78

On Object Oriented Programming Languages as a Tool for a Domain Decomposition Method with Local Adaptive Refinement

Brit Gunn Ersland and Magne S. Espedal

1 Introduction and Model Problem

The main objective for this work is to show how Object Oriented programming languages like C++ can simplify the implementation of a complex model where domain decomposition and local adaptive refinement is used. As an example we present a simulator for two phase fluid flow (oil,water) in a porous media, where the library DIFFPACK [Lan94b, Lan94a, Dho] is extensively used. We start by constructing the base classes for the solvers, and use these as building bricks in a more complex system, where different equations are solved on different meshes with domain decomposition on the finest mesh. The decomposed domain is regarded as an array of solvers which compute the solution to an equation on a single domain with appropriate boundary conditions.

For incompressible immiscible displacement of oil by water in a reservoir the following equations yield

$$\nabla \cdot \mathbf{u} = q_1(\mathbf{x}, t) \tag{1.1}$$

$$\mathbf{u} = -\mathbf{K}(\mathbf{x}) M(S, \mathbf{x}) \cdot \nabla p \tag{1.2}$$

$$\phi \frac{\partial S}{\partial t} + \nabla \cdot (f(S)\mathbf{u}) - \epsilon \nabla \cdot (D(S, x) \nabla S) = q_2(\mathbf{x}, t) \,. \tag{1.3}$$

We will use Neumann type of boundary conditions.

$\mathbf{u}$ is the total Darcy velocity, which is the sum of the velocity of the oil and water phase. $\mathbf{K}(\mathbf{x})$ is the permeability which depend on the porous medium, $M(S, \mathbf{x})$ denotes

Ninth International Conference on Domain Decomposition Methods
Editor Petter E. Bjørstad, Magne S. Espedal and David E. Keyes

the total mobility of the phases,

$$M(S, \mathbf{x}) = \lambda_w(S) + \lambda_o(S),$$

and p is the total fluid pressure. ϕ is the porosity of the porous media which is considered constant. The fractional flow function $f(S)$ is a nonlinear function of the saturation and is given as

$$f(S) = \frac{\lambda_w(S)}{\lambda_w(S) + \lambda_o(S)} \tag{1.4}$$

where the mobility of oil and water,

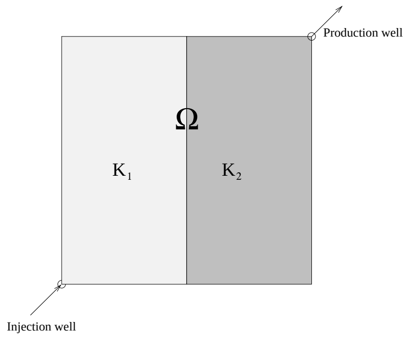
$$\lambda_l, \ l \in \{o, w\}$$

is a given function of S.

The diffusion coefficient $D(S, x)$ depend on the capillary pressure and the permeability, while ε is a small parameter. For a complete survey and justification of the model we refer to [CJ86].

Here, we regard a rectangular domain which consist of two different sediments as depicted in Figure 1. Water is injected in the lower left corner and oil is produced in the upper right corner. Initially we assume that we have an established shock somewhat away from the injection well. A combination of (1.1) and (1.2) give an

Figure 1 The figure shows a computational domain Ω with an injection well and a production well.



elliptic equation for the total pressure. This equation is solved by a finite element method on a coarse mesh, then the velocity is derived from the pressure with a second order method [Sæv90].

Since ε is a small parameter the nonlinear saturation equation (1.3) is dominated by the convective part of the equation, which denotes the main transport. It is well known that a nonlinear equation like (1.3) will establish shock like solutions even from smooth initial conditions, therefore we split the function $f(S)$ into two parts [EE87, DR82].

$$f(S) = \bar{f}(S) + b(S)S, \tag{1.5}$$

where $\bar{f}(S)$, the convex hull of the function. multiplied by $\mathbf{u}$ denotes the displacement of an established shock. Hence, we split the saturation equation in two parts. The convective part

$$\Psi(\mathbf{x})\frac{\partial S}{\partial \tau} \equiv \phi\frac{\partial S}{\partial t} + \bar{f}'(S)\mathbf{u}\cdot\nabla S = 0, \tag{1.6}$$

is solved by the Modified Method of Characteristics [DR82]. The elliptic part

$$\Psi(\mathbf{x})\frac{\partial S}{\partial \tau} + \nabla\cdot(b(S)S\mathbf{u}) - \varepsilon\nabla\cdot(D(S,\mathbf{x})\cdot\nabla S) = 0 \tag{1.7}$$

is solved a finite element method with optimal testfunctions [BM84]. Since the saturation develop shock-like solutions a fine resolution is needed to resolve the shock. However, the elliptic part of the saturation equation (1.7) only contribute to the solution in vicinity of sharp saturation gradients, therefore we want to solve this equation in these areas only.

2 Solution Procedure

Since different resolution is needed for the total pressure and the water saturation, we need to define different grid levels and a mapping between the coarse grid, fine subgrids and a global fine grid. For this problem we have used the mapping shown in

Figure 2 *Mapping between fine grid Ω_F, fine subgrids Ω_m and coarse grid Ω_C .*

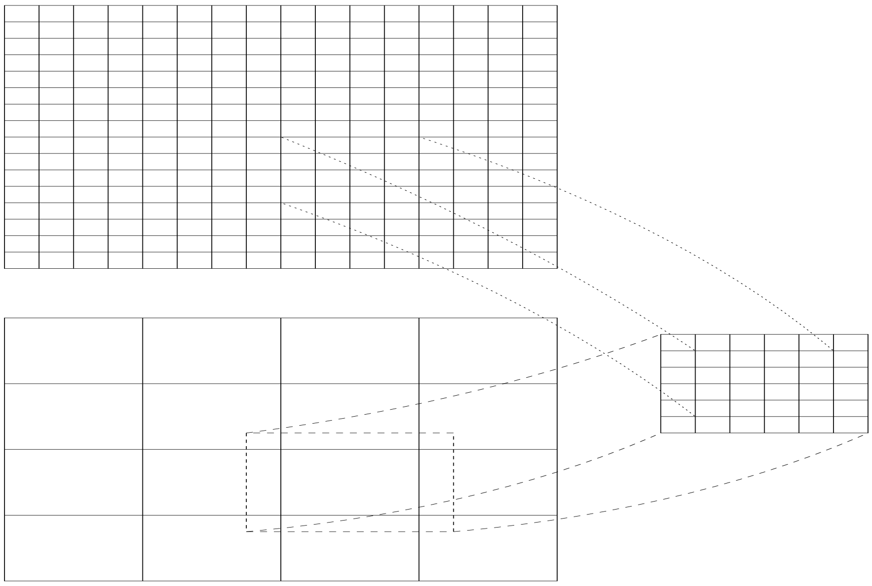


Figure 2, where Ω_C is the coarse grid, Ω_F is the global fine grid and the fine subgrids are denoted Ω_m . A sequential timestepping procedure is used [Sæv90], to decouple the equations. Hence, at every time step:

1. Solve pressure on Ω_C and determine the velocity on Ω_C .
2. Solve the hyperbolic saturation equation on Ω_C and on subgrids Ω_m , with large saturation gradients.
3. Begin
   for m = 1; number of subdomains do
   if Ω_m have large gradients
   ∘ update the boundary conditions
   ∘ solve the elliptic equation
   endif
   endo
4. Return to 3 until convergence or a maximum number of iterations is reached.
5. Map solution to Ω_F .

Implementation

In this section we will describe the design and some implementation aspects. A main objective is to build a design where each solver may be debugged independently. Now, (1.1), (1.2), (1.6) and (1.7) depend on the mobility function of oil and water, therefor we implemented these functions together with all the methods which uses the mobilities in a separate base class **FracFunc**. The permeability $\mathbf{K}(\mathbf{x})$, which depend on the porous medium is implemented as a separate class. In our test case a domain as depicted in Figure 1 is used, where the permeability depend on (x, y) . However, in order to use random permeability or several layers with different sediments only **Permeab** need to be changed.

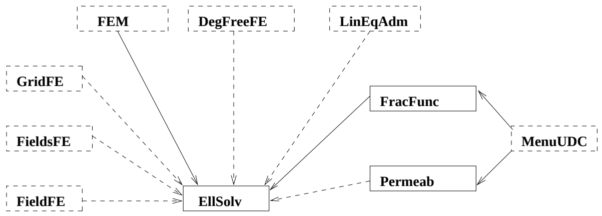
The solution procedure which was described in the previous section requires four solvers, we will just give a brief overview over them before we discuss some details regarding the implementation of the domain decomposition method.

- **Press** - inherit **FracFunc** and contain the data and methods which are needed to solve the pressure equation.
- **Velocity** - inherit **Press**, and use the pressure to derive the velocity.
- **CharSol** - inherit **FracFunc** and use the velocity to integrate backward along the characteristics to solve (1.6). First on the coarce domain Ω_C, then on the fine subdomains Ω_m where the saturation gradient is large. Here the saturation on the uniform fine mesh Ω_F at previous time level is used when we search the new solution on different grid levels.
- **EllSolv** - inherit **FracFunc**, and has a data Type **Permeab**. The class contain all the data and methods which are needed to solve the elliptic part of the saturation equation (1.7) on a single domain. Here, Dirichlet boundary conditions, Neumann type of boundary conditions, or different boundary conditions on different boundaries can be chosen.

Both **Press** and **EllSolv** is derived form the DIFFPACK base class **FEM** [Lan94a] which is a base class for Finite Element programming with DIFFPACK. In addition both **Press** and **EllSolv** has a **LinEqAdm** which is an Abstract Data Type which administers the solvers for a linear system of equations, see figure 3. **FieldFE**, **FieldsFE** and **GridFE** is scalar field, vector field and a finite element grid in the

DIFFPACK library. **EllSolv** contains a method which solves the elliptic part of the

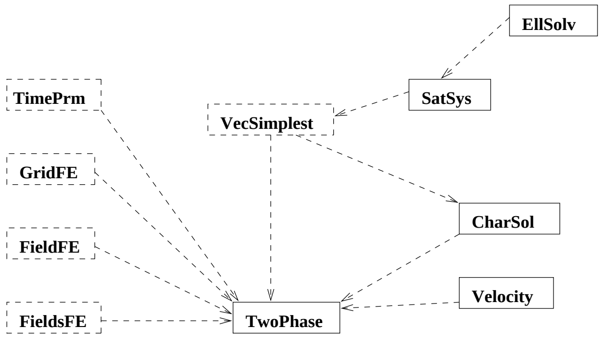
Figure 3 *Data Types and dependencies. Dotted lines indicate a "has a" relationship, while solid lines indicate a "is a" relationship. Solid boxes are newly constructed Data Types, while dotted boxes is included in DIFFPACK.*



saturation equation on a given domain Ω_m. Instead of decomposing the computational domain inside **EllSolv** and by this altering the methods in **EllSolv**, we construct a domain decomposition method by making multiple copies of **EllSolv** where the data differs. Now, to use a domain decomposition method like additive or multiplicative Schwarz, some information about the surrounding subdomains are needed. Since the elliptic saturation equation only need to be solved on subdomains where the saturation gradients are large, a boolean variable is needed to indicate if the equation need to be solved on current subdomain at this time level. Therefore, a new class **SatSys** is implemented which has a data type **EllSolv**, an array which keep track of the neighboring subdomains and a boolean variable. In addition SatSys contain methods to initiate the local domain Ω_m and mark boundaries which are at the outer boundary, where Neumann type of boundary conditions are used.

Since we want to solve the elliptic saturation by a Schwarz method, we need methods which control our subdomains. To do this we use a general vector in DIFFPACK, **VecSimplest** which can take user defined data types as argument. In our case we use **SatSys** as data type in **VecSimplest**. Let us call the method which solves

Figure 4 *Relationships and dependencies for the two phase reservoir simulator with adaptive local refinement. Dotted lines indicate a "has a" relationship, while the solid lines indicate a "is a" relationship. Solid boxes are newly constructed Data Types, while dotted boxes indicate Data Types from the DIFFPACK package.*



the elliptic saturation equation on a single domain *solveAtThisTimeLevel()* and the vector of subdomain solvers *localSys* while **EllSolv** inside **SatSys** is called *ellsolv*. In our code the method which handles the interior boundary conditions is called *updateLocBoundaries(i)* which mean that the boundary condition for domain i is updated. Hence, our multiplicative Schwarz procedure among refined grids are:

```
for( i=1; i <= nel; i++ ){
if( localSys(i).Active == TRUE) {
updateLocBoundaries(i);
localSys(i).ellsolv.solveAtThisTimeLevel();
   }
}
```

The relationship between the different classes that have been implemented is shown in Figure 4. In order to avoid multiple data, both **EllSolv** and **CharSol** point at the same objects on the same subdomain Ω_m . Likewise do **CharSol** and **Velocity** on the coarse domain Ω_C .

3 Numerical Result

The algorithms which is described in the preveous sections have been tested in 2d on the domain depicted in Figure 1, where the interface between two sediments are inside some of the subdomains Ω_m . The methods which are used to handle the interface conditions are teated in [Ers96] which also contain more numerical results and description of the experiment that is shown here. The results obtained with adaptive local grid refinement in Figure 5 shows good agreement with the results obtained on a single domain, see [Ers96].

We have shown that a domain decomposition method is easy to construct when a object oriented language as C++ is used. Some of the methods which are in **EllSolv**, **Press** and **Velocity** are initially written by K. G. Frøysa, but modified to some extent.

Acknowledgement

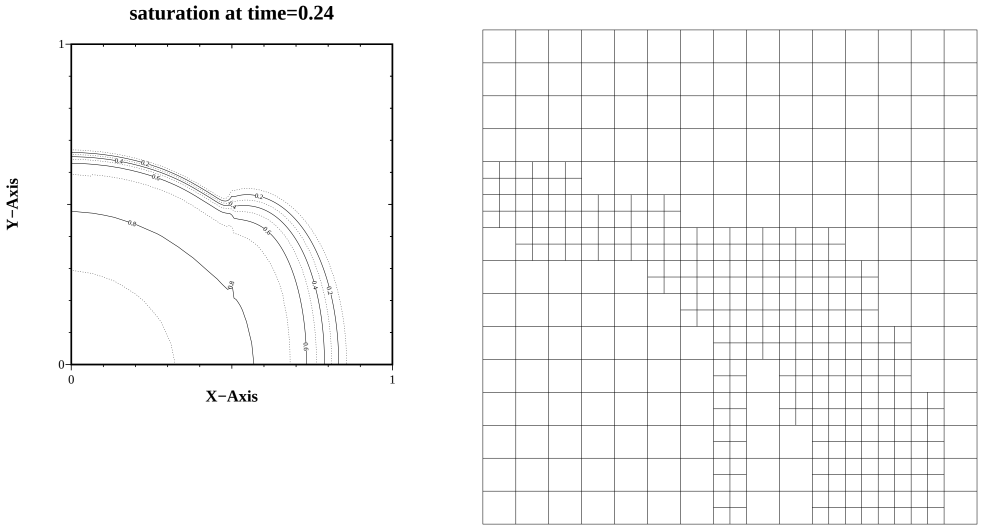
This research was supported by the Norwegian Research Counsel and the University of Bergen.

REFERENCES

[BM84] Barrett J. and Morton K. (1984) Approximate symmetrization and petrov-galerkin methods for diffusion-convection problems. *Computer Methods in Applied Mechanics and Engineering* 45: 97–122.

[CJ86] Chavent G. and Jaffre J. (1986) *Mathematical models and finite elements for reservoir simulation*. North-Holland.

Figure 5 *Computed results with adaptive local grid refinements at time level* $t = 0.24$. *The refined area are marked with a cross.*



[Dho] http://www.oslo.sintef.no/avd/33/3340/diffpack. The Diffpack WWW home page.

[DR82] Douglas J. and Russell T. (1982) Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures. *SIAM Journal on Numerical Analysis* 19: 871–885.

[EE87] Espedal M. and Ewing R. (1987) Characteristic petrov-galerkin subdomain methods for two-phase immiscible flow. *Computer Methods in Applied Mechanics and Engineering.* 64: 113–135.

[Ers96] Ersland B. (1996) *On Numerical Methods for Including the Effect of Capillary Pressure Forces on Two-phase, Immiscible Flow in a Layered Porous Medium.* PhD thesis, University of Bergen. Department of Mathematics, University of Bergen.

[Lan94a] Langtangen H. P. (1994) Details of finite element programming in diffpack. Technical report, SINTEF, informatics Oslo.

[Lan94b] Langtangen H. P. (1994) Diffpack: Software for partial differential equations. In Vermeulen (ed) *OON-SKI'94.* Proceedings of the Second Annual Object-Oriented Numerics Conference.

[Sæv90] Sævareid O. (1990) *On Local Grid Refinement Techniques for Reservoir Flow Problems.* PhD thesis, Department of Applied Mathematics, University of Bergen.