Parallel Programming Of A Reservoir Simulator

MARIYAMNI AWANG
Jabatan Kejuruteraan Petroleum,
Universiti Teknologi Malaysia,
Jalan Semarak, 54100,
Kuala Lumpur.

ABSTRACT

This study concerns applying parallel programming to reservoir simulation using a 32-Mbyte, 12-processor parallel computer. The effects of number of processes, granularity, load balancing and program structure were studied.

The model simulated was a two-dimensionals, two-phase, black oil model with a fully-implicit formulation. The differenced equations were solved by the Newton-Raphson method and, Gaussian elimination was used to solve the Jacobian matrix. Matrix generation was parallelized using monitors as macros to synchronize calculations. The performance of the simulator was measured by the speed up.

The speed ups of the matrix generation time increased almost linearly with increasing number of processes. For all of the models tested, the speed ups ranged from 3.5 to 4.0 for four processes and 7.0 to 7.9 for eight processes.

INTRODUCTION

Complex reservoir models provide a demand for computers having large memory and fast execution, such as supercomputers. The disadvantage of supercomputers is the cost that ranges in the millions of dollars. Instead of expensive supercomputers, parallel computers are a cheaper alternative. Although investigations on reservoir simulation using parallel machines have increased, there are areas of parallel programming which have not been fully investigated with respect to reservior simulation. In this study, a shared memory parallel computer was used to study several factors in the parallel programming of a simulator.

The finite-difference techniques used in reservoir simulation result in many independent calculations, which are amenable to parallel execution. Using these techniques, a reservoir is discretized into many grid block. The discretized equations representing the model yields a matrix, which is refered to as the Jacobian. The matrix is computed from the fluid properties, rock properties, relative permeabilities and the physical dimensions of the model. Fluid properties and relative permeabilities are functions of pressures and saturations, and the values are obtained by the interpolation of tables of data or table look-up. At a given time, the pressure and saturation distributions of a reservoir are dependent only on the spatial location, therefore table look-ups can be concucted in parallel. Matrix generation has been reported to constitute substantial part of the computing time[11,8]. This work involved the parallelization of the matrix generation part of a simulator.

The overall execution time of a simulator is significantly affected by the matrix solver. Direct linear solvers, such as Gaussian elimination, require large memories. The computation time increases rapidlly with increasing matrix size[9]. Some interative techniques, such as the preconditioned conjugate gradient method, the Gauss-Siedel method and the SOR method, have been modified to be used with parallel or vectorizing machines. Application of these methods have been reported to result in considerable reduction of execution time[5,6]. Since considerable literature on matrix solution has been reported in reservoir simulation and other fields, this work did not aim to develop a parallel matrix solver.

An important factor in parallel programming a simulator is the structure of the program, which directly affects load balancing and granularity. Load balancing is the term applied to the division of tasks among the processors. The number of tasks allocated to each processor is the granularity. The granularity should be sufficiently large to offset overhead costs, and load balancing should ensure that no processor is idle at any time. In this investigation, a process was created when the parallel program was executed. The

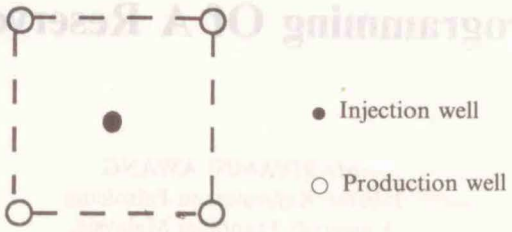

Figure 1 : Five-spot pattern

process (or master process) then created several slave processes. Load balancing and granularity considerations in multiprocessing are similar to multiprocessor programming. The improvement in execution rate due to parallel programming is measured by the speed up, which is the ratio of the CPU time of sequential excution to the CPU time using multiprocessing. Using the same computer as in this investigation, Barua[4] found that unequal load balancing caused a reduction in the speed up. However, no methods were suggested to overcome the problem. Scott et al[12] reported poor speed ups in the matrix solution execution due to the overheads such as synchronizing the processes. Similarly, no granularity or load balancing effects were related to the synchronizing time. In this study, two different program structures were applied to matrix generation to study the relationships between load balancing, granularity and synchronization.

The objective of this investigation was to reduce the computation time of a reservoir simulator through the use of a shared memory parallel computer. Among the factors that were considered are:

(1) The number of processes.
(2) The structure of the program.
(3) Granularity and load balancing.

An oil-gas two-dimensional reservoir model was simulated. A series of programs written in FORTRAN were parallelized using the ANL macros[1]. Investigations on the parallelized simulators were conducted using a shared-memory, 12-processor computer. Speed ups of the parallel programs were measured for one, four and eight processes. Two structures of parallel program were compared. One structure used the sequential style that minimized synchronization and maximized granularity. The other structure used the stepwise style that results in smaller granularity and more synchronization.

PARALLEL COMPUTERS

Shared memory parallel computers are MIMD (multiple-instruction multiple-data) machines that are characterized by a common memory area which is accessible to all processors. These machines have been used for general application and for studies in parallelism. This has resulted in a variety of software support for these machines, ranging from operating systems to automatic parallel compilers. At the current level of hardware and software technology, these computers have been able to achieve the speed of minisupercomputers at less cost[2]. Another advantage is that no major restructuring of sequential programs is necessary since programming is based on shared-memory.

The 32-Mytbe computer used in this study is a small, multiuser machine that is used for parallel programming studies. Big simulators could not be programmed for the machine. However, it was adequate for research purposes. This computer does not have vectorizing facilities, and so the reduction in computing time is due only to the parallel execution of instruction. The computer ia a highly-coupled, shared-memory multiprocessor[10]. It can be configured with two to 20 microprocessors, four to 128 Mbytes of shared-memory and one to ten ethernet/mass strorage I/O channels. The bipolar bus has a bandwidth of 100 Mbytess to ensure sufficient capacity. The computer allows symmetric multiprocessing, which means all processors are equal and have the same priority. The performance ranges from 1.5 to 15 MIPS (million instructions per second).

Another subgroup is the distributed memory parallel processor, such as the hypercube machines. Typically, each node is a microcomputer connected to the neighbouring nodes communication channels.

The main problem with this system is difficulty in programming, since the message-passing operating system requires major restructuring of a program.

DESCRIPTION OF A SIMULATOR

The model used was an areal, homogeneous, oil-gas model with a five-spot pattern configuration (shown in Figure 1). The oil phase, gas phase and rocks were compressible. Capillary and gravitational effects were not considered, and the injection rate was kept constant. The production rate was kept constant until the reservoir pressure was too low to maintain the production rate, which was then varied.

The equations for various types of simulators are derived from considerations of material balance, and are discussed in detail by Aziz and Settari[11] The discretized flow equations that represent the simulator are:

$$\Delta_x TX_l \Delta_x p_l + \Delta_y TY_l \Delta_y p_l = \frac{V_{i,j}}{\Delta t} \Delta_t(\phi S_l b_l) + q_{l\,i,j} \qquad \text{Eqn. 1}$$

where l is the oil or gas phase,

i, j is the block index,
$V_{i,j}$ the block volume,
$q_{l\,i,j}$ is the source term of the block,
TX_l is the phase transmissivity in the x direction, and similarly
TY_l is the l phase transmissivity in the y direction.
The definitions for the terms are given in the Nomenclature.

Using the Newton-Raphson method, the following equation was solved iteratively to obtain the unknown pressures and saturations:

$$J^{n+1\,v}\,\vec{u}^{\,n+1\,v+1} = -\vec{F}^{\,n+1\,v} \qquad \text{Eqn. 2}$$

where J is the jacobian matrix,

n is the time step,
v is the iteration level,
F is the residuals vector,
u is the vector of unknown pressures and saturations.

The algorithm of the solution is as follows:

(1) At time step n, iteration level v, the Jacobian is evaluated.
(2) The pressures and saturations are evaluated using the Gaussian elimination.
(3) All calculations that are dependent on pressures and saturations, such as fluid properties and transmissivities, are recalculated using the new pressures and saturations.
(4) The material balance, as given by Equation 1 is tested for convergence. For convergence, the simulation proceeds to the next time step, otherwise the iteration is repeated.

PARALLELIZATION MECHANISMS

The synchronization mechanisms used in this research were monitors[3] written as macros that were developed at the Argonne National Laboratory. Details of the development of these macros were reported by Lusk and Overbeek[1]. The ANL macros were written to be used with shared-memory machines which have their own synchronization primitives. The macros are processed by the m4 preprocessor, which is a preprocessor for use with UNIX systems. The basic ANL marco[1] library was formed using machine

specific synchronization primitives. Therefore, it is necessary it recode the basic library for different machines.

PROGRAM STRUCTURE

The structure and the style of three programs were considered with respect to abstraction, speed of execution, synchronization and ganularity. The programs were a sequential program and two parallel programs, that is stepwise and sequential style. The sequential program was used to calculate the speed ups of the parallel programs.

Sequential program

In sequential programming, common tasks are written as subroutines and usage of local variables instead of global variables are emphasized. These two approaches allow abstraction, and changes can be made easily to any of the subroutines without affecting the whole program. A modular program that is easily understood and modular subroutines that may be reused are obtained. However, with more abstraction, the execution of a program becomes slower. In this research, for example, calculation of the oil properties and gas properties were written as separate subprograms *oil* (*arg*), *gas* (*arg*). The algorithm of *oil* (*arg*) is given in Figure 2.

```
Procedure Oil (p, S, Bo, Viso, Rso)
Declarations
/* p, S are input values fro pressure and saturation */
/* Bo, Viso and Rso are calculated values */
      table-look up(p, S, ind)
      /* ind is output of table-look-up*/
      Mathematical computations
End (oil)
```

Figure 2: Algorithm of subprogram *oil (arg)*

Each of the four subprograms that calculated the Jacobian matrix coefficients passed in the pressures and saturations and returned the cofficients. *Oil* and *gas* were called in each subprogram. No common blocks were used to store the calculated variables, hence abstraction was maximized. Figure 3 shows an example.

The main disadvantage of this approach was that it required the calculation of variables that had already been calculated in another subroutine. An alternative that saved computation time but reduced abstraction was to save values that were used frequently in common memory.

```
Procedure matrixA (coefficients A)
Declarations
      oil (arg)
      gas (arg)
      Mathematical computations
End (matrixA)
```

Figure 3 : Abstraction in subprogram matrix A

Parallel Program: Stepwise style

The structure of the program required the processes to execute groups of instruction concurrently and to wait until all processes had completed execution before proceeding to the next group of instructions. These 'steps' form a synchronized movement of the processes.

This method uses many global variables, communication and synchronization constructs and is suitable for a shared-memory machine since it requires many global variables and data access. This method is also suitable for vectorizing machines, since the program can be structured into many loops.

Parallel Program: Sequential style

In the sequential programming approach, the processes executed asynchronously from the beginning when each process picked up a grid block. Each process calculated the grid block properties and the properties of the four neighbouring grid blocks. Therefore, only the saturations and the pressures of the connecting neighbouring grids were accessed during execution. The processes were required to synchronize once at the end of the matrix generation portion to ensure that all coefficients had been generated before proceeding to the matrix solution, In essence, each process ran its own program until all coefficients were generated. There was little communication or synchronization, and the granularity was larger than the granularity of the stepwise method. This method is suitable for message-passing computers which are loosely coupled, and therefore require more time to synchronize of communicate. Fiqure 4 shows a comparison between the sequential-style and stepwise-style of programming.

RESULTS AND DISCUSSION

Similar level f difficulty was experienced when programming in parallel and in sequence. The main concerns during programming were:

(1) ensuring that all processes reached the same synchronizing stage before proceeding to the next stage.
(2) The design of the macros requires that all processes to be released together. Therefore, if some processes remained outside the parallel loops, the program hung.
(3) identifying the variables that have to be globally known.

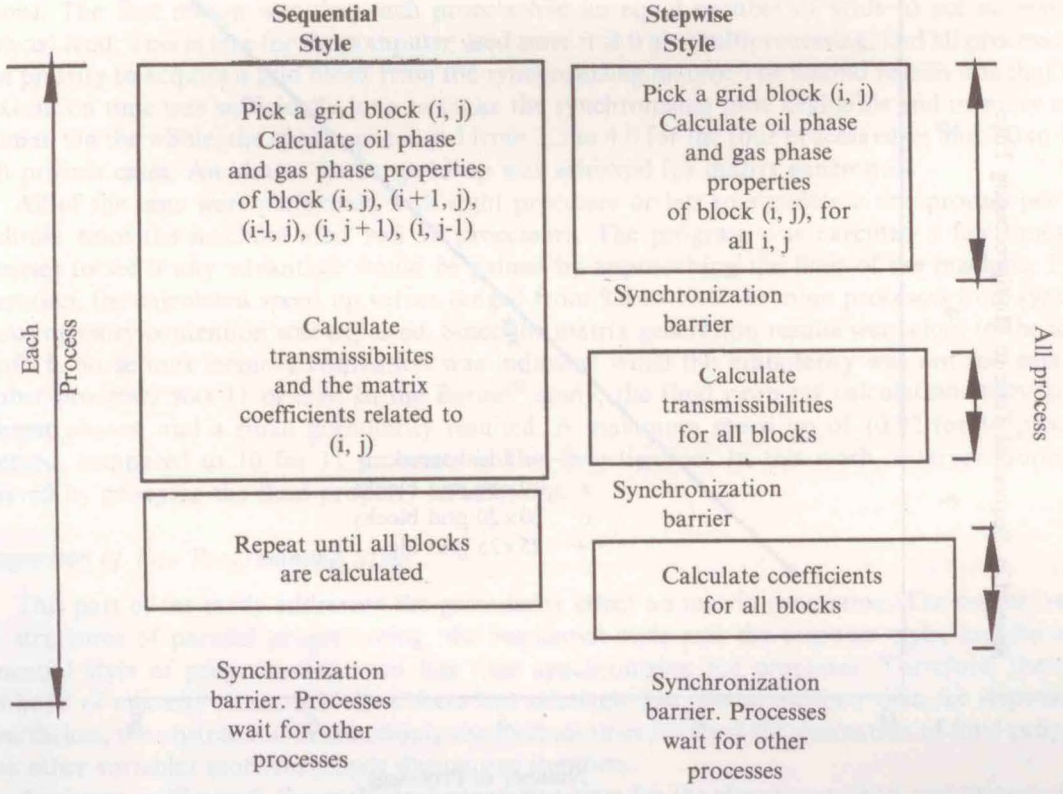

Figure 4: Comparison of the sequential style and stepwise style of programming

The speed up of the calculations can be estimated theoretically. There are N_xN_y sets of independent calculations for a simulator that has N_xN_y grid blocks. If there are *Nprocs* processes, there would be *Nprocs* computations at a given time. The speed up would be *Nprocs*, less synchronization and communication effects.

Effects of varying the number of processes and grid blocks

The following discussion covers the results of ten year simulation tests on three sizes of simulators, 10×10, 20×20 and 25×25 grid blocks. The results between duplicate test varied between five to ten percent. For most of the cases, the variation was six percent. The speed ups were plotted in Figure 5 showing the effect of parallelization on coefficient generation time.

From the results, the cases that were executed with four processes did not show a marked difference when the number of grid blocks increased. The average speed up value was 3.9. When the number of processes was increased to eight, the average speed up for the 10×10 case was less then the other two cases. When the number of grid blocks was 10×10 and the number of processes was four, all four processes would finish execution more or less at the same time. There would be some time lag between the first process and the last process because the synchronizing macro[1] sequentially allocates a grid block to each process. Also, some grid blocks with wells require more computation. When the number of processes was increased to eight, the last four grid blocks were computed by four processes, and the remaining four processes were idle. Load imbalance caused a loss in the speed up. Loss owing to synchronization time was minimal because the shared memory architecture of the computer allowed rapid access and data transfer.

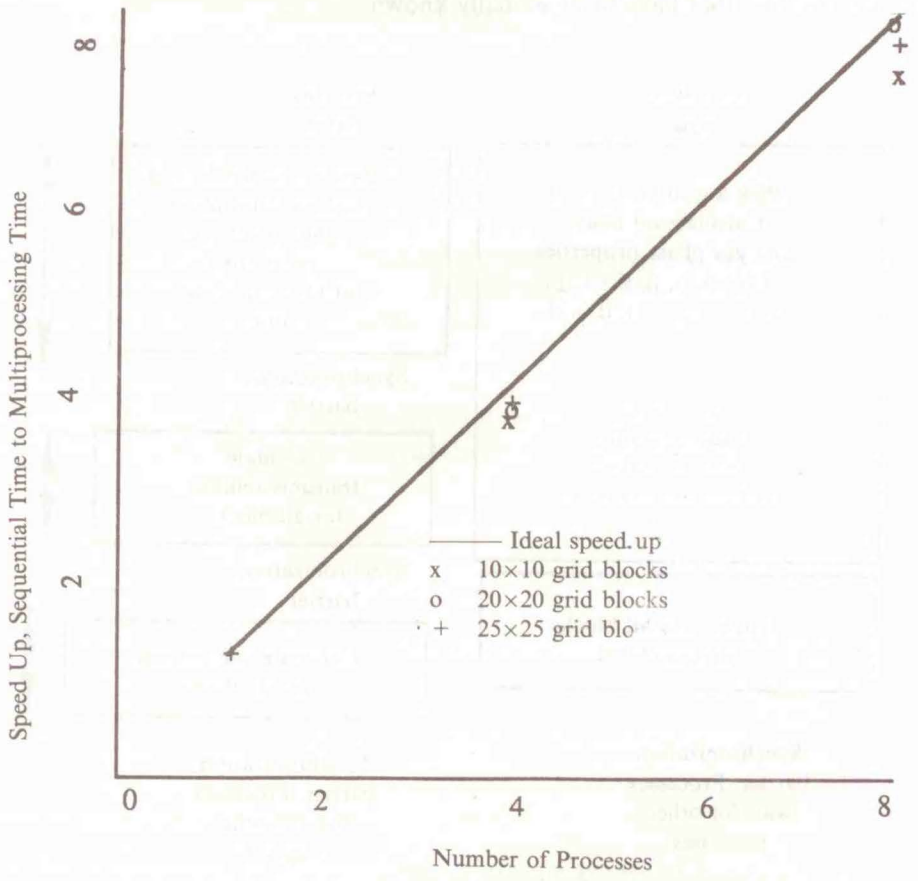

Figure 5: Speed up of matrix generation time

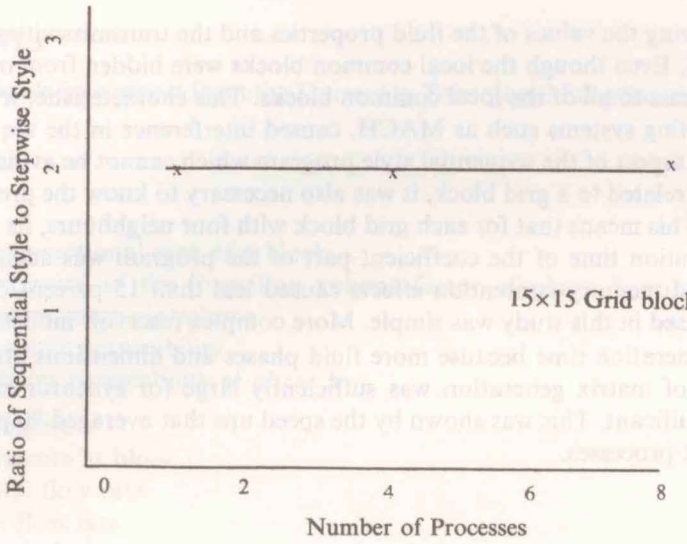

Figure 6: Comparison of the time required by the stepwise method and the sequential method.

The single shared variable method that is used in the ANL synchronizing macro[1] caused processes to try to access one memory address simultaneously and time loss due to memory contention occurred. Due to the small number of processes that were tested, memory contention was not expected to be a major problem in this work. The total loss was also significant for a small model such as 10×10 grid blocks owing to the short execution time. From the values of speed up, the 20×20 model gave the best performance for two reasons. The first reason was that each process had an equal number of grids to act on resulting in a balanced load. This is true for the computer used since it is truly multiprocessing, and all processes have the same priority to acquire a grid block from the synchronizing macro. The second reason was that the length of execution time was sufficiently long to make the synchronizing time negligible and memory contention minimal. On the whole, the speed ups ranged from 3.5 to 4.0 for the four process cases and 7.0 to 7.9 for the eight process cases. An almost linear speed up was achieved for matrix generation.

All of the tests were performed with eight processes or less to simulate a one process per processor condition since the machine used had 12 processors. The program was executed a few times using 11 processes to see if any advantage would be gained by approaching the limit of the machine. For matrix generation, the calculated speed up values ranged from 9.6 to 10.0. As more processes were synchronized, serious memory contention was expected. Since the matrix generation results were close to the ideal speed up of 11, no serious memory contention was indicated when the granularity was not too small and the number processes was 11 or less. In the Barua[4] study, the fluid property calculations were grouped by different phases, and a small granularity resulted. A maximum speed up of 10.92 for 14 processes was observed, compared to 10 for 11 processes in this investigation. In this work, a larger granularity was achieved by grouping the fluid property calculations.

Comparison of Two Programming Style

This part of the study addressed the granularity effect on matrix generation. The results from testing two structures of parallel programming, the sequential style and the stepwise style, are discussed. The sequential style of programming used less time synchronizing the processes. Therefore, there was less likelihood of memory contention since there was relatively less shared memory than the stepwise method. Nevertheless, the abstraction of commonly used subroutines required the evaluation of fluid properties and some other variables more than once during one iteration.

As shown in Figure 6, the coefficient generation time for the stepwise method was twice as fast as the sequential method when applied to an oil and gas reservoir. The sequential style program was expected to

require less time by saving the values of the fluid properties and the transmissivities in common blocks that were local to a process. Even though the local common blocks were hidden from other slave processes, the master process had access to all of the local common blocks. This characteristic, which is present in UNIX or UNIX-based operating systems such as MACH, caused interference in the sequential style program.

An unfavourable aspect of the sequential style program which cannot be avoided was that to calculate the matrix coefficients related to a grid block, it was also necessary to know the properties of its connecting neighbouring blocks. This means that for each grid block with four neighbours, its values will be calculated four times. The calculation time of the coefficient part of the program was adequately large so that the synchronizing time and memory contention effects caused less than 15 percent deviation from the ideal speed up. The model used in this study was simple. More complex reservoir models are expected to require a larger coefficient generation time because more fluid phases and dimensions are involved.

The granularity of matrix generation was sufficiently large for synchronization time and memory contention to be insignificant. This was shown by the speed ups that averaged 90 percent of the ideal speed ups for four and eight processes.

Overall Speed Up

From the results of the sequential program of a 15 × 15 grid block model, the percentage of the total time spent on matrix generation was 34.0 and on matrix solution was 60.0. Six percent of the total time was spent on convergence test and updating common variables. Therefore, the speed up of the overall execution time was dominated by the matrix solver. In order to benefit fully from parallel programming, a parallel solver should be used. The minimum execution time, t_{min}, for a 15 × 15 model of the simulator used in this study can be estimated by

$$t_{min} = \frac{0.34}{sp_{mg}} t + 0.06\, t + \frac{0.60}{sp_{ms}} t \qquad \text{Eqn. 3}$$

where t is the total execution time,
sp_{mg} is the maximum speed up for matrix generation,
and sp_{ms} is the maximum speed up for matrix solution.

The maximum possible speed up of matrix generation is ten for 11 processes as measured in this work. The maximum possible speed up of matrix solution is 8.25 for 11 processes if the matrix solver used by Barua[4] is used. Assuming that the convergence test and updating were carried out sequentially, and subtituting the pertinent values into Equation 3, the minimum excution time is 0.167 t. The maximum possible speed up the simulator is 5.99 for 11 processes or a sixfold increase in the computation rate could be achieved.

CONCLUSIONS

Matrix generation can be parallelized to give a maximum speed up of ten for 11 processes. An almost linear speed up was obtained for different sizes, specifically 10×10, 20×20 and 25×25 grid blocks. The highest deviation from the ideal speed up was observed for the 10x10 case. The averaged speed up was 0.90 of the ideal speed up.

The sequential style of programming was slower than the stepwise method due to the recalculation of variables. This method seemed to be suitable for a message-passing machine but not for a tightly coupled shared-memory machine.

Load imbalance is another factor that can cause considerable slow down in the excution time. Some optimization of load balancing and granularity should be carried out before a simulation run.

For a 15×15 grid block reservoir model, a maximum overall speed up of 5.99 for 11 processes is possible if the stepwise structure was used for matrix generation and the matrix solution method in Barua[4]'s work was used.

ACKNOWLEDGEMENTS

This work was made possible by support from the University Teknologi Malaysia and Stanford University.

NOMENCLATURE

Symbol

A	cross-sectional area of a block
b_1	reciprocal of the formation volume factor of phase 1, volume at standard conditions/reservoir volume
k	absolute permeability
k_{r1}	relative permeability of phase 1
p	pressure
p_i	pressure at block i
q	total flow rate
q_o	oil flow rate
q_w	water flow rate
S	saturation
S_1	saturation of 1 phase
$TX_1 = \lambda A/\Delta x$	transmissivity of 1 phase in the x direction
$TY_1 = \lambda A/\Delta Y$	transmissivity of 1 phase in the y direction
t	time
Δt	time step change
V_b	bulk volume
$\lambda = k\, k_{r1}/\mu_1$	mobility of 1 phase
$\mu 1$	viscosity of 1 phase
ϕ	porosity

Abbreviations

MIMD	multiple instruction multiple data
MIPS	million instructions per second
SOR	successive over relaxation
GOR	Gas-oil ratio, volume of gas volume of oil

REFERENCES

1. E. L. Lusk and R. A. Overbeek, Implementation of Monitors with Macros: A Programming for the HEP and Other Parallel Processors, Argone National Laboratory, Argonne, Illinois, 1983.
2. G. Bell, The Future of High Performance Computers in Science and Engineering, *Communications of the ACM*, vol 32, no 9, pp 1091–1101, September, 1989.
3. G. R. Andrews and F.B. Schneider, Concepts and Notations for Concurrent Programming, *Computing Surveys*, vol 15, no 1, March 1983.
4. J. Barua, A study on Newton Related Nonlinear Methods in Well Test Analysis, Production schedule Optimization and Reservoir Simulation, PhD Thesis, Stanford University, 1989.
5. J. E. Killough and M.F Wheeler, Parallel Iterative Linear Equation Solvers: An Investigation of Domain Decomposition algorithms for Reservoir Simulation, *SPE 16021*, Ninth SPE Simposium on Reservoir Simulation, San Antonio, Texas, February 1–4, 1987.
6. J.R Walis, J.A. Foster and R.P. Kendall, A New Parallel Iterative Linear Solution Method for Large Scale Reservoir Simulation, *SPE 21209*, Eleventh SPE Symposium on Reservoir Simulation, Anaheim, California, 1991.
7. K. Aziz and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Publishers, London, England, 1979.
8. L.C Young, Equation of State Compositional Model, *SPE 16023*, Ninth SPE Symposium on Reservoir Simulation, San Antonio, Texas, February 1–4 1987.
9. M. Awang, *Application of Parallel Programming to Reservoir Simulation*, PhD Thesis, Stanford University, 1991.

10. P. Lee, *Major Advances in Parallel Processing*, Editor C. Jesshope, Technical Press, Brookfield, England, Ch. 16, pp 204-210, 1989.
11. R.P. Kendall, J.S. Nolen and P.L. Stanat, The Impact Of Vector Processors on Petroleum Reservoir Simulation, *Proceedings of the IEEE*, vol 72, no 1, pp 85–89, January 1984.
12. S.L. Scott, R.L. Wainwright, R. Raghavan and H. Demuth, Application of Parallel (MIMD) Computers to Reservoir Simulation, *SPE 16020*, Ninth SPE Symposium on Reservoir Simulation, San Antonio, Texas, February 1–4, 1987.

APPENDIX

A.1 Properties of Oil-Gas Model

The dimensions of the model and flow rates of the wells are:

(1) Reservoir area = 10000 ft × 10000 ft.
(2) Thickness = 100 ft.
(3) For 10 × 10 grid block model:
    (a) Grid size, × direction = 2000.00 ft.
    (b) Grid size, y direction = 2000.00 ft.
(4) Injection rate = 100 MMSCF/day.
(5) Total production rate = 157330.2 cu.ft day.
(6) Absolute permeability in the x direction = 215 md.
(7) Absolute permeability in the y direction = 215 md.

A.2 Oil and gas relative permeabilities

	Relative permeabilities	
Oil saturation	Oil phase	Gas phase
0	0	1.0
0.001	0	1.0
0.020	0	.997
0.050	0.005	.980
0.12	0.025	.700
0.20	0.075	.35
0.25	0.125	.200
0.3	0.19	.09
0.4	0.41	.021
0.45	0.6	.01
0.5	0.72	.001
0.6	0.87	.0001
0.7	0.94	.000
0.85	0.98	.000
1.0	1.0	.000

A.3 Oil properties

Pressure psia	FVF bbl/STB	Viscosity cp	GOR SCF/STB
14.7	1.062	1.04	1.0
264.7	1.15	0.975	90.5
514.7	1.207	0.91	180.0
1014.7	1.295	0.83	371.0
2014.7	1.4350	0.695	636.0
2514.7	1.500	0.641	775.0
3014.7	1.565	0.5940	930.0
4014.7	1.695	0.510	1270.0
9014.7	1.5790	0.70	1270.0

A.4 Gas properties

Pressure psia	FVF bbl/STB	Viscosity cp
14.7	.166666	0.008
264.7	.012093	0.0096
514.7	.0062274	0.0112
1014.7	.003197	0.014
2014.7	.001614	0.0189
2514.7	.001294	0.0208
3014.7	.001080	0.0228
4014.7	.000811	0.0268
5014.7	.000649	0.0309
9014.7	.000386	0.047