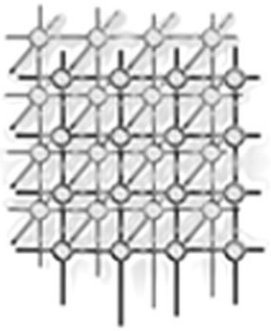
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2005; **17**:1387–1414
Published online 8 April 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.899

Enabling interactive and collaborative oil reservoir simulations on the Grid



Manish Parashar[1,*,†], Rajeev Muralidhar[1], Wonsuck Lee[2], Dorian Arnold[3], Jack Dongarra[4] and Mary Wheeler[5]

[1]*The Applied Software Systems Laboratory, Rutgers University, Piscataway, NJ 08854, U.S.A.*
[2]*Computing Science Research, Bell Laboratories, Murray Hill, NJ 07974, U.S.A.*
[3]*Computer Science Department, University of Wisconsin, Madison, WI 53706, U.S.A.*
[4]*Computer Science Department, University of Tennessee at Knoxville, Knoxville, TN 37996, U.S.A.*
[5]*Center for Subsurface Modeling, University of Texas at Austin, Austin, TX 78712, U.S.A.*

SUMMARY

Grid-enabled infrastructures and problem-solving environments can significantly increase the scale, cost-effectiveness and utility of scientific simulations, enabling highly accurate simulations that provide in-depth insight into complex phenomena. This paper presents a prototype of such an environment, i.e. an interactive and collaborative problem-solving environment for the formulation, development, deployment and management of oil reservoir and environmental flow simulations in computational Grid environments. The project builds on three independent research efforts: (1) the IPARS oil reservoir and environmental flow simulation framework; (2) the NetSolve Grid engine; and (3) the Discover Grid-based computational collaboratory. Its primary objective is to demonstrate the advantages of an integrated simulation infrastructure towards effectively supporting scientific investigation on the Grid, and to investigate the components and capabilities of such an infrastructure. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: oil reservoir simulation; Grid-based computational collaboratory; interactive monitoring and steering; IPARS; Discover; NetSolve

*Correspondence to: Manish Parashar, Department of Electrical and Computer Engineering, Rutgers, The State University of New Jersey, 94 Brett Road, Piscataway, NJ 08854, U.S.A.
†E-mail: parashar@caip.rutgers.edu

Contract/grant sponsor: National Science Foundation (CAREERS, NGS, ITR); contract/grant numbers: ACI9984357, EIA0103674 and EIA0120934
Contract/grant sponsor: Department of Energy/California Institute of Technology (ASCI); contract/grant numbers: PC 295251 and 1052856

Received 30 August 2003
Revised 12 January 2004
Accepted 5 February 2004

1. INTRODUCTION

High-performance simulations play a critical role in all areas of science and engineering. As the complexity, costs and resource requirements of these simulations grow, it has become important for scientists and engineers to leverage the emerging computational Grid infrastructure [1,2]. Computational Grid technology provides many benefits including the abilities to launch simulations in an automated way on remote, geographically-distributed resources, and to seamlessly (and collaboratively) access, monitor and interactively control the simulations at runtime. Such a Grid-enabled simulation infrastructure can significantly increase the scale, cost-effectiveness and utility of simulations, and has the potential for enabling highly accurate simulations providing in-depth insight into complex phenomena.

A high-level Grid-enabled simulation infrastructure provides many advantages to scientists and engineers including:

- a high-level computational framework for developing simulation models and formulations, configuring simulations, and specifying their runtime input parameters;
- relief from the tedium associated with finding and maintaining software libraries, toolkits, programs and other computing resources;
- dynamic resource selection providing increased performance efficiency;
- simulation deployment from thin clients to powerful hardware resources;
- interaction and steering capabilities, which allow experts to drive the discovery process by observing intermediate results and dynamically changing parameters to lead the simulation to more interesting results, explore 'what-if' scenarios, detect and correct unstable situations and terminate uninteresting runs early; and
- dynamic human collaborations with geographically dispersed participants to analyze and discuss the results of increasingly complex and multi-disciplinary simulations as they run.

As a result, Grid-enabling infrastructures are critical for transforming these simulations into true research modalities.

The primary objective of this paper is to demonstrate the advantages of an integrated simulation infrastructure towards effectively supporting scientific investigation on the Grid, and to investigate the components and capabilities of such an infrastructure. In this paper, we present a prototype interactive and collaborative problem-solving environment to enable the formulation, development, deployment and management of oil reservoir and environmental flow simulations in computational Grid environments. Our effort leverages three independent research projects, detailed below.

- *Integrated Parallel Accurate Reservoir Simulator (IPARS)* is an oil reservoir and environmental flow simulation framework. It is designed to study fluid transport and multi-phase fluid dynamics through porous media, or more specifically, underground structures. Using IPARS, petroleum engineers can simulate the oil recovery processes, explore the plans for optimal oil field management and achieve maximum hydrocarbon compound recovery. Environmental flow models in IPARS help scientists to understand groundwater flow, solute transport and its interaction with surface flow.
- *NetSolve* provides the ability to efficiently and securely access hardware and software computational resources that need not be administered by the users of the resources.

Using NetSolve, we are able to provide a simple, user-friendly interface to the complex IPARS simulator and run it on clusters of parallel processors far more powerful than the client host on which the interface is accessed.

- The *Discover computational collaboratory* enables collaborative monitoring and control of large distributed Grid applications. Using Discover, we are able to seamlessly and securely access, monitor, interact with and steer the IPARS simulation as it runs on the Grid. Discover's detachable portals also provide collaboration capabilities.

The rest of this paper is organized as follows. Sections 2–4 provide overviews of IPARS, NetSolve and Discover, respectively. Section 5 details the integration of these systems to provide a single, coherent environment that allows scientists to collaboratively launch, monitor and steer advanced oil reservoir simulations on a powerful computational Grid. The paper concludes with a discussion of the resulting infrastructure and the implications of the work presented in Section 6.

2. OIL RESERVOIR SIMULATION USING IPARS

An oil reservoir is a porous geologic formation that contains at least one hydrocarbon (oil or gas) phase in addition to water within its pore space [3]. Examination of the physical processes such as transport, phase change and reaction in an oil reservoir are primary interests among petroleum engineers and scientists. Normally, scientists admit macroscopic description of the porous matrix to study the physics of fluid flow through it. More precisely, a general approach describes the aforementioned phenomena at a small length scale that is yet larger than the linear dimension of the pore [4]. With a mathematical definition of porous medium (see Chapter 2 of [3] for complete details), one can properly treat phenomena in porous media using the continuum approach. Using a porous medium model, the equations of conservation of mass, momentum and energy in a fluid continuum are averaged to deduce the governing equations of the fluid flow through porous media. A reservoir simulation employs this mathematical statement, that is a system of partial differential equations and a set of initial and boundary conditions, to numerically solve the equations efficiently.

2.1. IPARS

IPARS [5,6] is a parallel reservoir simulation framework for fluid flows in porous media developed at the Center for Subsurface Modeling (CSM), The University of Texas at Austin. The simulator supports three-dimensional transient flows of multiple phases containing multiple components plus immobile phases (rock and adsorbed components). There are currently *ten physical models* in IPARS, including multiphase gas–oil–water, air–water flow, and reactive transport models. IPARS is primarily implemented in Fortran, C and C++.

Unlike conventional reservoir simulation software, it supports multiple physical models which constitute one oil reservoir. In other words, an oil reservoir can be subdivided into several subdomains and each subdomain can be associated with an independent physical model that best describes the characteristics of it. These models are coupled across interfaces by a set of matching or approximately matching conservation equations and constitutive laws [7]. Also, different subdomains can be simulated with different numerical methods for high computational efficiency. For instance, two-phase

(oil and water) flow model using fully implicit scheme and the same model with sequential model [8] may co-exist for one reservoir simulation task [9].

The basis of multi-model and multi-methods implementation is the multi-block algorithm. The multi-block algorithm in IPARS consists of decomposition of the simulation domain into multiple non-overlapping subdomains or blocks [10]. As we discussed, each block can be associated with its own physical model, numerical method and Grid structure that are independent from neighboring blocks. Furthermore, the computation can be distributed among massively parallel computers or clusters of workstations. The IPARS framework provides all the memory management, message passing, linear solver and nonlinear solution methods, etc. [11,12].

In the following sections, we take a closer look at the mathematics of fluid flow through porous media to reveal the essence of reservoir simulation and the computational complexities associated with it. However, those seeking an in-depth description of the matters should be directed to [3,8,13] and the references therein.

2.2. Mathematical models of fluid flow in an oil reservoir

The IPARS framework supports three-dimensional transient flow of multiple phases with various components through immobile phases (rock/soil). For the sake of brevity, we present one of the most simple forms of governing equations, that is the mathematical formulation of oil–water two-phase flow through an oil reservoir, used in IPARS.

Immiscible multi-phase flow [14] through a porous media can be described using the mass conservation equation for each fluid phase f, in which the convection occurs according to Darcy's phase velocity [3],

$$\frac{\partial(\phi\rho_{\mathrm{f}}S_{\mathrm{f}})}{\partial t} = \nabla \cdot U_{\mathrm{f}} + q_{\mathrm{f}} \tag{1}$$

Here the subscript f represents fluid phase, f = w, o, for water and oil, respectively. ϕ denotes porosity of the porous media, which may vary with water pressure and location. ρ_{f} is the density of phase f. S_{f} and U_{f} are the saturation and the mass flux of phase f, respectively. Source and sink are applied as q_{f}, which represents mass flow rate; the mass flow rate is negative at production wells and positive at injection wells in (1).

The aforementioned Darcy's law is

$$U_{\mathrm{f}} = -\frac{K k_{\mathrm{rf}}}{\mu_{\mathrm{f}}}\rho_{\mathrm{f}}(\nabla P_{\mathrm{f}} - \rho_{\mathrm{f}} g \nabla D) \tag{2}$$

In (2), K is the absolute permeability and k_{rf} is the relative permeability of phase f. Relative permeabilities are functions of the wetting phase saturation and location. μ_{f} is the viscosity and P_{f} is the pressure of the fluid phase f. Gravity is denoted by g and D is the depth from datum. Gravity can be considered in an arbitrary direction in this model.

Equation (1) provides a set of two partial differential equations governing the simultaneous, immiscible flow of water and oil. These equations contain four dependent variables: P_{w}, P_{o}, S_{w} and S_{o}. Therefore, two additional equations are needed to provide a unique solution to the system. They are

$$S_{\mathrm{w}} + S_{\mathrm{o}} = 1 \tag{3}$$

$$P_{\mathrm{c}}(S_{\mathrm{w}}) = P_{\mathrm{o}} - P_{\mathrm{w}} \tag{4}$$

The sum of volume fractions of each phase is equivalent to 1 (3). Capillary pressure, P_c, is introduced by (4). It may vary with location and water saturation.

Fluid density, ρ_f, is defined by the mass of the fluid per unit volume. In general, it depends on fluid pressure, P_f, and temperature, T; however, we adopt a simpler constitutive law. We assume that the density of water phase can be described with water compressibility.

$$\rho_w(P_w) = \rho_{wo} \exp(c_w(P_w - P_{wo})) \tag{5}$$

where the subscript o in (5) indicates an initial steady state at standard condition. c_w is the compressibility of water phase and regarded as a constant. The density of oil phase, ρ_o, can be defined similarly.

We use the state of pressure equilibrium as for the initial condition. From the input data, water pressure and water saturation at a specific depth are obtained. Pressure and saturation of each Grid element in the entire domain can be calculated for the oil and water phases using (3)–(5). At static equilibrium, all phase fluxes are zero and phase pressures and saturations are distributed for depth only. Above the water table, a two-phase region exists in equilibrium with the assigned capillary pressure and water saturation relationship. Below the water table, oil saturation is set to zero.

Both pressure specified and no-flow boundaries can be employed. For instances, the outer infinite-acting boundary is represented by a no-flow boundary condition by setting transmissibility to zero at the boundary. A reservoir overlaid on an impermeable base can be imposed with a no-flow condition. Mathematically, the potential gradient normal to the boundary is set to zero. For complete detail, see [5,15,16].

In (1), q_f is a source or sink term. IPARS supports an arbitrary number of vertical source and sink with specified bottom pressure or mass flow rate. A well may penetrate through the full vertical domain or only a part of it. A source is able to contain only a single phase (water), but a sink may have both water and oil phases. The radial geometry is used for the source and sink so that they are equivalent to an injection and production well. IPARS well models were developed by Wheeler [5].

Here we present a detailed discussion of the well models used in IPARS since the interaction and steering of the IPARS simulations are driven via well parameters, which are the computational objects to be discussed in Section 4.

The mass flow rate of phase f from source to a Grid element i is

$$q_{f,i} = \rho_{f,i} G_i \Delta x_i K_i k_{rf,i} (P_{WB} - \bar{P}_{f,i})/\mu_{f,i} \tag{6}$$

where G_i is dimensionless geometric factor. It is defined by

$$G_i = 2\pi/(\ln(r_{eq}/r_w) + s) \tag{7}$$

$$r_{eq} = 0.208 \times \sqrt{\frac{\text{Element volume}}{\text{Wellbore length in the Grid element}}} \tag{8}$$

Here r_{eq} is equivalent radius of the Grid element center, which is a strong function of Grid size. r_w is wellbore radius and s is the skin factor. P_{WB} is wellbore pressure at the center of the open interval in element i. This quantity is related to bottom pressure by

$$P_{WB} = P_{BH} + \rho_{WB} g(D_{WB,i} - D_{BH}) \tag{9}$$

where subscript BH denotes bottom hole and WB denotes wellbore. Therefore, ρ_{WB} is average fluid density in the wellbore and $D_{\mathrm{WB},i}$ is the depth at the center of the wellbore open interval in element i.

$\bar{P}_{\mathrm{f},i}$ is formation pressure of phase f in element i at the depth of the center of the open interval in element i. It may be necessary to adjust the element center pressure for depth:

$$\bar{P}_{\mathrm{f},i} = P_{\mathrm{f},i} + \rho_{\mathrm{f},i} g (D_{\mathrm{WB},i} - D_i) \tag{10}$$

where $P_{\mathrm{f},i}$ is phase pressure at the center of element i. Therefore, the mass flow rate of a single phase source is

$$q_{\mathrm{f}} = \sum_i \left(\frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i}}{\mu_{\mathrm{f},i}} \right) P_{\mathrm{BH}} + \sum_i \left(\frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} (D_{\mathrm{WB},i} - D_{\mathrm{BH}})}{\mu_{\mathrm{f},i}} \right) g \rho_{\mathrm{WB}} - \sum_i \left(\frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} (P_{\mathrm{f},i} + \rho_{\mathrm{f},i} g (D_{\mathrm{WB},i} - D_i))}{\mu_{\mathrm{f},i}} \right) \tag{11}$$

In order to make a sink term formulation, we need to consider a two-phase flow model. By definition, the mass flow rate q is negative for sink. Wellbore density of sink term is

$$\rho_{\mathrm{WB}} = q_{\mathrm{T}} \Big/ \sum_{\mathrm{f}} (q_{\mathrm{f}} / \rho_{\mathrm{f,WB}}) \tag{12}$$

where $q_{\mathrm{T}} = \sum_{\mathrm{f}} q_{\mathrm{f}}$ is total mass flow rate and ρ_{f} is the density of phase f, which is calculated by

$$\rho_{\mathrm{f,WB}} = \rho_{\mathrm{f}} (P_{\mathrm{BH}} + g \rho_{\mathrm{WB}} (D_{\mathrm{WB}} - D_H)) \tag{13}$$

Phase density ρ_{f} can be evaluated by successive replacement using a value of ρ_{WB} from the previous nonlinear iteration or timestep in (12). At the beginning of the first timestep, the wellbore density required for this purpose can be obtained from

$$\rho_{\mathrm{WB}} = \frac{\sum_{\mathrm{f}} \sum_i G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} / \mu_{\mathrm{f},i}}{\sum_{\mathrm{f}} \sum_i G_i \Delta x_i K_i k_{\mathrm{rf},i} / \mu_{\mathrm{f},i}} \tag{14}$$

Since flow rates are unknown values, wellbore density cannot be directly computed from (12). The explicit form of ρ_{WB} can be derived by combining (11) and (13).

$$\rho_{\mathrm{WB}} = \frac{(-B + \sqrt{B^2 - 4AC})}{2A} \tag{15}$$

where

$$A = g \sum_{\mathrm{f}} \left(\sum_i \left(\frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} (D_{\mathrm{WB},i} - D_{\mathrm{BH}})}{\mu_{\mathrm{f},i}} \right) \Big/ \rho_{\mathrm{f}} \right)$$

$$B = -g \sum_{\mathrm{f}} \left(\sum_i \frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} (D_{\mathrm{WB},i} - D_{\mathrm{BH}})}{\mu_{\mathrm{f}}} \right) + \sum_{\mathrm{f}} \left(\sum_i \frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i}}{\mu_{\mathrm{f},i}} P_{\mathrm{BH}}, - \sum_i \frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} (P_{\mathrm{f},i} + \rho_{\mathrm{f},i} g (D_{\mathrm{WB},i} - D_i))}{\mu_{\mathrm{f},i}} \right) \Big/ \rho_{\mathrm{f}}$$

$$C = -\sum_{\mathrm{f}} \left(\sum_i \frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i}}{\mu_{\mathrm{f},i}} \right) P_{\mathrm{BH}} + \sum_{\mathrm{f}} \left(\sum_i \frac{G_i \Delta x_i K_i k_{\mathrm{rf},i} \rho_{\mathrm{f},i} (P_{\mathrm{f},i} + \rho_{\mathrm{f},i} g (D_{\mathrm{WB},i} - D_i))}{\mu_{\mathrm{f},i}} \right)$$

The mass flow rates from each element are now evaluated explicitly using (6). Total flow rates from the sink must be checked with (11) to ensure the net flow of each phase out of the formation.

2.3. Numerics and solution procedure

Using finite difference method, a system of partial differential equations (1) is solved simultaneously for the primary unknown variables, namely *simultaneous solution method*. Among the choices of the primary unknowns, we pick P_w and S_w as primary variables by a reason which will be given later in this section.

The simultaneous solution method was first proposed by Garvin *et al.* [17] and later extended by Peaceman *et al.* [18], Terhune *et al.* [19] and Sheffield [20]. This method was initially applied to multiphase immiscible flow problems where there is no mass exchange between the phases. The basic idea is to generate the expansion of the derivatives with respect to primary variables in the conservation equations.

Properties of the simultaneous solution method have been examined for many types of nonlinear multiphase problems and fully implicit schemes have been used successfully. A fully implicit approach for the solution of the immiscible flow equations was first introduced by Blair and Weinaug [21]. Aziz and Settari [22] performed a stability analysis of the linearized semi-implicit and fully implicit methods and concluded that they all satisfy material balance and are unconditionally stable. Peaceman showed a more refined, nonlinear stability analysis of the linearized method [8].

With a fully implicit scheme, the discretization leads to a set of nonlinear algebraic equations and these equations should be solved iteratively for computational efficiency. A full Newton iteration procedure is used, in which a system of nonlinear equations is approximated by a system of linear equations. The residual vector is the subtraction of the right-hand side from the left-hand side of the mass conservation equation (1). It should be noted that we discretize the mass conservation equations (1) on the rectangular Grid geometry and use the one point upstream weighting scheme for the transmissibility terms.

We have a certain freedom to choose primary unknowns as long as they span the solution space. However, appropriately chosen primary variables shall deduce a simple Jacobian formulation as well as computationally cost-effective algorithm. Here we select water pressure, P_w, and water saturation, S_w, as primary variables. These two variables make the formulation of the elements of the corresponding Jacobian matrix simple, and so does the evaluation of those elements at each iteration.

By applying the idea mentioned above, the resulting system equation is

$$\Delta \mathbf{u}_m = -(J_m(\mathbf{u}_m))^{-1} F(\mathbf{u}_m) \tag{16}$$

Here $\mathbf{u}_m = (S_w^m, P_w^m)^T$, $\Delta \mathbf{u}_m = (\Delta S_w^{m+1}, \Delta P_w^{m+1})^T$, and $\Delta S_w^{m+1} = S_w^{m+1} - S_w^m$, $\Delta P_w^{m+1} = P_w^{m+1} - P_w^m$. m represents mth Newtonian iteration level, F is a residual vector from the oil and water mass conservation equation, and J_m is Jacobian matrix of residual vector F evaluated at $\mathbf{u}_k$ at time level n. Here S_w^m is a representing value of the ijk Grid element, i.e. $S_w^m = S_{w,ijk}^m$ and $P_w^m = P_{w,ijk}^m$.

To construct the set of Grid elements constituting the simulation domain, we used a rectangular Grid formulation. A right prism with small control volume is our basic Grid element or cell. All the prisms can be identical or each prism can have different lengths (Δx_i), widths (Δy_j), and/or heights (Δz_k) for ijk cell.

A cell centered finite differencing is used. It can be understood as a mixed finite element with quadrature as described in [23,24]. The position of the cell center is the arithmetic mean of the lengths of the edges of the two adjacent prisms. For the absolute permeability at the cell center, harmonic

averages of the absolute permeabilities between two elements are used:

$$\Delta x_{i+1/2} = (\Delta x_i + \Delta x_{i+1})/2$$
$$\Delta y_{j+1/2} = (\Delta y_j + \Delta y_{j+1})/2$$
$$\Delta z_{k+1/2} = (\Delta z_k + \Delta z_{k+1})/2$$
$$\lambda_{\mathrm{f}} = \frac{K k_{\mathrm{rf}}}{\mu_{\mathrm{w}}}$$
$$H_{i+1/2,jk} = 2\Delta y_j \Delta z_k \left(\frac{\Delta x_i}{K_{x,i}} + \frac{\Delta x_{i+1}}{K_{x,i+1}} \right)^{-1}$$

for fluid phase f. $H_{i,j+1/2,k}$ and $H_{ij,k+1/2}$ are similarly defined.

Following the idea mentioned above and the general procedure of the cell centered finite differencing or a mixed finite-element method with quadrature [23], (1) and (2) may be fully implicitly discretized as follows:

$$\begin{aligned}
&\Delta x_i \Delta y_j \Delta z_k [(\phi \rho_{\mathrm{f}} S_{\mathrm{f}})^{n+1,m+1} - (\phi \rho_{\mathrm{f}} S_{\mathrm{f}})^n]_{ijk} \\
&\quad = \Delta t^n \Delta x_i \Delta y_j \Delta z_k q_{\mathrm{f},ijk}^{m+1} \\
&\qquad + \Delta t^n \Delta y_j \Delta z_k \left\{ \lambda_{\mathrm{f},i+1/2} \frac{P_{\mathrm{f},i+1} - P_{\mathrm{f},i}}{\Delta x_{i+1/2}} - (\lambda_f \rho_{\mathrm{f}} g)_{i+1/2} \frac{D_{i+1} - D_i}{\Delta x_{i+1/2}} \right. \\
&\qquad \left. - \lambda_{\mathrm{f},i-1/2} \frac{P_{\mathrm{f},i} - P_{\mathrm{f},i-1}}{\Delta x_{i-1/2}} + (\lambda_{\mathrm{f}} \rho_{\mathrm{f}} g)_{i-1/2} \frac{D_i - D_{i-1}}{\Delta x_{i-1/2}} \right\}_{jk}^{n+1,m+1} \\
&\qquad + \Delta t^n \Delta z_k \Delta x_i \left\{ \lambda_{\mathrm{f},j+1/2} \frac{P_{\mathrm{f},j+1} - P_{\mathrm{f},j}}{\Delta y_{j+1/2}} - (\lambda_{\mathrm{f}} \rho_{\mathrm{f}} g)_{j+1/2} \frac{D_{j+1} - D_j}{\Delta y_{j+1/2}} \right. \\
&\qquad \left. - \lambda_{\mathrm{f},j-1/2} \frac{P_{\mathrm{f},j} - P_{\mathrm{f},j-1}}{\Delta y_{j-1/2}} + (\lambda_{\mathrm{f}} \rho_{\mathrm{f}} g)_{j-1/2} \frac{D_j - D_{j-1}}{\Delta y_{j-1/2}} \right\}_{ki}^{n+1,m+1} \\
&\qquad + \Delta t^n \Delta x_i \Delta y_j \left\{ \lambda_{\mathrm{f},k+1/2} \frac{P_{\mathrm{f},k+1} - P_{\mathrm{f},k}}{\Delta z_{k+1/2}} - (\lambda_{\mathrm{f}} \rho_{\mathrm{f}} g)_{k+1/2} \frac{D_{k+1} - D_k}{\Delta z_{k+1/2}} \right. \\
&\qquad \left. - \lambda_{\mathrm{f},k-1/2} \frac{P_{\mathrm{f},k} - P_{\mathrm{f},k-1}}{\Delta z_{k-1/2}} + (\lambda_{\mathrm{f}} \rho_{\mathrm{f}} g)_{k-1/2} \frac{D_k - D_{k-1}}{\Delta z_{k-1/2}} \right\}_{ij}^{n+1,m+1} + \Delta t^n q_{\mathrm{f},ijk}^{n,m+1}
\end{aligned} \tag{17}$$

Here the superscripts n and m denote the time level and the Newtonian iteration level, respectively. $\Delta t^n = t^{n+1} - t^n$ is the timestep at time level n. In the discretization written above, all the variables are the functions of both location and time. However, the finite difference equation is not complete since the right-hand side of (17) includes the terms which require $m + 1$ iteration level evaluations of the pressure variable. The $m + 1$ iteration level evaluation can be obtained by the first-order Taylor expansion, for instance, a variable Θ of phase f at the $m+1$ level iteration has the following expression:

$$\Theta_{\mathrm{f}}^{m+1} \approx \Theta_{\mathrm{f}}^m + \frac{\partial \Theta}{\partial P_{\mathrm{f}}} \Delta P_{\mathrm{f},ijk}^{m+1} + \frac{\partial \Theta}{\partial S_{\mathrm{f}}} \Delta S_{\mathrm{f},ijk}^{m+1} \tag{18}$$

All of the terms in Equation (17) should basically experience the same procedures to perform Newtonian iterations.

To end this section, we give a brief description of the simulation: IPARS computes the initial condition of the reservoir. The simulation begins with the given initial timestep size. The system of nonlinear equation (18) is solved which may involve several linear solutions of (17). If the linear solver or the Newton iteration fails, the simulator cuts the timestep size and resolves the system. The timestep size can be increased by a user specified factor if the solutions are found for a certain number of successive steps.

2.4. Computational aspects of IPARS simulations

IPARS simulates oil and gas reservoirs, which often have very complex geological formations. Mathematical and numerical challenges are significant to handle highly nonlinear and coupled flows with multiscale, multiphase, multicomponent and multiphysics features.

Under these circumstances, in order to accurately and efficiently simulate a reservoir with, say, 1000 km^2 lateral area with 100 m depth, the requirements of computational resources are huge. Such a reservoir needs millions of computational Grid elements to be simulated accurately. In light of this, the primary goal of a typical simulator is to support realistic and high-resolution reservoir studies with a million or more Grid elements.

To address this challenge, IPARS is designed for parallel distributed memory machines. On multiprocessor machines, the domain Grid is distributed among the processors such that each processor is assigned a subset of the total Grid system. Dynamic domain decomposition is used to distribute Grid elements among the processors. Although a million-Gridblock hydrocarbon compositional model, simulating a complicated oil recovery process, may have taken months on a decent workstation, the simulation can be done in just an hour on a reasonable parallel computer system, for instance, a 64-node PC cluster.

3. THE NETSOLVE GRID COMPUTING SYSTEM

The primary aim of the NetSolve project is to create an infrastructure that provides seamless, reliable access to loosely-connected, organizationally and/or geographically distributed hardware and software resources. NetSolve enables these services by running a set of service daemons, servers and agents, on participating computer hosts that can be accessed by client programs via the NetSolve programming application programmer interface (API). A brief overview of the NetSolve system is discussed below. A complete discussion of NetSolve and NetSolve-related topics may be found in [25,26]; other NetSolve publications and the software is available at [27].

3.1. NetSolve overview

The typical instance of a *NetSolve Grid* is a set of heterogeneous computer nodes connected via some networking infrastructure. The geographical expanse may range from a single multi-processor environment such as an IBM SP with a high-speed interconnect to widely remote nodes connected via the commodity Internet or exclusive high-performance networks such as Abilene and vBNS.

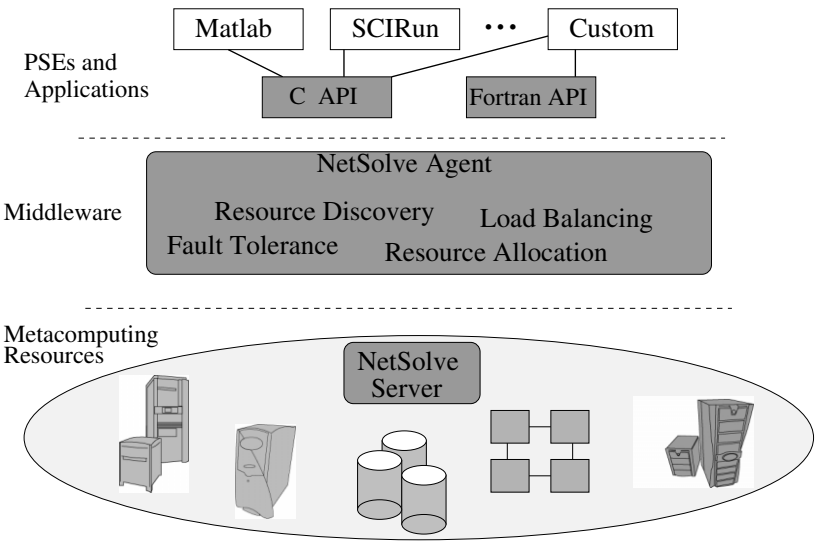


Figure 1. An architectural overview of the NetSolve system.

NetSolve uses a client/agent/server model and is available for all popular variants of UNIX; parts of the system are also available for the Microsoft Windows operating system.

The major components of the NetSolve system are the *NetSolve agent*, an information service and resource manager, the *NetSolve server*, a networked computer configured to allow access to computational hardware and software, and the *NetSolve client* libraries, which allow users to instrument their application code with requests for remote computational services. Figure 1 shows the infrastructure of the NetSolve system and its relation to the applications that use it. NetSolve and systems like it are often referred to as Grid middleware. The shaded parts of the figure represent the NetSolve components that act as the intermediate or 'middle' layer that binds the application or user to the hardware and software services needed to perform useful computations. A number of command-line and Web-based tools exist to allow NetSolve system administrators (maintainers of running NetSolve servers and agents) to detect and modify the system's configuration. These tools allow for querying the configuration and modifying it by terminating agents or servers, or reconfiguring servers with new or different services, etc. The NetSolve philosophy is that client-users need not be administrators of a NetSolve server pool. Pools can be hosted by individuals, departments or organizations and the appropriate administrators can use access control mechanisms to determine the 'openness' of the pool, i.e. which clients can access which servers/services. Furthermore, servers from

multiple administrative domains can be combined into a single pool—each individual server having the ability to selectively render services to clients[‡].

3.2. The NetSolve components

The NetSolve client API libraries allow an application to leverage the computational hardware and software resources served by the NetSolve Grid using a remote procedure call (RPC) model. In the simplest case, NetSolve transmits input data from a client to a server, performs the requested service and transports output data back to the requesting client. NetSolve supports many data types, including vectors, matrices and sparse matrices of native types. Client requests may be synchronous (blocking) or asynchronous (non-blocking) and NetSolve supports many other features to provide reliability, fault tolerance and optimal performance.

A computer host becomes a NetSolve server when it is configured to run the NetSolve server daemon. During server initialization, the server determines the hardware capabilities (performance rating, number of processors, etc.) and software services it has been configured to serve. The server registers this information with the NetSolve agent and then waits for incoming service requests. The configuration parameters that a NetSolve server administrator may specify include a server's CPU load threshold, scratch space for temporary files, number of simultaneous service requests permitted and the domains and users from which requests will be accepted.

To keep NetSolve as general as possible, we define a formal problem description that allows server administrators to describe new services they wish to provide via their servers.

From a client's perspective, a problem is a 3-tuple: <*name*, *inputs*, *outputs*> where:

- *name* is a character string containing the name of the problem;
- *inputs* is a list of input objects;
- *outputs* is a list of output objects.

These items are specified in a problem description file (PDF) that also specifies libraries containing the implementations of any underlying functions or services being interfaced by NetSolve and a code that uses some pre-defined macros to extract data elements from the NetSolve input objects, pass them to the underlying service functions, and place output results back into NetSolve output objects. A tool called the `code generator` parses this PDF to create actual C code that is then linked with NetSolve auxiliary libraries (that transfer data to/from client programs and handle other generic service tasks) and those libraries specified by the PDF to create a *service executable*. NetSolve's PDF mechanism is similar to IDLs as used in systems such as CORBA.

The *NetSolve agent* acts as a client's entry point into the system and also manages server resources. As an information service, the agent maintains a comprehensive view of the status of all NetSolve server components and their interactions with clients. It keeps track of the software capabilities of the servers and is able to schedule client requests to appropriate servers. As a resource allocator at request time, the agent uses server-provided performance ratings and dynamic network benchmarks to

[‡]Such a NetSolve pool, encompassing machines all over the world is accessible via the NetSolve agent running at http://netsolve.cs.utk.edu. This system is an extremely open system accessible to anyone using the NetSolve system.

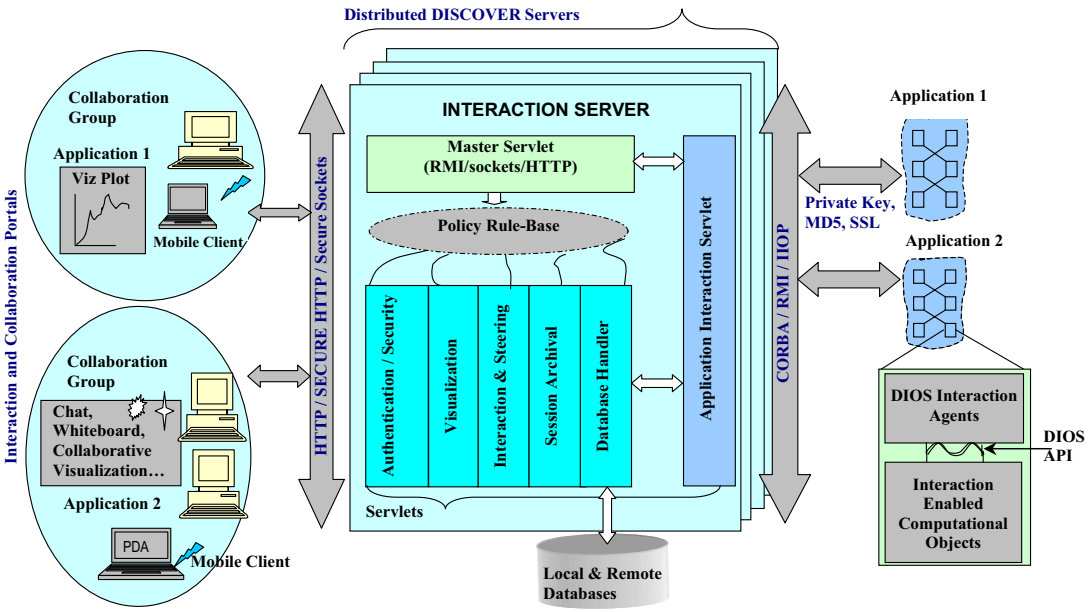


Figure 2. Architectural schematic of the Discover computational collaboratory.

calculate an estimated turn-around time for each service request. The most efficient and capable server is elected for the task.

4. THE DISCOVER COMPUTATIONAL COLLABORATORY

Discover [28] is an interactive computational collaboratory that combines Grid-enabled middleware services, application control networks with sensors and actuators and pervasive portals, and enables seamless and secure access to geographically distributed resources, applications and data on the Grid. Using the Discover collaboratory, scientists and engineers can discover and access applications and services on the Grid as long as they have appropriate privileges and capabilities. Furthermore, they can form or join collaboration groups and can securely, consistently and collaboratively monitor, interact with and steer these applications based on their privileges and capabilities. Discover is currently operational and is being used to provide interaction capabilities to a number of scientific and engineering applications, including oil reservoir simulations, computational fluid dynamics, seismic modeling and numerical relativity.

4.1. Discover architecture

A conceptual overview of the Discover architecture is presented in Figure 2. It is composed of three key components: (1) middleware substrate that integrates Discover interaction and collaboration servers

and enables interoperability with external Grid services; (2) application control network combining sensors, actuators and interaction agents to enable interactive monitoring and steering of distributed applications; and (3) pervasive detachable portals for collaborative access to the Grid applications. These components are briefly described below.

4.1.1. Discover middleware substrate

The Discover middleware substrate [29] defines interfaces and mechanisms for a peer-to-peer integration and interoperability of services provided by domain specific collaboratories on the Grid. This means an interoperability between geographically distributed instances of the Discover collaboratory. Furthermore, the middleware substrate integrates Discover collaboratory services with the Grid services provided by the Globus Toolkit [30] using the CORBA Commodity Grid (CORBACoG) Kit [31]. Clients can use the middleware substrate to locate available resources on the Grid, allocate required resources, run applications on these resources, and connect to and collaboratively monitor, interact with and steer these applications. The middleware substrate enables Discover interaction and steering servers as well as Globus servers to dynamically discover and connect to one another to form a peer network. This allows clients connected to their local servers to have global access to all applications and services across all the servers in the network based on their credentials, capabilities and privileges. Details about the implementation and operation of the current Discover middleware substrate can be found in [29,32].

4.1.2. DIOS interactive object framework

DIOS is a distributed object infrastructure that enables the development and deployment of interactive Grid applications. It addresses three key challenges: (1) definition and deployment of interaction objects that extend distributed and dynamic computational objects with sensors and actuators for interaction and steering; (2) definition of a scalable control network that interconnects interaction objects and enables object discovery, interrogation and control; and (3) definition of an interaction gateway that enables remote clients to access, monitor and interact with applications.

DIOS is composed of two key components: (1) interaction objects that encapsulate sensors and actuators; and (2) a hierarchical control network composed of *Discover agents*, *base stations* and an *interaction gateway*. Interaction objects extend the application's computational objects (data structures used by the application) with monitoring and steering capabilities by providing abstractions for creating sensors and actuators. Interaction objects are created by deriving the computational objects from a virtual interaction base class of the interaction object library. The derived objects define a set of *views* that they can provide and a set of *commands* that they can accept. Interaction objects can be either local to a single computational node, distributed across multiple nodes or shared between some or all of the nodes. Furthermore, interaction objects can be dynamically created or deleted during application execution, can migrate between computational nodes or modify its distribution by notifying appropriate interaction agents.

The control network is a hierarchical structure composed of: (1) Discover agents on each node; (2) base stations for each interaction cell; and (3) an interaction gateway that connects to the interaction server and provides a proxy to the entire application. The network is automatically configured at run-time using an underlying messaging environment (e.g. MPI) and the available number of processors.

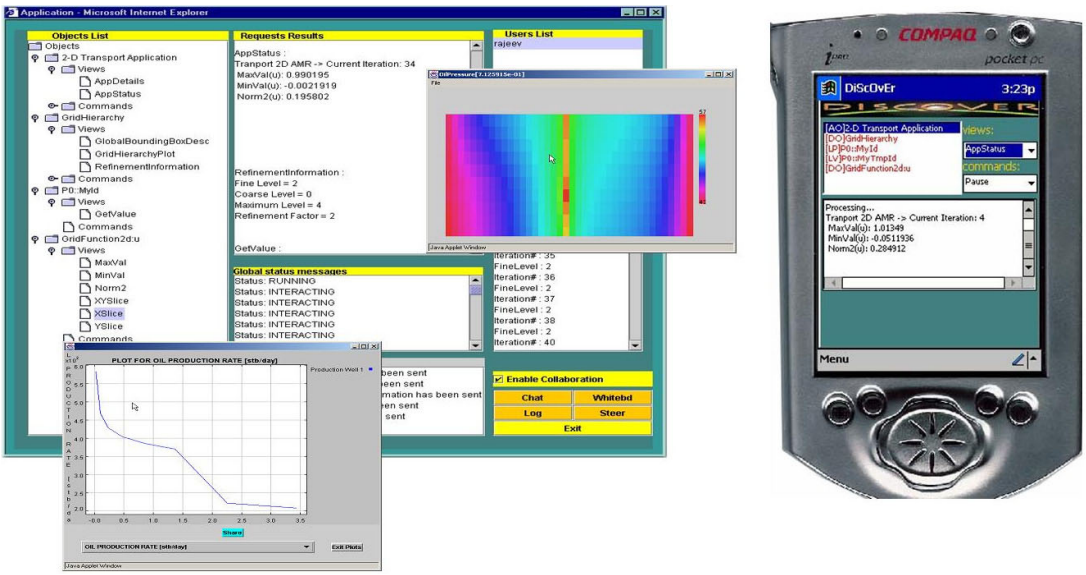


Figure 3. The Discover collaborative interaction/steering portal.

Discover agents, base stations and the gateway each maintain registries of interaction objects registered in their respective domains (node, cell, entire application, respectively). The gateway is additionally responsible for interfacing with the interaction server, delegating interaction requests to the appropriate interaction agents (Discover agents and/or base stations), and collecting their responses. In the case of distributed objects, the gateway also performs a gather operation for collating the responses arriving from the corresponding nodes. A recent extension to DIOS allows clients to define and deploy rules to automatically monitor and control applications and/or application objects. The conditions and actions of the rules are composed using the exported view/command interfaces. A distributed rule-engine is built in the control network that authenticates and validates incoming rules, decomposes the rules and distributes triggers to appropriate application objects, and manages the execution of the rules. A more detailed description of the DIOS framework can be found in [33].

4.1.3. Discover interaction and collaboration portals

The Discover portal enables remote, collaborative access to applications, application objects and Grid services and provides scientists and engineers with an anytime/anywhere capability for seamlessly and securely launching, accessing, monitoring and controlling Grid applications. It also provides a replicated shared workspace architecture and integrates collaboration tools such as chat and whiteboard. Furthermore, it integrates ‘collaboration streams’, which maintain a navigable record of all client–client and client–applications interactions and collaboration. A screenshot of the current Discover portal is presented in Figure 3.

5. ENABLING INTERACTIVE OIL RESERVOIR SIMULATIONS ON THE GRID

In this section, we describe the integration of our three building blocks (IPARS, NetSolve and Discover) outlined above, and present the engineering of a prototype interactive and collaborative problem-solving environment for Grid-based oil reservoir simulations. Specifically, we first describe how the IPARS simulator is enhanced by Discover to enable runtime simulation analysis, interaction and collaboration. We then present the transformation of the Discover-enhanced IPARS simulator into a NetSolve service. We conclude this section by demonstrating how the completed integration environment may be used in practice.

5.1. Interactive and collaborative IPARS simulations using Discover

The goal of the IPARS–Discover integration is to transform traditional batch IPARS simulations into more interactive and collaborative simulations. Interactive IPARS simulations provide many benefits. For example, petroleum production engineers working with oil reservoir simulations often want to change the number of active wells, their types, their locations and some of the conditions in field activity. Similarly, environmental hydrologists working with simulations of the transport of groundwater over a period of dry and rainy seasons can effectively study a real-world situation by interactively changing conditions at soil boundaries.

The IPARS–Discover integration augments IPARS with runtime monitoring and tracking capabilities that provide users with a 'window-in-the-oven' view of the simulations, and allow them to visualize and track the simulation domain. Furthermore, it provides interactive steering capabilities that allow the state of simulation to be controlled and managed at runtime. In order to ensure 'safe steering' and that the state of the simulation remains consistent, users are only allowed to control specific parameters and objects and in controlled ways. Furthermore, interaction/steering is only allowed at well-defined interaction points in the simulation. For instance, one cannot arbitrarily change the values of the primary unknowns as this can cause the results to be completely non-physical, and could break convergence properties or possibly the whole simulation. Similarly, if users are allowed to change parameters in the middle of time steps, an abrupt change of a variable may make the linear system ill-conditioned.

In our implementation, interaction points and interaction objects/parameters are defined by the algorithm/simulation designer so as to ensure consistency and provide maximum flexibility within the application domain. In the case of IPARS, wells and boundary conditions are the candidate objects for steering. The simulation of the petroleum engineering application is driven by the wells placed at different geographical locations with different settings. Critical simulation parameters for wells include the number of wells, the type of the wells (e.g. pressure specified water injection, oil mass rate specified production, etc.), the well-bore diameter, the injection/production rate and the bottom hole depth of the well, etc.

The overall IPARS–Discover integration consists of four steps. The process is described in the following sections.

(1) *Registration*. IPARS simulations are registered with the Discover system using the Discover Web registration form to receive a unique identifier for each simulation. Authorized clients and their capabilities and access privileges are specified during registration.

(2) *Creation of interaction objects*. The DIOS library is used to construct interaction objects from IPARS computational data-structures and to export their interaction interfaces to a Discover server.
(3) *IPARS simulation execution*. Once the simulation is launched, the IPARS identifier obtained during registration is used to authenticate the application with the Discover middleware. During execution, application information is automatically exported and updated at the servers using the DIOS control network.
(4) *IPARS simulation interaction and steering*. Distributed clients discover and connect to running IPARS simulations using the Discover Web portals. Clients can initiate or join interaction and/or collaboration sessions and can monitor, interact with and steer the simulation based on their capabilities and privileges.

5.1.1. Constructing interactive IPARS simulations

Enabling the monitoring and interactive steering of parallel and distributed applications requires the definition and deployment of *sensors* and *actuators* required to monitor and control the application objects (i.e. the wells and boundary conditions in IPARS simulations). Defining these interaction interfaces and mechanisms in a generic manner, and co-locating them with the application's computational objects can be non-trivial. This is because the structures of application computational objects vary significantly, and the objects can span multiple processors and address spaces. Another issue is the construction of a *control network* that interconnects these sensors and actuators so that commands and requests can be routed to the appropriate set(s) of computational objects (depending on the current distribution of the object), and the returned information can be collated and coherently presented. Finally, the interaction and steering interfaces presented by the application need to be exported so that they can be accessed remotely, using standard distributed object protocols, to enable application monitoring and control.

The construction of interactive IPARS simulations is described below. Note that IPARS simulations are primarily Fortran-based requiring that the application data structures be *wrapped* by C++ wrappers before they can be integrated with DIOS. This process involves the following steps for every data structure that needs to be monitored and steered.

(1) Identify the computational data that need to be monitored and steered.
(2) Objectify the data structure, i.e. create C++ object wrappers for each computational data-structure. Care must be taken while mapping Fortran datatypes and arrays to C++.
(3) Derive the newly formed class from the appropriate DIOS base class depending on whether it is distributed or not.
(4) Override the virtual function `exportInterfaces()` to export the views and commands for the data structure using the DIOS API functions `addView()` and `addCommand()`.
(5) Override the virtual function `processMessage()` to process interaction requests.
(6) Create instances of the interaction objects within the application and register them with the DIOS interaction agents.

These steps are described below.

5.1.1.1. Identifying computational objects. The first step in making an application interactive is to identify the important data structures of the application that are necessary to steer the simulation. In oil reservoir simulations, wells placed at different geological locations drive the dynamics of fluids flow inside the reservoir. The transient behavior of the fluids can be widely different depending on the number of wells, their types, and their characteristics. As a result, these are the critical simulation parameters that should be monitored and steered. Therefore, we identify an oil well as a computational object that can be used to drive the simulation and will use the IPARS well as an example in the discussion below. Key IPARS simulator well parameters (defined as Fortran data types) are summarized below.

```
WELLTOP(,s,t) = X,Y,Z OF TOP OF INTERVAL s FOR WELL t (FEET)
WELLBOT(,s,t) = X,Y,Z OF BOTTOM OF INTERVAL s FOR WELL t (FEET)
WELLDIAM(s,t) = WELLBORE DIAMETER OF INTERVAL s FOR WELL t (FEET)
DEPBOT(t) = BOTTOM HOLE DEPTH OF WELL t (FEET)
DEPTOP(t) = TOP HOLE DEPTH OF WELL t (TOP OF COMPLETION) (FEET)
NUMWEL = NUMBER OF WELLS
NWELPRC(t) = PROCESSOR THAT OWNS WELL t
WELNAM(t) = WELL NAME OF WELL t
WELBHP(t) = WELL BOTTOM HOLE PRESSURE IN WELL t (PSI)
KWELL(t) = CURRENT WELL TYPE OF WELL t (e.g. SHUT IN, WATER INJECTION,
= PRODUCTION, GAS INJECTION, etc.)
TITHIS(h) = DESCRIPTION OF WELL HISTORY DATA TYPE h
= (e.g. WATER/GAS INJECTION RATE, OIL/WATER PRODUCTION RATE,
= GAS PRODUCTION RATE, WATER/OIL RATIO, GAS/OIL RATIO, etc.) ..........
..........
```

5.1.1.2. Creating object wrappers. Now that we have identified the critical well parameters that are required for driving the simulation, we proceed to create a computational well object. In order to do this, we develop one or more C++ wrapper classes that will contain the relevant well data/parameters. This process consists of the following steps.

(1) The Fortran data elements are mapped to corresponding types in C/C++. For example `REAL*4` is mapped to `float`, `REAL*8` to `double` and `INTEGER` to `int`.
(2) A C++ class is created that maintains references to all relevant parameters with an IPARS simulation. This class is the first class instantiated when the application initializes and is called the `IPARSApplication` class. It is instantiated only once at startup. Subsequent classes needed to maintain simulation data use these references. A bare bone skeleton of the class is listed below (note that not all data members are shown here). It can be seen that the `IPARSApplication` class has a constructor that takes in all the relevant references to the data elements of the simulation.

```
#include <Discover.h>
class IPARSApplication
{
private :
        char* welnam; /* name of well */
```


```
        int* numwel; /* well number */
        float* deptop; /* top hole depth of well */
        float* depbot; /* bottom hole depth of well */
        int* kwell; /* current well type of well */
        int* nwelprc; /* processor that owns this well */
        float* welldiam;/* wellbore diameter for each interval */
        float* welltop; /* x,y,z of top of each interval for well t */
        float* wellbot; /* x,y,z of bottom of each interval for well t */
        double* welbhp; /* well bottom hole pressure */
        .......
public :
       IPARSApplication(char* welname, int* numwell, float* topdep,
                        float* botdep, int* welltype, int* nprcwell,
                        float* welldia, float* welltop, float* wellbot,
                        double* wellbhp, ...);
       ~ IPARSApplication();
};
```

(3) Now, a class is created that contains all information necessary to maintain the simulation parameters of a well. This class is called the `IPARSWell` class and only contains the information pertaining to a single well. It is constructed from the `IPARSApplication` class. A typical simulation will have multiple instances of the `IPARSWell` class, which may be dynamically created at runtime. Note that this class is derived from an appropriate DIOS base class depending on whether the object is distributed or not. A sample `IPARSWell` class is listed below.

```
#include <Discover.h>
class IPARSWell : public DIOS_IObject
{
private :
        char* welnam; /* name of well */
        int numwel; /* well number */
        float deptop; /* top hole depth of well */
        float depbot; /* bottom hole depth of well */
        int kwell; /* current well type of well */
        int nwelprc; /* processor that owns this well */
        float welldiam; /* wellbore diameter for each interval */
        float welltop; /* x,y,z of top of each interval for well t */
        float wellbot; /* x,y,x of bottom of each interval for well t */
        double welbhp; /* well bottom hole pressure */
        .......
public :
       IPARSWell(int wellnum, IPARSApplication& theApp);
       /* all member variables are initialized using the
       IPARSApplication pointer */
       ~ IPARSWell();
       /* All other member functions here */
       .......
       /* overridden virtual functions */
       .......
};
```


5.1.1.3. Creating sensors and actuators. The next step consists of adding view (sensors) and command (actuators) interfaces to each computation object created. In the case of IPARS, this involves adding *methods* to the `IPARSWell` class that can be used to monitor and steer each well object in the simulation.

Adding view interfaces. The views that an object exports depend on the nature of object and the types of interactions desired. We illustrate the process below using simple examples. More complex monitoring functionalities, for example, generating two- and three-dimensional visualization plots or movie files of the simulation process, can be similarly defined. In our example, we add the following methods to the `IPARSWell` class for processing view requests.

(1) A method to extract complete information for the well object. This information is a formatted text stream that can be easily interpreted and displayed at the client end.
(2) A method to extract only the well information that is computed in the current iteration of simulation.
(3) A method to generate two-dimensional plots for well parameters such as oil/water production ratio, oil production rate, etc. These parameters are plotted as evolving parametric plots and provide information about the behavior of the well.

Adding command interfaces. Command interfaces can be similarly defined to enable the desired interaction by adding methods to modify relevant well parameters. In the example below, commands are added to enable each of the well parameters to be controlled.

The resulting `IPARSWell` class with sensors and actuators defined is listed below.

```
class IPARSWell : public DIOS_IObject
{
private :
        /* all member variables */
        .......
public :
       /* constructors, etc. */
       .......
       /* overridden virtual functions */
       void exportInterfaces();
       void processMessage(Message* msg);
       /* VIEW INTERFACES */
       String WellDescr();
       String PlotWellParams();
       .......

       /* COMMAND INTERFACES */
       String SetTopHoleDepth(char*argv[]);
       String SetBottomHoleDepth(char*argv[]);
       String SetWellType(char*argv[]);
       String SetWellBoreDiameter(char*argv[]);
       String SetWellTop(char*argv[]);
       String SetWellBottom(char*argv[]);
       String SetBottomHolePressure(char*argv[]);
       .......
};
```


5.1.1.4. Creating interaction objects. The next step in enabling interaction is to transform the created computational object into an interaction object. This essentially involves exporting the interaction information (views and commands) to the external interactivity system. This is achieved by overriding two virtual functions of the `DIOS_IObject` base class. The first function, `exportInterfaces()`, exports the interaction interfaces of the object. The second function, `processMessage()`, invokes the appropriate interaction interface (view or command method) in response to an interaction request (message). This is illustrated in the code segment below.

```
void IPARSWell::exportInterfaces()
{
     setType(DIOS_GLOBAL_NONSCALAR);
     /* Export VIEWS */
     addView("WellDescription", "TEXTDISPLAY");
     addView("PlotWellParams", "POINTPLOT");
     .......

     /* Export COMMANDS */
     addCommand("SetTopHoleDepth(float)","Top Hole Depth", "float");
     addCommand("SetBottomHoleDepth(float)","Bottom Hole Depth", "float");
     addCommand("SetWellType(int)", "Well type","int");
     addCommand("SetWellBoreDiameter(interval,float)","Well Bore Dia",
                "int,float");
     addCommand("SetWellTop(interval,x,y,z)","Well Top Coords",
                "int,float,float,float");
     addCommand("SetWellBottom(interval,x,y,z)","Well Bottom Coords",
                "int,float,float,float");
     addCommand("SetBottomHolePressure(double)", "Bottom Hole Pressure",
                "double");
     .......
}


void IPARSWell::processMessage(Message* request)
{    /* request contains the name of the interface to be
     invoked and arguments to be passed, for commands */
     String interface = request->getInterface()->getName();
     String rettype = request->getInterface()->getReturnType();
     String response;
     if (interface == "WellDescription")
        response = WellDescr();
     else if(interface == "PlotWellParams")
          response = PlotWellParams();
     else if(interface == "SetTopHoleDepth")
          response = SetTopHoleDepth(request->args());
     else if(interface == "SetBottomHoleDepth")
          response = SetBottomHoleDepth(request->args());
     else if(interface == "SetWellType")
          response = SetWellType(request->args());
     else if(interface == "SetWellBoreDiameter")
          response = SetWellBoreDiameter(request->args());
     else if(interface == "SetFaultBlockNumber")
          response = SetFaultBlockNumber(request->args());
```



```
      else if(interface == "SetWellTop")
           response = SetWellTop(request->args());
      else if(interface == "SetWellBottom")
           response = SetWellBottom(request->args());
      else if(interface == "SetBottomHolePressure")
           response = SetBottomHolePressure(request->args());

      .......

      setResponse(interface, rettype, response);
}
```

5.1.1.5. Registering interaction objects. Instances of interaction objects can be dynamically created and registered with the interaction agent during the execution of the simulation. In the simplest scenarios, objects are registered at startup by the application using the function `registerObject()` as follows:

```
/* The IPARSWell instance oilwell is registered with name "OilWell". */
Discover->registerObject(&oilwell, ''OilWell'');
```

Note that every object has to be registered with a unique name. `Discover` is a reference to the interaction agent initialized by the application at startup. The `oilwell` interaction object can be deregistered as follows:

```
Discover->deregisterObject(''OilWell'');
```

Once deregistered, it will no longer be accessible at the client portal.

5.1.2. Collaborative interaction and steering

The DIOS enabled IPARS simulations constructed as described above can now be globally accessed for collaborative interaction and steering via the Discover middleware infrastructure using Discover portals. Current interaction/steering capabilities include application control (start, stop and pause), checkpoint/rollback, query and control of parameters such as well diameter, pressure, oil/water injection rate, etc. Figure 4 presents a screen dump of the portal showing plots tracking changes in well parameters of interest for an IPARS implicit hydrology model simulation.

5.2. Grid-based IPARS simulations using NetSolve

This section describes the simplicity with which complex simulation systems such as IPARS can be enhanced to leverage the Grid services that NetSolve can provide. The result of such an integration is access to resource management, scheduling, deployment and computational hardware services from thin clients. These widely distributed services are accessed in the same way as local services.

When creating NetSolve services, one must consider the granularity of the services, computation especially, in relation to the likely amount of input and output data to be transferred. Our choices were to create a coarse-grained IPARS service that would execute the entire simulation code remotely, or to run the IPARS simulator on the local client and export finer-grained requests for the

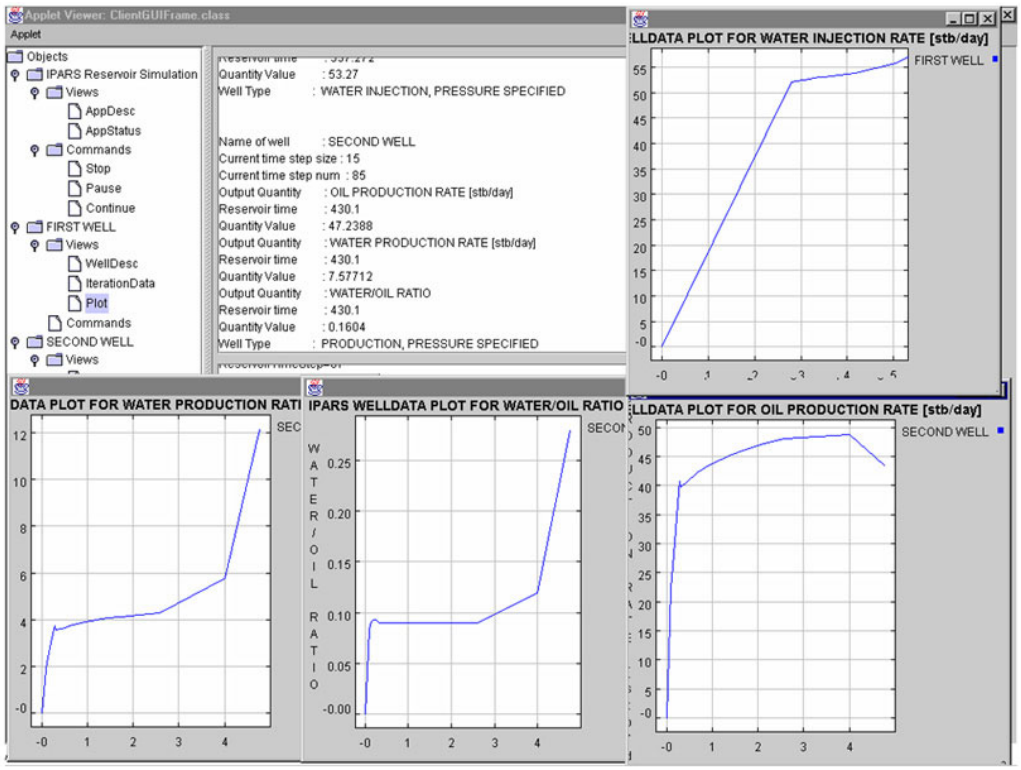


Figure 4. Interactive visualization and steering of IPARS oil reservoir simulations.

computationally-intensive numerical solvers that IPARS leverages, many of which already exist as NetSolve services. Several factors affected our decision to create a coarse-grained IPARS service.

(1) Although most of the finer-grained solvers already exist as NetSolve services, the time to develop a new IPARS service would be significantly less than the time to modify the IPARS simulator to make remote requests for its computational services.
(2) Multiple remote accesses, each with input/output data transmission, for finer-grained services coupled with necessary system overhead would be less efficient than a single service request. Any benefit to be achieved by NetSolve's capability to service requests simultaneously would be minimal since IPARS leverages MPI to achieve parallelism.
(3) Running the entire simulation remotely allows for the client host to be extremely thin. This also allows the simulator to be accessed from systems to which it has not been ported.

5.2.1. *Creating an IPARS service in NetSolve*

The two components that NetSolve provides to create a new service are the PDF (overviewed in Section 3) and the NetSolve *code generator*. The PDF is used to completely specify the intended IPARS service, and the code generator transforms the PDF specification into a program that will be executed by NetSolve servers configured to run this service.

IPARS is designed to receive a single file input, containing the simulation parameters and field data definitions, and produce several output files. One output file contains the results and numerical values of the simulation parameters in ASCII format. The other output files contain visualization support data. An artifact of NetSolve's historical motivation (to provide access to numerical solvers) is the fact that the PDF expects a service to be invoked via a function call. To facilitate this PDF convention, we implement a wrapper function to invoke the IPARS simulator. This wrapper also adds the convenience of allowing us to post-process the simulation output to generate plots, figures and animations, and make them accessible to the client user as service output.

A segment of the IPARS service PDF (simplified for presentation) appears below:

```
1. @PROBLEM ipars
2. @DESCRIPTION
3. Parallel Sub-Surface Flow Simulator.
4. @INPUT 2
5. @OBJECT STRING CHAR model
6. Physical model to use
7. @OBJECT FILE CHAR input_file
8. IPARS simulation input file

...

9. @CODE
10. ipars_wrapper( model, input_file );
11. @ENDCODE
```

Most of this specification is straightforward. Line 1 defines the name that will be used to identify this service within NetSolve. Lines 4–8 specify that the service has two input parameters, a character string defining the physical IPARS simulation model to use, and an ASCII simulation input file. Eventually, the IPARS wrapper function is called with the service input parameters, at line 10. For brevity of discussion, the output file parameters have been untreated, but are specified in a similar manner to input parameters.

With the IPARS service specification, the NetSolve code generator must be invoked to create a IPARS service program to be launched by NetSolve servers. The job of the code generator is to produce appropriate C code specified by the PDF that is linked with NetSolve auxiliary functions (to transfer data to/from client programs and handle other generic service tasks) to be compiled into an *IPARS service executable*. This service program will be instantiated whenever requests for the IPARS service are needed.

5.2.2. *Enabling IPARS-ready NetSolve servers*

During a NetSolve server's initialization phase, the server reads a configuration file that specifies a set of PDFs designating the services the server will provide. Service metadata (name, number and types

of inputs and outputs, etc.) is registered with the NetSolve agent specified in the configuration file. Whenever a client makes a request for the IPARS service to an agent, the agent uses static registration data with dynamic status and performance data to determine which server(s) should be tasked with fulfilling the request. In a wide-area Grid, the agent can efficiently and reliably choose among multiple IPARS-ready NetSolve servers based on static parameters such as rated CPU performance and number of CPUs, and dynamic parameters such as server workload, and network bandwidth and latency.

The IPARS service executable is appropriately invoked by the designated NetSolve server once any requisite authentication and authorization have been successful. This program establishes a network connection with the requesting client for receipt of input data. The program then invokes the code specified in the IPARS PDF with appropriate parameters; upon completion, output data is transferred to the client on the established connection and the program exits.

5.2.3. Accessing the IPARS service via NetSolve

We now describe how clients access the IPARS service via the NetSolve interface and infrastructure. NetSolve has function-based APIs available to different programming languages and environments including C, Matlab and Mathematica. From the C interface, the IPARS simulation service using the 'black oil' reservoir model might be requested as follows:

```
int status = netsolve( ``black oil'', input_file, output_file );
```

Upon success, this call invokes the IPARS service to run the 'black oil' model on some server using the specified input and output files, ignoring visualization output for simplicity. Observe that the single call encapsulates a battery of distributed service interactions from the user who simply makes a library function call. We acknowledge the unwieldiness of a function-based API to access a complete program such as the IPARS simulator. To make the interface less cumbersome, we quickly developed a Web browser interface from which the IPARS service could be invoked via NetSolve. The browser interface provides many advantages:

- Web browsers are ubiquitous providing access to the IPARS service from practically any platform;
- the Discover services integrated into the IPARS simulator are also accessible via Web browsers;
- we can leverage HTML forms and CGI scripts available in browsing environments to create specialized menus and options for composing and configuring IPARS simulations rather than forcing users to learn the syntax of IPARS input files; the developed menu allows the selection of physical models, engineering parameters, domain-Grid, numerical algorithmic parameters and variables for the visualization; and
- Web browsers provide a convenient and familiar environment for viewing the simulation output visualization files.

The result is a uniform, seamless and intuitive browser interface to all configuration, deployment, management, interaction, steering and collaboration services.

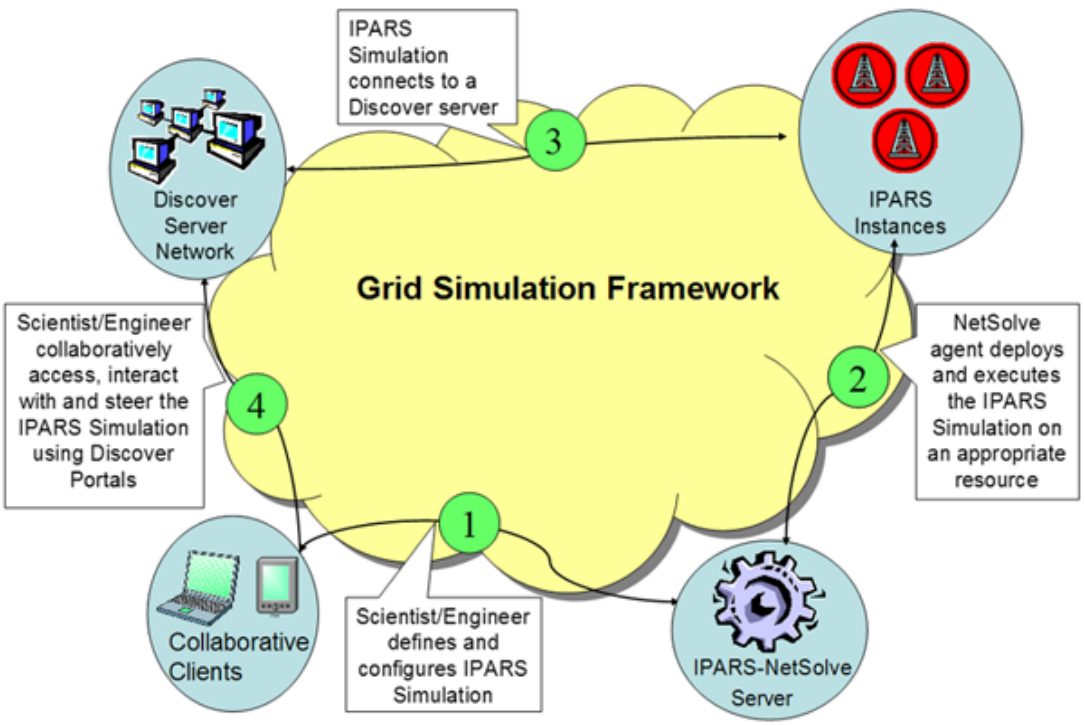


Figure 5. Schematics description of the work scenario.

5.3. An illustrative scenario

The sample usage scenario of the Grid-based interactive and collaborative oil reservoir simulations environment presented in this paper is illustrated in Figure 5. The scenario consists of the following steps.

(1) *Simulation configuration*. The scientist/engineer connects to the IPARS–NetSolve simulation Web site from their local machine. Using the customized NetSolve interface, the engineer configures the IPARS simulation by selecting the IPARS model and input parameters. The engineer may upload an input file stored locally.
(2) *Simulation deployment*. The engineer deploys the simulation on a Grid resource by clicking a button on the Web interface. A registered NetSolve agent processes the deployment request. It selects an appropriate resource from currently available Grid resources and deploys the simulation from that resource. Note that the selected resource may be a single-processor machine or a multi-processor cluster.

(3) *Simulation access*. The simulation connects to a local Discover server and exports its interaction interfaces. The simulation can now be remotely discovered and accessed using the Discover middleware substrate.
(4) *Simulation interaction and steering*. Geographically distributed scientists and engineers can now connect to and interact with the simulation using Discover portals. The engineers can steer it provided they have appropriate credentials and privileges. Access privileges are assigned by the owner of the simulation. An IPARS steering session may include the following operations (based on the view/command interfaces exported via DIOS/Discover):

    - increase/decrease the water pressure at injection wells;
    - increase/decrease the flow rate at production wells;
    - open-up injection and/or production wells;
    - shut-off injection and/or production wells;
    - change boundary conditions.

    Note that the simulation can always be rolled back if the steering process results in an incorrect state.
(5) *Collaboration*. The distributed scientists and engineers can collaborate with each other while interacting with and steering the simulation. They can share data and views returned by the simulation, annotate these using whiteboard capabilities, coordinate commands sent to the simulations (using locks), and chat with one another.
(6) *Post-simulation*. Once the simulation is completed, all the participants can access the output files from the simulation.

6. SUMMARY AND CONCLUSIONS

In this paper, we have presented a prototype interactive and collaborative problem-solving environment to enable the formulation, development, deployment, and management of oil reservoir and environmental flow simulations in computational Grid environments. The primary objective of this research effort was to demonstrate the advantages of such an integrated simulation infrastructure in effectively supporting scientific investigation on the Grid, and to investigate the components and capabilities of such an infrastructure. The prototype problem-solving environment essentially integrated the capabilities of three independent research projects: (1) IPARS oil reservoir and environmental flow simulation framework; (2) NetSolve Grid computing engine; and (3) Discover computational collaboratory. The operation of the infrastructure was demonstrated using a sample application scenario.

ACKNOWLEDGEMENTS

We would like to thank the members of the Discover team, V. Bhat, S. Kaur, V. Matossian and S. Verma for their contributions to this project. We also thank M. Peszyńska, J. Wheeler and S. Bryant for their invaluable help in bringing this project to fruition.

REFERENCES

1. Foster I. Internet computing and the emerging Grid, December 2000. http://www.nature.com/nature/webmatters/grid/grid.html.
2. Foster I, Kesselman C. (eds.). *The Grid, Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Mateo, CA, 1998.
3. Bear J. *Dynamics of Fluids in Porous Media*. Elsevier: New York, 1972.
4. Kaviany M. *Principles of Heat Transfer in Porous Media* (2nd edn). Springer: New York, 1995.
5. Wheeler J. Ipars simulator framework user's guide. *Technical Report*, Center for Subsurface Modeling, The University of Texas, Austin, TX, 1998.
6. The IPARS Project Web site. http://www.ices.utexas.edu/CSM/web/ipars.html.
7. Lu Q, Peszyńska M, Wheeler MF. Multiphysics coupling of codes. *Computational Methods for Subsurface Flow and Transport* (*Computational Methods in Water Resources*, vol. 1), Brebbia CA, Gray WG, Bentley LR, Sykes JF, Pinder GF. (eds.). A. A. Balkema: Rotterdam, 2000; 175–182.
8. Peaceman DW. *Fundamentals of Numerical Reservoir Simulation*. Elsevier: New York, 1977.
9. Lu Q, Peszyńska M, Wheeler MF. Coupling different numerical algorithms for two phase fluid flow. *Mathematics of Finite Elements and Applications X (MAFELAP 1999)*, Whiteman JR. (ed.). Elsevier: Amsterdam, 2000; Chapter 12, 205–214.
10. Lu Q, Peszyńska M, Wheeler MF. A parallel multi-block black-oil model in multi-model implementation. *Society of Petroleum Engineers Journal* 2002; **7**(3):278–287.
11. Wheeler MF, Arbogast T, Bryant S, Eaton J, Lu Q, Peszyńska M, Yotov I. A parallel multiblock/multidomain approach to reservoir simulation. *Proceedings of the 15th SPE Symposium on Reservoir Simulation*. Society of Petroleum Engineers: Houston, TX, 1999; 51–62.
12. Wheeler JA, Wheeler MF, Peszyńska M. A distributed computing portal for coupling multi-physics and multiple domains in porous media. *Computational Methods for Subsurface Flow and Transport* (*Computational Methods in Water Resources*, vol. 1), Brebbia CA, Gray WG, Bentley WG, Sykes JF, Pinder GF (eds.). A. A. Balkema: Rotterdam, 2000; 167–174.
13. Ewing RE, (ed.). *The Mathematics of Reservoir Simulation* (*Frontiers in Applied Mathematics*, vol. 1). SIAM: Philadeliphia, PA, 1984.
14. Dagan G. *Flow and Transport in Porous Formations*. Springer: New York, 1989.
15. Wheeler MF, Lee W, Noh M-H. Air-water flow simulation in unsaturated porous media. *Computational Methods for Subsurface Flow and Transport* (*Computational Methods in Water Resources*, vol. 1), Brebbia CA, Gray WG, Bentley LR, Sykes JF, Pinder GF, (eds.). A. A. Balkema: Rotterdam, 2000; 93–100.
16. Noh M-H. A twophase air-water subsurface flow model. *Masters Thesis*, The University of Texas at Austin, TX, 1999.
17. West WJ, Garvin WW, Sheldon SW. Solution of the equations of unsteady state two-phase flow in oil reservoirs. *Transactions of the AIME* 1954; **201**:217–219.
18. Peaceman DW, Douglas J Jr, Rachford HH. A method for calculating multi-dimensional immiscible displacement. *Transactions of the American Institute of Minerals, Metallurgy and Petroleum Engineering* 1959; **216**:297–308.
19. Terhune MH, Coats KH, Nielsen RL, Weber AG. Simulation of three-dimensional, two-phase flow in oil and gas reservoirs. *Society of Petroleum Engineers Journal* 1967; **7**:377–388.
20. Sheffield M. Three phase flow including gravitational, viscous and capillary forces. *Society of Petroleum Engineers Journal* 1969; **9**:255–269.
21. Blair PM, Weinaug CF. Solution of two-phase flow problems using implicit difference equation. *Society of Petroleum Engineers Journal* 1969; **9**:417–424.
22. Aziz K, Settari A. *Petroleum Reservoir Simulation*. Applied Science Publishers: London, 1979.
23. Arbogast T, Dawson C, Wheeler MF. A parallel algorithm for two phase multicomponent contaminant transport. *Applications of Mathematics* 1995; **3**:163–174.
24. Russell TF, Wheeler MF. Finite element and finite difference methods for continuous flows in porous media. The mathematics of reservoir simulation. *Frontiers in Applied Mathematics* 1983; **1**:35–106.
25. Arnold D, Casanova H, Dongarra J. Innovations of the NetSolve Grid computing system. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15):1457–1479.
26. Arnold D, Agrawal S, Blackford S, Dongarra J, Miller M, Sagi K, Shi Z, Vahdiyar S. Users' guide to NetSolve V1.4. *Technical Report UT-CS-01-467*, Computer Science Department, University of Tennessee, TN, July 2001.
27. The NetSolve Project Web Site. http://icl.cs.utk.edu/netsolve.
28. The Discover Computational Collaboratory. http://www.discoverportal.org.
29. Mann V, Parashar M. Engineering an Interoperable Computational Collaboratory on the Grid. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15):1569–1593.
30. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications* 1997; **11**(2):115–128.

31. Parashar M, von Laszewski G, Verma S, Gawor J, Keahey K, Rehn N. A CORBA commodity Grid kit. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15):1057–1074.
32. Bhat V, Parashar M. A middleware substrate for integrating services on the Grid. *Journal of Supercomputing (Special Issue on Infrastructures and Applications for Cluster and Grid Computing Environments)*.
33. Muralidhar R, Parashar M. A distributed object infrastructure for interaction and steering. *Concurrency and Computation: Practice and Experience* 2003; **15**(10):957–977.

