Distribution of The CPU Time of a Reservoir Simulator

Dr. Mariyamni Awang
Jabatan Kejuruteraan Petroleum

Abstract

The aims of this investigation was to obtain the time consuming computations in a reservoir simulator. A fully implicit oil-gas model was tested. The Newton-Raphson technique and the Gaussian elimination method were used in the simulation. To reduce the overall CPU time, the results showed that small simulators need efficient table look-up methods, while large simulators need efficient matrix solvers.

Introduction

In programming a mathematical model, an important consideration is the computer time that is required to run the program. The choice of algorithms is often an optimisation of efficiency and accuracy.

The CPU time needed by a reservoir simulator may be reduced substantially, using fully implicit models [Aziz and Settari (1979)] and high perfomance or parallel computers [Awang (1991)]. More CPU time can be saved by modifying or optimizing subroutines that are time consuming.

The objective of this study was to identify the high CPU usage areas in a typical reservoir simulator using the debugging program.

Measurement were made using IBM 4831 and the reservoir model tested was a to-dimensional two-phase black oil simulator. The program was written in FORTRAN.

Description of an Oil-Gas Reservoir Simulator

The formulation of different types of simulators are described in detail by Aziz and Settari (1979) and Peaceman (1977). The model programmed for this work was a two-dimensional fully implicit oil-gas model. The following assumptions were applied in this work:

1. The oil phase. gas phase and the rocks are compressible
2. The models are an areal, two-dimensional model.
3. Capillary and gravitational effects are not included.
4. Absolute permeability is constant.

The Newton-Raphson technique was used to solve equations and, as a result, a block pentadiagonal matrix was formed. The matrix was solved was solved by Gaussian elimination. The algorithm of the program and the function of each subroutine are given in the appendix.

Description of the debugging Software

The computer used in this work has a timing subprogram. The accuracy of the timing subprogram is not adequate for short execution times due to the layers of subprograms between the hardware level clock and the timing subprogram. Better measurement may be obtained by specialized timing methods.

The procedure that is available on most IBM mainframes is a debugging option, and is referred to as a 'hotspotting' option. Among other capabilities, 'hotspotting' indicates the percentage of CPU time that is spent in each subroutine during excecution. The CPU time is not measured directly, but is deduced from sampling measurements. At every ten microsecond interval, a sample is taken from each subroutine. The number of samples that has been taken is reported as a percentage at the end of the execution. Since the length of time that is spent in a subroutine is related to the number of samples taken, the percentage is a measure of the CPU time. The subroutines that are 'hotspots' or bottlenecks would give higher percentage than other subroutines. The percentage profile does not indicate the absolute CPU time. Therefore, a comparison of execution time of differen test is not possible.

Result

The advantage of using the debugging program is that the number of samples taken from every subroutine during execution is automatically recorded. The alternative of using timing subprograms would involve placing CALL statements before and after every subroutine, which is tedious and inaccurate. Test were made using 10X10, 20X20, and 30X30 grid block models.

High CPU usage areas

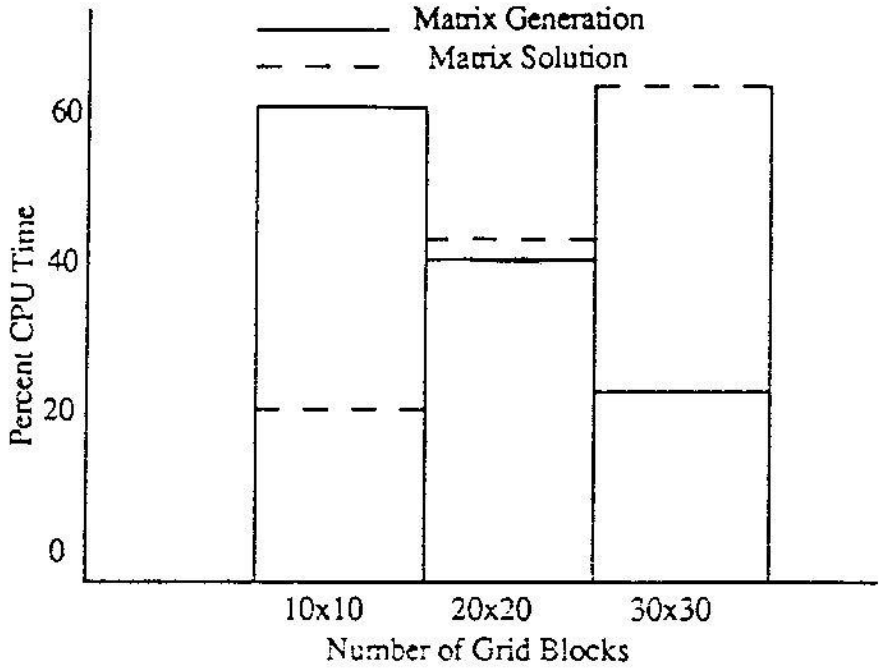
From the 10X10 grid block test, 46.63 percent of a CPU time was spent on looking up data and 20.65 percent of the CPU time was spent on solving the matrix. Efficient table look-up methods and fast matrix solvers should reduce this bottlenecks.

Other time consuming area were found to be the calculation of the derivatives of the fluid properties and the derivatives of the transmissivities. The evaluation of the derivatives involves finding the gradient of the properties at the desired pressure or saturation. This means searching at least two values of each property and dividing by either the pressure change or the saturation change. Efficient table look-up methods will reduce this portion of CPU time. The calculation time of the transmissivities and their derivaties can be reduced by vectorising. Also, the use of COMMON blocks in the program will result in less calculation since no recalculation will be necessary. Since transmissivities and the derivaties of transmissivities are dependent on fluid properties and their derivatives, greater saving can be achieved by hastening fluid property calculations.

Effect of Grid Blocks

A comparison of the CPU times of the 10X10, 20X20 and 30X30 grid block models is given in Figure 1. The matrix generation step, which included table look-up, formed only 28.00 percent of the total CPU time for the 30X30 grid block model. In compaarison, the 10X10 grid block model used 63.50 percent of the CPU time to generate the matrix and 22.00 percent of the CPU time to solve it. This means that the dominant step changed as the number of grid blocks increased. Therefore, for a 30X30 grid block and larger models, a more efficient matrix solver must be used in order to reduce the overall computation time.

Fig. 1 Effects of Grid Blocks
on Percent CPU Time



Conclusions

1. For a 10X10 grid model, the table look-up step was dominant. A more efficient table look-up method would reduce the CPU time.

2. For a 30X30 grid block model, the matrix solver was the most time consuming subroutine. A more efficient matrix solver must be used for large models.

3. The debugging program allowed a simple way of identifying the areas that controlled the overall computation time of a program.

References

1. M. Awang (1991), Application of Parallel Programming to Reservoir Simulation, PhD thesis, Stanford University.

2. K. Aziz and A. Settari (1979), Petroleum Reservoir Simulation, Applied Science Publishers, London.

3. D. W. Peacement (1977), Fundamentals of Numerical Reservoir Simulation, Elsevier Scientific Pub. Company, Amsterdam.

APPENDIX

The algorithm of the simulator is as follows:

1. Initialization: Input data, such as fluid properties and initial conditions, were read.
2. The fluid properties, transmissivities and their derivatives were calculated for each block.
3. The elements of the matrix were calculated.
4. The matrix was solved for the pressure and saturation of each block.
5. Convergence test: If true, the calculation proceeded to the next time step, else the calculations were repeated for the next iteration. Repetition began at step (2).

Table A1. Functions of Subroutines in a Simulator

STEP	SUBPROGRAM	FUNCTION
Matrix Generation	Table look up	Using bisectional search to get index
	Oil, gas, relative permeabilities	Interpolation of data using index
	Derivatives, transmissivities	Evaluation of derivatives, transmissivities
	Jacobian, residuals	Evaluation of matrix elements
Matrix solver	Store	Storage of arrays as vector
	Reduction, Inversion	Solution of matrix