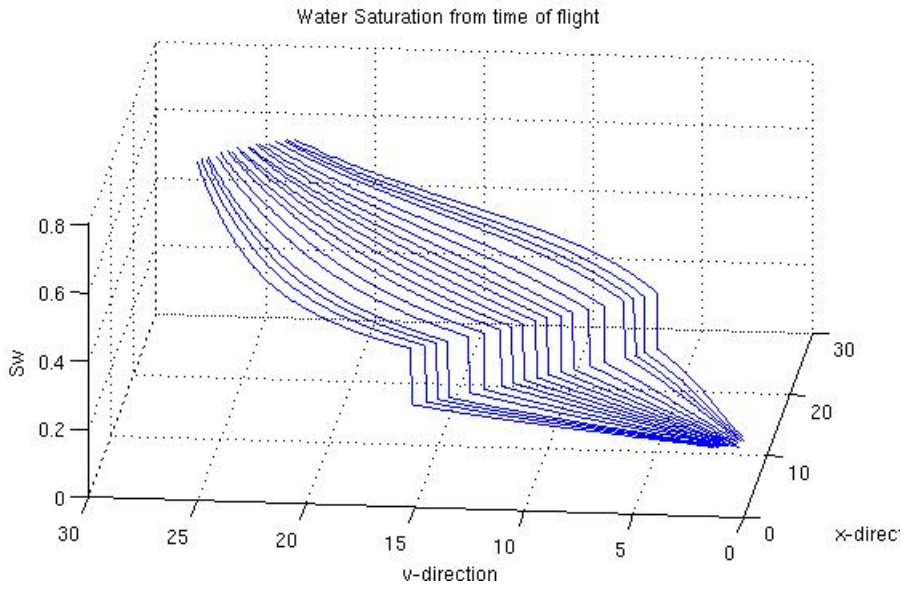

Figure 2.1: Streamline simulation water saturation plot. Water Saturation is derived from time of flight, calculated from **BlackOil** visualized with MatLAB

In addition to basic optimization, some OpenMP shared memory optimization pragmas were added throughout the **BlackOil** code. This added another layer of parallelism to the existing MPI (distributed memory) parallelism and enables hybrid parallel runs. The **Streamline** thorn was the first to receive the OpenMP (Chapman, Jost, and van der Pas 2007) treatment which resulted in slight (less than 10%) but promising performance gains. This was accomplished by adding multithreading support to the stream tracing routine and the time–of–flight update routine.

As mentioned earlier, **BlackOil** is a parallel reservoir simulator; this means that the execution time of the simulation is reduced significantly by running on more than just one processor. In the case of **BlackOil**, we have successfully run the code on more than a few thousand processors. Figure 2.2 shows the results of a weak scaling test with **BlackOil**. In a weak scaling test, the problem size increases with the number of processors. Ideally, the time to completion should be a flat line. Due to increasing time spent in communication between processors, the time to completion tends to increase. The operational limits exist in the memory copy to–and–fro with PETSc, which contributes to a large portion of the increase in time to completion. This places the performance of **BlackOil** as average amongst other PETSc based applications (Salawdeh et al. 2008; Anderson et al. 1999) and Cactus applications (Kamil, S. et al 2006).

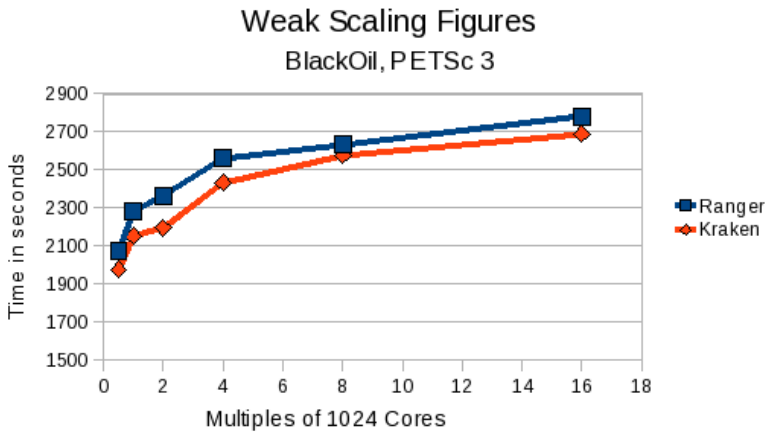

Figure 2.2: Weak scaling of **BlackOil**. These runs were made with a constant local problem size of 50x50x50 grid blocks, on Ranger and Kraken

2.3 Defiant

2.3.1 Governing Equations

Armed with experience from writing **BlackOil**, we set out to write an easy to understand, develop and maintain reservoir simulator in **Defiant**. This being the case, special care has been taken in making the code as readable and obvious as possible. To that end, we adopted the notation and formulation used by Chen (Chen 2007).

The differential form of the equations we are trying to solve is still the same as in **BlackOil**, effectively equations 2.5, 2.6, 2.7. In compact form:

$$\frac{\partial}{\partial t}(\frac{\phi S_w}{B_w}) = \nabla \cdot (\boldsymbol{T}_w[\nabla p_w - \gamma_w \nabla z]) + \frac{q_{Ws}}{B_w} \tag{2.20}$$

$$\frac{\partial}{\partial t}(\frac{\phi S_o}{B_o}) = \nabla \cdot (\boldsymbol{T}_o[\nabla p_o - \gamma_o \nabla z]) + \frac{q_{Os}}{B_o} \tag{2.21}$$

$$\begin{aligned}\frac{\partial}{\partial t}(\phi(\frac{S_g}{B_g} + \frac{R_{so}S_o}{B_o})) = \nabla \cdot (\boldsymbol{T}_g[\nabla p_w - \gamma_w \nabla z] \\ + R_{so}\boldsymbol{T}_o[\nabla p_o - \gamma_o \nabla z]) + \frac{q_{Gs}}{B_g} + \frac{R_{so}q_{Os}}{B_o}\end{aligned} \tag{2.22}$$

Where:

$$\boldsymbol{T}_\alpha = \frac{k_{r\alpha}}{\mu_\alpha B_\alpha}\boldsymbol{K}, \quad \alpha = w, o, g \tag{2.23}$$

and

$$\gamma_\alpha = \rho_\alpha G, \quad \alpha = w, o, g \tag{2.24}$$

Equations 2.20, 2.21 and 2.22 are in standard volumes. The volumetric flow rates at the wells are as follows:

$$q_{Ws} = \tilde{J}\frac{k_{rw}}{\mu_w}\left[p_{bh} - p_w - \gamma_w(z_{bh} - z)\right]\delta(\boldsymbol{x} - \boldsymbol{x}_{well}) \tag{2.25}$$

$$q_{Os} = \tilde{J}\frac{k_{ro}}{\mu_o}\left[p_{bh} - p_o - \gamma_o(z_{bh} - z)\right]\delta(\boldsymbol{x} - \boldsymbol{x}_{well}) \tag{2.26}$$

$$q_{Gs} = \tilde{J}\frac{k_{rg}}{\mu_g}\left[p_{bh} - p_g - \gamma_g(z_{bh} - z)\right]\delta(\boldsymbol{x} - \boldsymbol{x}_{well}) \tag{2.27}$$

where the well index W_{index} is given by:

$$\tilde{J} = \frac{2\pi h_1\sqrt{k_{22}k_{33}}}{ln(r_e/r_w) + s} \tag{2.28}$$

For wells in the x_1 direction. For wells in the x_2 and x_3 directions, the well indices are respectively:

$$\tilde{J} = \frac{2\pi h_2\sqrt{k_{11}k_{33}}}{ln(r_e/r_w) + s} \tag{2.29}$$

$$\tilde{J} = \frac{2\pi h_3\sqrt{k_{11}k_{22}}}{ln(r_e/r_w) + s} \tag{2.30}$$

The effective well radius r_e in the x_1, x_2 and x_3 directions is:

$$r_e = \frac{0.14((k_{33}/k_{22})^{1/2}h_2^2 + (k_{22}/k_{33})^{1/2}h_3^2)^{1/2}}{0.5((k_{33}/k_{22})^{1/4} + (k_{22}/k_{33})^{1/4})} \tag{2.31}$$

$$r_e = \frac{0.14((k_{33}/k_{11})^{1/2}h_1^2 + (k_{11}/k_{33})^{1/2}h_3^2)^{1/2}}{0.5((k_{33}/k_{11})^{1/4} + (k_{11}/k_{33})^{1/4})} \tag{2.32}$$

$$r_e = \frac{0.14((k_{22}/k_{11})^{1/2}h_1^2 + (k_{11}/k_{22})^{1/2}h_2^2)^{1/2}}{0.5((k_{11}/k_{22})^{1/4} + (k_{22}/k_{11})^{1/4})} \tag{2.33}$$

These equations (2.25, 2.26, 2.27) are known as the Peaceman model equations (Aziz and Settari 1979; Chen 2007) for anisotropic wells.

2.3.2 Two Phase IMPES

The approach in solving the two phase IMPES formulation of the black oil equations in **Defiant** is considerably different from the same formulation in **BlackOil**. In **Defiant** the oil pressure: p_o is the pressure we are solving for, and is used interchangeably with p in our equations. We follow the approach in (Chen 2007), starting with equations 2.20 and 2.21 and using the saturation and capillary pressure constraints:

$$S_w + S_o = 1, \quad p_{w,i,j,k} = p_{o,i,j,k} + p_{c,i,j,k} \tag{2.34}$$

We can eliminate the left hand sides and with some algebra arrive at the equation:

$$-\nabla\cdot(\boldsymbol{K}\lambda\nabla p) = \widetilde{q} - \nabla\cdot(\boldsymbol{K}(\lambda_w\nabla p_c + (\lambda_w\gamma_w + \lambda_o\gamma_o)\nabla z) \tag{2.35}$$

Using finite difference discretization and some algebraic manipulation we can derive the block–centered seven–point stencil scheme for the pressure equation (equation 2.35):

$$\lambda = \lambda_o + \lambda_w \tag{2.36}$$

$$\overline{\gamma} = \lambda_o\gamma_o + \lambda_w\gamma_w \tag{2.37}$$

$$\begin{aligned}
&-(\frac{A_1\lambda k_{11}}{h_1 B_o})_{i+1/2,j,k}(p_{i+1,j,k} - p_{i,j,k}) + (\frac{A_1\lambda k_{11}}{h_1 B_o})_{i-1/2,j,k}(p_{i,j,k} - p_{i-1,j,k}) \\
&-(\frac{A_2\lambda k_{22}}{h_2 B_o})_{i,j+1/2,k}(p_{i,j+1,k} - p_{i,j,k}) + (\frac{A_2\lambda k_{22}}{h_2 B_o})_{i,j-1/2,k}(p_{i,j,k} - p_{i,j-1,k}) \\
&-(\frac{A_3\lambda k_{33}}{h_3 B_o})_{i,j,k+1/2}(p_{i,j,k+1} - p_{i,j,k}) + (\frac{A_3\lambda k_{33}}{h_3 B_o})_{i,j,k-1/2}(p_{i,j,k} - p_{i,j,k-1}) \\
=\ &-(\frac{A_1\lambda_w k_{11}}{h_1 B_w})_{i+1/2,j,k}(p_{c,i+1,j,k} - p_{c,i,j,k}) + (\frac{A_1\lambda_w k_{11}}{h_1 B_w})_{i-1/2,j,k}(p_{c,i,j,k} - p_{c,i-1,j,k}) \\
&-(\frac{A_2\lambda_w k_{22}}{h_2 B_w})_{i,j+1/2,k}(p_{c,i,j+1,k} - p_{c,i,j,k}) + (\frac{A_2\lambda_w k_{22}}{h_2 B_w})_{i,j-1/2,k}(p_{c,i,j,k} - p_{c,i,j-1,k}) \\
&-(\frac{A_3\lambda_w k_{33}}{h_3 B_w})_{i,j,k+1/2}(p_{c,i,j,k+1} - p_{c,i,j,k}) + (\frac{A_3\lambda_w k_{33}}{h_3 B_w})_{i,j,k-1/2}(p_{c,i,j,k} - p_{c,i,j,k-1}) \\
&-(\frac{A_1\lambda_o\gamma_o k_{11}}{h_1 B_o})_{i+1/2,j,k}(z_{i+1,j,k} - z_{i,j,k}) + (\frac{A_1\lambda_o\gamma_o k_{11}}{h_1 B_o})_{i-1/2,j,k}(z_{i,j,k} - z_{i-1,j,k}) \\
&-(\frac{A_2\lambda_o\gamma_o k_{22}}{h_2 B_o})_{i,j+1/2,k}(z_{i,j+1,k} - z_{i,j,k}) + (\frac{A_2\lambda_o\gamma_o k_{22}}{h_2 B_o})_{i,j-1/2,k}(z_{i,j,k} - z_{i,j-1,k}) \\
&-(\frac{A_3\lambda_o\gamma_o k_{33}}{h_3 B_o})_{i,j,k+1/2}(z_{i,j,k+1} - z_{i,j,k}) + (\frac{A_3\lambda_o\gamma_o k_{33}}{h_3 B_o})_{i,j,k-1/2}(z_{i,j,k} - z_{i,j,k-1}) \\
&-(\frac{A_1\lambda_w\gamma_w k_{11}}{h_1 B_w})_{i+1/2,j,k}(z_{i+1,j,k} - z_{i,j,k}) + (\frac{A_1\lambda_w\gamma_w k_{11}}{h_1 B_w})_{i-1/2,j,k}(z_{i,j,k} - z_{i-1,j,k}) \\
&-(\frac{A_2\lambda_w\gamma_w k_{22}}{h_2 B_w})_{i,j+1/2,k}(z_{i,j+1,k} - z_{i,j,k}) + (\frac{A_2\lambda_w\gamma_w k_{22}}{h_2 B_w})_{i,j-1/2,k}(z_{i,j,k} - z_{i,j-1,k}) \\
&-(\frac{A_3\lambda_w\gamma_w k_{33}}{h_3 B_w})_{i,j,k+1/2}(z_{i,j,k+1} - z_{i,j,k}) + (\frac{A_3\lambda_w\gamma_w k_{33}}{h_3 B_w})_{i,j,k-1/2}(z_{i,j,k} - z_{i,j,k-1}) \\
&+\widetilde{Q}_{w,i,j,k} + \widetilde{Q}_{o,i,j,k}
\end{aligned} \tag{2.38}$$

As is evident from equation 2.38 we need to interpolate properties at the block–faces. To interpolate relative permeabilities, we use two–point upstream weighting:

$$k_{rw,i-1/2,j,k} = \begin{cases} (1+\beta_{i-1})k_{rw,i-1,j,k} - \beta_{i-1}k_{rw,i-2,j,k} & \text{if} \quad \Delta\Phi_{w,i-1/2,j,k} < 0 \\ (1+\beta'_i)k_{rw,i,j,k} - \beta'_i k_{rw,i+1,j,k} & \text{if} \quad \Delta\Phi_{w,i-1/2,j,k} > 0 \end{cases} \tag{2.39}$$

$$\beta_{i-1} = h_{i-1}/(2h_{i-3/2}) \tag{2.40}$$

$$\beta'_i = h_i/(2h_{i+1/2}) \tag{2.41}$$

Similar expressions can be derived for all grid–block faces and all phases. The two–point upstream weighting is second order accurate, and gives sharp solutions to fronts on a small number of grid–blocks (Chen 2007).

The (kA/h) term is treated as a harmonic (series) mean:

$$(\frac{A_1 k_{11}}{h_1})_{i\pm1/2,j,k} = \frac{2(A_1k_{11})_{i,j,k}(A_1k_{11})_{i\pm1,j,k}}{(A_1k_{11})_{i,j,k}(h_1)_{i\pm1,j,k} + (A_1k_{11})_{i\pm1,j,k}(h_1)_{i,j,k}} \tag{2.42}$$

and similar expressions can be derived for the x_2 and x_3 directions.

The density by viscosity term: $(\rho/\ \mu)$ is interpolated using pore volume fraction as follows:

$$\beta = \frac{\phi_{i,j,k}h_{1,i,j,k}h_{2,i,j,k}h_{3,i,j,k}}{(\phi_{i,j,k}h_{1,i,j,k}h_{2,i,j,k}h_{3,i,j,k} + \phi_{i\pm1,j,k}h_{1,i\pm1,j,k}h_{2,i\pm1,j,k}h_{3,i\pm1,j,k})}$$
$$(\frac{\rho}{\mu})_{i\pm1/2,j,k} = \beta(\frac{\rho}{\mu})(p_{i,j,k}) + (1-\beta)(\frac{\rho}{\mu})(p_{i\pm1,j,k}) \tag{2.43}$$

This is a weighted evaluation described in Chen 2007. We use compressibility equations for all fluids and the porous media

$$\phi = \phi^0(1 + c_R(p - p^0)) \tag{2.44}$$

The implicit solution of the pressure equation is typically the most expensive operation (Chen 2007). Since the pressure changes in porous media are less rapid than saturation changes, we can take a larger time step for pressure than we do for saturation. In **Defiant**, this is implemented as an adaptive time step improved IMPES (Chen 2007). The formulation is straightforward: for a positive integer N let $0 = t^0 < t^1 < t^2 < \ldots < t^N = T$ be the time domain partitions for the pressure solve. Let time partition: $J^n = [t^n, t^{n+1}]$ be the time partition for the saturation updates, and let J^n be partitioned into M saturation update partitions such that $J^{n,0} \rightarrow J^{n,M}$ span J^n, i.e. at time $t^{n,m}$ we are operating with the t^n pressure solution and the mth saturation update.
In improved adaptive–time step IMPES, the aim is to calculate $\delta t_s^{n,m+1}$ such that:

$$(\frac{\partial S^{n,m}}{\partial t})_{\max} = \max_{i,j,k}(\frac{S^{n,m+1} - S^{n,m}}{\delta t}) < \Delta S_{\max} \tag{2.45}$$

where $\Delta S_{\max}$ is the maximum permissible rate of change in saturation over the entire

domain. Therefore:

$$\delta t^{n,m+1} = \Delta S_{\max}/(S^{n,m+1} - S^{n,m}) \tag{2.46}$$

In terms of actual implementation, we use the saturation update equation to find the maximum rate of change of saturation across the domain, then divide the maximum permissible rate of change by that value. The resulting value would be the new $\delta t^{n,m+1}$ and we can use that to update the saturation. A more detailed analysis of the improved adaptive time–step IMPES formulation can be found in (Chen 2007).

2.3.3 Three Phase Newton–Raphson

Since **Defiant** was designed with extensibility and flexibility as primary features, it was an easy task to re–use most of the subroutines written for the two phase IMPES solver for a three phase IMPES formulation as well as two and three phase Newton–Raphson fully implicit solvers. Since we will be working with coupled nonlinear equations we chose to use a Newton–Raphson formulation (Kreyszig 1998; Chapra and Canale 2000; Chen 2007). The residuals are as follows:

$$\begin{aligned}
R^l_{w,i,j,k} = & \frac{1}{\delta t}(V[(\frac{\phi S_w}{B_w})^l - (\frac{\phi S_w}{B_w})^n])_{i,j,k} \\
& - T^l_{w1,i+1/2,j,k}(p^l_{w,i+1,j,k} - p^l_{w,i,j,k}) - T^l_{w1,i-1/2,j,k}(p^l_{w,i,j,k} - p^l_{w,i-1,j,k}) \\
& - T^l_{w2,i,j+1/2,k}(p^l_{w,i,j+1,k} - p^l_{w,i,j,k}) - T^l_{w2,i,j-1/2,k}(p^l_{w,i,j,k} - p^l_{w,i,j-1,k}) \\
& - T^l_{w3,i,j,k+1/2}(p^l_{w,i,j,k+1} - p^l_{w,i,j,k}) - T^l_{w3,i,j,k-1/2}(p^l_{w,i,j,k} - p^l_{w,i,j,k-1}) \\
& - (T_w\gamma_w)^l_{1,i+1/2,j,k}(z_{w,i+1,j,k} - z_{w,i,j,k}) - (T_w\gamma_w)^l_{1,i-1/2,j,k}(z_{w,i,j,k} - z_{w,i-1,j,k}) \\
& - (T_w\gamma - w)^l_{2,i,j+1/2,k}(z_{w,i,j+1,k} - z_{w,i,j,k}) - (T_w\gamma_w)^l_{2,i,j-1/2,k}(z_{w,i,j,k} - z_{w,i,j-1,k}) \\
& - (T_w\gamma - w)^l_{3,i,j,k+1/2}(z_{w,i,j,k+1} - z_{w,i,j,k}) - (T_w\gamma_w)^l_{3,i,j,k-1/2}(z_{w,i,j,k} - z_{w,i,j,k-1}) \\
& - \widetilde{Q}^l_{Ws,i,j,k}
\end{aligned} \tag{2.47}$$

$$
\begin{aligned}
R^l_{o,i,j,k} = \frac{1}{\delta t}(V[(\frac{\phi S_o}{B_o})^l - (\frac{\phi S_o}{B_o})^n])_{i,j,k} \\
&- T^l_{o1,i+1/2,j,k}(p^l_{o,i+1,j,k} - p^l_{o,i,j,k}) - T^l_{o1,i-1/2,j,k}(p^l_{o,i,j,k} - p^l_{o,i-1,j,k}) \\
&- T^l_{o2,i,j+1/2,k}(p^l_{o,i,j+1,k} - p^l_{o,i,j,k}) - T^l_{o2,i,j-1/2,k}(p^l_{o,i,j,k} - p^l_{o,i,j-1,k}) \\
&- T^l_{o3,i,j,k+1/2}(p^l_{o,i,j,k+1} - p^l_{o,i,j,k}) - T^l_{o3,i,j,k-1/2}(p^l_{o,i,j,k} - p^l_{o,i,j,k-1}) \\
&- (T_o\gamma_o)^l_{1,i+1/2,j,k}(z_{o,i+1,j,k} - z_{o,i,j,k}) - (T_o\gamma_o)^l_{1,i-1/2,j,k}(z_{o,i,j,k} - z_{o,i-1,j,k}) \\
&- (T_o\gamma_o)^l_{2,i,j+1/2,k}(z_{o,i,j+1,k} - z_{o,i,j,k}) - (T_o\gamma_o)^l_{2,i,j-1/2,k}(z_{o,i,j,k} - z_{o,i,j-1,k}) \\
&- (T_o\gamma_o)^l_{3,i,j,k+1/2}(z_{o,i,j,k+1} - z_{o,i,j,k}) - (T_o\gamma_o)^l_{3,i,j,k-1/2}(z_{o,i,j,k} - z_{o,i,j,k-1}) \\
&- \widetilde{Q}^l_{Os,i,j,k} \qquad (2.48)
\end{aligned}
$$

$$
\begin{aligned}
R^l_{o,i,j,k} = \frac{1}{\delta t}(V[(\frac{\phi S_g}{B_g} + \frac{R_{so}\phi S_o}{B_o})^l - (\frac{\phi S_g}{B_g} + \frac{R_{so}\phi S_o}{B_o})^n])_{i,j,k} \\
&- T^l_{g1,i+1/2,j,k}(p^l_{g,i+1,j,k} - p^l_{g,i,j,k}) - T^l_{g1,i-1/2,j,k}(p^l_{g,i,j,k} - p^l_{g,i-1,j,k}) \\
&- T^l_{g2,i,j+1/2,k}(p^l_{g,i,j+1,k} - p^l_{g,i,j,k}) - T^l_{g2,i,j-1/2,k}(p^l_{g,i,j,k} - p^l_{g,i,j-1,k}) \\
&- T^l_{g3,i,j,k+1/2}(p^l_{g,i,j,k+1} - p^l_{g,i,j,k}) - T^l_{g3,i,j,k-1/2}(p^l_{g,i,j,k} - p^l_{g,i,j,k-1}) \\
&- (T_g\gamma_g)^l_{1,i+1/2,j,k}(z_{g,i+1,j,k} - z_{g,i,j,k}) - (T_g\gamma_g)^l_{1,i-1/2,j,k}(z_{g,i,j,k} - z_{g,i-1,j,k}) \\
&- (T_g\gamma_g)^l_{2,i,j+1/2,k}(z_{g,i,j+1,k} - z_{g,i,j,k}) - (T_g\gamma_g)^l_{2,i,j-1/2,k}(z_{g,i,j,k} - z_{g,i,j-1,k}) \\
&- (T_g\gamma_g)^l_{3,i,j,k+1/2}(z_{g,i,j,k+1} - z_{g,i,j,k}) - (T_g\gamma_g)^l_{3,i,j,k-1/2}(z_{g,i,j,k} - z_{g,i,j,k-1}) \\
&- (R_{so}T_o)^l_{o1,i+1/2,j,k}(p^l_{o,i+1,j,k} - p^l_{o,i,j,k}) - (R_{so}T_o)^l_{o1,i-1/2,j,k}(p^l_{o,i,j,k} - p^l_{o,i-1,j,k}) \\
&- (R_{so}T_o)^l_{o2,i,j+1/2,k}(p^l_{o,i,j+1,k} - p^l_{o,i,j,k}) - (R_{so}T_o)^l_{o2,i,j-1/2,k}(p^l_{o,i,j,k} - p^l_{o,i,j-1,k}) \\
&- (R_{so}T_o)^l_{o3,i,j,k+1/2}(p^l_{o,i,j,k+1} - p^l_{o,i,j,k}) - (R_{so}T_o)^l_{o3,i,j,k-1/2}(p^l_{o,i,j,k} - p^l_{o,i,j,k-1}) \\
&- (R_{so}T_o\gamma_o)^l_{1,i+1/2,j,k}(z_{o,i+1,j,k} - z_{o,i,j,k}) - (R_{so}T_o\gamma_o)^l_{1,i-1/2,j,k}(z_{o,i,j,k} - z_{o,i-1,j,k}) \\
&- (R_{so}T_o\gamma_o)^l_{2,i,j+1/2,k}(z_{o,i,j+1,k} - z_{o,i,j,k}) - (R_{so}T_o\gamma_o)^l_{2,i,j-1/2,k}(z_{o,i,j,k} - z_{o,i,j-1,k}) \\
&- (R_{so}T_o\gamma_o)^l_{3,i,j,k+1/2}(z_{o,i,j,k+1} - z_{o,i,j,k}) - (R_{so}T_o\gamma_o)^l_{3,i,j,k-1/2}(z_{o,i,j,k} - z_{o,i,j,k-1}) \\
&- \widetilde{Q}^l_{Gs,i,j,k} \qquad (2.49)
\end{aligned}
$$

where the l subscript denotes the value at the lth Newton–Raphson iteration between

the n and $n+1$ time–steps. We define the unknown and residual vectors:

$$\boldsymbol{y} = (p, S_w, S_o)^T, \quad \boldsymbol{R}^l_{i,j,k} = (R^l_{w,i,j,k}, R^l_{o,i,j,k}, R^l_{g,i,j,k})^T \tag{2.50}$$

The superscript T indicates a transpose. The application of the Newton–Raphson iteration to equations: 2.47, 2.48 and 2.49, yields a linear system of equations in terms of $\delta \boldsymbol{y}^{l+1}$:

$$\begin{aligned}
&\frac{\partial \boldsymbol{R}^l_{i,j,k}}{\partial \boldsymbol{y}^l_{i,j,k-1}} \delta \boldsymbol{y}^{l+1}_{i,j,k-1} + \frac{\partial \boldsymbol{R}^l_{i,j,k}}{\partial \boldsymbol{y}^l_{i,j-1,k}} \delta \boldsymbol{y}^{l+1}_{i,j-1,k} \\
&+ \frac{\partial \boldsymbol{R}^l_{i,j,k}}{\partial \boldsymbol{y}^l_{i-1,j,k}} \delta \boldsymbol{y}^{l+1}_{i-1,j,k} + \frac{\partial \boldsymbol{R}^l_{i,j,k}}{\partial \boldsymbol{y}^l_{i+1,j,k}} \delta \boldsymbol{y}^{l+1}_{i+1,j,k} \\
&+ \frac{\partial \boldsymbol{R}^l_{i,j,k}}{\partial \boldsymbol{y}^l_{i,j+1,k}} \delta \boldsymbol{y}^{l+1}_{i,j+1,k} + \frac{\partial \boldsymbol{R}^l_{i,j,k}}{\partial \boldsymbol{y}^l_{i,j,k+1}} \delta \boldsymbol{y}^{l+1}_{i,j,k+1} = -\boldsymbol{R}^l_{i,j,k+1}
\end{aligned} \tag{2.51}$$

which is the block seven-point stencil in the increment $\delta \boldsymbol{y}^{l+1}$ and Jacobian:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} = \begin{pmatrix} \frac{\partial R_w}{\partial p} & \frac{\partial R_w}{\partial S_o} & \frac{\partial R_w}{\partial S_w} \\ \frac{\partial R_o}{\partial p} & \frac{\partial R_o}{\partial S_o} & \frac{\partial R_o}{\partial S_w} \\ \frac{\partial R_g}{\partial p} & \frac{\partial R_g}{\partial S_o} & \frac{\partial R_g}{\partial S_w} \end{pmatrix} \tag{2.52}$$

We solve the following system of equations for $\delta \boldsymbol{y}^{l+1}$:

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{y}} \delta \boldsymbol{y} = -\boldsymbol{R} \tag{2.53}$$

The solution to the system of equations is updated:

$$\boldsymbol{y}^{l+1} = \boldsymbol{y}^l + \delta \boldsymbol{y}^{l+1} \tag{2.54}$$

until $\boldsymbol{y}^{l+1}$ is equal to $\boldsymbol{y}^l$. In **Defiant** we use a matrix coloring routine (Balay et al. 2001; Hossaina and Steihaug 2004), modified from a built–in finite difference coloring routine, to compute the Jacobian numerically. While computationally more expensive than a symbolic calculation of the Jacobian, it allows us to handle table–based interpolated values for capillary pressures and relative permeabilities.

2.3.4 Approach: PETSc

PETSc (Balay et al. 2001), the "Portable Extensible Toolkit for Scientific Computing" is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. PETSc contains scalable parallel preconditioners, Krylov subspace solvers, parallel Newton–based nonlinear solvers, parallel time–stepping (ODE) solvers as well as parallel linear algebra routines and data structures. Since PETSc solvers and data structures use the standard message passing interface (MPI) for parallel computing allowing it to scale well on thousands of processors, it made a logical choice as an underlying infrastructure for a high performance reservoir simulator.

PETSc supports distributed meshes and arrays: parallel data–structures that are very useful for geometric domain–decomposition (divide and conquer) based parallelism. Therefore we made a design decision to exclusively use PETSc building blocks in **Defiant**. PETSc also contains many vector, matrix and array routines that allow us to write dense, short code that is very close in formulation to the reference material (Chen 2007) that is easy to read, understand and maintain. Finally, PETSc also contains interfaces to other packages that provide everything from algebraic multigrid (AMG through BoomerAMG and HYPRe; Falgout, Jones, and Yang 2006) to visualization through VTK (Moore 1998).

Extensibility, flexibility and scalability are at the core of **Defiant**, as well as abstractions. As PETSc internal functionality improves (solvers, preconditioners, data structures), the improvements will percolate to **Defiant** without any code modification. This includes graph based matrix coloring algorithms (Hossaina and Steihaug 2004; Balay et al. 2001) for efficient Jacobian calculations (as opposed to our modified finite difference coloring routine). The coloring routines assign the same color to columns of a matrix that do not share rows (i.e. have no non–zero elements in that row; Hwang and Cai 2005). The Jacobian terms of columns with the same color can be computed simultaneously, which is faster than computing the Jacobian element by element across the entire matrix.

Using PETSc native distributed meshes and arrays removes the performance issue of memory copies to and from solver routines that **BlackOil** suffered from: the PETSc distributed meshes and arrays are fully compatible with the matrix and vector data structures that we use for solving the linear system of equations. Those are the same parallel/distributed matrix and vector data structures that we use in the parallel implementation of the ensemble Kalman filter thus unifying communication between ensembles and the EnKF implementation.

Common file formats, memory to memory copies and data streaming/fetching are possible data transfer options. We can pass vector pointers from ensemble members to the EnKF for direct memory access, provide local files or remote files (compressed

or uncompressed) through ftp/http as well as stream data through sockets. With this variety of options, it is possible to perform history matching studies in various modes of simulation coupling: tight, medium and loosely coupled simulations and varied levels of distribution: local, shared file–system and remote.

2.3.5 Capability

As it stands, **Defiant** has all the functionality required to solve 2 and 3 phase black oil systems, through both IMPES and fully implicit Newton–Raphson formulations. **Defiant** also has improved adaptive time–step IMPES, linearized coupled well systems, monitoring well, coupling hooks to hydraulic well models (more on that in the future work section), as well as coupled reservoir simulation capabilities. The object–oriented design allows us to couple two or more reservoir simulations into one larger system that can be solved implicitly through any common parameter.

In short, **Defiant** is:

- An abstract, extensible, parallel, scalable, high performance reservoir simulation platform
- Supports non–uniform geometry, possible extension to adaptive mesh refinement through SAMRAI (Wissink et al. 2001)
- Has advanced interpolation routines: two–point weighted upstreaming for relative permeability to resolve sharp features with a low number of grid blocks
- Contains several built–in example simulators: 2/3 phase adaptive time–step improved IMPES and 2/3 fully implicit Newton Raphson simulators
- Numerically computes the Jacobian allowing full nonlinear treatment of table–based PVT relations
- Includes directional, anisotropic, non–uniform Peaceman well models, extensible to other well models with hooks to hydraulic models
- Supports geometric multigrid with current restrictions on well and perforation handling
- Supports algebraic multigrid with the BoomerAMG preconditioner in HYPRe (Falgout, Jones, and Yang 2006)
- Supports implicit treatment of perforated grid blocks connected through wells, which should allowthe incorporation of fish–spine and multi–lateral wells (Thuwaini, Shenawi, and Yuen 2009)

- Has built in support for tables including PVT tables and capillary–pressure and saturation tables
- Includes several 2 and 3 phase relative permeability models: Corey, Narr and Henderson, Narr and Wygal, Stones models I and II (Chen 2007)
- Contains the computational kernel to compute the Jacobian for compositional flow, and work is progressing on finishing the compositional model support
- Uses PETSc vector data structures and distributed arrays, which allows for fast local and remote IO, checkpointing and memory–to–memory copies as well as full compatibility with the parallel EnKF in **Valiant**
- Follows the exact variable naming notation in Chen's "Reservoir Simulation: Mathematical Techniques in Oil Recovery" (Chen 2007), making the book the effective manual and documentation for the code
- Provides support to coupling analysis simulations: coupling to the OpenFOAM (OpenCFD Corporation 2000) computational fluid dynamics package for CFD well modelling and to OpenLB (Latt 2006) for lattice Boltzmann simulations of perforations, gravel–packs, and sand control(more on this in the future work section)

This is not to say **Defiant** does not have limitations. We have a very high number of collective operations in MPI, which is a major bottleneck that needs to be addressed. **Defiant** cannot support unstructured meshes, it needs an outer layer of boundary grid points that increases as the problem size increases, wasting valuable memory space. This cost will increase significantly when we start working with multicomponent systems. **Defiant** also does not have any initial data generation routines, nor does it have any aquifer support models or complex boundary conditions. Those are all missing features that are being implemented.

2.3.6 Validation

To validate **Defiant** we ran a series of canonical cases, including a convergence study of the Buckley–Leverett problem (Buckley and Leverett 1942) with varying grid block numbers and time–steps. The same experiment was conducted in the x_1, x_2 and x_3 directions (after turning off gravity) to ensure no programming errors or index errors were introduced and that the formulation is consistent in all directions. The results in figure 2.3 show excellent convergence as well as the ability of the two–point upstream weighting interpolation of relative permeability to resolve the sharp front. This confirms that we get the same answer as we change the number of grid blocks and change the time–step. This is important as we run larger and more complex problems. Figure 2.3

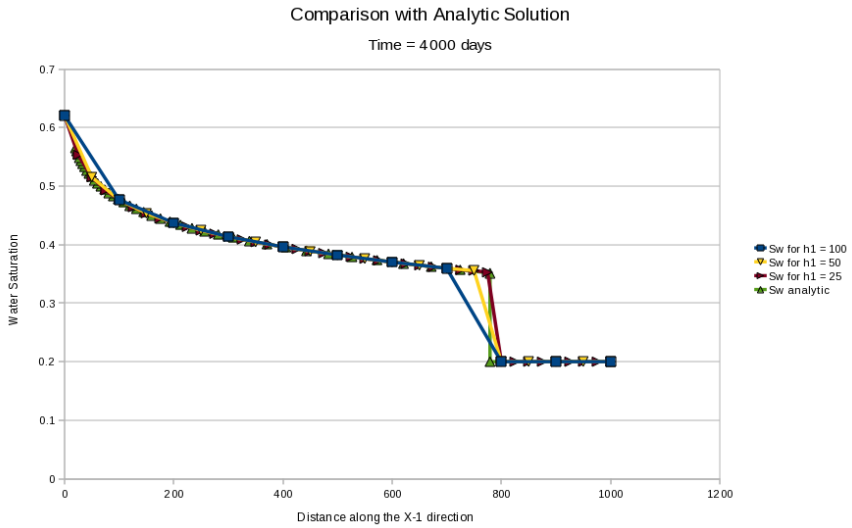

Figure 2.3: Buckley–Leverett convergence test. This problem is solved using **Defiant** on 11, 21 and 41 grid blocks at $T_p = 4000$ days and compared against analytical solution

also includes the analytical solution for comparison. The analytical solution was kindly provided by Professor Chris White.

We also ran a quarter five–spot pattern reservoir in three planes x_1, x_2, x_1, x_3 and x_2, x_3. The results at different times are in figures 2.4, 2.5, 2.6.

To compare against a standard implementation, we ran "Test Case 1" from Aziz and Settari (Aziz and Settari 1979). This is a simple Buckley–Leverett problem of a reservoir 1000 ft long, 10,000 ft cross–sectional area, and 426.5 cubic feet injection and production rates. The water saturation profile for the 10 and 20 grid point problems at $T = 1500$ days are presented in figures 2.7 and 2.8 respectively. We managed to capture the correct front location using adaptive time step improved IMPES.

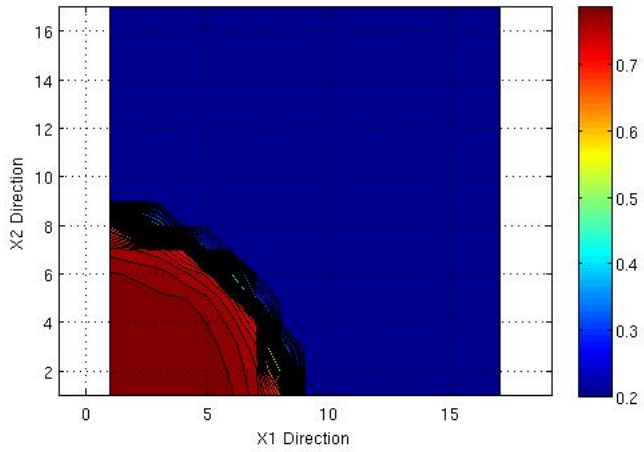

Figure 2.4: Five–spot pattern reserovir at T=15 days. Here we show the water saturation, captured well by the two–point upstream weighting method

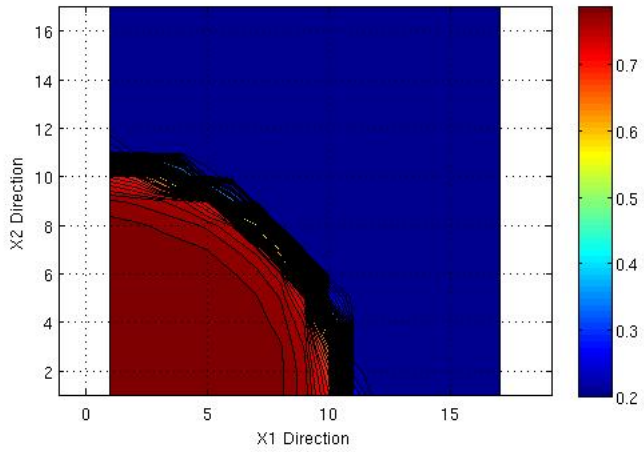

Figure 2.5: Five–spot pattern reserovir at T=25 days

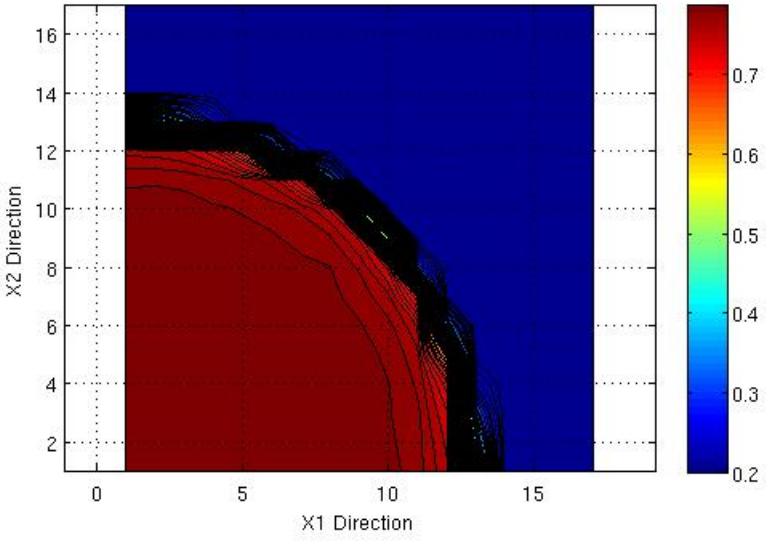

Figure 2.6: Five–spot pattern reserovir at T=40 days

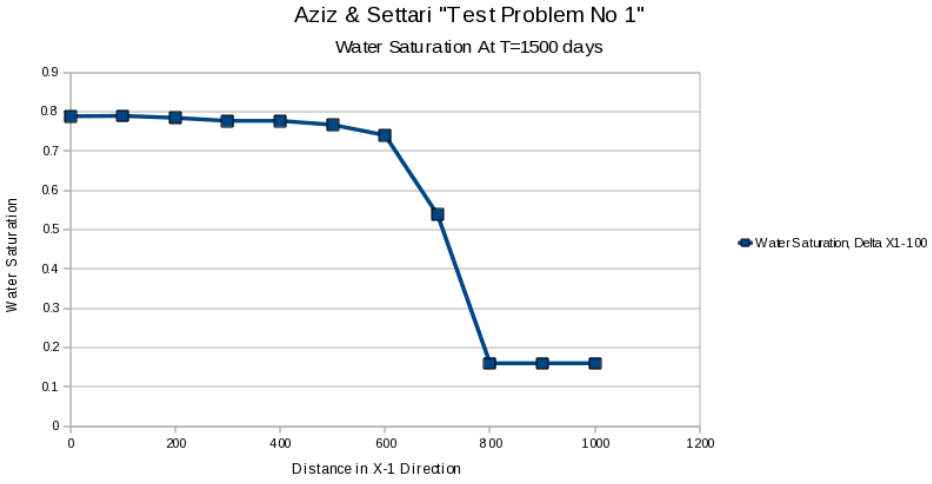

Figure 2.7: Test Case 1 from Aziz and Settari, 10 grid points

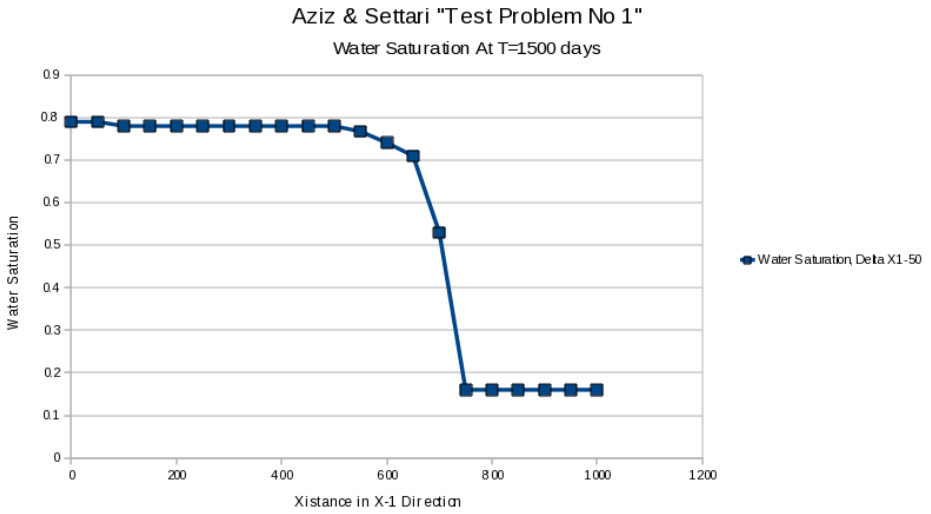

Figure 2.8: Test Case 1 from Aziz and Settari, 20 grid points

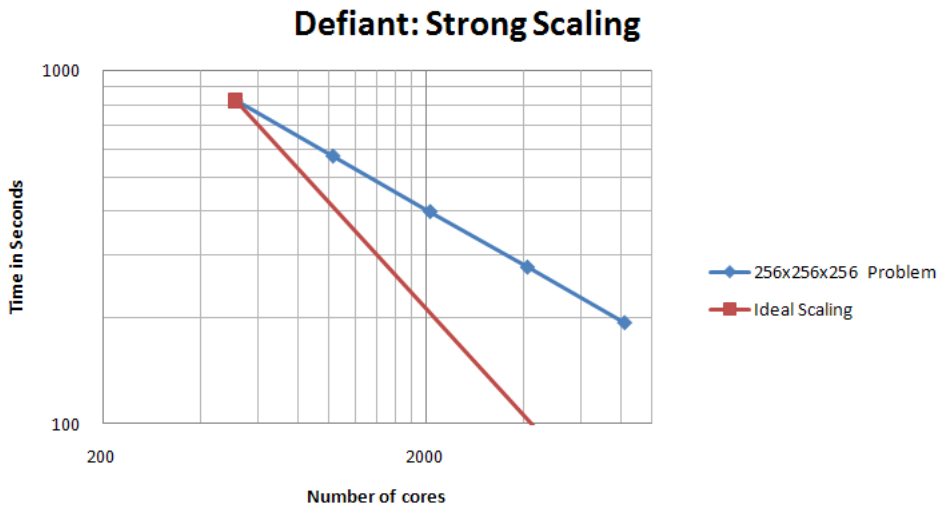

Figure 2.9: Strong Scaling for **Defiant**. These runs were made on Ranger with a 256x256x256 problem

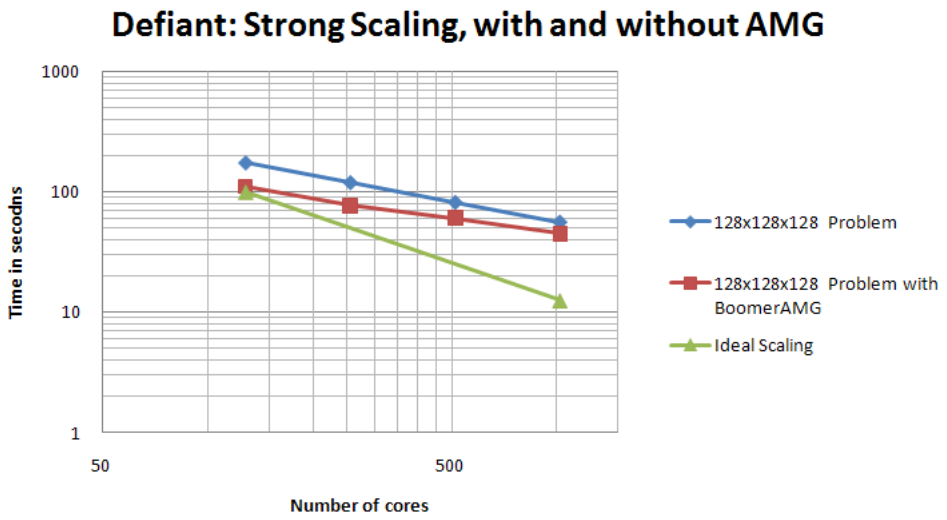

Figure 2.10: Strong Scaling for **Defiant** with AMG. These runs were made on Ranger with a 128x128x128 problem with and without algebraic multigrid

2.3.7 Performance

Early benchmarks of **Defiant** are promising. With an optimized compilation of PETSc we are able to scale up to 8192 cores without considerable difficulty. Figures 2.9 and 2.10 show the results of strong scaling benchmarks run on Ranger (TACC UT Austin 2006). In strong scaling benchmarks, the size of the problem remains constant as we increase the number of cores. Plotting the results on a log–log scale, the ideal scaling case would appear as a straight line with a -1 slope.

For these benchmarks, we used a block Jacobi preconditioner and a conjugate gradient Krylov subspace solver. We also tried using the HYPRe (Falgout, Jones, and Yang 2006) BoomerAMG preconditioner to solve the system of equations with an algebraic multigrid preconditioner and a Richardson iterative solver (figure 2.10). The performance improvements were considerable for low core counts but as the number of cores increased, the time spent in communication increased. Our experience thus far shows that BoomerAMG is best suited for smaller problems running on smaller number of cores.

There are some interesting statistics regarding communication: on a sample 256x256x256 problem run on 512 cores we encountered an inordinately high number of calls to the function MPI_Allreduce, a total of $2.71928e+07$ calls, making up 94.73% of the time spent in MPI communication and 27.24% of the total time in the simulation. While the

function MPI_Allreduce is optimized in the mvapich2 (MVAPICH Development Team, OHU 2002) implementation of MPI (MPI Forum Standards Committee 1994) used on Ranger (TACC UT Austin 2006), this many function calls is indicative of a bottleneck that will be investigated. We also noticed a relatively small number of grid points that we could fit in the 2GB/core: 33,554,432 in a 256x256x512 was the largest size problem we could fit in 1024 cores. Using memory profiling tools and event logging in PETSc, it should be possible to determine the cause of the increase in memory footprint.

2.4 Practical Considerations

A 100x100x100 grid point problem run for one iteration with **Defiant** takes 32.73 seconds to complete on 8 cores with Boomer algebraic multigrid and HYPRe. The same problem run with **BlackOil** takes 1 minute 19 seconds with a conjugate gradient solver and a block Jacobi preconditioner. **Defiant** has performance advantages over **BlackOil**, and as PETSc, Boomer and HYPRe improve, the **Defiant** will make use of these improvements with no changes to the source code, which is not the case with **BlackOil**. At the moment, **Defiant** has a larger memory footprint than **BlackOil**. This is another performance issue that will be addressed when the optimization development cycle starts.

Consider our 32.73 seconds per iteration 1 million grid point problem. Using a one day Δt_p, we can finish a one year simulation of the reservoir in 3.2 hours (in, for instance, an ensemble inversion or optimization method; EnKF, EnOpt, or EnRML) with **Defiant** (7.8 hours with **BlackOil**) running on 8 cores. Running 128 simulations concurrently with Lazarus, we would need 1024 cores. If Lazarus requests 1024 cores for 48 hours (a typical job duration on Ranger), it would be able to perform a history matching study of the reservoir over 14.8 years, or a forecast of 128 ensemble members for the next 14.8 years (provided we have assimilated all the previous historical data).

Consider a hypothetical scenario where we start with a ten day deadline before a development decision is due. To make this decision, detailed reservoir management studies spanning production forecast for the next fifteen years need to be performed. With **Defiant**, the reservoir modeler would finish the study in two days and have eight days for production optimization studies or certainty analysis. Using **BlackOil**, running the same fifteen years worth of forecasts will take 4.8 days, leaving slightly over five days for certainty analysis and optimization. If the reservoir modeler were to start from scratch and generate new models and perform history matching, of the past fifteen years in addition to the forecasts, a total of four days would be required with **Defiant** and with **BlackOil**, the deadline would be missed.

The few days saved when running a faster simulator give the reservoir modeler a substantial advantage in reservoir management and optimization, especially in cases where initial studies indicate the need for more detailed simulations, or for critical scenarios

where multiple studies need to be performed. These simulations will naturally be more detailed and require longer time to solution that would be only be available with fast reservoir simulators.

Chapter 3

The Ensemble Kalman Filter

3.1 Approach

The state of the model x_k can be described as a linear/linearized function of its previous state x_{k-1}. The state vector x_k consists of variables for rock and fluid properties in every grid block. This includes grid block permeabilities, porosities, saturations, pressures and PVT properties. $g(x_k)$ is the simulated data from the model state x_k. The measurements, d_{obs} are related to the state variables through $g(x_k)$. This is described in equation 3.1.

$$\begin{aligned} x_k &= A_k x_{k-1} + w_{k-1} \\ d_{obs} &= g(x_k) + \varepsilon_k \end{aligned} \tag{3.1}$$

In the case of a two phase reservoir, the state vector x_k is as follows:

$$x_k = [\phi, \ln(k_{11}), \ln(k_{22}), \ln(k_{33}), S_w, P_w] \tag{3.2}$$

The collection of ensemble state vectors can be placed into a single column of an ensemble matrix as follows:

$$\begin{aligned} \Psi_k &= \{y_{k,1}, y_{k,2}, y_{k,3}, \ldots, y_{k,N_e}\} \\ y_{k,i} &= [(x_{k,i})^T, g(x_{k,i})^T]^T \end{aligned} \tag{3.3}$$

In the forecast stage, we run the reservoir simulator. Effectively, we are advancing forward in time for all ensemble members:

$$y^f_{k,i} = f(y_{k-1,i}) \ \forall i \in \{1, \ldots, N_e\} \tag{3.4}$$

where $f(.)$ is the reservoir simulator.

In the data assimilation stage: we update the variables describing the state of the system to honor the observations:

$$y_j = y_j^f + K_e(d_j - Hy_j^f) \; \forall j \in \{1, \ldots, N_e\} \tag{3.5}$$

Where K_e is the ensemble Kalman gain, and H is the measurement operator that extracts the simulated data from the state vector y^f: $H_k = [0 \quad I]$. d_j is a vector of observational data that has been perturbed with random errors from the same distribution as the measurement error: $d_j = d_{obs} + \varepsilon_j \; \forall j \in \{1, \ldots, N_e\}$.

The ensemble Kalman gain is given by:

$$K_e = C_{\Psi,e} H^T (H C_{\Psi,e} H^T + C_D)^{-1} \tag{3.6}$$

To compute the Kalman gain matrix, we use the following formulation:

$$\begin{aligned} L_e &= \frac{1}{\sqrt{N_e - 1}} (\Psi - \bar{y}^f) \\ K_e &= L_e (HL_e)^T [(HL_e)(HL_e)^T + C_D]^{-1} \end{aligned} \tag{3.7}$$

There are at least two distinct approaches to implementing the data assimilation stage. The serial approach is where for all ensemble members, the state vector is read from disk, the mean vector updated and the HL_e matrix computed (this is done to reduce memory overhead of having to load all state vectors). The parallel approach treats the entire problem as a linear algebra problem that is written into C code using PETSc vectors and matrices, and PLAPACK (Alpatov et al. 1997) for the dense matrix inversion. Using a parallel layout for the state vectors, the observations, and all the matrices, allows us to load a larger number of significantly larger ensembles. The parallel implementation of the EnKF, the PEnKF, has been implemented in the **Blanc** thorn, and uses the **Bleu** thorn for IO. The **Bleu** thorn is also used by the **BlackOil** reservoir simulator for IO.

The second implementation of the PEnKF is the **Valiant** (El-Khamra 2009d) platform. Building on the success of the architecture in **Defiant**, we set out to implement a similar platform in **Valiant**: a library and an implementation with reusable modular components with flexibility and extensibility at its core, highest practical level of abstraction as well as strict adherence to a clear, concise notation and documentation. The reference material for **Valiant** in this case is "Inverse Theory for Petroleum Reservoir Characterization and History Matching" by Oliver and Liu 2008. **Valiant** contains functions to read state vectors and observations from **Defiant** (compatible with NetCDF and ADCIRC formats; Unidata 1984; Luettich and Scheffner 1992), functions to forward scatter and gather state vectors into ensemble columns and, of course, the scalable imple-

mentation of the parallel ensemble Kalman filter. In **Valiant**, the dense ensemble matrix is kept in RAM, (as opposed to read and written from and to disk), and is partitioned across cores column–wise and row–wise (equation 3.8).

$$\Psi_k = \left[\begin{array}{cc|cc} x_{0,procid=0} & x_{1,procid=0\cdots} & x_{N_e-1,procid=nprocs-4} & x_{N_e,procid=nprocs-4} \\ x_{0,procid=0} & x_{1,procid=0\cdots} & x_{N_e-1,procid=nprocs-4} & x_{N_e,procid=nprocs-4} \\ x_{0,procid=0} & x_{1,procid=0\cdots} & x_{N_e-1,procid=nprocs-4} & x_{N_e,procid=nprocs-4} \\ \hline x_{0,procid=1} & x_{1,procid=1\cdots} & x_{N_e-1,procid=nprocs-3} & x_{N_e,procid=nprocs-3} \\ x_{0,procid=1} & x_{1,procid=1\cdots} & x_{N_e-1,procid=nprocs-3} & x_{N_e,procid=nprocs-3} \\ \hline x_{0,procid=2} & x_{1,procid=2\cdots} & x_{N_e-1,procid=nprocs-2} & x_{N_e,procid=nprocs-2} \\ x_{0,procid=2} & x_{1,procid=2\cdots} & x_{N_e-1,procid=nprocs-2} & x_{N_e,procid=nprocs-2} \\ x_{0,procid=2} & x_{1,procid=2\cdots} & x_{N_e-1,procid=nprocs-2} & x_{N_e,procid=nprocs-2} \\ x_{0,procid=2} & x_{1,procid=2\cdots} & x_{N_e-1,procid=nprocs-2} & x_{N_e,procid=nprocs-2} \\ \hline x_{0,procid=3} & x_{1,procid=3\cdots} & x_{N_e-1,procid=nprocs-1} & x_{N_e,procid=nprocs-1} \\ x_{0,procid=3} & x_{1,procid=3\cdots} & x_{N_e-1,procid=nprocs-1} & x_{N_e,procid=nprocs-1} \\ \hline x_{0,procid=4} & x_{1,procid=4\cdots} & x_{N_e-1,procid=nprocs} & x_{N_e,procid=nproc} \end{array}\right] \tag{3.8}$$

In equation 3.8 we assemble the ensemble member state vectors x_k into the full ensemble matrix Ψ_k. For example, x_0 is the state vector of the first ensemble member and resides in the first column of the Ψ matrix. Ψ_k is then partitioned row–wise and column–wise across the available processors, leading to chunks of the ensemble state vector $x_{k,procid}$ residing on processors with rank *procid*. In our example x_0 is partitioned across processors 0,1,2,3 and 4.

Valiant has the capability to handle large systems containing millions of degrees of freedom. This should not be the sole motivation to assimilating large numbers of ensemble members frequently. According to Li (2008), the EnKF can become unstable when the error covariance is:

- Underestimated: magnitude of variances were reduced as more data were assimilated
- Overestimated: the observations are in a distance dependent manner where the noise (the error) of the observations is larger than the signal (magnitude of the covariance)

These issues can be addressed to some extent with covariance inflation and localization (Li 2008).

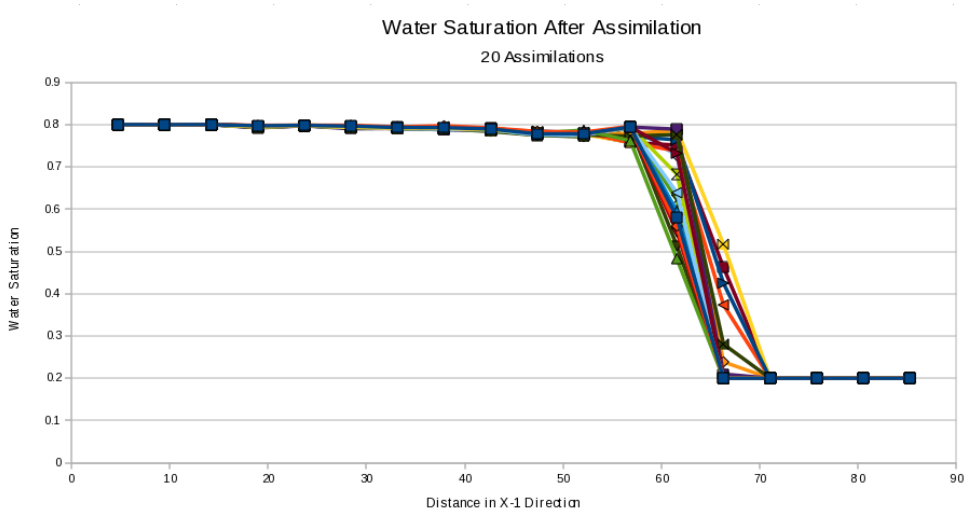

Figure 3.1: Water saturation after EnKF assimilation. These are the profiles of an ensemble of 20 members, with progressive observation data assimilation from the base case using **Valiant**

3.2 Test Case

To test **Valiant**, we conducted a small history matching study of a perturbed Buckley Leverett problem. Using a random number generator from the SPRNG (Mascagni and Srinivasan 2000) routines in PETSc, we added noise to the porosity and natural logarithm of permeability. We then proceed to run the ensembles along with the base case, assimilating observations from the base case every 10 iterations (Figure 3.1). For comparison we also run the original perturbed models without data assimilation (Figure 3.2).

Comparing figures (Figure 3.1) and (Figure 3.2), we can see a narrower spread across the front. In this test study, **Valiant** and **Defiant** were run in parallel on the Ranger supercomputer using **Relentless**. While the problem is not impressive in size or complexity (1090x20 degrees of freedom) , it paves the way to running larger, more complex problems now that the infrastructure is in place.

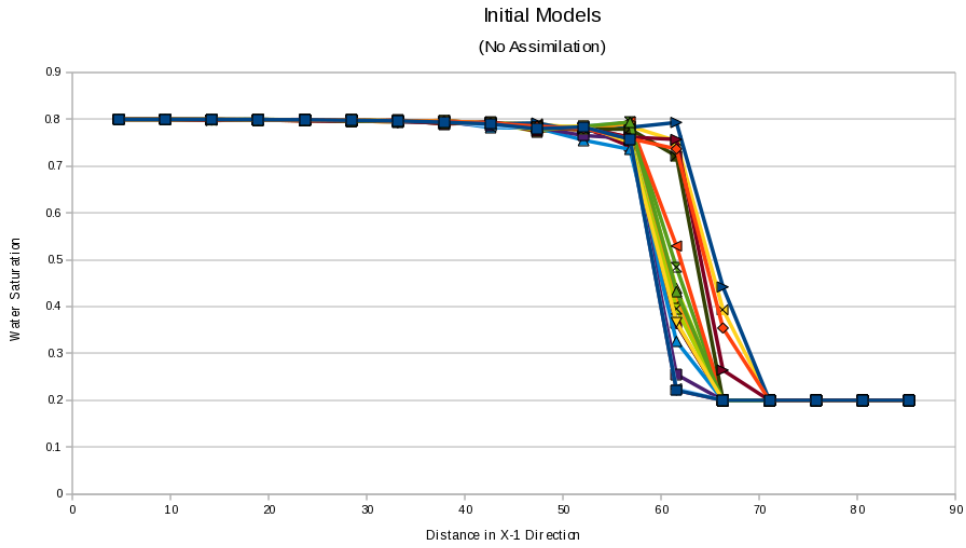

Figure 3.2: Water saturation without EnKF assimilation. These are the water saturation profiles of the original perturbed ensemble members

Chapter 4

The Workflow Manager

To submit simulations of ensemble members to supercomputing resources and collect their data for the data assimilation stage, we need a reliable job manager that is lightweight, portable and capable of handling various computational resources: grids, Condor pools and clouds. To that end we developed Lazarus.

Lazarus has advanced autonomic features such as built in fault tolerance: it checks the output files of the reservoir simulations for errors and resubmits failed jobs. It also has an integrated interface to the Batch Queue Prediction "BQP" tools. This allows Lazarus to find out which supercomputer/queue combination has the least waiting time for simulations of a given size and duration hence significantly reducing the total time to completion. Lazarus is also scalable: it has submitted tens of thousands of simulations to several supercomputers and Condor pools.

In its current form, Lazarus runs EnKF workflows, and will be integrated in a wider range of workflow management tools: **Relentless** (El-Khamra 2009b). **Relentless** is a set of common workflow scripts that include ensemble Kalman filter workflows, ensemble randomized maximum likelihood filter workflows, ensemble optimization workflows and ensemble reservoir operational lifetime workflows (CO2 sequestration studies). **Relentless** will also make use of various SAGA adapters including globus, Condor and cloud adapters

4.1 About SAGA

The Simple API for Grid Applications (SAGA) (Goodale and Shantenu 2005) is an API standardization effort within the Open Grid Forum (OGF) an international standards development body concerned primarily with standards for distributed computing. The specification and implementations of the SAGA API has been guided by detailed examination of the requirements expressed by existing and emerging distributed com-

puting applications in order to find common themes and motifs that can be reused across more than one use–case. The main governing design principle for the SAGA API is the 80:20 Rule: "Design an API with 20% complexity that serves 80% of the application use cases". They are intended to cover the most common application–level distributed computing programming constructs such as file transfer, and job management. In general, SAGA embodies the most commonly required features derived from a broad survey of the community and provides the most common grid programming abstractions that were identified by several use cases.

SAGA provides several capabilities that we use extensively through its Python bindings: the file transfer capability to move data files across machines and the job submission capability that we use to submit jobs to supercomputers. Lazarus (El-Khamra and Jha 2009) submits reservoir simulation jobs to supercomputers, collects the data files they produce on a single resource to run the EnKF, submits the EnKF job to be run, then launches the next stage of reservoir simulations.

4.2 Approach

To reliably launch hundreds of reservoir simulations and collect the data they produce repeatedly across different machines, we needed a fast, easy to use abstraction: this led to the development of the BigJob abstraction (Luckow et al. 2009). Lazarus builds upon earlier work, and extends the BigJob abstraction to explicitly use autonomic decision making based upon BQP, as well as fault–tolerance. This is the basis of the general purpose autonomic framework Lazarus.

After losing many jobs to various errors such as running out of allocated disk space, hardware failure and file–copy failure, we concluded that autonomy and self–healing are important features that we needed to incorporate in Lazarus to ensure all large scale history matching runs complete successfully. With fault tolerance, we no longer wasted thousands of Service Units (SU) running simulations with missing/corrupt ensemble members, as they are resurrected. On the other hand, the autonomy incorporated in Lazarus, while basic, ensures that we optimize the job sizes to have a higher chance of getting through the queue without delay.

For the current implementation, we use simple, portable scripts – based upon SAGA, to handle job–launching and data migration. Lazarus uses the BigJob abstraction to launch simulations, the SAGA file adapter Python bindings to move files from one machine to another and BQP data from a small Python wrapper around the BQP command line tool to retrieve the optimal location and number of BigJobs required to satisfy the computation power required (figure 4.1).

Lazarus's healing ability is based on user–defined tests on the data to make sure it is not corrupt, missing, or otherwise defective. If a simulation is found to be defective

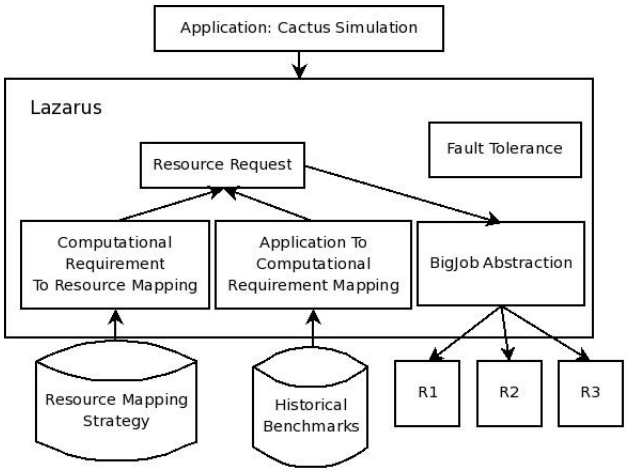

Figure 4.1: Lazarus architecture diagram. The "Resource Mapping" Strategy input is provided by BQP. Given the size of the individual tasks, the "Historical Benchmarks" help determine the size of the sub–jobs. (For the specific workload chosen, it so happens that the size of each sub–job is 16).

it is resubmitted to be run again. This self resurrection ability overcomes some of the catastrophic errors encountered when running large numbers of jobs such as exceeding wall clock time, running out of disk space (resubmit on a different machine) and of course node hardware failure.

4.3 Control Flow

Figure 4.2 outlines a typical EnKF history matching study managed by Lazarus. To prepare for launching Lazarus, a model generator is required to create the initial data (initial state vectors) for all the ensemble members. This is typically done before running Lazarus, and performed on one resource then synchronized against all others to ensure consistency.

Before launching Lazarus, some input parameters need to be specified. These are the executable names, working root directory, simulation directories, number of simulations per stage, number of stages, simulation size and so on. The current implementation supports varying simulation sizes across a single stage and from stage to stage.

Given the number of jobs per stage, and historical benchmark information available on various resources, an estimate for the total SUs required is computed and sent to the

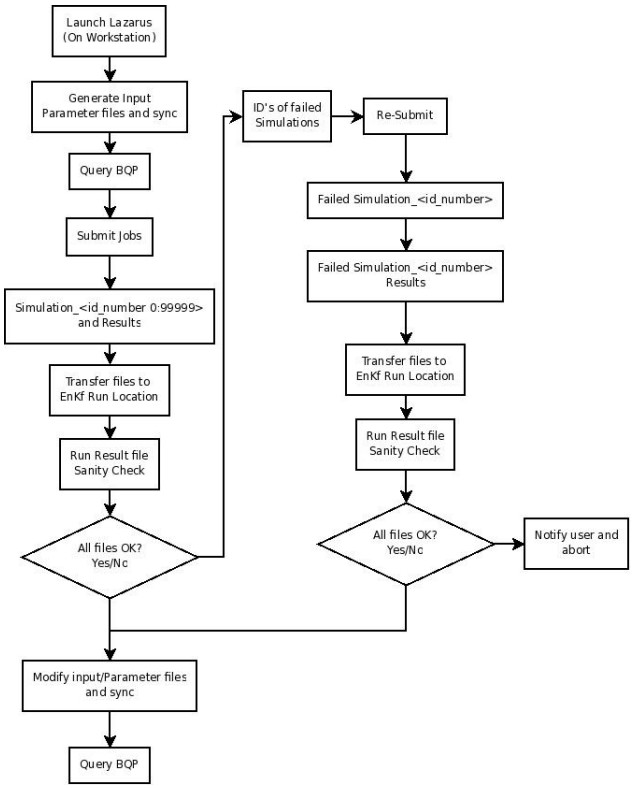

Figure 4.2: Control flow in Lazarus

Lazarus resource manager. The resource manager queries BQP for the optimal sizes and durations and creates a list of BigJobs, their sizes, durations and the resource/queue where they will be submitted. If we have more than one resource available, the SU requirement is satisfied based on a round–robin algorithm, iterating over resources and adding BigJobs until no more BigJobs need to be submitted.

Once the resource list is formulated, Lazarus uses the BigJob abstraction to submit jobs to all BigJobs in the resource list. The status of the jobs and BigJobs is reported on regular intervals through a logging mechanism. Upon completion of all the subjobs, Lazarus performs a calibration test where it runs the user–specified output–check on a pre–validated file, to ensure all the tools are in place and in working order. It then proceeds to check the output from all simulations to ensure sanity of output. Simulations that have faulty output are then collected into a list and resubmitted. The output is again checked for sanity: if all is well we proceed with the EnKF run (i.e. the analysis) and if not Lazarus will abort with a detailed log file of the error. This is mainly to safeguard against wasting SUs on corrupt runs. Once the analysis is complete, Lazarus repeats the forecast iteration with the newly modified state vectors and the history matching continues.

4.4 Autonomy

The Lazarus framework contains several aspects of autonomy: it has self–configuration (deployment on resources), self optimization (using BQP data), self monitoring (checking its own output) and of course self healing (resubmission of faulty simulations). These aspects have been implemented with varying levels of intelligence: the self optimization for example is a basic algorithm that uses BQP data to assign big–jobs to resources, but does not take into consideration bandwidth requirement and the cost of copying files across machines. The self–healing on another hand can typically resurrect jobs that fail due to node failure as opposed to software failure. Many autonomic features of Lazarus will find their way into the FAUST framework, where they will be improved to incorporate sophisticated inherent intelligence.

4.5 Failure Modes

As early as the first nontrivial Lazarus run we encountered failures. They varied in nature (hardware/environment) and severity. As Lazarus progressively used more resources, many more failure modes were encountered. Many were errors that were irrecoverable and resulted in a cold restart from scratch; others were recoverable with a simple simulation result–checking fault tolerant component built into Lazarus.

Representative hardware failures encountered:

- Lost a compute node from the pool
- Lost a network connection to a machine, BQP or the advert service
- Could not write to /scratch because it was taken down for maintenance

Amongst the modes of failure, we found that we reliably recovered from node failure and failure to write data to disk, but not network failure. The status of any given job is reported in a one–way poll for current state.

Representative software environment failures encountered:

- Missing or wrong shared libraries
- Wrong or inadequate environment variables
- Run out of quota on disk, wrote too many files to the same directory
- Run out of SU's in the allocation
- No internet connection to the compute nodes (no BQP, no advert service)
- MPI error that causes a simulation to stall (this happened because of a faulty installation and was corrected)

In software environment failures, typically jobs terminated with errors or were killed, before the simulations' results were written to disk. Unsurprisingly we ran into all of these failures while developing Lazarus, and were able to recover from all of them except for the network connectivity in the compute nodes, as this would register the job's state as "unknown". This would be solved with the soon–to–be–implemented heart–beat monitoring system.

Representative simulation failures encountered:

- Missing parameter files or executable
- Wrong/Non–existent parameters, parameter files, or erroneous setup of parameter files
- Divergence in the solver, causing NaN shuffling and leading to exceeding the requested wall clock time
- Under–estimation of the required wall clock time

- Hitting /scratch or /home too often (because of checkpointing), leading to very low access speeds and exceeding the requested wall clock time

- Numerical errors resulting in unintended behavior or bad results, NaN's or Inf's (this can be caused by wrong model parameters or divergence in the solver)

Some of the simulation failures are relatively easy to guard against, namely the wrong parameter file or missing executable. The other errors are harder to detect and rectify.

At every stage, we run a user–defined number of ensembles on HPC and HTC resources. As the ensemble members of a given stage run to completion, a second set of error–checking jobs are launched that ensure all the ensemble members ran to completion before the Kalman gain calculation is performed. This error–checking mechanism attempts to resubmit the failed jobs, and if met with failure a second time, the stage is halted to allow for user intervention. This seemingly un–important feature safeguards against runaway failed simulations that consume service units with no good reason.

The numerical errors are hard to detect: result files full of wrong values will have the same size as result files with useful data. With some customization, the output file check can be extended to scan for NaN's and Inf's in the result file. This is a minor improvement and will be IO intensive (therefore costly). A better solution would be to add sanity checks in the actual simulation code through the liberal and extensive use of error and warning messages. These messages are collected by Lazarus in the output and error files. For example in **Defiant**, all calls made to PETSc use the "CHKERRQ" and "CHKMEMQ" functions to check for computational and memory errors. The error checking functionality can be turned on and off through command line options that are specified in Lazarus. It is possible to run the first few stages of the EnKF with full error checking then switch it off after experience shows it is not needed, to avoid computational overhead.

4.6 Fault Tolerance

Given the many different failure modes, fault–tolerance provides Lazarus with built in self–healing capabilities. These capabilities rely on a tool–check/calibration test and output file checks. After simulations have finished and their output files are copied, Lazarus proceeds to perform a tool–check on an undamaged file. The purpose of the tool–check is to ensure the tools that will be used in checking the output files from the simulations are available and behave as expected. This is important as we move from one machine to another with a different environment, tool versions and tool output. Once the tools are verified, they are used to check the output of all the simulations. If a simulation is found to have missing, incomplete or otherwise faulty output, it is flagged

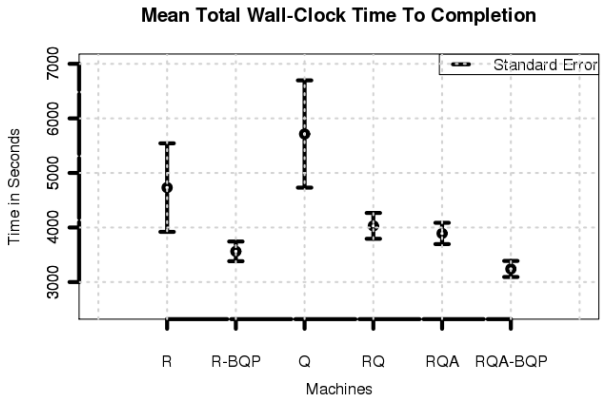

Figure 4.3: Time to completion with Lazarus. From Left to Right: (i) Ranger (ii) Ranger when using BQP, (iii) QueenBee, (iii) Ranger and QueenBee, (iv) Ranger, QueenBee and Abe, (v) Ranger, QueenBee and Abe when using BQP.

for resubmission. After all output is checked, the faulty simulations are resubmitted and upon termination, all output is checked again, and upon success, Lazarus proceeds.

4.7 Grid Awareness

Since Lazarus uses the SAGA Python bindings, it is natively grid–aware, Condor–aware and cloud–aware, if the globus, Condor and cloud adaptors are present. All jobs can be submitted to different supercomputers and Condor pools, all files can be transferred from one resource to another, and each EnKF stage can be distributed across multiple resources. The Lazarus script can run on any machine, say a researcher's workstation and monitor the status of submitted jobs. Figure 4.3 outlines the total time to completion for five stages of 100 ensemble members running on single and multiple resources, with and without BQP optimization. Optimizing job launching with BQP leads to a reduction of up to 50% of total time to completion. This translates to the reservoir modeler spending 50% less time waiting for results of history matching, or running twice as many history matching studies.

Chapter 5

Integration and the Closed Loop

5.1 Unifying the IO

One of the major tasks undertaken in this effort is to unify the input and output formats when working with different programs and different packages. To that end we chose to use the PETSc file input/output functions. This provides direct control over the format (HDF5, NetCDF or PETSc native format) through parameters that are passed at runtime. Since the reservoir simulator **Defiant** and the ensemble Kalman filter (**Valiant**) both use PETSc data structures directly, it was was an obvious and convenient choice. PETSc input/output routines also allow for remote reads and writes as well as socket interfaces for streaming.

5.2 Integration of Sensor Data

With a high performance reservoir simulator, a parallel, scalable ensemble Kalman filter, and a workflow manager that provides access to an abundance of high performance and high throughput resources optimized for lowest total–time–to–completion, we are well equipped to perform live, real–time reservoir characterization based on direct sensor data. To that end we implemented a communication layer based on data streaming from the data acquisition (sensor) platforms to a data spool (MySQL database). The EnKF application (**Valiant** or **Blanc**) queries the data spool for the latest available sensor platform (field observations) data. Once the new field observations are retrieved they are labeled as "assimilated" to differentiate them from the next–in–line observations.

The use of an intermediate data–spool is necessary for many reasons: it is difficult to guarantee resource availability and synchronization with the EnKF. It is also prohibitively costly to maintain an active EnKF session throughout the life–time of the reservoir. With a data–spool we can also adjust the update rate at the EnKF session

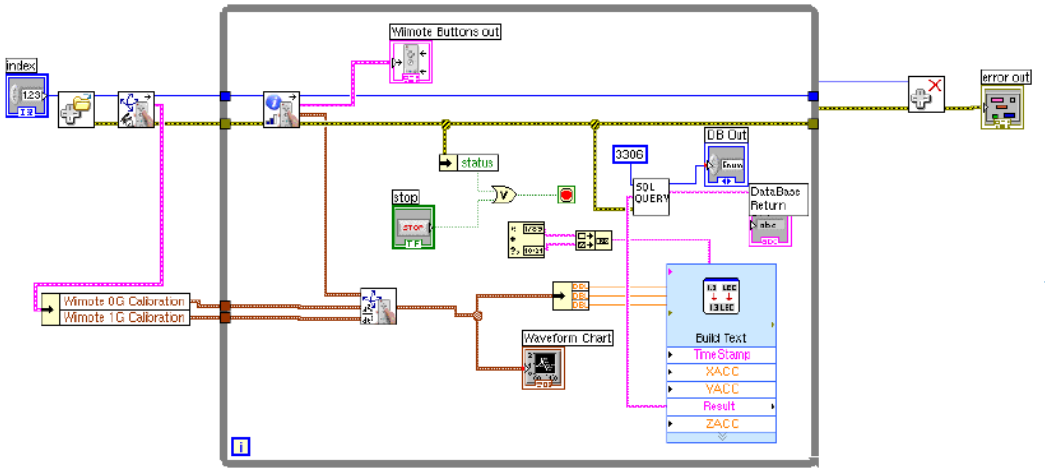

Figure 5.1: Sensor side LabVIEW vi. To stream data from sensor platforms to data spools we use a LabVIEW vi that makes calls to a shared library containing an interface to the database (MySQL). Initial tests were conducted using a Wii mote 3–axis accelerometer. Image courtesy of Richard G. Duff

without losing any data from the reservoir. Updates to the models can be as immediate as the moment data is broadcast from the sensors in the field, or as late as when the reservoir is being abandoned. With immediate model updates, we can run forecast and optimization studies. Lazarus, with its BQP optimization and reduced total time to completion, will make this possible.

Special care had to be taken when interfacing to sensor platforms: there is no unified specification for communicating abstract sensor information. The simplest approach was to develop an interface to a common data acquisition application: LabVIEW from National Instruments (National 2009). To that end we developed a virtual instrument (vi) that makes function calls to a shared library which in turn interfaces to the MySQL database (Figure 5.1). The vi has an adjustable streaming rate for polling sensor information and relaying it to the database, and is modular so it can be integrated in existing vi's as a sub–vi. This translates to painless integration with existing field and laboratory sensor platforms, in particular, the UCoMs sand tank experiment (Lei et al. 2006).

It is worth mentioning that the intention here was to develop an abstract interface to sensor data that can accommodate everything from drill–string modelling (Duff), well–heads, weather stations and buoys. This is one of the reasons why a ProdML (ProdML) was not the first choice of interfaces. However, should the need arise, we can develop a direct ProdML to PETSc interface which will have no significant impact on the closed loop performance.

Chapter 6

Conclusions and Future Work

6.1 Defiant

As a finished product, **Defiant** will be able to handle non–isothermal, multicomponent multiphase flow and serve as a research platform for reservoir simulation, geothermal energy, CO2 sequestration and enhanced recovery. Due to its PETSc based implementation, **Defiant** is also an interesting applied mathematics engine that can be used as a test platform for new solvers and preconditioners and adaptive mesh refinement algorithms. Finally, **Defiant** is interesting from a computer science perspective in its ability to run multiple, coupled simulations with varying degrees of granularity. For these reasons, special care has been taken in making **Defiant** as extensible and flexible as possible, without sacrificing utility. As it stands today, the stable branch of **Defiant** contains over 8,000 lines of code, and the development branch well over 15,000. Development at the moment is focused on:

- General black oil reservoir simulation test case: A **Defiant** based reservoir simulator with XML input support for convenient problem specification
- Multicomponent support (through PETSc degrees–of–freedom interface in distributed arrays)
- Newton–Raphson solver support for multicomponent reservoir simulation
- Non–isothermal reservoir simulation
- Support for fracture mechanics models
- Support for multicomponent reservoir simulation in the **Valiant** interface
- Better use of the geometric multigrid routines in PETSc (the DMMG), removing current well handling restrictions

- Identifying the performance bottleneck with the MPI_Allreduce function calls
- Identifying the cause of the large memory consumption
- Coupling with the PVT modelling package: **Reliant** (El-Khamra 2009c) that is under development to facilitate model preparation

6.2 Valiant

The ultimate aim behind **Valiant** is to have a parallel ensemble Kalman filter implementation that is publicly available, scalable, fast, able to handle a large number of file formats and interoperable with the reservoir simulator. Building on the success of the initial trials, **Valiant** will be extended to handle ensemble optimization (Oliver and Liu 2008), iterative ensemble Kalman filters (Oliver and Liu 2008; Evensen 2006) and ensemble randomized maximum likelihood filters (Oliver and Liu 2008). Furthermore, **Valiant** has a wealth of features to inherit from implementations of local expertise (Li et al. 2007) including support for normalization and localization. These filters and features will use the same IO and scattering routines that exist today in **Valiant**, and will enable us to handle highly nonlinear simulations (Oliver and Liu 2008). We will also maintain the neutrality of **Valiant** and attempt to use the parallel ensemble Kalman filter with other simulators such as atmospheric simulation and weather forecasting.

Other planned extensions to **Valiant** include parallel model generation routines. At the moment we have to rely on external packages, which do not share a compatible IO layer with **Defiant**. This presents a major limitation in model preparation. The solution is simple: include model generation routines in **Valiant**. This will guarantee IO compatibility and make automated reservoir management studies possible.

6.3 Relentless

With ensemble optimization and ensemble randomized likelihood filters under development, the workflow manager design will undergo a thorough over–haul to accommodate these new workflows. **Relentless** will continue to be developed with the SAGA Python bindings and with the FAUST (Weidner and Jha 2008) framework. The Lazarus framework will be fully integrated in **Relentless** (El-Khamra 2009b), which will retain autonomics, fault tolerance and abstraction as defining qualities. **Relentless** will also support running coupled simulations for hydraulic well models and coupled reservoir simulations.

Bibliography

AB MySQL. 1995. MySQL: A relational database management system. http://www.mysql.com/.

Alpatov, Philip, Greg Baker, Carter Edwards, John Gunnels, Greg Morrow, James Overfelt, Robert van de Geijn, and Yuan-Jye J. Wu. 1997. "PLAPACK: parallel linear algebra package design overview." *Supercomputing '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM).* New York, NY, USA: ACM, 1–16.

Anderson, W. K., W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. 1999. "Achieving high sustained performance in an unstructured mesh CFD application." *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM).* New York, NY, USA: ACM, 69.

Aziz, K., and A. Settari. 1979. *Petroleum Reservoir Simulation.* Society of Petroleum Engineers.

Balay, Satish, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. 2001. PETSc Web page. http://www.mcs.anl.gov/petsc.

Brevik, John, Daniel Nurmi, and Rich Wolski. 2006. "Predicting bounds on queuing delay for batch-scheduled parallel machines." *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming.* New York, NY, USA: ACM, 110–118.

Buckley, and Leverett. 1942. "Mechanism of fluid displacements in sands." *Transactions of the AIME (146): 107116.*

Carlson. 2003. *Practical Reservoir Simulation.* Tulsa: PennWell.

Chapman, Barbara, Gabriele Jost, and Ruud van der Pas. 2007, October. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation).* The MIT Press.

Chapra, Steven, and Raymond Canale. 2000. *Numerical Methods for Engineers.* McGraw-Hill.

Chen, Yan, Dean S. Oliver, and Dongxiao Zhang. 2008. "Efficient Ensemble-Based Closed-Loop Production Optimization." *SPE/DOE Symposium on Improved Oil Recovery.*

Chen, Zhangxin. 2007. *Reservoir Simulation: Mathematical Techniques in Oil Recovery.* Philadelphia: Society for Industrial and Applied Mathematics.

Datta-Gupta, Akhil, and M. J. King. 2007. *Streamline Simulation: Theory and Practice.* Society of Petroleum Engineers, Textbook Series.

Duff, Richard G. Drilling Lab: Drillstring Vibration Experiments. Available online at http://www.youtube.com/drillinglab.

El-Khamra, Yaakoub, and Shantenu Jha. 2009. "Title: Developing Autonomic Distributed Scientific Applications: A Case Study From History Matching Using Ensemble Kalman-Filters." *Sixth International Conference on Autonomic Computing, 2009. ICAC '09 (Barcelona).* IEEE.

El-Khamra, Yaakoub Y. 2009a. Defiant: A High-Performance Reservoir Simulation Platform. http://github.com/yye00/Defiant.

———. 2009b. Relentless: High-Throughput, High-Performance Autonomic Workflow Management Tools. http://github.com/yye00/Relentless.

———. 2009c. Reliant: Lightweight PVT Solver. http://github.com/yye00/Reliant.

———. 2009d. Valiant: A Generic Parallel Statiscal Ensemble Method Tools Platform. http://github.com/yye00/Valiant.

Evensen, Geir. 2006. *Data Assimilation: The Ensemble Kalman Filter.* Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Falgout, R.D., J.E. Jones, and U.M. Yang. 2006. "The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners,." *Numerical Solution of Partial Differential Equations on Parallel Computers, A.M. Bruaset and A. Tveito, eds.*

Fanchi, John. 2005. *Principles of Applied Reservoir Simulation, Third Edition.* City: Gulf Professional Publishing.

Forsythe, G.E., and W.R. Wasow. 1960. *Finite Difference Methods for Partial Differential Equations.* New York: Wiley.

Goodale, Tom, and Jha Shantenu. 2005. "SAGA: A Simple API for Grid Applications, High-Level Application Program- ming on the Grid." *Computational Methods in Science and Technology, Special Issue, Grid Applications: New Challenges for Computational Methods*, Volume 8.

Hilderbrand, F.B. 1968. *Finite Difference Equations and Simulations.* Prentice Hall.

Holstein, Edward D. 2007. *Petroleum Engineering Handbook, Vol. 5 Reservoir and Petrophysics.* City: Society of Petroleum.

Hossaina, Shahadat, and Trond Steihaug. 2004. "Graph coloring in the estimation of sparse derivative matrices: Instances and applications." *ISMP 2003.*

Hwang, Feng-Nan, and Xiao-Chuan Cai. 2005. "A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations." *J. Comput. Phys.* 204 (2): 666–691.

Jansen, J.D., S.D. Douma, P.M.J. Van den Hof, O.H. Bosgra, and A.W. Heemink. 2009. "Closed Loop Reservoir Management." *SPE Reservoir Simulation Symposium.*

Jha, Shantenu, Hartmut Kaiser, Yaakoub El-Khamra, and Ole Weidner. 2007. "Design and Implementation of Network Performance Aware Applications Using SAGA and Cactus." *Accepted for 3rd IEEE Conference on eScience2007 and Grid Computing, Bangalore, India.*

Kalman, R.E. 2005. "A New Approach to Linear Filtering and Prediction Problems."

Kamil, S. et al. 2006. "Reconfigurable Hybrid Interconnection for Static and Dynamic-Scientific Applications." *International Conference for High PerformanceComputing, Networking, Storage and Analysis.* Tampa, FL.

Kreyszig, Erwin. 1998. *Advanced Engineering Mathematics.* Wiley.

Lapipus, L., and J. H. Seinfeld. 1971. *Numerical Solution of Ordinary Differential Equations.*

Latt, Jonas. 2006. Open source lattice Boltzmann code. http://www.openlb.org/.

Lazarus3. Lazarus/Relentless webpage. http://www.github.com/Lazarus.

Lei, Zhou, Dayong Huang, Archit Kulshrestha, Santiago Pena, Gabrielle Allen, Xin Li, Christopher White, Richard Duff, John R. Smith, and Subhash Kalla. 2006, May. "ResGrid: A Grid-aware Toolkit For Reservoir Uncertainty Analysis." *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid06).* Singapore.

Li, Xin. 2008. "Continuous Reservoir Model Updating by Ensemble Kalman Filter on Grid Computing Architectures." *Ph.D dissertation, Louisiana State University.*

Li, Xin, Christopher White, Zhou Lei, and Gabrielle Allen. 2007, August. "Reservoir Model Updating by Ensemble Kalman Filter-Practical Approaches Using Grid Computing Technology." *Petroleum Geostatistics 2007.* Cascais,Portugal.

Luckow, Andre, Shantenu Jha, Joohyun Kim, Andre Merzky, and Bettina Schnor. 2009. "Adaptive Distributed Replica-Exchange Simulations." *Theme Issue of the Philosophical Transactions of the Royal Society A: Crossing Boundaries: Computational*

Science, E-Science and Global E-Infrastructure Proceedings of the UK e-Science All Hands Meeting 2008, vol. 367.

Luettich, R.A., Jr. J.J. Westerink, and N.W. Scheffner. 1992. "DCIRC: an advanced three-dimensional circulation model for shelves coasts and estuaries, report 1: theory and methodology of ADCIRC-2DDI and ADCIRC-3DL." *Dredging Research Program Technical Report DRP-92-6, U.S. Army Engineers Waterways Experiment Station*. http://www.unc.edu/ims/adcirc/.

Mascagni, M., and A. Srinivasan. 2000. "Algorithm 806: SPRNG: A Scalable Library for Pseudorandom Number Generation." *ACM Transactions on Mathematical Software*.

Mattax, Calvin. 1990. *Reservoir Simulation*. City: Society of Petroleum.

Moore, James C. 1998. "Visualizing with VTK." *Linux Journal*, p. 5.

MPI Forum Standards Committee. 1994. "Message Passing Interface Standard." http://www.mpi-forum.org/.

MVAPICH Development Team, OHU. 2002. MVAPICH: MPI over InfiniBand and 10GigE/iWARP. http://mvapich.cse.ohio-state.edu/.

National, Instruments. 2009. "LabVIEW: A graphical programming environment used by engineers and scientists to develop sophisticated measurement, test, and control systems." http://www.ni.com/labview/.

Oliver, Dean S., Albert C. Reynolds, and Ning Liu. 2008. *Inverse Theory for Petroleum Reservoir Characterization and History Matching*. Cambridge University Press.

OpenCFD Corporation. 2000. Open Field Operation and Manipulation CFD Toolbox. http://www.opencfd.co.uk/openfoam/.

Patankar, Suhas. 1980. *Numerical Heat Transfer And Fluid Flow*. Taylor And Francis.

Peaceman, Donald W. 1977. *Practical Reservoir Simulation*. Amsterdam and New York: Elsevier Scientific Pub. Co. : distributors for the U.S. and Canada, Elsevier North-Holland.

ProdML. Production Markup Language. Energistics: The Energy Standards Resource Centre http://www.prodml.org/prodml/Default.asp.

Saad, Yousef. 2003, April. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics.

Salawdeh, I., E. César, A. Morajko, T. Margalef, and E. Luque. 2008. "Performance Model for Parallel Mathematical Libraries Based on Historical Knowledgebase." *Euro-Par '08: Proceedings of the 14th international Euro-Par conference on Parallel Processing*. Berlin, Heidelberg: Springer-Verlag, 110–119.

Seidel, Edward. 1999. "Technologies for Collaborative, Large Scale Simulation in Astrophysics and a General Toolkit for solving PDE's in Science and Engineering." In *Forschung und wissenschaftliches Rechnen*, edited by T. Plesser and P. Wittenburg. Max-Planck-Gesselschaft, München.

TACC UT Austin. 2006. Ranger Sun Constellation Supercomputer. http://www.tacc.utexas.edu/resources/hpc/#constellation.

Thuwaini, Jamil, Shamsuddin Shenawi, and Bevan Yuen. 2009. "SS: Simulation Optimization of Wells with Complex Architecture." *Offshore Technology Conference*.

Unidata, NetCDF Development Team. 1984. "NetCDF: Network Common Data Form." http://www.unidata.ucar.edu/software/netcdf/.

Wang, Chunhong, Gaoming Li, and Albert Coburn Reynolds. 2007. "Production Optimization in Closed-Loop Reservoir Management." *SPE Annual Technical Conference and Exhibition*.

Weidner, Ole, and Shantenu Jha. 2008. Famework for Adaptive Ubquitious Scalable Tasks (FAUST). http://saga.cct.lsu.edu/index.php?option=com_content&task=view&id=98&Itemid=174.

Wissink, Andrew M., Richard D. Hornung, Scott R. Kohn, Steve S. Smith, and Noah Elliott. 2001. "Large scale parallel structured AMR calculations using the SAMRAI framework." *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM, 6–6.

Vita

Yaakoub El–Khamra received his Bachelor of Engineering degree from the American University of Beirut in 2002. In 2003 he became a staff member of the Center for Computation & Technology CCT at Louisiana State University. He later joined the graduate program at the Craft and Hawkins Petroleum Engineering Department.