SPE 93274

An Extensible Architecture for Next Generation Scalable Parallel Reservoir Simulation

D. DeBaun, T. Byer, SPE, ChevronTexaco; P. Childs, J. Chen, F. Saaf, M. Wells, J. Liu, H. Cao, L. Pianelo, V. Tilakraj, P. Crumpton, and D. Walsh, SPE, Schlumberger; H. Yardumian, R. Zorzynski, K.-T. Lim, M. Schrader, and V. Zapata, SPE, Chevron Texaco; J. Nolen,* SPE, Schlumberger; and H. Tchelepi,** SPE, ChevronTexaco

* Retired
** Now at Stanford University

This paper was prepared for presentation at the 2005 SPE Reservoir Simulation Symposium held in Houston, Texas U.S.A., 31 January 2005 – 2 February 2005.

Abstract

We discuss an object-oriented, component-based architecture for a parallel reservoir simulator. The architecture successfully balances the need for extensibility, maintainability, and reuse with the need for efficient computation. Parallelism is hidden from the application developer via a set of abstractions and a unifying framework that supports static and dynamic load balancing on unstructured and structured grids. We explain how the simulator architecture supports black oil and compositional models within a general formulation, and how it achieves parallel scalability for all the major components: property calculation, Jacobian construction, linear solver, well model, input/output etc. Some initial results are discussed for a selection of models ranging from standard benchmarks to large complex field cases.

Motivation

Current generation reservoir simulators typically have a lifespan of at least 10 years. As a result, many of the simulators in widespread use have their roots in petroleum engineering, software engineering and computing technologies that were originally targeted at structured grids and scalar architectures. Advances in software and hardware technologies, parallel computing, advanced gridding and modern software engineering techniques have created new opportunities. At the same time, evolving workflows involving complex geological models, large full-field simulations, near well studies, production and facilities optimization, and stochastic modeling have placed new demands on reservoir simulation technology.

Although object-oriented design has been the main staple of software development for quite some time, its application to scientific programming has been hindered somewhat by the perceived inefficiency of run-time polymorphism. However, in recent years the emergence of compilers that are in full compliance with the C++ standard has made use of generic programming techniques commercially practical. Generic programming offers the benefits of encapsulation and code reuse without the overhead of virtual function calls[1,2]. Object-oriented and generic programming techniques using C++ have been shown to be useful in the development of scientific and numerical applications[3,4].

Large-scale object-oriented frameworks for scientific computing have been developed in many contexts[5], and parallelism has been addressed for example in the POOMA project[6]. Object-oriented architectures for flow simulations have been developed in other industries[7] and these techniques have been applied to reservoir simulators by several practitioners within academia[8,9,10,11,12] and also in industry[13].

Introduction

In order to take advantage of the trends in reservoir simulation and computing technology described earlier, we have invested in a next generation simulator. The key features of this simulator, referred to as Intersect (or IX) are:

1. parallel scalability
2. the ability to handle large and/or complex reservoirs
3. unified treatment of structured and unstructured grids
4. multi-purpose implementation
5. adaptable software architecture.
6. coupled surface networks

In this context, complexity can arise from modeling of structure, advanced wells, heterogeneity, compositional fluid behavior etc. Underpinning our simulator is extensive research in collaboration with industry and academia on: architectural frameworks; formulation of nonlinear equations in the reservoir and wells; linear solvers; parallel communication models; compositional phase behavior; object oriented design and framework development; unified well model; load balancing; and longer term initiatives such as multi-scale modeling[14].

We chose an approach that takes advantage of the object-oriented and generic programming capabilities of C++ to provide maximum reuse and extensibility without sacrificing computational efficiency.

In order to maximize the speed of simulation, it is important that a simulator be able to make effective use of single-program multiple-data (SPMD) architectures. Programming for these architectures is inherently more complex than serial programming. In addition, the parallelism in the reservoir simulation problem is inherently more complicated that in many other domains. In the past, most applications have been designed as serial code and then ported to parallel platforms later in their lifecycle. This generally involves considerable refactoring and suboptimal parallel performance. Our architecture and all associated algorithms have been built to run in parallel from the outset.

Our simulator is based on an unstructured graph topology that accommodates structured grids, local grid refinements, PEBI grids, and fully unstructured grids. This provides generality and extensibility for contemporary scientific practice without loss of efficiency and is ideally suited for effective parallelism. The encapsulation provided by the object-oriented approach means flexibility, hiding parallel details from application developers and allowing plug-and-play implementation of property models. We have used XML-based interfaces and code generation to support rapid integration with workflow tools around the simulator.

Recognizing that successful simulation codes have a long life, the architecture must be flexible enough to support and accommodate ongoing research and development. While these requirements are not necessarily defined at this time, it is clear that the simulation should be controllable from separate components including those developed by outside parties. To accomplish this, various well-defined interfaces are established to allow for extensibility, both at compile time and run time.

Layered object oriented architecture

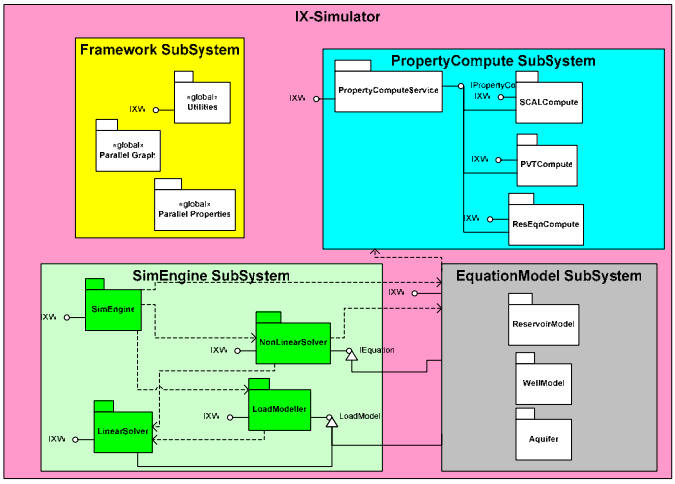

Fig. 1 High level package diagram of the simulator engine architecture

Fig. 1 shows a high level view of how the architecture is split into subsystems: The Framework subsystem is responsible for providing graph algorithms, implementing the parallel communication and various utilities. The *PropertyCompute* package encapsulates PVT and SCAL calculations as well as wrappers to ensure a common interface separating algorithms from data in the engine. The *SimEngine* package provides the core simulator including linear and nonlinear solvers and Jacobian setup. Lastly, the *EquationModel* system provides equations for reservoir objects, well objects and aquifer objects which are composed and solved by the SimEngine subsystem.

The simulator has been designed to support plug-in functionality for research and development purposes. For example, our PVT and linear solver packages may be replaced by 3rd party packages conforming to our API.

At the core of our architecture lies the parallel framework which is designed to provide algorithmic scalability utilizing a large number of distributed memory processors. The framework supports a message passing communication model, but the design also supports the efficient use of shared memory architectures.

The architecture employs object-oriented software design patterns to address common design problems[15]. The framework uses many of these patterns to address problems associated with algorithmic dependencies and extending functionality. Lastly, the architecture provides a framework which allows addition of more resources by separating out workflow processes from the simulation engine via callbacks.

In order to promote extensibility while maximizing code reuse and maintainability, the framework has made judicious use of generic programming techniques. As these techniques operate at compile time, they allow for maximal code reuse via code generated "by the compiler" without sacrificing computation speed.

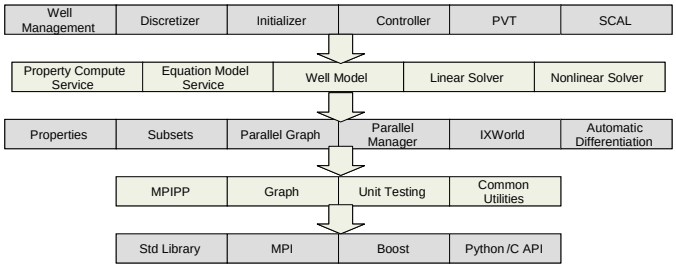

Fig. 2 Architectural layers in the simulator system

Some of layers in the architecture are shown in **Fig. 2**. Among these are

- Parallel framework
- Unit test framework
- Graph framework
- Utilities
- Well model
- PVT model (EOS and black oil)
- Linear solver
- Linearizer and Automatic Differentiation (AD)
- Formulation (Implicit, IMPSAT, nonlinear updater)
- Timestepping

- Scale model
- Load balance

Parallel communication model

The simulator uses the standard SPMD paradigm where the program is replicated on each processor, each solving a different part of the problem domain. The distribution of data between available processors is determined by the load balancing package.

Communication between processors is performed using a message passing interface, currently based on MPI[16,17]. Access to MPI is through a custom object oriented abstraction layer that provides for the possibility of using other standards in the future, motivated by OOMPI[18] but written in-house specifically for the requirements of a reservoir simulator. The bulk of the communication involves gathering and scattering cell- and well-based properties with connections to more than one processor.

Parallel graph

A distributed parallel graph representation underpins the simulation framework. It is the basis for encapsulation of data and algorithms because it provides the abstractions to hide parallel complexity. Use of a graph structure to represent abstractions in the solution of partial differential equations has become widespread in the scientific community[19]. Parallel graph and property data structures support arbitrary grid connections so that algorithms are independent of the grid structure. In this model, structured grid models are handled as a subset of unstructured models and the simulator is written for an unstructured grid without any specific code to deal with structured topologies (except on input and output, in order to fit within the current generation of workflow tools). The key concepts of the graph framework are the graph representation and creation of properties based on the graph. These properties include node properties, subset properties, and edge properties for flux computations.

One important aspect of the decomposition is that no processor is required to keep a global image of the graph or the properties on it. In this way the system can accommodate much larger models and effectively use many more processors than would otherwise be possible because the additional memory and processing overhead to deal with them is kept small. In this way excellent scalability is achieved for a greater number of processors.

Some key components that utilize the parallel graph framework are: the linear solver; the load balancer; Jacobian generation; and the well model. (The load balancer provides the framework for determining the load of significant components of the simulator and determining the partition of the reservoir and wells amongst the available processors to balance the overall load.)

Parallel communicators can be global or based on groups for wells. Parallel updates are naturally handled on the distributed graph in the usual fashion via “ghost” cells (see **Fig. 3**). The graph framework can create links between the overall graph and graphs representing certain subset or well information to support generic graph-based algorithms.

The framework provides services for data movement to support load rebalancing and also provides a number of utilities. These include serialization and deserialization of the graph and properties on the graph (serialization refers to creating a view suitable for a serial application, such as a controller). The facility to create a global graph or properties by deserialization is required for reporting field data with the current generation of workflow tools. Both wells and connections are represented by graphs to promote sharing of generic linear algebra code between the reservoir and well components when dealing with the coupled system. Additionally, the graph framework has facilities for algebraic coarsening and providing the data structures for interpolation and restriction operations in an algebraic multigrid linear solver[20].

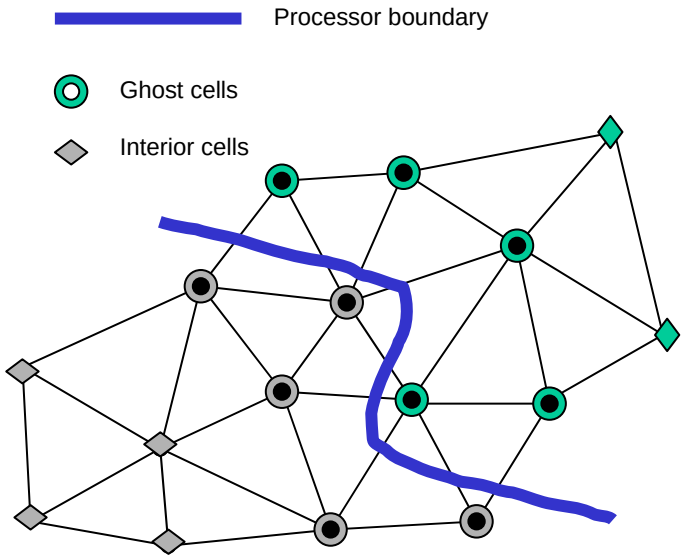

Fig. 3 Parallel graph, showing nodes, edges and ghost cells

Automatic Differentiation

The computation of derivatives for the Jacobian is of fundamental importance in reservoir simulation. Traditional simulators often have thousands of lines of hand-coded chain rule expressions. This code is error prone and difficult to maintain, and small errors can severely affect performance. In recent years, automatic differentiation (AD) tools have becoming increasingly popular as greater emphasis is placed on general purpose extensible simulation codes[21].

The two common approaches to AD[21] are: (1) source code converters and (2) operator overloading. Source code converters process existing code to generate additional source that calculates derivatives. This code is then compiled and linked into the application. However, modern languages such as C++ provide a variety of constructs that can be used to develop an automatic differentiation component that is run time configurable. Our AD library is based on operator overloading and expression templates[22,23]. This is used to

instantiate a computational graph containing expressions required for the differential expression[22] which is then traversed recursively to evaluate the derivatives. For example the expression required to compute the linearized fluxes is given by:

```
F1EdgeExprCompute(flux,
   trans * UsWt_Trans(mob,pot)*(pot),
   graph,
   domain_begin, domain_end, range_begin,
   independent_vars);
```

`F1EdgeExprCompute` is a helper method which takes iterator parameters that define the graph domain and independent variables for differentiation. `Flux` is the output derivative container. The second argument is the expression for the flux and is very similar to the symbolic representation $T * \lambda * \Phi$

The expression supports any fluid model, and any level of adaptive implicit (AIM) strategy, and allows for properties with arbitrary dependencies. This permits considerable reuse of the code of a given set of governing equations for different property models. The upstream weighting expression `UsWt_Trans(mob,pot)` is a custom expression. Likewise, operators for specialized operations such as obtaining derivatives via table lookup can be defined.

The resulting derivative calculations are in general nearly as fast and in some cases faster than their hand-coded counterparts. In addition, the AD package can be tested and verified independently of the simulator, thus reducing the possibilities of error and increasing confidence in the results.

Nonlinear Formulation

The usual equations for mass balance in the reservoir are written in terms of natural variables

$$\{w_i, y_i, x_i\} \quad i = 1, nc$$

$$p$$

$$S_o, S_g, S_w$$

as

$$\frac{V}{\Delta t}\Delta_t\left[\phi\left(\rho_o S_o x_i + \rho_g S_g y_i + \rho_w S_w w_i\right)\right] = -(q_o + q_g + q_w) +$$

$$\Delta_{xyz}\left[T\rho_o x_i \frac{kr_o}{\mu_o}\left(\Delta p - \Delta Pc_{go} - \gamma_o \Delta Z\right)\right] +$$

$$\Delta_{xyz}\left[T\rho_g y_i \frac{kr_g}{\mu_g}\left(\Delta p - \gamma_g \Delta Z\right)\right] +$$

$$\Delta_{xyz}\left[T\rho_w w_i \frac{kr_w}{\mu_w}\left(\Delta p - \Delta Pc_{go} - \Delta Pc_{wo} - \gamma_w \Delta Z\right)\right]$$

For the relaxed volume method[24], an additional constraint is imposed on the sum of saturations

$$1 - \Sigma S_j = 0$$

We use a natural variable formulation similar to that proposed by Coats[24]. The motivation is to avoid conversions to mass variables and to allow ready support for arbitrary levels of implicitness in the model. The formulation supports IMPSAT and adaptive implicit (AIM) to provide numerical stability at reduced computational cost in the spirit of Young[25]. We use IMPSAT[26] (Implicit pressure and saturation) approaches as well as an AIM approach based predominantly on IMPSAT[27] where the timestep selection is made according to the stable timestep criteria[28]. The split of primary and secondary variables, as defined by Coats[24] is configurable at runtime according to the phases present and the state of the fluid model.

Similar flexibilities are required by the nonlinear solution updating component where there is a large choice of algorithms depending on use of the Coats' relaxed volume update[24] or a more usual Newton (or modified Newton) update. The nonlinear solver can use either a relaxed volume approach or a Newton based approach to achieve convergence with convergence criteria based on the volume balance error as well as changes in solution variables from the linear solver component in the usual way.

In either case, however, the nonlinear solver is protected from implementation details through interfaces to algorithm strategies. A common formulation service supplies all information relating to the equation set so that all abstractions about the compositional or black oil fluid model are unified.

The simulator uses the unified compositional formulation[24] to simplify matters. Since the details of the fluid property calculation for black oil or compositional are hidden behind a PVT model provider, the core simulator is insulated from any details of the PVT model.

The nonlinear solver contains several points of configurability that include property calculations, equation linearization, linear solve, timestepping and nonlinear updating. The nonlinear solver contains a loose coupling between these components and the strategy pattern is again used to define the component algorithms and allow for interchangeable algorithms by creating sequences of compute strategies. All these strategies use the compute methods required for assembling an algorithm. For example, the fluid characterization may require a different set of computes for either black oil or compositional fluids.

Linear solver

The choice of linear solver is critical in being able to achieve scalable parallel performance. For the efficient and flexible solution of the linear system for a next generation reservoir simulator, we employ a multilevel, multistage linear solver.

Object oriented programming is naturally suited to define a number of such strategies to allow runtime selection and configuration of both preconditioner and solver algorithms. The linear solver is a component which itself is composed of many configurable parts expressed with a clearly defined interface. The parallel graph and communication framework underpins much of our linear solver.

Algebraic multigrid[20] is chosen here as the current paradigm which allows scalable performance. The pressure equation in reservoir simulation is elliptic in nature and can be solved effectively with this approach. From multigrid theory[29], excellent scalability can be achieved if there is a good load balance. Because there are predominantly local operations involved, this can be implemented very efficiently. We have written an efficient parallel algebraic multigrid (PAMG) solver to make effective use of special purpose algorithms required for reservoir simulation.

The parallel graph effectively supports all the algebraic operations required for the multigrid coarsening. The pressure preconditioner is PAMG with a GMRES iteration[30]. Well terms are treated in the usual way via a Schur complement approach. The full linear solver supports nested preconditioners - for example, a PAMG pressure solver together with a constrained pressure residual technique (CPR) approach of Wallis[31]. A local preconditioner is used for the non-pressure parts of the system together with an FGMRES outer iteration. This approach leads to an overall system which scales very well in parallel.

Because of the underlying graph, arbitrary unstructured grids are handled naturally with this approach; indeed the parallel graph framework simplifies the coding considerably. The linear solver makes extensive use of the parallel graph to represent the coupled system of reservoir and wells.

PVT

Typically, the flash component in compositional reservoir simulation is required for each grid element, in each nonlinear iteration of each time-step and the cost involved comprises a significant fraction of the overall run-time, often as much as 30-50%. Several techniques have been employed to provide considerable algorithmic speedup over iterative flash implementations in existing simulators. These algorithms will be more fully described in a later publication.

Most commercial reservoir simulators, given a long enough lifespan, ultimately evolve into a somewhat bewildering array of options and extensions, many of which are related to aspects of the physics being modeled. To achieve our objective of extending the simulator to include thermal modeling, chemical reactions, and other mechanisms, it is essential that the property calculations are encapsulated behind extensible interfaces. Property calculations relating to EOS etc. are performed by simulator invocation of *compute methods*, which present a uniform interface and operate over subsets of the overall computational graph. Containers representing a particular property also have the abstraction of derivative dependencies - for example oil viscosity may be a function of pressure, temperature and the composition of the oil phase. This helps simplify the management of property derivatives for Jacobian construction. This modular design has allowed us to wrap several external flash packages written in Fortran while preserving a common interface to the simulator.

The PVT module is responsible for determination of those physical properties of the fluid system relevant to the construction of the reservoir equations and their subsequent linearization. Each property is associated with a phase and can generally be viewed as a function of the natural variables relevant to the phase in question. The constraints of thermodynamic equilibrium are imposed in the standard EOS form (similarly for black oil)

$$r_i^f \equiv x_i \phi_i^L(p,T,x) - y_i \phi_i^V(p,T,y)$$

In the work described here, the iterative approach to flash calculations (resolving $r_i^f = 0$) has been pursued because decoupling makes possible the design of a cohesive, modular application, shareable with workflow components. Also the additional work involved in enforcing the flash constraint is likely more than offset by savings due to faster and more reliable convergence of the simulator iteration.

With few exceptions, PVT computations are node-based and parallel communication is not an issue; load-balancing however is. The load imposed by PVT computations is a strong function of the number of phases present and somewhat of the implicit state and will be balanced “on average” through the use of PVT load models describing this cost to the load balance component.

Property Framework and Data Model

A common API to the PVT and SCAL packages is provided to support the physical modeling and algorithm requirements of simulation studies. We make use of the factory pattern[15] to support abstract algorithm object creation and runtime configurability. Factories can be initiated from runtime configured plug-ins, which provides a mechanism to incorporate future functionality, such as different SCAL models, linear solver, or well model. This provides loose coupling by abstracting the creation and initialization of algorithms from the core simulator, thereby allowing implementations to change over time with minimal impact on the core.

In order for collections of property and equation calculations to be processed at the common abstract level, we wrap specialized methods for computing properties associated with the parallel graph to express them with a common API. This achieves separation of interface and implementation: for example, the flux equation setup needs access to relative permeabilities but does not require details of how it was calculated; and the nonlinear solver needs a Jacobian but not fluid model details.

The compute methods are actually realizations of the strategy pattern[15] which encapsulates algorithmic variation. This has advantages over the use of inheritance, as it avoids a potential combinatorial explosion in classes by separating algorithms and data as far as possible. This is accomplished by using callbacks, combined with generic programming, in order to apply algorithms to the underlying data. The concept of iterators[32,1] is used to promote separation of the algorithms and data containers as far as possible. The underlying data model is tree-structured, allowing grouping of data by objects and providing for clear memory management strategies by resolving lifetime and ownership issues of dynamic data kept on the heap.

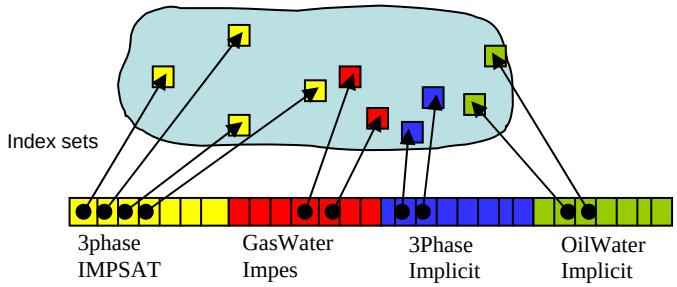

Fig. 4 Example use of index subsets to express groups of cells with common characteristic in an algorithm

Typically, in compositional reservoir simulation, PVT properties are only calculated on subsets of the reservoir, based on the number and types of phases etc. present in each subset. On a graph, a subset is a group of nodes or edges with some property in common, such as a phase or implicit state, or well equipment type (**fig. 4**). This procedure has been common in most simulators since the vector processing era[33,34,35]. Algorithms defined over subsets are layered independently as far as possible from data structures via iterators, though this does not preclude the use of tighter coupling where required for reasons of performance. Depending on the type of calculation, different algorithms are attached at run time to subsets which may process data using either indirect addressing, or by copying data into and out of contiguous memory blocks.

The use of graphs is a natural concept for creating sparse Jacobian data. Both data and algorithms are arranged in an object-oriented fashion via tree structures. For example, all reservoir field properties are held together in property maps in a "snapshot" structure which allows effective memory management as well as access to serialization and reporting utilities. The reservoir Jacobian is now assembled piecewise from a sequence of strategies building on top of the property calculations and AD.

Well model

The motivation here was to develop a model for production/injection systems from perforations to surface facilities. The well model is treated as a component with a well defined interface. It is a hierarchy of laterals, completions, devices etc. that lends itself very naturally to an object-oriented framework. Within a graph-based framework, the well trajectory and inflow paths and associated properties are modeled using special graphs linked to the reservoir graph.

The advent of control and completion techniques available in unconventional wells (e.g. multi-laterals), obliges the need to model such flow and control features by providing an open architecture capable of incorporating new devices as part of the well model (Holmes[36]). The state of the art approaches to modeling wells[37, 38] include conventional well models (CWM), uniform well models (UWM), and multi-segmented models (MSW). All of these approaches try to solve the same problem with different assumptions and/or simplifications. The well model in this work is based on a general formulation applicable to any configurations of well trajectory and control devices and is implicitly coupled to the reservoir.

The architecture provides a modular, object-oriented framework, extensible to implement new devices and equipment with appropriate interfaces and provides excellent scalability**.** A given well (which may be vertical or multi-lateral) is partitioned among several processes depending on the location of the well trajectory and the reservoir grid partition. Parallelism is achieved using a separate MPI communicator for each well in order to minimize latency and to support distributed solves which balance the workload for wells. There is no restriction on which processor is used to solve a given well, and as far as communication with the reservoir, inflow calculations are done locally according to the current domain split.

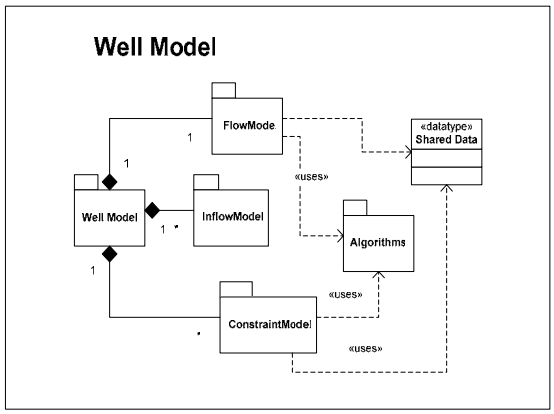

Fig. 5 UML package design of the well model

Fig. 5 summarizes the categories in the well model architecture: FlowModel is responsible for fluid flow and fluid property calculations within the well-bore and uses components (Algorithms Module) such as pressure drop, with a multi-phase flow correlation, and a linear solver. InflowModel calculates flow between well-bore segment and reservoir grid blocks. ConstraintModel is responsible for setting various constraints on user designated nodes: For every well, at least one constraint is needed at the well head or bottom-hole. The SharedData module consists of data like injection stream, PVT tables etc.

Load Balancing

It is critical to maintain an even computation and communication load across the system for efficient and scalable performance where latency is minimized. Not surprisingly, given the growth in parallel computing, load balancing is currently a fertile area of research in the scientific software community[39].

Our simulator employs a graph based approach to determine and maintain optimal partitioning, balancing load distributions, and minimizing total communication volume. With appropriate load modeling this achieves excellent parallel scalability, and is ideally suited to modeling in the presence of complex wells, inactive cells and unstructured grids.

The load balancing is performed on the coupled system of reservoir, wells, and aquifers. Load modeling techniques allow us to handle multiple and competitive objectives in a balanced fashion instead of simply distributing an equal number of active cells to each processor. The load is not only balanced at the beginning of a simulation, but can also be rebalanced during simulation when the system becomes imbalanced. The partitioning algorithms are run in parallel and are very efficient.

Several software packages are available for load balancing based on a graph concept. The general approach employed here is described by Karypis and others[40,19] Choice of node and edge weights determines a good initial static load balance such that all components are balanced as well as possible on average. Knowing up front where the well trajectories intersect the reservoir grid allows us to improve load balancing for the coupled system including well and reservoir equations. We have incorporated abstractions in our data model to support dynamic load balancing and data movement. Initial data has to be distributed to all processors and there is a bootstrapping approach to enable initialization and the initial load balancing to work in unison. Serialization uses framework utilities for reading and writing parallel data.

Starting an application on a parallel machine when the data distribution is not known at startup is a bootstrap process. In other words, data must be loaded and distributed so that the final data distribution can be calculated. Calculation of the final partition map is made using the graph, initial properties, and an expression of the computational load from PVT, linear solver and well model packages. This comprises what is known as the load model.

Often, the re-distribution of the problem from the original serialized front end is more efficient than attempting to dynamically move the data. This is founded on the knowledge that the initial distribution used to bootstrap the system will be very different from the calculated distribution. In this case, the amount of data to move dynamically is much larger than the amount of data we anticipate moving when we are dynamically rebalancing the system.

The load balancing component is independent from simulation details and is extensible to new partitioning algorithms and simulator components as long as the load can be accurately modeled by weights on the graph. **Fig. 6** shows an improved 3D partitioning: unstructured grid with 0.76M active cells, 3D partitioning for 16 CPUs of a generic model representing a salt dome. The improvement in load balancing over ECLIPSE[41] is shown by **Fig. 7**.

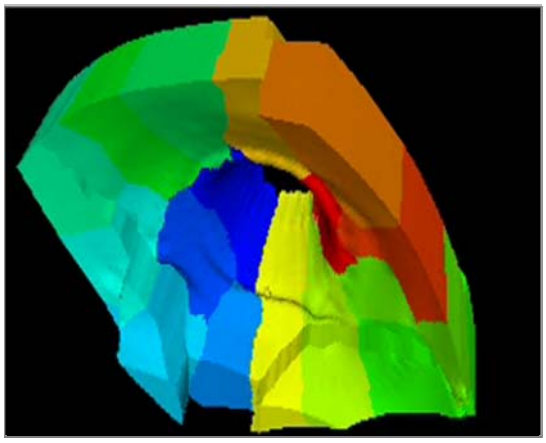

Fig. 6 Domain split for load balancing on a generic salt-dome model

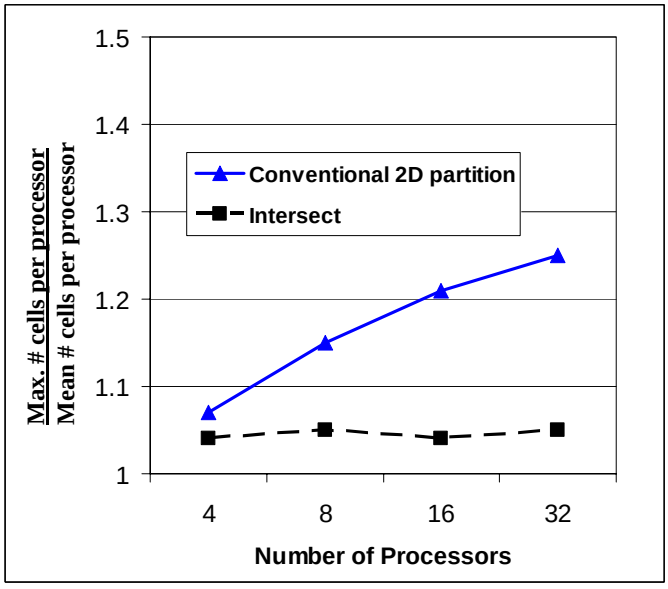

Fig. 7 Improvement in load balance for the generic salt dome model relative to an existing simulator using conventional 2D partitioning

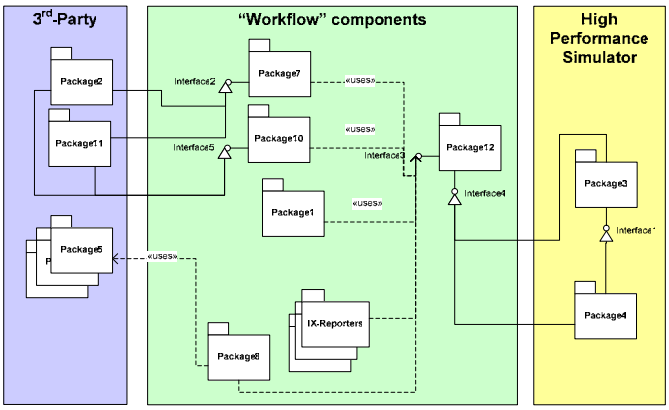

Fig. 8 Components interacting with the simulator engine

In this section, we describe the interfaces and additional components required to support emerging workflows with both loose and tight integration within the simulation environment. We have chosen a non-keyword based approach, using a hierarchical schema expressed in XML and special purpose APIs for bulk data transfer. The interface to the workflow tools is serialized, with the simulator engine providing the parallel serialization and deserialization tools. In this manner, we reflect the workflow interface and the simulator interface from the same description.

In our design, various components express a set of functionality to represent various aspects of a reservoir such as grid, fluid, well descriptions and how these change over time. The environment accommodates the need to be able to export as much as the user requests to the simulator. There are specific export algorithms to convert between its API and the internal representations in the simulator.

There are many parameters that are used to initialize and control a reservoir simulation run. These parameters and controls are expressed in XML, which allows a complete description of the interfaces. Code generation is employed to automatically generate C++ interfaces from this simple description. In this way the changes to the interface are reflected in the code automatically. Documentation is automatically generated from this description, meaning it is always up-to-date. Another benefit is that more than one type of interface can be generated, for example in addition to a C++ interface; interfaces to other programming languages such as Python can be generated[42].

The API specifies data access as well as validation methods which are reflected (implemented) in the simulator engine code. The simulator provides a rich set of validators ranging from simple generic ones (such as number out of range) to very specific checks on PVT and SCAL properties; and they are documented as part of the API such that they are callable from the environment. In this way external components such as the user interface have access to the very same validations as the simulator. This prevents the duplication of validation code that would otherwise occur, and reduces the likelihood that an invalid model will be sent to the simulator.

Within this design, (see **Fig. 8**) components are separate from the simulator engine, but interact with it through a defined set of APIs, referred to as IXWorld. Examples of these components include reporters, discretizer, initializer and well management.

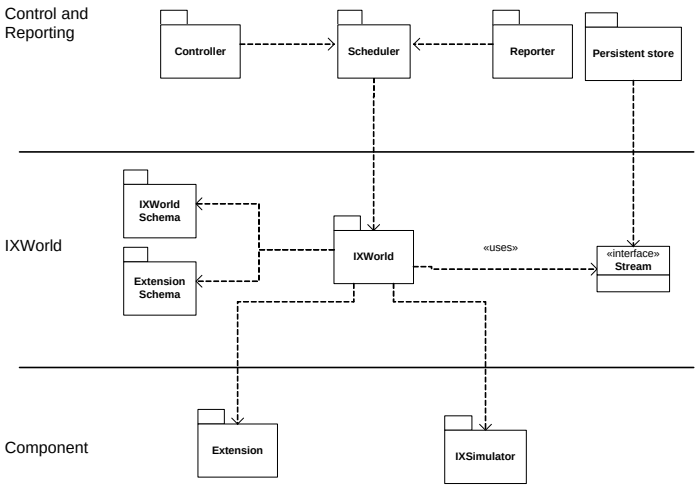

Fig. 9 Architectural representation of control and reports using the simulator engine component

During execution, the simulator will need to interface with controlling and reporting components that may be operating in a serial mode. This interaction is accomplished via the published APIs and illustrated in **Figs. 9** and **10**.

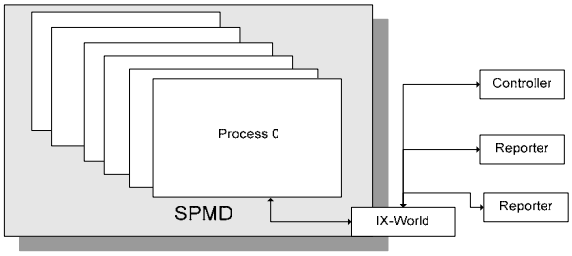

Fig. 10 Interface to serial programs

An important example of a controller is the well management system. It can sit on top of multiple simulator models, possibly spanning several reservoirs, aggregating and synchronizing them in order to produce effective well management strategies[43].

Results

Validation and benchmarking has been carried out on a number of realistic and proprietary full-field models as well as standard benchmark models. Here we present a few of our initial results. We are continuing to gain access to and benchmark larger and ever more complex proprietary models with unstructured grids, local grid refinement and multilateral wells – results for these will be published later. Here, we demonstrate excellent parallel scalability across the range from relatively small models to large geostatistical models. Scalability on smaller models is against the trend for existing

technology where scalable benefits are only approached for larger model sizes.

1. SPE10

The 10th SPE benchmark case is a standard test case which has proved challenging for many existing reservoir simulators[44] which do not cope well with the high degree of heterogeneity in the reservoir and are dominated by the linear solver component. This model benefits from the algebraic multigrid approach which deals effectively with heterogeneity and provides scalable performance without being limited to a geometric partition. Here we present a version of the SPE10 model, with original porosities and permeabilities run with a full three phase black oil fluid description, that has been. modified to be highly incompressible, to further stress the simulator. The model has about 1.1 M cells, and is run in implicit mode for 2000 days. It has 4 producers (BHP controlled, modified for multiple completions) and 1 injector (under rate control).

Fig. 11 demonstrates the initial load balance, split into 8 domains where the load is modeled using edge weights based on transmissibility and well trajectories. The vertical splits are seen due to shale barriers, which is somewhat different from the more usual pure areal decompositions.

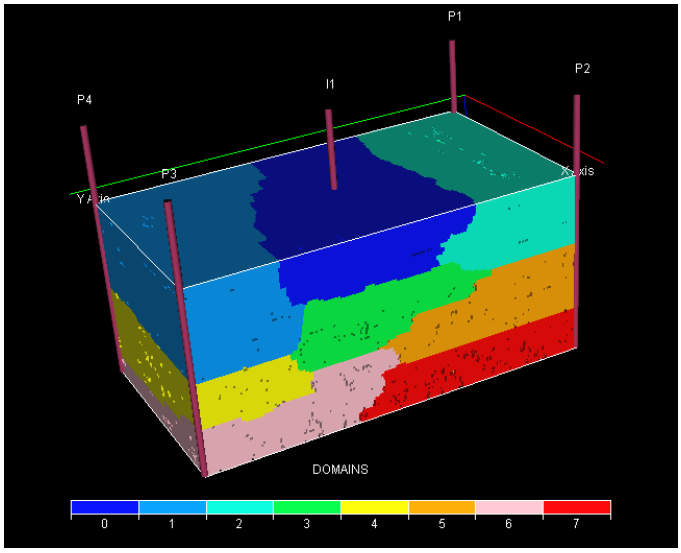

Fig. 11 Initial domain partition for modified SPE10 model

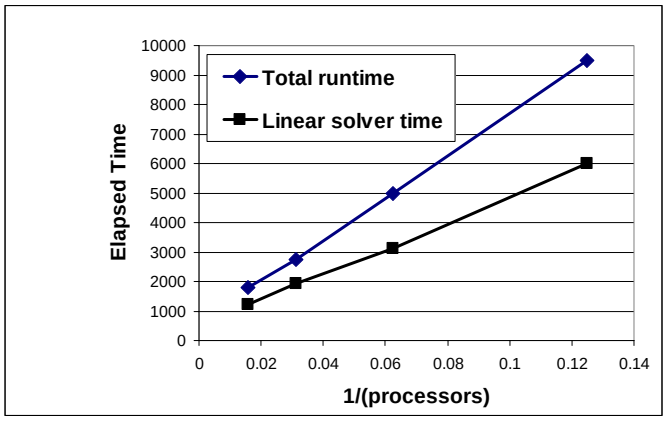

Fig. 12 Scalability speedup on modified SPE10 model

Fig. 12 shows run times plotted against 1/(number of processors) for both the full system and linear solver component on up to 64 processors, which is the limit for our in-house cluster. We have plotted elapsed time versus 1/(number of processors). Results show excellent parallel scalability and illustrate the feasibility of running such highly heterogeneous models. The low intercept with the y-axis shows we have minimized serial work.

It is a clearly desirable property for a parallel reservoir simulator that the solutions and convergence properties are relatively insensitive if not independent on the number of processors. **Fig. 13** illustrates that the convergence properties of the linear solver are independent on the number of processors for this model. The figure shows convergence history for a snapshot linear solution.

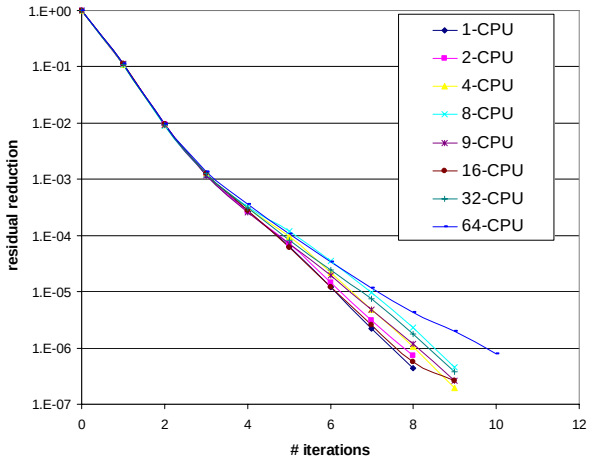

Fig. 13 Convergence independence on number of domains

2. Gullfaks

The models presented here have been derived from data pertaining to the Gullfaks oil drive project operated by Statoil ASA during 1998 to 2002. Our first model is a 19 layer prediction case with water injection, 80k active cells, black oil, 3 SCAL regions, 9 equilibrium regions and has 40 producers, 14 water injectors and 6 gas injectors. All wells start with rate control, and a prediction run to 5000 days is performed in implicit mode. A second compositional model is also

presented with 8 components and historical rate controls. **Fig. 14** illustrates the geometry of the reservoir and wells.

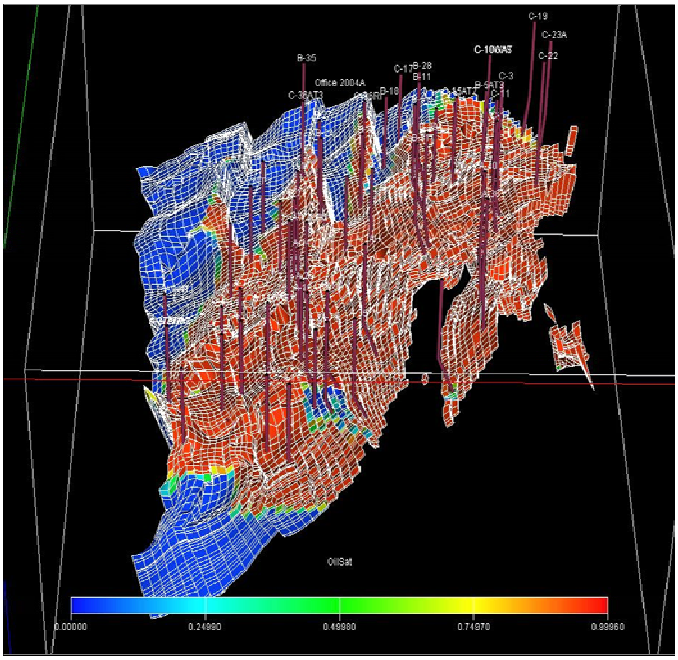

Fig. 14 View of the Gullfaks model showing wells and fault blocks

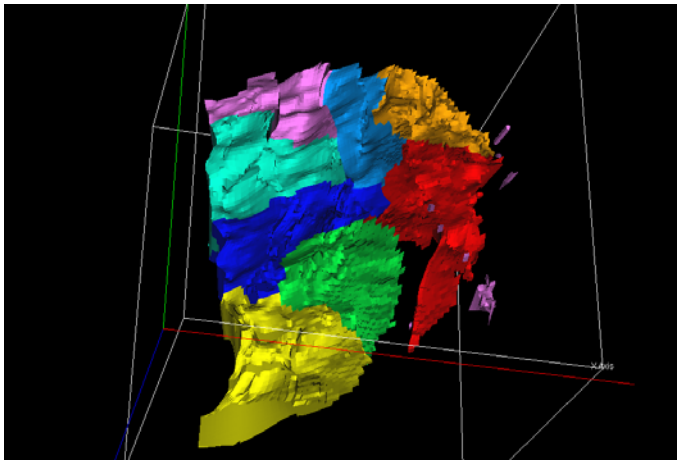

Fig. 15 Domain split into 8 domains for the Gullfaks model. Note the disconnected domain

Fig. 15 shows the domain split for 8 processors. **Fig. 16** illustrates results for the black oil Gullfaks model. In scalar mode, our simulator executes 5 times faster than ECLIPSE Blackoil, growing to 11 times faster when both simulators are allowed to choose as many processors as they wish. This performance, with a relatively small number of active cells per processor, clearly demonstrates the opportunity and potential benefit for using parallel simulation on small to medium sized models.

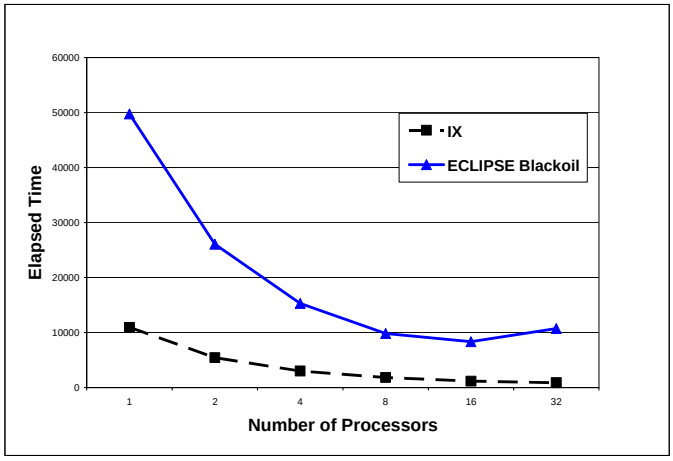

Fig. 16 Overall run times of the simulator for the black oil version of the Gullfaks model on up to 32 processors and comparison with ECLIPSE Blackoil

Speedup for the compositional Gullfaks on up to 32 processors model is shown in **Fig. 17.** We see excellent performance gains over ECLIPSE Compositional in this case. The result for each simulator is normalized by its runtime on 4 processors.

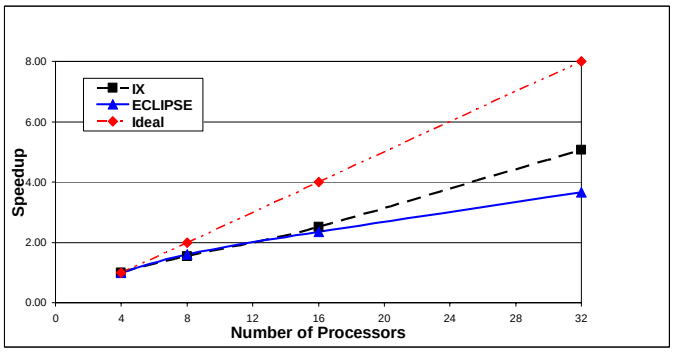

Fig. 17 Overall speedups for the compositional version of the Gullfaks model on up to 32 processors

Unstructured models

Benchmarking of field scale unstructured grid models is at a fairly early stage. This reflects the fact that end-to-end workflows involving unstructured gridding are not yet commonplace. For now, we present a small but hopefully representative unstructured grid sector model. **Fig. 18** shows our model, which has several fault blocks and 14k cells. It is run using the black oil formulation on a single processor. A fivefold speedup over existing simulators is obtained, as shown by **Fig. 19**.

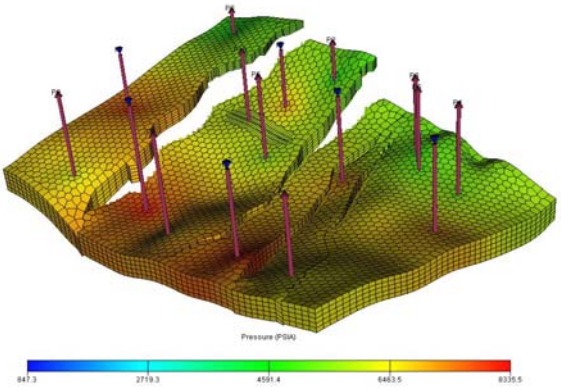

Fig. 18 Unstructured sector model

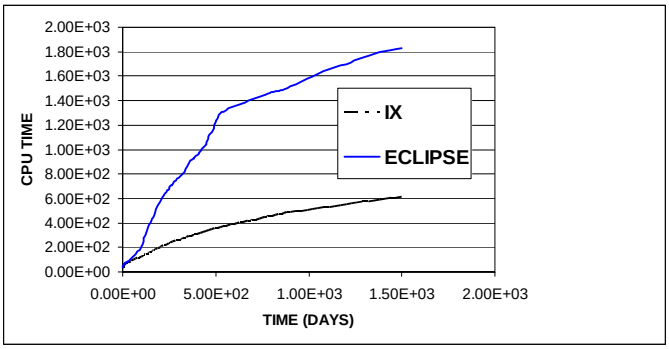

Fig. 19 Speedup for unstructured sector model relative to a commercial simulator

Conclusions

Next generation scalable parallel performance has been achieved for a range of benchmark and proprietary models. The results presented in this paper witness significant gains in parallel scalability and algorithmic performance over existing simulators in widespread use.

We have further demonstrated the application of state of the art software engineering techniques and adoption of software industry standards. In addition, we have demonstrated that the development of a flexible, extensible, parallel and object-oriented framework for reservoir simulation supports the extensibility to add new physical models. The framework carefully balances the requirements of performance with capabilities to add new physics, provide for plug-and-play of new physical models and components, while also exploiting ongoing research and providing integration with next generation workflow tools.

In the future, we will be capitalizing on the framework to deliver dynamic load balancing and to support development of new physical models and engineering features. Ongoing research is aimed at improving EOS and thermal modeling, advanced well management and optimization. Further details will be presented in future publications

Acknowledgements

The work described here is a collaboration of many people at Schlumberger and ChevronTexaco. Much of the model preparation and additional benchmarking was carried out by Shabhaz Sikandar and Paul Fjerstad (Schlumberger) and Will daSie (ChevronTexaco). Thanks are also due to many external consultants, including among others, Andrew Lumsdaine and John Wallis.

We express our appreciation to Statoil ASA for providing the Gullfaks model and kind permission to publish the results. Lastly, it is our pleasure to thank both Schlumberger and ChevronTexaco for permission to publish this work.

Nomenclature

nc	total number of components
x_i	liquid mole fraction for component i
y_i	vapor mole fraction
w_i	aqueous mole fraction
ρ_α	mass density of phase α
b_α	molar density of phase α
ϕ	porosity
V	initial pore volume
Pc_α	capillary pressure between phases α and α
μ_α	viscosity of phase α
kr_α	relative permeability of phase α
Z	vertical depth
γ_α	gravity density for phase α
ϕ^L, ϕ^V	fugacities for liquid and vapor phases
Δt	timestep
Δ_t	temporal differencing operator
Δ_{xyz}	spatial differencing operator

subscripts

o,w,g	oil, water, gas phases
α	phase

References

1. Austern, M. H.: *Generic Programming and the STL – Using and Extending the C++ Standard Template Library*, Professional Computing Series. Addison-Wesley, 1999.
2. Alexandrescu, A.: *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley, 2001.
3. Barton, J. and Nackman, L.: *Scientific and Engineering C+"*, Addison-Wesley, 1994.
4. Siek, J. and Lumsdaine, A.: "A Modern Framework for Portable High-Performance Numerical Linear Algebra", in *Advances in Software Tools for Scientific Computing, Lecture Notes in Computational Science and Engineering"*, pp 1-55 Springer-Verlag, 2000.

5. Henderson, M., Anderson, C, and Lyons, S. (eds): "*Object oriented methods for interoperable scientific and engineering computation*", Society for Industrial and Applied Mathematics, 1999.
6. Oddham, J., "Scientific Computing Using POOMA", C++ Users Journal, November 2002.
7. Langtangen, H. P.: *Computational Partial Differential Equations, Numerical Methods and Diffpack Programming, Lecture Notes in Computational Science and Engineering*, Springer-Verlag, 1999.
8. Verma, S. and Aziz, K., "FLEX: an Object-Oriented Reservoir simulator,", SPE paper 36007, presented at the Petroleum Computer Conference, Dallas, 1996
9. Verma, S.:"Flexible Grids for Reservoir Simulation", PhD Thesis, Dept. of Pet. Eng, Stanford Univ., 1996.
10. Byer, T.J., Edwards, M.G. and Aziz, K,: "A Preconditioned Adaptive Implicit Method for Reservoirs with Surface Facilities,", SPE 51895, presented at the SPE Reservoir Simulation Symposium, Houston, 1999.
11. Nogaret, C., "Implementation of an Network-based Approach in an Object-Oriented Reservoir Simulator," Msc Thesis, Dept. of Pet. Eng., Stanford Univ, 1996.
12. Parashar, M., Wheeler, A., Pope, G., Wang., K, Wang, P.: "A New Generation EOS Compositional Reservoir Simulator: Part II – Framework and Multiprocessing", SPE 37977, presented at the Reservoir simulation Symposium, Dallas, 1997.
13. Beckner, B.L., Hutfilz, J.M., Ray, M.B. and Tomich, J.F., "EM^{power}: New Reservoir Simulation System" ,SPE paper 68116, presented at the SPE Middle East Oil show, Bahrain, 2001.
14. Jenny, P., Lee, S., Tchelepi, H.A.: "Adaptive Multiscale Finite-Volume Methods for Multiphase Flow and Transport in Porous Media", SIAM Multiscale Modeling and Simulation, Vol3. no. 1, pp. 50-64, 2004.
15. Gamma, E., Helm, R., Johnson, R. and Vlissides, J: "*Design Patterns: Elements of Reusable Object-Oriented software*", Addison-Wesley, 1994.
16. Message Passing Interface Forum. "MPI: A Message Passing Interface", Proc. Supercomputing, pp. 878–883. IEEE Computer Society Press, Nov. 1993.
17. Geist A., Gropp W., Huss-Lederman S., Lumsdaine A., Lusk E., Saphir W., Skjellum T., Snir M.: "MPI-2: Extending the Message Passing Interface", *Euro-Par '96 Parallel Processing, Lecture Notes in Computer Science Number 1123*, pp. 128–135. Springer Verlag, 1996.
18. Squyres, J., McCandless B., Lumsdaine, A.: "Object oriented MPI: A Class library for the message passing interface", in proc. Parallel Object Oriented Methods and Applications (POOMA), Santa Fe, 1996.
19. Schloegel, K., Karypis, G. and Kumar, V.: "Graph Partitioning for High Performance Scientific Simulations", in *CRPC Parallel Computing Handbook*, eds. J. Dongarra et al., Morgan Kaufmann (2000).
20. Stüben, K.: "A Review of Algebraic Multigrid", Journal of Computational and Applied Mathematics, Vol. 128, pp. 281-309, 2001.
21. www.autodiff.org/
22. Aubert, P., Di Cesari, N., Pironneau, O.: "Automatic Differentiation in C++ using Expression Templates and Application to a Flow Control Problem", Computing and Visualization in Science, Vol. 3, No. 4, pp.197-208, 1999
23. Velhuizen, T.: "Expression Templates," C++ Report, Vol 7, no. 5, June 1995, pp 26-31
24. Coats, K.H.. Thomas, L.K. and Pierson, R.G.: "Compositional and Black Oil Reservoir simulation" , SPE paper 50990, SPEREE, August 1998
25. Young, L.C. and Russell, T.F.: "Implementation of An Adaptive Implicit Method", SPE paper 25245, presented at the SPE Reservoir Simulation Symposium, New Orleans, 1993.
26. Quandalle, P., Savary, D: "An Implicit in Pressure and Saturations Approach to Fully Compositional simulation", SPE 18423, presented at the 10th Reservoir Simulation Symposium, Houston, 1989.
27. Cao, H. and Aziz, K.: "Performance of IMPSAT and IMPSAT-AIM Models in compositional simulation", SPE paper 77720, presented at the SPE ATCE, San Antonio, 2002.
28. Coats, K.H. "IMPES Stability: Selection of stable timesteps", SPE paper 84924, SPEJ, June 2003.
29. Trottenberg, U., Schuller, A, Oosterlee, C., "*Multigrid*", Academic Press, 2001.
30. Saad, Y: "*Iterative Methods for Sparse Linear Systems*", Second Edition, Society for Industrial and Applied Mathematics, 2003.
31. Wallis, J.R.: "Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration". SPE paper 12265, presented at the SPE Reservoir Simulation Symposium, San Francisco, 1983.
32. Stepanov, A. and Lee, M.: "The Standard Template Library", Technical Report HPL-95-11 (R.1), Hewlett-Packard Laboratories, Nov. 1995
33. Young, L.C.: "Full-Field Compositional Modeling on Vector Processors," SPE paper 17803, SPERE, Feb 1991.
34. Chien, M.C.H., Wasserman, M.L., Yardumian, H.E., Chung, E.Y.: "The Use of Vectorization and Parallel Processing for Reservoir Simulation", SPE 16025, presented at the SPE Reservoir Simulation Symposium, San Antonio, 1987.
35. Lim.K.T., Sciozer, D.J. Aziz, K. :"A New Approach for Residual and Jacobian Array Construction in Reservoir simulators", SPE paper 28248, presented at the SPE Petroleum Computer Conference, 1994, Dallas.
36. Holmes, J.A.: "Modeling Advanced Wells in Reservoir Simulation," *JPT*, Nov. 2001.
37. Holmes, J.A., Barkve, T., and Lund, O.: "Application of a Multisegment Well Model to Simulate Flow in

Advanced Wells," SPE PAPER 50646 presented at the SPE European Petroleum Conference, the Hague, the Netherlands, 20-22 Oct, 1998.
38. Stone, T.W., Bennett, J., Law, D.H.S., and Holmes, J.A.: "Thermal Simulation with Multisegment Wells," SPE paper 66373, presented at the SPE Reservoir Simulation Symposium, Houston, TX, 11-14 Feb, 2001.
39. Hendrickson, B., Devine, K.: "Dynamic Load Balancing in Computational Mechanics", Computer Meths. In Applied Mechanics & Engineering, vol. 184, issues 2-4, pp. 485-500, April 2000.
40. Karypis, G. and Kumar, V.: "Parallel Multilevel K-Way Partitioning Scheme for Irregular Graphs, Graph Partitioning and Sparse Matrix Ordering,",SIAM Review, vol 41., No. 2, pp 278-300, 2003.
41. ECLIPSE reference manual 2003, Schlumberger Information Systems.
42. www.python.org/
43. Ghorayeb, K., Holmes, J,. Torrens, R., Grewal, B.: "A General Purpose Controller for Coupling Multiple Reservoir simulations and surface Facility Networks", SPE paper 79702, presented at the SPE Reservoir Simulation Symposium, Houston, 2003.
44. Christie, M.A., Blunt, M.J.: "Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques", SPE paper 72469, presented at the SPE Reservoir Simulation symposium, Houston, 11-14 February 2001.