Parallel Reservoir Simulation with Nested Factorisation

Richard Burrows

Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD, United Kingdom

Dave Ponting

Schlumberger Geoquest, Abingdon Business Park, OX14 1DZ, United Kingdom

Lindsay Wood

Sciencesoft Ltd, Dalziel Business Centre, ML1 1YE, United Kingdom

Presented at the 5th European Conference on the Mathematics of Oil Recovery, Leoben, Austria 3-6 Sept. 1996

1 Abstract

To obtain worthwhile gains from massively parallel computers in the area of reservoir simulation we require a high degree of parallelisation and efficient treatment of data input and output. We discuss a domain-based approach in which the reservoir is split into areally defined domains, each treating its own cells and wells. A message passing protocol such as PVM is used as the programming model.

This requires a scalable parallel linear solver. We use a nested factorisation solver for each domain with a black-white ordering in the overall matrix. On each domain the equations are solved by planes of cells - this enables latency problems to be overcome as data may be sent to the neighbouring domain as each plane is solved but is not required until the sweep is complete. The solver supports IMPES, adaptive implicit (AIM), implicit saturation and fully implicit solution methods. To avoid slowing the simulator for results output each domain only sends out information as messages to a master task. This may occupy a parallel machine node or a separate front end machine. It collects and collates the results whilst the simulation nodes continue working and can act as an interactive user interface to the reservoir model. The flexibility in the selection of domain boundaries enables these to be moved during the simulation to load balance the processors. A diffusion based method for doing this is described.

An alternative method of using such a master slave architecture exists - running a number of independent simulation realisations simultaneously. We discuss the advantages of combining these approaches.

2 Introduction

We discuss the exploitation of parallel processing in compositional reservoir simulation. This involves a change to the central linear solution algorithm. The nested factorisation (Ref. 1) method has proved effective in solving the linear systems which arise in reservoir simulation, which are usually dominated by large vertical transmissibilities. However, nested factorisation is a highly recursive algorithm which is not immediately suitable for parallel processing. By using an areal division of the reservoir into domains and a black-white ordering of the planes of cells within the domains it is possible to generalise nested factorisation to obtain a scalable parallel algorithm which maintains the absence of error terms in the vertical direction.

With a parallel linear solver in place, the remaining aspects of parallel decomposition may be considered. We wish to retain a single version of the code with a few compact routines to treat the parallel pro-

cessing mode. Compositional simulation is in some respects better adapted to parallel processing than black oil simulation, with a more even distribution of computing effort across the elements of the algorithm. The linear solver is generally less dominant, but the production system and the well model are more significant. We wish to parallelise the entire code including well and initialisation operations to obtain efficient performance on larger parallel systems.

To achieve this the entire simulation code is run on each processing element, but treating only its own section of the field. The only requirement for a master process is to launch the slave simulations, to send the input data and to collate the output. It is convenient to use the same program as master and slave, allowing the code to detect whether it is in master or slave mode on startup.

With a high degree of parallel decomposition the treatment of input and output becomes important. To prevent delays whilst each processor accesses the input data we read this into the master task and pass it via the message passing system to the individual processors. In the same way output results are sent back to the master from the processors and only assembled in the master.

This message passing architecture has a simple generalisation which applies to the engineering problems encountered in reservoir simulation. In parallel simulation each slave is sent a copy of the data and processes a domain of the field. Alteratively, we may send different copies of the data and process the whole field. In most simulation studies the geology of the rocks and the nature of the fluid properties are not well known. Often a number of plausible realisations of the reservoir exist and all may be run to give an indication of the uncertainly in the predicted results. Sensitivity runs may also be performed to establish the effect of different modelling assumptions. This is a naturally parallel task which may be treated using the same message-passing approach. A large number of runs may be defined from a single data description and submitted to a set of processors in parallel (Ref. 2). Once the runs have finished the results can be collated and reported. The two approaches may be combined - this limits the loss of total scalar efficiency encounted in highly decomposed studies. For example 4 independent realisations might be performed with each running as a 4 processor parallel task using a total of 16 processors. This allows the most flexible and efficient use of the available parallel computing capacity.

3 The Message Passing Programming Model

Two main methods exist to set up parallel version of a simulation code. In a global memory model we perform parallel operations in a single program which accesses data over the processors. The compiler or a programming language such as HPF Fortran passes data across the processors as required. In a message passing model we start a task on each processor, and pass information between the processors when it is needed. These are passed as message packets - a library of message passing routines is employed for this purpose. The library used in this project was PVM (Ref. 3), a widely available set of routines, but called from interface routines so that extension to other possibilities is simple.

Highly parallel computers usually employ a distributed memory, with some attached to each processor. However, the message passing picture may be also used on hardware in which all the memory is genuinely available to all the processors equally.

A distributed memory architecture thus requires that information be passed between processors as messages rather than by direct access to memory. In any computer system we can term the time taken to initiate this data access as latency which is a system dependent quantity. On shared memory systems the latency is usually in the range of 1 - 100 clock cycles, but on distributed memory systems this can be several thousand clock cycles. An algorithm can only run effi-

ciently on the latter if the time spent on computational work between message exchanges is greater than the total latency times associated with this work. This is equivalent to minimising the number of message passing operations and to managing the messages so that no processor is held in a wait state while a fetch operation is done. A primary aim of the linear solver design is thus to structure the computations so that useful work is performed while messages already sent are delayed by latency and transfer times.

4 Structure of the Simulator

A reservoir simulator solves the equations governing the flow of hydrocarbon components and water within oil and gas reservoirs. In black oil mode there are just two components - stock tank oil and stock tank gas. In thermal mode there is an additional energy equation. The main computational sections are:

1. Oil and gas property evaluation from the cell composition.
2. Evaluation of flow rates between cells.
3. Evaluation of well injection and production rates.
4. Assembling the Jacobian matrix for the non-linear equations.
5. Solving the linear equation system.

To run the simulator in parallel, we split the reservoir into domains and assign a processor to each domain. If cells in each domain were completely independent this would lead to a naturally parallel program.

The property calculations are naturally parallel. Evaluation of the flows requires knowledge of cells on other domains. We wish to access the required values without having exception coding. To do this we use the method of over-dimensioning.

Normally a reservoir simulator only stores solution values for active cells - those with a non-zero pore volume. Internally all arrays are dimensioned to N_{active}, the number of active cells, and operations involve loops from 1 to N_{active}. Active ordering is normally a rotated natural ordering with inactive cells compressed out.

In the over-dimensioning method we allow a separate count of cells as N_{total}, whilst most loop operations still run from 1 to N_{active}. Overdimension cells in the range N_{active}+1 to N_{total} represent neighbouring cells in other domains, and are numbered after the active cells. A single parallel exchange operation is performed after the property evaluation step to fill these over-dimension cells with their active cell values on neighbouring processors.

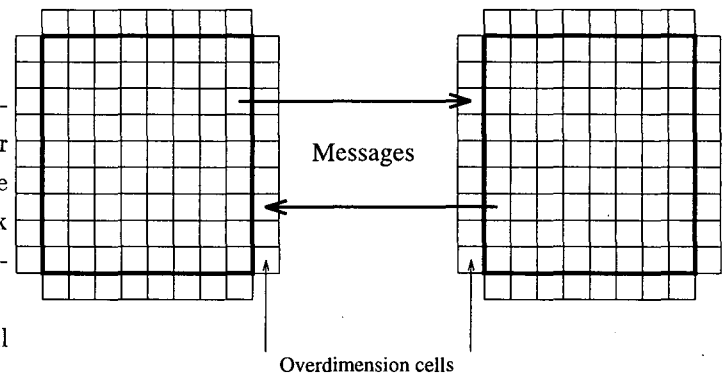

Figure 1. Setting up over-dimension cells

Note that in a two dimensional array of domains an active cell may be copied into more than one neighbouring over-dimension grid cell. This causes no problems as long as the messages go from the active cells to the over-dimension cells. Operating on an over-dimension cell and copying this into an active cell would cause ordering problems.

The residuals $R_a(X_a, X_o)$ for the flow equations are constructed for the active cells only - these involve active and over-dimension cell solution variables X_a and X_o. We construct the Jacobian blocks:

$$A_{aa} = \partial R_a / \partial X_a$$

$$A_{ao} = \partial R_a / \partial X_o$$

$$A_{oa} = \partial R_o / \partial X_a$$

With these blocks in place all linear solver column sum and back-substitution operations may be performed by

passing solution array values only.

5 The parallel linear solver

The nested factorisation method is a natural starting point for a parallel algorithm.

We consider the set of linear equations:

$$Ax = R$$

$$A = d + l_1 + u_1 + l_2 + u_2 + l_3 + u_3$$

where d is a diagonal matrix, and l, u are lower and upper bands each direction. The elements of A are sub-matrices with dimension dependent on the number of implicit variables per cell. The derivation of these linear equations is described in appendix 1.

The usual nested factorisation preconditioning matrix is:

$$B = (P + l_3).(1 + P^{-1}u_3)$$

$$P = (T + l_2).(1 + T^{-1}u_2)$$

$$T = (\gamma + l_1).(1 + \gamma^{-1}u_1)$$

γ is a block diagonal matrix defined by:

$$\gamma = d - l_1\gamma^{-1}u_1 - colsum(l_2)T^{-1}u_2 - colsum(l_3)P^{-1}u_3$$

B is used as a preconditioning matrix in a ORTHOMIN accelerated conjugate gradient linear solution scheme (Ref. 6). Colsum is an operator which sums the elements of a matrix in columns.

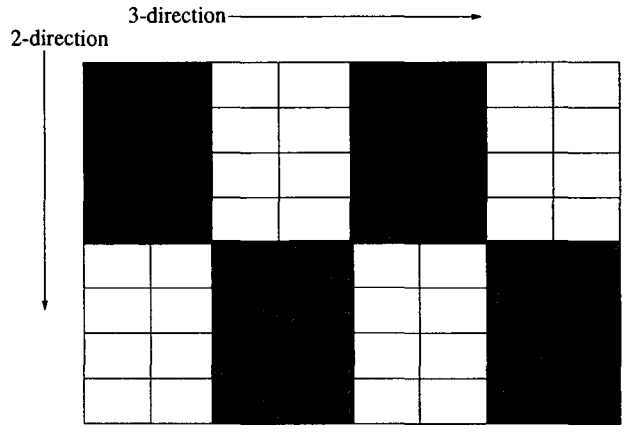

Figure 2. Definition of black and white planes

The parallel solver is a two-colour method. Half the planes on each domain are black planes and are chosen to form the chessboard pattern shown in Figure 2. This example has four domains, and only the active (not overdimension) cells are shown The global ordering used in the linear solver numbers the black planes before the white planes. This results in the matrix structure shown in figure 3.

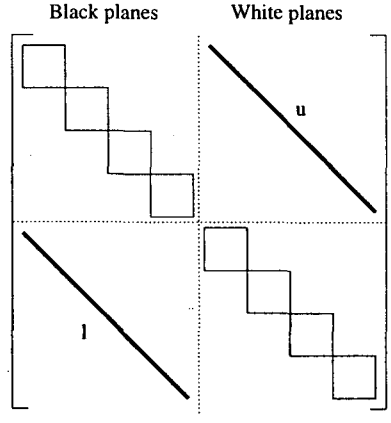

Figure 3. Two coloured domain matrix structure

This results in the matrix:

$$A = d + l_1 + u_1 + l_2^i + u_2^i + l_3^i + u_3^i + l^e + u^e$$

where l_2^i, u_2^i, l_3^i and u_3^i are the interior matrix elements in the 2 and 3-directions which are internal to the black

or the white planes of a domain. The external matrix elements connecting black and white planes are:

$$u^e = l_2^{bw} + u_2^{bw} + l_3^{bw} + u_3^{bw}$$

$$l^e = l_2^{wb} + u_2^{wb} + l_3^{wb} + u_3^{wb}$$

Note that both the lower l^e and upper u^e bands have matrix elements corresponding to both positive and negative directions in both 2 and 3 directions. When performing elimination operations all four possibilities must be checked. This is a change from normal nested factorisation in which lower bands only occur in the negative direction and upper bands only occur in the positive direction.

The preconditioning is then similar to nested factorisation, except that black-white and white-black terms in both the 2- and 3-directions are included in the outer nesting:

$$B = (P + l_3^i + l^e).(1 + P^{-1}(u_3^i + u^e))$$

$$P = (T + l_2^i).(1 + T^{-1}u_2^i)$$

$$T = (\gamma + l_1).(1 + \gamma^{-1}u_1)$$

We can express γ as:

$$\begin{aligned}\gamma = d &- l_1\gamma^{-1}u_1 \\ &- colsum(l_2^i)T^{-1}u_2^i \\ &- colsum((l^e + l_3^i)P^{-1}(u^e + u_3^i)\end{aligned}$$

This preconditioning maintains the main feature of nested factorisation in having no error terms in the strong inner direction. The inner 1-direction is always chosen as this dominant z-direction in the parallel case. The computation of γ is therefore a two step process which allows all the black planes to be processed in parallel followed by all the white planes in parallel.

i. Gamma sweep for black planes

$$\begin{aligned}\gamma = d &- l_1\gamma^{-1}u_1 \\ &- colsum(l_2^i)T^{-1}u_2^i \\ &- colsum(l_3^i)P^{-1}u_3^i\end{aligned}$$

ii. Gamma sweep for white planes

$$\begin{aligned}\gamma = d &- l_1\gamma^{-1}u_1 \\ &- colsum(l_2^i)T^{-1}u_2^i \\ &- colsum(l^e + l_3^i)P^{-1}(u^e + u_3^i)\end{aligned}$$

The sequence of operations is important here. The value of $C = colsum(l^e + l_3^i)P^{-1}$ is built up in an auxillary array as γ is obtained. For each plane the operations are:

i. Subtract $C.(u^e + u_3^i)$ from γ for this plane. This involves C values from previous planes.
ii. Obtain γ values for this plane.
iii. Construct $colsum(l^e + l_3^i)$ for this plane.
iv. Post-multiply to obtain $C = colsum(l^e + l_3^i).P^{-1}$ for this plane.

The only message passing operation required is to copy the C vector values on black planes into the over-dimension cells on the white planes of neighbouring processors. This can be done as a single operation between the black and white plane sweeps.

A similar two step process can be applied to both the forward and backwards sweeps over planes in the solution of the approximate inverse system.

$$Bx = (P + l_3^i + l^e).(1 + P^{-1}(u_3^i + u^e))x = R$$

This is solved in two stages:

$$Bx = (P + l_3^i + l^e).y = R$$

$$(1 + P^{-1}(u_3^i + u^e))x = y$$

In the forward sweep we have:

$$y^b = P^{-1}.(R^b - l_3^i.y^b)$$

$$y^w = P^{-1}.(R^w - l_3^i.y^b + l^e.y^w)$$

After treating the black planes a message passing operation is required to copy y-values on neighbouring domains into the overdimension cells to allow them to be 'seen' by the white planes.
In the backward sweep we have:

$$x^w = y^w - P^{-1}u_3^i x^w$$

$$x^b = y^b - P^{-1}(u_3^i x^b + u^e x^w)$$

These are processed in reverse order, with the white planes first, and treating the planes in decreasing 3-index. After treating the white planes a message passing operation is required to copy x-values on neighbouring domains into the overdimension cells to allow them to be 'seen' by the black planes.

The parallel solver can thus be implemented in its simplest form with just one message passing operation in the γ calculation and two for each ORTHOMIN iteration. However it is possible to sequence the same messages rather differently. Consider the forward sweep of the approximate inverse. A given white plane can be processed if the required messages have been received, so that the appropriate over-dimension cells have been filled in. These will come from the corresponding black planes on neighbouring processors. If these messages are sent as soon as those planes have been completed, the messages have the time required to process the higher 3-index black planes and the lower 3-index white planes before they are required. This processing time may be used to overcome the latency delay.

As only the solution values are copied over, and as all the matrix elements required for the colsum and back-substitution operations are held on the processors, several aspects of the simulator are automatically included in this solver:

i. Non-neighbour connections due to faults.
ii. Local grid refinements contained within a single domain. Each local grid would be placed on the processor which contains the section of global grid in which it lies.
iii. Matrix elements due to multiply completed wells contained within a single domain.

The same solver code can be used for the parallel algorithm as for normal nested factorisation, with two enhancements:

i. A permutation array is used to define the order in which the 3-planes are treated.
ii. A marker array is updated as each plane is processed. This enables the lower and upper bands to be located - a direction to a marked plane is lower on a forward sweep.

To obtain normal nested factorisation the permutation array is simply set to natural, rather than black-white, order. Each iteration of the parallel solver involves slightly less work than in normal nested factorisation as some 2-direction matrix elements are in the outer rather than the middle nesting and are swept once rather than twice per iteration.

Some results from tests of the linear solver are described in appendix 2.

6 Load Balancing

As the processes are synchronised after the property exchange and each linear solver iteration, the progress of the simulation is defined by the slowest process. Particularly for compositional simulation the computing time per cell may vary widely depending on the phase state. Two phase cells which use a flash calculation will take more time than single phase hydrocarbon cells. This imbalance may change during the progress of the simulation. To prevent load imbalance effects from severely limiting the parallel performance we use a dynamic load balancing technique.

In parallel reservoir simulation a 3D grid is partitioned in the two horizontal directions into rectangular domains, each controlled by one processor. As the simulation progresses the workloads of each cell will change and the partitioning may be no longer suitable.

It may then be necessary to change the partitioning so that the work is more evenly distributed.
The diffusion-based scheme for dynamic load balancing uses the following iterative procedure:

i. Each processor $P_{i,j}$ determines the total work for $U_{i,j}$ for cells in its domain and receives corresponding values from its neighbours.

ii. Each processor determines the desired work diffusion in each of the four cartesian directions (diffusion across the grid outer boundary is zero). For example, in the positive x direction:

$$q^x_{i,j} = 0.5(U_{i+1,j} - U_{i,j})$$

iii. Each processor determines the local boundary shift $\delta B^d_{i,j}$, $d = x, y$, to achieve the required flow of work. These shifts are sent to the master processor.

iv. The master determines the required partition shifts as (for a P_x by P_y processor arrangement):

$$\delta S_x = \frac{1}{P_y} \sum_{j=1..P_y} \delta B^x_{i,j}$$

$$\delta S_y = \frac{1}{P_x} \sum_{i=1..P_x} \delta B^y_{i,j}$$

v. The shifts are made by storing the current simulation solution, restarting the simulation with the new domain partitioning and reloading the solution. Convergence is reached when δS_x and δS_y have integer values of zero.

7 Conclusions

A method has been described of parallelising an existing reservoir simulation code with minimal changes to the existing structure. The entire decomposed code was implemented on the IBM SP2 parallel system. Typical overall all speed ups were obtained of 4.6 on 6 processors and 10.6 on 16 processors. Typical problem sizes were 1600 cells with 9 components. The lack of 'ideal' parallel efficiency (a speed up of 6 on 6 processors etc) is due to three factors:

i Residual load balancing errors

ii The additional cost of storing over-dimension cell values and forming the Jacobian blocks A_{ao} and A_{oa}

iii The additional linear iterations required by the parallel linear solver.

A generalisation of the nested factorisation solver allows the algorithm to scale well on parallel processors whilst maintaining the strengths of the original algorithm. Load balancing is an important issue and may be treated using a diffusion based dynamic balancing method.

8 Nomenclature

A	Jacobian matrix
b^m_p	Molar density
d	Diagonal band of Jacobian
D	Cell depth
f	Local flow/unit area vector
F	Flow of fluid through a cell interface
g	Gravity constant
K_{rp}	Relative permeability of phase p
N_{active}	Number of active cells
l	Lower band of Jacobian
Q	Well production rate from a grid block
R	Residual of non-linear equations
ρ^{ij}_p	Average density of phase p in cells i and j
Δt	Length of time step
μ_p	Viscosity
$T^{i \to j}$	Transmissibility from cell i to cell j
u	Upper band of Jacobian
x_{pc}	Mole fraction of component c in phase p
X	Primary solution
x	Change in X over a non-linear iteration
Y	Secondary solution

Sub- and superscripts

c	Component index
p	Phase index

9 Acknowledgements

Part of this work was supported by ESPRIT under research contract P9601.

10 References

1. Appleyard, J.R. and Cheshire, I.M., SPE 12264, Proc. of 7th SPE Symp. on Res. Sim., San Francisco, 1983.
2. Faidi, S., Ponting, D.K. and Eagling, T, SPE 36005, Proc. Petroleum Computer Conference, Dallas, June 96.
3. Geist, S. G., Beguelin, A., Dongarra, J. Jiang, W., Manchek R. and Sunderam V., PVM 3 Uers's Guide and Reference Manual, Oak Ridge National Laboratory, 1993.
4. Bhogeswara, R. and Killough J.E., SPE 25240, Proc 12th Symp. on Res. Sim., New Orleans, 1993.
5. Trangenstein, J.A. and Bell, J.B., SPE 13520, SPE 8th Reservoir Simulation Symposium, Feb 1985.
6 Vinsome, P.K.W., SPE 5729, Proc. SPE-AIME 4th Symp on Num. Sim.,Los Angeles,1976.
7. Young, L.C., SPE 16023, Proc. 9th Symp. on Res. Sim.,Feb 1987,San Antonio.
8. Kenyon, D.E. and Behie, A., SPE 12278, Proc. 7th Symp. on Res. Sim.,Nov 1983,San Francisco.

11 Appendix 1: The Non-Linear Equations Defining a Time Step

Suppose the simulation takes a time step from t to $t+\Delta T$. The primary solution variables are the pressure and reservoir molar densities of each component and water in each cell:

$$X = (P, m_c, m_w) \; c = 1, .., N_c$$

The discretised conservation equations may be written in residual form:

$$R_{ci} = (V_p m_c)_i^{T+\Delta T} - (V_p m_c)_i^t + \Delta t.(\sum_j F_c^{i \to j} + \sum_w Q_c^w) = 0$$

where $F_c^{i \to j}$ is the flow from cell i to j, Q_c^w the well injection or production rate.
The flow of a component is obtained by summing over phases:

$$F_c^{i \to j} = T^{i \to j} M_{cp}.(P_i - P_j + P_{cp_i} - P_{cp_j} - g(d_i - d_j)\rho_p^{ij})$$

M_{cp} is the generalised mobility of component c in phase p, given by $x_{pc} K_{rp}(S_p).b_p^m/\mu_p$.
One additional equation is required to complete the set, and this is taken as a volume balance:

$$R_v = V_p - V_p.(U_o + U_g + U_w) = 0$$

where U_p are phase volumes per unit reservoir volume. The phase equilibrium condition which defines the the phase volumes, saturations and other quantities appearing above is obtained from the Gibbs energy minimum condition :

$$\partial G^i / \partial Y_c^i = 0$$

These are solved with respect to the secondary variables Y in two phase cells, and total derivatives with respect to the primary variables obtained. The methods used follow those of Ref. 5.

The $N_c + 2$ equations are in residual form, and can be solved by Newton's method. At each Newton step we change the solution vector by:

$$x = -A^{-1} R$$

where A is the Jacobian $\partial R / \partial X$. This is the linear equation system discussed in section 5.

The generalised mobility M_{cp} and the capillary pressure P_{cp} may be treated in three ways:

- **Implicit case:** The mobility and capillary pressure are a function of the solution at the end of the time step:

$$P_{cp} = P_{cp}(X^{t+\Delta t})$$

$$M_{cp} = M_{cp}(X^{t+\Delta t})$$

The resulting dependence of the flows on all the unknown solution variables is reflected in the Jacobian A: the band terms coupling the cells are full $N_c + 2$ by $N_c + 2$ submatrices.

- **Explicit case:** The mobility and capillary pressure are held at the start of time state values. The only coupling between cells is through the cell pressures. Strictly this is not an IMPES (IMplicit pressure, Explicit Saturation) method, rather an IMPEM (IMplicit Saturation Explicit Mobility) method.
The Jacobian only has band terms in the pressure equation and can be reduced to a single variable per cell by using the conservation equations to eliminate the terms in molar density from the volume balance condition.

- **The Implicit Saturation Case:** We introduce three new variables: X_o, X_g and X_w. These will be treated implicitly and represent the phase saturations. Three new equations are required to define these extra variables and ensure that they represent the required saturations. These are the saturation balance conditions:

$$R_o = V_p.(S_o(P,m) - X_o) = 0$$

$$R_g = V_p.(S_g(P,m) - X_g) = 0$$

$$R_w = V_p.(S_w(P,m) - X_w) = 0$$

$S_p(P,m)$ are the saturations as determined by the usual Gibbs energy conditions. Note that these are diagonal equations: they do not involve solution variables in any neighbouring grid cells. The conditions that the residuals go to zero ensure that X_o, X_g, X_w converge to the true saturations.
The capillary pressure and relative permeabilities are now made functions of X_o, X_g, X_w:

$$P_{cp} = P_{cp}(X^{T+\Delta T})$$

$$K_{rp} = K_{rp}(X^{T+\Delta T})$$

The Jacobian only has band terms in the P, X_o, X_g, X_w and can be reduced to a four variable per cell system.

Implicit and explicit mobility treatments may be combined in adaptive implicit methods.

12 Appendix 2: Results of Linear Solver Tests.

An extensive set of data sets were used to evaluate the performance of the linear solver, but only a representative sample of these are reported here. We investigated the convergence of the linear solver as a function of domain size. All results quoted here are for the default reduction in linear solver residual of 1.0E-10 which is approximately six orders of magnitude smaller than that used by Bhogeswara and Killough (Ref. 4).

Test 1: Convergence dependence on domain size.

We used the test set SPE3X10, (Ref. 7), which is built up of 10 SPE3 data sets (Ref. 8). The model has dimension 45x18x4 and 3240 active cells. There are 10 production and 10 injection wells.

We ran a series of studies for the AIM solution method and totaled the number of linear iterations

required for each case. These results are summarised in table 1.

	Non-Linear Iterations	Linear Iterations	Linears /Non-linear
Single domain	101	1690	17
5 domains	101	2911	29
15 domains	101	3678	36
45 domains	100	6488	65

Table 1.

For the worst case (45 domains) each domain has only 36 active cells, but the total number of linears is less than four times that of the single domain. Using domains with 200-300 cells increases the number of linear iterations by 70%.

Test 2: Implicit black oil parallel test set.

This is a fully implicit black oil model. The grid size is 40x40x3 with 4800 active cells.

	Non-Linear Iterations	Linear Iterations	Linears /Non-linear
1 domain	218	2384	11
15 domains	218	3080	14.1

Table 2.

The increase in linear iterations is 30%.

Test 3: AIM North sea compositional model.

The grid size is 87x34x11 with 20637 active cells. There are fault non- neighbour connections in both the x and y directions plus pinch-out connections with a total of 263 non-neighbour connections.

	Non-Linear Iterations	Linear Iterations	Linears /Non-linear
1 Domain	2827	57652	20
9 domains	2888	63715	22

Table 3.

The 10% increase in linear iterations is relatively small which can be explained by the fragmented nature of the reservoir. There are several regions which are connected areally by only a few grid blocks and hence an areal decomposition has a small effect on the total number of linear iterations.

Test 4: AIM North sea compositional model with horizontal wells.

The grid dimension is 21x19x22 with 7444 active cells including non-neighbour connections.

	Non-Linear Iterations	Linear Iterations	Linears /Non-linear
1 domains	372	3215	8.6
9 domains	364	3854	10.6

Table 4.

The increase 20% increase in iterations is similar to that for test 3.