Parallel Reservoir Simulation Based on Domain Decomposition Techniques.

Petter Bjørstad[1], Randi Moe[1], Rudi Olufsen[2], and Eero Vainikko[1]

[1] Para//ab, Institutt for Informatikk,
University of Bergen, Høyteknologisenteret N-5020 Bergen, Norway.
Petter.Bjorstad@ii.uib.no
Randi.Moe@ii.uib.no
Eero.Vainikko@ii.uib.no
[2] Technical Software Consultants AS
Gaustadalléen 21, N-0371 Oslo, Norway.
rudi@tscas.no

Abstract. We have studied parallelisation methods for the pressure equation in the reservoir simulator FRONTSIM *** for three-dimensional reservoirs. In order to obtain a faster simulator and finer resolution, we have studied the use of domain decomposition methods combined with the use of parallel processing. Both the additive and multiplicative Schwarz methods have been implemented together with coarse grid solver techniques. The code is parallelised using the PVM message passing library for communication and has been tested on a cluster of workstations as well as on a tightly coupled multiprocessor machine. This effort is part of the EC-ESPRIT III / EUROPORT2 project. Some implementational results are given.

1 Introduction

The reliability of a reservoir model depends on the level of details in critical areas. To be able to exploit oil fields with only marginal economics, more reliable results are needed. Therefore, for future planning of oil fields, larger models are needed. One would like to be able to run models with up to one million grid blocks. Currently, such large models can only be run in parallel on a number of processors. FRONTSIM, developed by Technical Software Consultants AS (TSC), is a software package for petroleum reservoir simulation. FRONTSIM is rather fast on a large class of reservoir problems, and therefore it has been natural to develop a parallel code for FRONTSIM in order to apply it on models with up to several million grid blocks. Acceptable simulation time for such problems is important for the planning of future oil fields.

2 Code Description

The simulator FRONTSIM [2, 5] handles compressible, two-phase, black-oil problems for two- or three-dimensional heterogeneous reservoirs. The problem is dis-

*** FRONTSIM is developed by TSC AS.

cretised using the finite element method. An important part of the simulator is the solution of the pressure equation of incompressible flow which reads

$$\nabla \cdot \mathbf{u}_T = Q$$

where $\mathbf{u}_T = -\lambda_T K(\nabla P - \rho_T g \nabla D)$, K is a permeability tensor, λ_T is the total mobility, ρ_T is the average density, g is the gravitational acceleration, D is the vertical distance in the reservoir, and Q is the source term for the wells.

Discretisation of the equation gives a symmetric and positive definite matrix so that e.g. the method of conjugate gradients (CG) can be used to solve the resulting system of linear equations. In order to obtain faster convergence, especially in the case of larger problem sizes, the general strategy is to use an effective preconditioner.

2.1 Code Structure Outline

The main parts of FRONTSIM are the pressure and the saturation solvers. For a three-dimensional case, the pressure solver in the serial version of FRONTSIM typically uses 50–90% of the total CPU time, most in the case of compressible flow. As the problem size increases, the pressure solver part is also increasing. The underlying reason for this is that the pressure solver uses a three-dimensional grid where the computational work is growing very rapidly as the problem size is increased. The computational work in the saturation solver, however, normally grows proportional to the surface of the oil/water interface. This is the unique feature of the FRONTSIM package, and leads to increased accuracy in describing the interface and to less computational work in the saturation part of the solution procedure.

More important than the growth of the computational work, is the growth of the memory requirements as the problems are refined. Again, this growth is more severe for the pressure solver than for the saturation solver. Hence, the memory requirements of the pressure solver together with the computational work are the major bottlenecks for simulating larger problems for the end-users of FRONTSIM. Hence, in porting this code to a parallel platform we have focused on parallelising the pressure solver, and we have limited ourselves to the incompressible cases.

The FRONTSIM code can be divided into four basic blocks in addition to the initialisation part as formulated in Algorithm 1.

Algorithm 1 (Main loop of the FRONTSIM code)

Initialisation (1)
for *each time step*
    Assemble the matrix of the linear system for solving the pressure equation (2)
    Solve the pressure equation (3)
    Compute the velocity field and other pressure dependent parameters (4)
    Solve the saturation equation (5)
end for

The steps (2) and (3) in the algorithm are the vital parts of solving the pressure equation and hence the focus of our parallelisation effort in this project. The initialisation phase (1), the pressure dependent parameters (4), and the saturation solver (5) are still computed sequentially.

2.2 Domain Decomposition Methods as Preconditioner

Domain decomposition techniques have received much interest in recent years, *cf. eg.*, Chan et al. [3], and Keyes et al. [4]. This is to a large extent due to their potential on multiprocessor computing systems. These methods are based on a subdivision of the computational domain into subdomains of reduced size and sometimes simpler geometrical shape. A preconditioner (an approximation to the original problem) can then be derived by considering the problem restricted to subdomains plus an appropriate coupling mechanism between the new (and smaller) subproblems. An application of the preconditioner therefore involves the solution of a family of subproblems of lower discrete dimension. The resulting methods should almost always be accelerated by some Krylov space method, *ie.,* the Conjugate Gradient method or some appropriate nonsymmetric variant like GMRES or BICGSTAB. If the complexity of the subdomain solver grows faster than linearly in the number of unknowns, then the overall complexity is reduced as we "divide and conquer" the domain. Thus an improved performance can be obtained also in a sequential setting. Naturally, the potential gain is larger when the individual subdomains can be processed in parallel.

We will exclusively consider overlapping methods, often referred to as Schwarz type methods, in this paper. Both the additive and multiplicative Schwarz methods have been implemented. In order to restore favourable convergence properties when a large number of subdomains is used, two-level methods solving an extra coarse grid problem is introduced. This approach can be proved optimal in the sense that the condition number has an upper bound independent of the discretisation parameter and of the number of subdomains as long as the overlap is constant. See [1] for further details on the domain decomposition methods used. Two-level methods can be combined with both multiplicative and additive methods. Our best results regarding CPU time have been achieved using the two-level additive method while the best convergence rates have been achieved using the two-level multiplicative method.

2.3 Implementational Issues

In the current implementation, the global domain has been divided into n by m subdomains of approximately equal size. The reservoir has vertical production and injection wells. We assume that the reservoir is split into subdomains in such a way that wells do not pass through subdomain boundaries. Thus, with vertical wells we end up with subdomains with possible neighbours only in the xy plane. Usually, the number of grid blocks needed to model a reservoir is much smaller in the z-direction. The overlap between neighbouring subdomains is at least one extra grid block, *ie.,* two neighbouring subdomains have at least one common gridplane. Figure 1 shows an example domain divided into four subdomains with this overlap. A slightly larger overlap of two grid blocks has been used in the reported computational results.

In order to achieve an efficient implementation one should use an approximate solution technique to solve the subdomain problems and the coarse grid problem. We have implemented both an exact solver, using the Cholesky factorisation, and approximate solvers as an incomplete Cholesky factorisation without fill-in, the symmetric Gauss-Seidel iteration (SGS) and the multigrid method. In general, using approximate solvers in the preconditioning step increases the number of conjugate gradient iterations but when considering the total running time and memory consumption, it is usually much more efficient than exact solvers. Incomplete Cholesky factorisation and SGS have shown to be best on smaller problems sizes while the multigrid method has given the best result for solving the coarse grid problem in the case of large reservoirs.

When using multigrid as the coarse grid solver, some preliminary work is needed to build up the coarse matrix hierarchies. One could assemble the coarse problem stiffness matrices in the same way as it was done for fine grid matrices. However, we are building the coarse problem matrices through Galerkin formulation in order to avoid upscaling the geological data as well. This is done by using interpolation, restriction and matrix-vector operations.

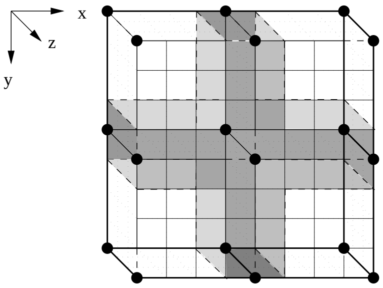

Fig. 1. A splitting of the reservoir into 4 subdomains with minimum overlap. The dark bullets denote the coarse grid nodes.

3 Parallel Implementation

The parallel version of FRONTSIM is implemented using the message passing library PVM and is based on the master-slave model. The master program executes all but the pressure equation solver sequentially. The first time the pressure equation is solved, the master process spawns the requested number of new processes and sends information about the problem sizes to the new processes. Algorithm 2 shows the part of the master program for solving the pressure equation.

Algorithm 2 (Additive Schwarz - master program)

```
begin
    if (more than one processor) then spawn new tasks
    if (first visit) then
        if (more than one processor) then send time independent data
        generate the local domain decomposition data structure
    endif
    for each time step
        if (more than one processor) then send time dependent data
        for each local subdomain Ω_i
            assemble the matrix A_i and the vector b_i
            if (twolevel) then generate the matrix A_{0,i}
        endfor
        if (twolevel) then
            receive A_{0,i} from the other tasks for all non-local subdomains Ω_i
            assemble the matrix A_0
        endif
        apply the preconditioned conjugate gradient method to compute x_i
        for each local subdomain Ω_i
            update the global pressure vector x based on x_i
        endfor
        if (more than one processor) then
            receive x_i's from the other tasks and update the global vector x
    endfor
end
```

For simplicity it is assumed in Algorithm 2 that the coarse grid solver is residing on the master process, however, one of the slave processes can also handle the coarse solver.

Algorithm 3 (Additive Schwarz - slave program)

```
begin
    receive time independent data from the master process
    generate the local domain decomposition data structure
    for each time step
        receive time dependent data from the master process
        for each local subdomain Ω_i
            assemble the matrix A_i and the vector b_i
        endfor
        if (twolevel) then generate the matrix A_{0,i} and send to coarse solver
        apply the preconditioned conjugate gradient method to compute x_i
        for each local subdomain Ω_i
            send the vector x_i to the master process
        endfor
    endfor
end
```

After the completion of the conjugate gradient iteration, the master process receives the parts of the solution vectors computed by the other processes. The new processes execute the program shown in Algorithm 3. All processes take an equal part in the computations with the master process as the coordinating process which also handles the interface to the rest of the simulator. The slave processes are hence idle during the saturation solver.

Algorithms 2 and 3 show that there is a need of a major transfer of information initially from the master to the slave processes in order to distribute static data read and computed by the master. With a large number of subdomains this transfer has to be organised in such a way that all slave processes are activated as soon as possible. Since there is one single source of this communication, namely the master process, this process can easily be a bottleneck to parallel efficiency. We have chosen to send smaller chunks of data to each process in a round-robin fashion.

Next load of communication will be at the beginning of each time step when the time dependent data has to be distributed from the master process to the slave processes. The communication will have to follow the same pattern as for the static data initially as again there is one source and several targets for the information to be transferred. As soon as a slave process has received time dependent data for a subdomain, it can start the assembling of the local matrix for this subdomain.

Assembling the subdomain matrices will now be local to each subdomain and hence no communication is needed during this phase. If a factorisation of the matrix is needed for the preconditioning step, this can be computed as soon as the assembling is completed. Within the solution of the pressure equation, the communication will follow a somewhat different pattern as the transfer will mainly be between various, and possible independent, pairs of slave processes.

At the end of each time step, the computed pressure will be transferred back to the master process. This time there will be one single receiver for the communication with all the slave processes as sources. The bottleneck will now be the master's capacity of receiving and processing the messages, however, the simulator will now enter a sequential phase anyway because of the sequential saturation solver. With a large number of subdomains, the matter of load balancing is solved by distributing a relatively equal part of the subdomains to each process where the relative sizes is also taken into account.

4 Performance Results

The main parallel results have been obtained using a cluster of workstation, *ie.,* up to 8 Dec Alpha 1000/2000 connected by FDDI and Gigaswitch. In addition, some tests have been run on a Parsytec GC/PowerPlus.

4.1 Test Cases

In order to illustrate the parallel performance of the current parallel implementation of FRONTSIM, we're showing results using a five-spot test case with homogeneous geology. The grid sizes are $128 \times 128 \times 10$ and $128 \times 128 \times 40$ grid blocks, *ie.,* in total 163 840 and 655 360 grid blocks, respectively. The producers are located in the four corners of the reservoir and the injector in the middle. All wells are vertical perforated through the reservoir. The case run one time step of ten years. The global domain is divided into 11×11, *ie.,* a total of 121 subdomains. Two-level additive Schwarz is used as preconditioner and as solvers on the subdomains and the coarse grid, respectively Incomplete Cholesky factorisation and multigrid is used.

4.2 Cluster of Workstations

Currently, test cases with more than 800 000 grid blocks have been run on the cluster of Dec Alpha workstations. However, the same code has been used to simulate successfully a test case of up to one million grid blocks on a cluster of SUN workstations.

Figure 2a shows the performance improvement of running the smaller test case of 163 840 grid blocks on an increasing number of processors. The timings refer to one time step of a simulation restricted to the parts of the code subject to parallelisation. The timings are split into three different parts. The initialisation part is work done before the time stepping loop and increases with increasing number of processors. This reflects the added overhead with several processors as input data is read by one processor and distributed to the others according to the splitting of the global domain. However, this part is done only once in each simulation and will be relatively less important in a full simulation which typically involves a large number of time steps. The timings of the assembling and the pressure solver refer to one invocation and will be repeated for each time step. These parts show a fairly good parallel efficiency, the speedup factor on eight processors is 5.7. As the problem size increases, similar speedup results are difficult to give as several processors are needed to be able to run such cases. However, Figure 2b shows timings for the largest test case using six and eight processors. The relative speedup is 1.4 in this case. In comparison, the same test case with 819 200 grid blocks run on eight processors consumed about 45% more CPU time on the corresponding parts of the simulation.

4.3 MIMD Computer

More difficulties were encountered when porting the parallel FRONTSIM code to the Parsytec platform. Due to the serial part of the simulation, the master process needs a large amount of internal memory. Hence, the master process could only reside on the front-end of the Parsytec computer. Thus, only Heterogeneous PVM could be used for message-passing and this product is still not fully matured. The communication between the front-end and the processor nodes is a substantial bottleneck in this project. We have completely avoided communication between the front-end and the nodes during the pressure equation solution. We can only give results of a small test case with $128 \times 128 \times 5$ grid blocks (See Figure 3). In this case, Incomplete Cholesky is used both as fine and coarse grid

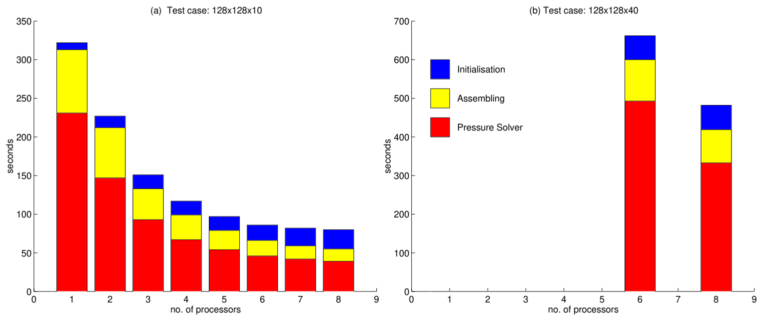

Fig. 2. Timings for two test cases run on a cluster of workstations.

solver. The number of subdomains is 5×5 and two-level additive Schwarz is used as preconditioner. We encounter speedup in the pressure equation solution times with adding processor nodes. In a full simulation the impact of the initialisation part will be less substantial. This shows that the Parsytec computer can be quite suitable to run the parallel FRONTSIM code if better communication libraries could be provided.

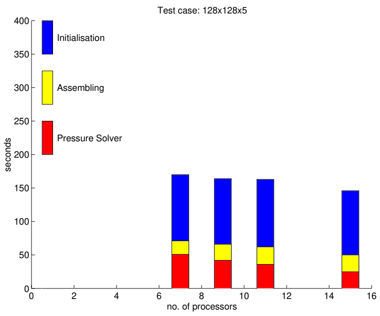

Fig. 3. Timings for a test case run on Parsytec GC/PowerPlus.

5 Conclusions

The parallel version of FRONTSIM using domain decomposition methods for solving the pressure equation allows larger models to be simulated within acceptable time limits. The memory requirements per processor is substantially lower than with the sequential version.

References

1. P. E. Bjørstad, R. Moe, R. Olufsen, and E. Vainikko, *Domain Decomposition Techniques in Parallelization of the 3-dimensional FRONTSIM code*, in Parallel Programming and Applications, P. Fritzson and L. Finmo, eds., Netherlands, 1995, IOS Press. Proceedings of the Workshop on Parallel Programming and Computation (ZEUS'95) and the 4th Nordic Transputer Conference (NTUG'95), Linköping, May, 1995.
2. F. Bratvedt, K. Bradtvedt, C. F. Buchholz, T. Gimse, H. Holden, L. Holden, and N. H. Risebro, *FRONTLINE and FRONTSIM; Two Full scale, Two-phase, Black Oil Reservoir Simulators Based on Front Tracking*, Surv. Math. Ind., (1993), pp. 185–215.
3. T. F. Chan, R. Glowinski, J. Périaux, and O. Widlund, eds., *Domain Decomposition Methods for Partial Differential Equations IV*, Philadelphia, 1991, SIAM. Proceedings of the Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Moscow, May, 1990.
4. D. E. Keyes, T. F. Chan, G. A. Meurant, J. S. Scroggs, and R. G. Voigt, eds., *Domain Decomposition Methods for Partial Differential Equations V*, Philadelphia, 1992, SIAM. Proceedings of the Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations, Norfolk, Virginia, May, 1991.
5. Technical Software Consultants A/S, Oslo, Norway, *FRONTSIM - A Front Tracking Reservoir Simulator*, 1992.