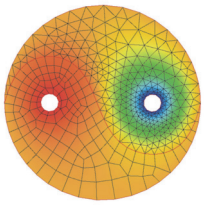
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE

Informatyka w Technologii Materiałów

Vol. 17, 2017, No. 2



A PARALLEL MIXED–HYBRID FINITE ELEMENT METHOD FOR TWO PHASE FLOW PROBLEMS IN POROUS MEDIA USING MPI

JAKUB SOLOVSKÝ*, RADEK FUČÍK

Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, Trojanova 13, 120 00 Prague, Czech Republic
**Corresponding author: Jakub.Solovsky@fjfi.cvut.cz*

Abstract

This paper deals with a numerical solution of two–phase flow problems in porous media. To solve this type of problems, we propose a numerical method based on mixed–hybrid finite element method. The method is briefly introduced for arbitrary dimension but only a 2D case is considered in this work.

We implement several variations of this method using different approaches to solve the resulting system of linear algebraic equations. Direct and iterative solvers are used and a parallel implementation of this method based on the domain decomposition method using MPI (Message Passing Interface) is described.

The accuracy and the computational efficiency of the method is verified using a problem with a known exact solution. Numerical experiments show that the errors are similar for all variations of the method. The method is convergent and the experimental order of convergence is slightly less than one. There are differences in the computational time in favor to the iterative solvers, especially when using finer meshes. For computations on fine meshes it is also advantageous to use the parallelism that significantly speeds-up the computation.

Key words: Mixed–Hybrid Finite Element Method, MPI, Parallel Implementation, Two Phase Flow in Porous Media

1. INTRODUCTION

Mathematical modeling of two phase flow in porous media can be used in many applications. Prediction of contaminant transport can be used for protection of water resources or for sanitation of dangerous substances leakage. Except for special cases, there is no known way how to solve these problems exactly, but with numerical methods, we can find at least a good approximation of the solution.

This paper focuses on the verification of the proposed numerical method using a two phase flow problem, for which an exact (semi-analytical) solution can be derived as reported by (McWhorter & Sunada, 1990; Fučík et al., 2007; Fučík et al., 2016). For computations on fine meshes, not only the accuracy, but also the computational time is an important aspect. Therefore, a parallel implementation of the method using MPI is also presented. The parallel version allows to efficiently solve problems on fine meshes.

2. NUMERICAL METHOD

Here, we present a brief description of the numerical scheme only. For the complete numerical scheme that can be used to solve broader classes of problems together with detail information about the method, refer to (Fučík et al., 2017). The method is designated to solve a system of n partial differential equations in the following coefficient form ($i = 1, \dots, n$):

$$\sum_{j=1}^{n} N_{i,j} \frac{\partial z_j}{\partial t} + \nabla \cdot \left[m_i \left(- \sum_{j=1}^{n} \mathbf{D}_{i,j} \nabla Z_j + \mathbf{w}_i \right) \right] = f_i, \quad (1)$$

ISSN 1641-8581

where $Z_j = Z_j(\mathbf{x}, t)$, $j = 1, \ldots, n$, are unknown functions $(\forall t > 0,\ \forall \mathbf{x} \in \Omega)$, $\Omega \subset \mathbb{R}^d$ is the computational domain and d is the spatial dimension, $d \in \{1,2,3\}$. $N_{i,j}$ and m_i are scalar coefficients, $\mathbf{w}_i$ are vector coefficients and $\mathbf{D}_{i,j}$ are symmetric, second order tensors. The coefficients can be functions of time t and spatial coordinates $\mathbf{x}$, but also of the unknown functions Z_j.

The system of differential equations (1) can be supplemented with Dirichlet or Neumann boundary conditions or their combination on different parts of the domain boundary $\partial\Omega$. System (1) is supplemented with the following initial conditions:

$$Z_j(\mathbf{x}, 0) = Z_{j,0}(\mathbf{x}), \forall \mathbf{x} \in \Omega, \tag{2}$$

where $Z_{j,0}(\mathbf{x})$ is the initial condition for Z_j.

The considered domain $\Omega \subset \mathbb{R}^d$ is divided into elements. The elements are line segments in one dimension (1D), triangles in two dimensions (2D) and tetrahedrons in three dimensions (3D). For element edges and boundaries, the following notation is used:

$\mathcal{K}_h$	is the set of all elements,
$\mathcal{E}_h$	is the set of all edges,
K	denotes a given element,
∂K	denotes the boundary of the element K,
E, F	denotes given edges.

The numerical scheme is based on the mixed–hybrid finite element method. We assume that the vector terms belong to the functional space $H(\mathrm{div}, \Omega)$ and on each element $K \in \mathcal{K}_h$ they can be approximated in the lowest order Raviart-Thomas-Nédélec space $RTN_0(K)$ (Brezzi & Fortin, 1991). On simplices, the basis functions of $RTN_0(K)$ are given by:

$$\boldsymbol{\omega}_{K,E}(\mathbf{x}) = \frac{1}{d|K|_d}(\mathbf{x} - \mathbf{x}_E), \tag{3}$$

where the subscript K denotes the element and E corresponds to the parts of ∂K: faces in 3D, line segments in 2D, and points in 1D. The position vector $\mathbf{x}_E$ is the vertex facing the object E and $|\cdot|_d$ denotes the d-dimensional Lebesgue's measure.

Scalar terms are approximated using the lowest order discontinuous Galerkin approximation, i.e., using piece wise constant functions per elements.

Each inner edge is shared by exactly two neighboring elements. Further in this paper, we assume that the unknown functions are continuous across these inner edges in the sense of traces. This allows us to introduce one common value of the trace at each inner edge F denoted by $Z_{j,F}$ that satisfies:

$$Z_{j,F} = Z_{j,K_1,F} = Z_{j,K_2,F}, \tag{4}$$

where $F = \partial K_1 \cap \partial K_2$.

Using algebraic manipulations of the system (1), the values of $Z_{j,K}^{k+1}$ can be expressed as functions of $Z_{j,F}^{k+1}$ for all elements $K \in \mathcal{K}_h$ and $F \in \partial K \cap \mathcal{E}_h$ in a following way:

$$\mathbf{Z}_K^{k+1} = \sum_{F \in \partial K} \mathbf{Q}_K^{-1}\mathbf{R}_{K,F}\mathbf{Z}_F^{k+1} + \mathbf{Q}_K^{-1}\mathbf{R}_K, \tag{5}$$

where the matrices $\mathbf{Q}$, $\mathbf{R}_{K,F}$ and the vector $\mathbf{R}_K$ are functions of the local coefficients and also depend on mesh geometry (for details, refer to (Fučík et al., 2017)). Vector $\mathbf{Z}_K^{k+1}$ consists of the unknowns $Z_{j,K}^{k+1}$ at time level t_{k+1}. Equation (5) bounds the average values on elements together with the traces on element boundaries.

Expression (5) for the mean values $Z_{j,K}^{k+1}$ is substituted into the mass balance equations on edges and into the definition of the Neumann boundary conditions. Furthermore, the Dirichlet boundary conditions are also included. As a result, we obtain a linear system of algebraic equations for the unknown traces $Z_{j,F}^{k+1}$ that can be written as

$$\mathbf{M}\mathbf{Z}^{k+1} = \mathbf{b}, \tag{6}$$

where $\mathbf{Z}^{k+1} = \{\{Z_{j,F}^{k+1}\}_{j=1}^{n}\}_{F \in \mathcal{E}_h}$ is the vector of unknown traces $Z_{j,F}^{k+1}$.

The traces obtained by solving equation (6), i.e., $\mathbf{Z}^{k+1} = \mathbf{M}^{-1}\mathbf{b}$, allow to compute the mean values $Z_{j,K}^{k+1}$ from equation (5) which completes the time step.

3. COMPUTATIONAL ALGORITHM

In this section, a summary of the computational algorithm is given. While describing the implementation, the steps of the algorithm will be described in details and issues connected with performing these steps will be discussed.

1. Initialization:

First, the algorithm initialization is done using the initial conditions.

Set $t_0 = 0$, $k = 0$, and set Δt to a given value (constant during the computation).

Set $Z_{j,K}^0$ using the initial conditions.

2. Main loop:

The following steps are repeated until the final computational time is reached

2a. Compute the local coefficients (see (Fučík et al., 2017) for details) for all $K \in \mathcal{K}_h$, $E, F \in \partial K \cap \mathcal{E}_h$.

2b. Compute the matrices $\mathbf{Q}_K$, $\mathbf{R}_{K,F}$, invert $\mathbf{Q}_K$ to obtain $\mathbf{Q}_K^{-1}$ and compute the vectors $\mathbf{R}_K$ for all $K \in \mathcal{K}_h$, $F \in \partial K \cap \mathcal{E}_h$.

2c. Assemble the matrix $\mathbf{M}$ of the linear system (6) that expresses the mass balance on inner edges and the boundary conditions.

2d. Solve the system (6) to obtain $Z_{j,E}^{k+1}$ for all $j \in \{1, \ldots, n\}$, $E \in \mathcal{E}_h$.

2e. Using (5), compute $Z_{j,K}^{k+1}$ for all $j \in \{1, \ldots, n\}$, $K \in \mathcal{K}_h$.

2f. Set $t_{k+1} = t_k + \Delta t$.

2g. Set $k = k + 1$.

In this paper, the time step Δt is considered constant, small enough to ensure stability of the numerical scheme. Since the problems discussed in this paper are strongly non-linear, the numerical stability is verified heuristically only: we check for spurious oscillations of the solution and for possible failures of the linear system solver caused by the ill-posedness of matrix $\mathbf{M}$.

4. IMPLEMENTATION IN 2D

A 2D variant of the numerical method was implemented in C++ and parallelized using MPI[1] (The Open MPI Project, 2016). Triangular meshes were generated by Gmsh (Geuzaine & Remacle, 2009). All computations were carried out at the computational cluster Hyperion of the Department of Mathematics at the Faculty of Nuclear Sciences and Physical Engineering of the Czech Technical University in Prague. Each node of the cluster is equipped with two CPUs AMD Opteron 6272 ($16 \times 2{,}1$GHz) and 64 GB (4×16GB) DDR3 RAM.

4.1. Serial implementation of the numerical scheme

In this section, we focus on a serial implementation of the scheme which is less difficult and will be used as a reference for the comparison with the parallel one to demonstrate advantages of the parallelism. By the serial implementation, we mean the implementation running in a single thread on a single CPU core.

Two different approaches, direct and iterative, can be used to solve the resulting linear system with sparse matrix $\mathbf{M}$ in (6). First, a direct solver from numerical library UMFPACK (Davis, 2004) is used. For finer meshes and therefore larger matrices of the linear system (6), the solution using UMFPACK becomes too time consuming. To speed-up the computation, iterative solvers are used instead.

For the class of problems considered in this paper, the matrix of the linear system (6) is non-symmetric. This restricts the possible choices of iterative methods. Two methods based on the Krylov's subspaces methods are used: the restarted generalized minimal residual method (GMRES) and the stabilized bi-conjugated gradient method (BICGSTAB). A detailed description of these methods can be found for example in (Saad, 2003). We consider both methods to be pre–conditioned by the incomplete LU factorization (ILU). The simplest version of the ILU preconditioner denoted by ILU(0) is used that neglects matrix fill-ups and the matrix of the ILU factorization has the same sparse structure as the original matrix of the system (Saad, 2003).

In general, choosing a stopping criterion is a very important strategy while using iterative solvers. Here, iterations are terminated when the norm of the residual divided by the norm of the right–hand–side $\mathbf{b}$ of the linear system (6) is lower than a given threshold $\varepsilon > 0$, i.e.,

$$\frac{\|\mathbf{M}\mathbf{Z}_q^{k+1} - \mathbf{b}\|}{\|\mathbf{b}\|} \leq \varepsilon, \tag{7}$$

where $\mathbf{Z}_q^{k+1}$ is the solution obtained in the q-th iteration.

4.2. Parallel implementation of the numerical scheme

For the parallel implementation, we consider the BICGSTAB method only due to its better performance and easier implementation with respect to GMRES. BICGSTAB is used together with ILU(0) pre-conditioner which is difficult to parallelize if applied to the whole matrix. To overcome the difficulties, we divide the matrix into blocks and perform ILU(0) only on the diagonal blocks. These blocks are independent and can be processed in parallel at the same time.

The approach based on the domain decomposition method (Toselli & Windlund, 2005) is used for the division of the matrix into blocks and for the distribution among the CPU cores. A straightforward approach would be to divide the mesh according to the elements but since the primary unknowns in the linear system (6) correspond to the traces that reside

[1] Message Passing Interface

on element boundaries, division based on edges is better.

All edges are divided into a given number of groups (equal to the number of available CPU cores) and these groups are then mapped on processors. In order to uniformly distribute the workload among processors, the groups should contain approximately the same number of edges. We also want to minimize the number of edges on group boundaries (i.e. length of the group boundary). Values on these edges are used by multiple processors during the computation and we want to minimize the data transfer.

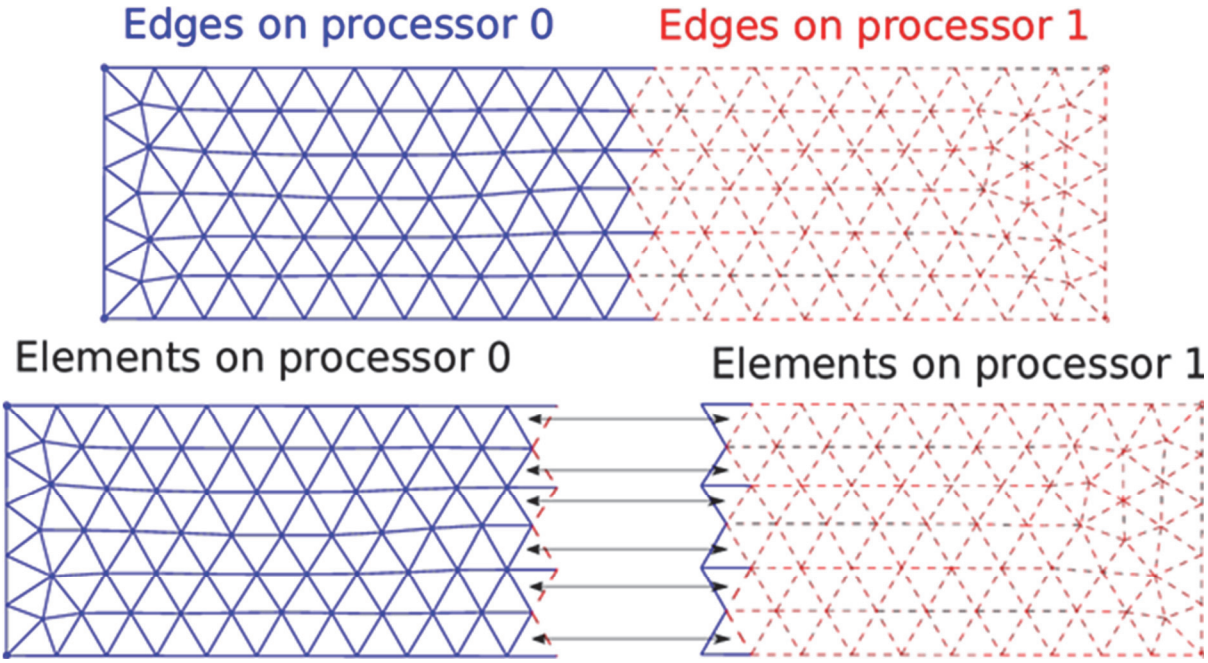


Fig. 1. *Example of the mesh division between two processors with duplicate elements marked by arrows.*

In this paper, a simple algorithm is used to divide the domain into groups: the whole domain is divided into a given number of squares (or rectangles) and the edges are assigned to the groups based on the position of their center. Use of a different algorithm is possible as there are no requirements on the shape of the subdomains.

Numerical scheme does not work with traces on edges only but also with average values over elements. To minimize the computational costs, all elements with at least one edge mapped on a given processor are mapped on that processor. Therefore, the elements with edges in different groups are stored redundantly (at most on three different processors in the case of triangles).

The division of the mesh between two processors with mapped edges and elements is shown in figure 1.

4.2.1. Communication

In this section, we go through the steps of the computational algorithm described in section 3 and decide in which steps the communication is necessary and which values must be transferred.

During the initialization from the initial conditions (steps **1a - 1b**), no data need to be transferred. All values needed to perform these steps are already stored on given processor.

During step **2a** – the computation of the local coefficients – no data need to be transferred.

The data transfer is first required in step **2b** – the computation of the local matrices $\mathbf{Q}_K$, $\mathbf{R}_{K,F}$ and vectors $\mathbf{R}_K$. To perform this step, mean values on elements and traces on element boundaries are needed. In the case of an element with edges in multiple groups, these traces must be transferred from the corresponding group. Otherwise, no data transfers are needed.

To assembly the matrix **M** of the system, there is no need of data transfer – the local matrices were already computed in the previous step **2b**. In order to efficiently solve the linear system, it is important to properly sort equations and unknowns during this step. Groups of edges are numbered as 0,1,2... and the ordering of rows is done as follows: first the mass balance equations on group 0 edges, then group 1 edges and so on. Equations in the groups are ordered as follows: first is the mass balance equations for Z_0, then follows the mass balance equation

for Z_1, etc. The ordering of the columns is chosen such that the structure of the matrix **M** is similar to the structure of a block-diagonal matrix (minimize the number of out-diagonal elements). It corresponds to the following ordering: first are the unknowns corresponding to group 0 edges, then group 1 edges and so on. The ordering in the groups is the same as the ordering of equations in the row ordering.

Each processor assembles a block of rows of the matrix and the right–hand–side vector that correspond to the mass balance equations on edges mapped on the processor.

The most complicated step in terms of data transfer is **2d** – solution of the linear system. Use of BICGSTAB requires to perform the following operations: the inner (scalar) vector product, the matrix–vector multiplication and the preconditioning (Saad, 2003). The easiest operation (that actually does not require data transfer) is the preconditioning. It requires a construction of the decomposition of the diagonal block and two backward steps. Diagonal blocks are stored as a whole on a single processor and, therefore, no communication is required.

To perform the inner product, each processor performs the inner product on the block that belongs to his group and these partial results are summed on one selected processor (for example the processor 0) via the MPI operation of *parallel reduction* and the result is then distributed back to all processors via the MPI operation called *scatter,* (The Open MPI Project, 2016).

At the beginning of the matrix vector product, each processor has a stored diagonal block of the matrix an the corresponding part of the vector due to how the vectors and matrices are assembled. To obtain the complete result, the MPI operation called *circular shift* (The Open MPI Project, 2016) is used. After the initial multiplication, each processor sends its part of the vector to the previous one (the first one sends data to the last one). Then, another block of the matrix with the corresponding part of the vector can be multiplied and the result is added to the result from previous step. These steps are repeated until the final result is obtained. For a division into p processors, $p-1$ steps are required. Only parts of the vector have to be transferred to perform the matrix–vector multiplication, all blocks of the matrix required on each processors are already stored there.

This process can be improved by taking advantage of the matrix structure. Matrix **M** contains zero blocks and therefore some of the transfers of the vectors to the processors for the matrix–vector multiplication can be omitted because the blocks of zeros in the matrix can be identified prior the multiplication. During the circular shift, data are transferred only to those processors where they are multiplied with the non-zero blocks of the matrix. This strategy can significantly reduce the amount of communication required especially when using a larger number of processors. Example of the mesh division among 12 processors is shown in figure 2 and the structure of corresponding matrix in figure 3. Computational times for both approaches using the benchmark problem described in next section are shown in table 1. Without this improvement, the computational time for 32 CPU cores is even longer than for 16 CPU cores and in this case the computation using 32 CPU cores is actually the slowest one.

Table 1. *Computational times in seconds for the comparison of two ways of communication using a mesh with 62 976 elements.*

Number of CPU cores	Circular shift	Circular shift to non-zero blocks
4	1 430	1 393
8	1 040	820
16	960	387
32	1 583	305

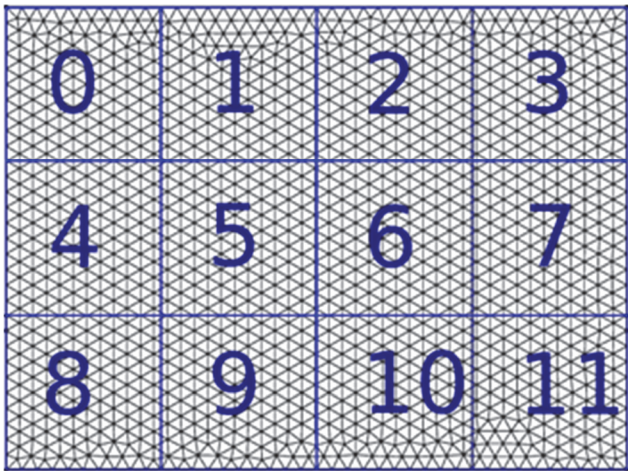


Fig. 2. *Example of the mesh distribution among 12 processors.*

Situation in step **2e** – the computation of the average values on elements $Z_{j,K}^{k+1}$ from traces on element boundaries $Z_{j,E}^{k+1}$ – is similar to the one in step **2b**. In each element, values on all its edges are needed and in the case of the element with edges in different groups, these values have to be transferred. Communication is not necessary in steps **2f** and **2g**.

The communication types in steps **2b** and **2e** are the same. The only difference is in the data trans-

ferred. It would be extremely inefficient to transfer the values from each edge as a standalone messages. At the beginning of steps **2b** or **2e**, a long message can be assembled that consists of all the values that will be transferred from one group to another and independently on the number of the edges only one message is transferred between these groups.

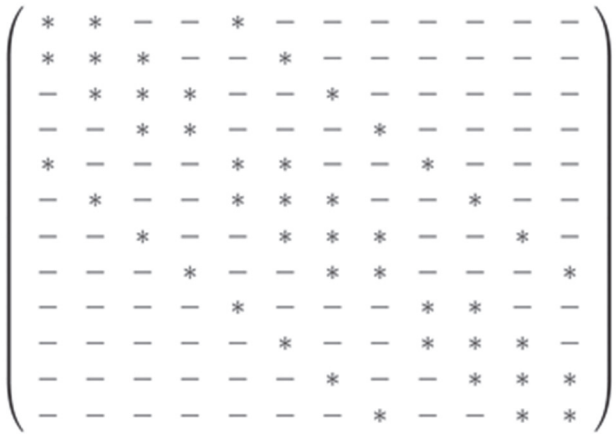


Fig. 3. *Structure of the matrix corresponding to the division in figure 2: "∗" non-zero blocks, "−" zero blocks.*

5. COMPUTATIONAL RESULTS

In this section we show computational results obtained using the described numerical method and compare the results for different implementations. The benchmark problem is the extension of a pure diffusion one-dimensional (1D) McWhorter–Sunada problem in homogeneous porous media (McWhorter & Sunada, 1990). This problem is defined on the interval $[0; +\infty)$ where the initial condition for the wetting phase saturation S_i is given and at the origin the boundary condition for the wetting phase saturation S_0 is prescribed. The exact solution can be found under the assumption of incompressible phases and neglected gravity, c.f. (McWhorter & Sunada, 1990; Fučík et al., 2007; Fučík et al., 2016). For the two dimensional (2D) variant of the numerical method, instead of a line, a rectangular domain is considered, where the exact solution is considered constant in the direction (x-axis) perpendicular to the line on which the original 1D problem is solved (y-axis).

5.1. Coefficients in the general form

As the primary unknowns Z_0 and Z_1, we use the capillary pressure p_c and the nonwetting phase pressure p_n, respectively. The capillary pressure is defined as the difference between the non-wetting phase and the wetting phase pressures $p_c = p_n - p_w$. The wetting phase saturation can be expressed as a function of capillary pressure p_c based on the Brooks–Corey (Brooks & Corey, 1964) or van Genuchten (van Genuchten, 1980) empirical relationships $S_w = S_w(p_c)$.

Then, the benchmark problem formulation can be described by the following choice of the coefficients in (1):

$$\mathbf{N} = \begin{pmatrix} -\Phi\rho_w \frac{dS_w}{dp_c} & 0 \\ -\Phi\rho_n \frac{dS_w}{dp_c} & \Phi S_n \frac{d\rho_n}{dp_n} \end{pmatrix}, \qquad \mathbf{m} = \begin{pmatrix} \rho_w \frac{\lambda_w}{\lambda_t} \\ \rho_n \frac{\lambda_n}{\lambda_t} \end{pmatrix},$$

$$\mathbf{D} = \begin{pmatrix} \lambda_t \mathbf{K} & -\lambda_t \mathbf{K} \\ 0 & \lambda_t \mathbf{K} \end{pmatrix}, \mathbf{w} = \begin{pmatrix} -\lambda_t \rho_w \mathbf{Kg} \\ \lambda_t \rho_n \mathbf{Kg} \end{pmatrix}, \mathbf{f} = \begin{pmatrix} -f_w \\ f_n \end{pmatrix},$$

where:
Φ $[-]$ is the porosity,
S_α $[-]$ is the α-phase saturation,
ρ_α $[\mathrm{kg \cdot m^{-3}}]$ is the α-phase density,
f_α $[\mathrm{kg \cdot m^{-3} \cdot s^{-1}}]$ are the sinks/sources,
$\mathbf{g}$ $[\mathrm{m \cdot s^{-2}}]$ is the gravity vector,
$\mathbf{K}$ $[\mathrm{m^2}]$ is the permeability tensor,
$k_{r\alpha}$ $[-]$ is relative permeability (Burdine (Burdine, 1953) or Mualem (Mualem, 1976) model),
μ_α $[\mathrm{kg \cdot m^{-1} \cdot s^{-1}}]$ is dynamic viscosity of the phase α,
$\lambda_\alpha = \frac{k_{r\alpha}}{\mu_\alpha}$ $[\mathrm{kg^{-1} \cdot m \cdot s}]$ is the α-phase mobility ($\lambda_t = \lambda_w + \lambda_n$),
p_α [Pa] is the α-phase pressure,
$\alpha \in \{w, n\}$ denotes the wetting or non–wetting phase.

These coefficients represents mass conservation law and Darcy law for both phases, for details refer to (Fučík & Mikyška, 2011).

The numerical solution is restricted to a domain of a finite length. Configuration of the computational domain, boundary denotation and the geometry of the coarse unstructured mesh are shown in figure 4. Exact and numerical solutions are compared at time $t = 60\ 000$ s when the head of the solution does not reach the neighborhood of the boundary representing infinity (Γ_1).

Water and air are considered as the wetting and the non-wetting phases, respectively. Properties of the fluids are given in table 2, properties of the porous media are given in table 3. In the following computations Brooks–Corey model for capillary pressure and Burdine model for relative permeability is used.

Table 2. *Fluid properties.*

	Density ρ [kg · m^3]	Dynamic viscosity μ [kg · m^{-1} · s^{-1}]
Water	997.78	$9.770 \cdot 10^{-4}$
Air	1.2047	$1.8205 \cdot 10^{-5}$

Table 3. *Porous media properties.*

Porosity	Φ	[−]	0.343
Intrinsic permeability	K	[m^2]	$5.168 \cdot 10^{-12}$
Residual water saturation	S_{wr}	[−]	0.04
Residual air saturation	S_{nr}	[−]	0
Entry pressure	p_d	[Pa]	4605.8
Brooks–Corey parameter	λ	[−]	2.857

The initial saturation is set to $S_i = 0.1$ and the following boundary conditions are prescribed:

$$\mathbf{u}_n \cdot \mathbf{n} = 0, \mathbf{u}_w \cdot \mathbf{n} = 0, \text{ on } \Gamma_1 \cup \Gamma_2 \cup \Gamma_4, \quad (8)$$

$$p_n = 10^5 \text{ Pa}, S_w = 0.6, \text{ on } \Gamma_3. \quad (9)$$

5.2. Numerical analysis

First, we compare the accuracy of different implementations of the numerical scheme. We introduce the following notation. The L_p norm in $\Omega \subset \mathbb{R}^d$ of some integrable function ψ is defined by:

$$\|\psi\|_p = \left(\int_\Omega |\psi(\mathbf{x})|^p \mathrm{d}\mathbf{x}\right)^{\frac{1}{p}}, \quad (10)$$

where we consider $p = 1{,}2$ in this paper. For a given mesh $\mathcal{K}_h$, the error E_h of the numerical solution is defined by:

$$\|E_h\|_p = \left\|Z^{ex} - Z^h\right\|_p, \quad (11)$$

where Z^{ex} and Z^h are the exact and the numerically approximated solution of the benchmark problem, respectively. The spatial step h is the largest circle diameter circumscribed around the elements in $\mathcal{K}_h$. Using this notation, we investigate the convergence and the accuracy of the numerical scheme using the experimental order of convergence eoc_p that approximates the order of convergence as:

$$eoc_p = \frac{\ln\|E_{h_1}\|_p - \ln\|E_{h_2}\|_p}{\ln h_1 - \ln h_2}, \quad (12)$$

where h_1 and h_2 denote the spatial steps of two different meshes $\mathcal{K}_{h_1}$ and $\mathcal{K}_{h_2}$, respectively.

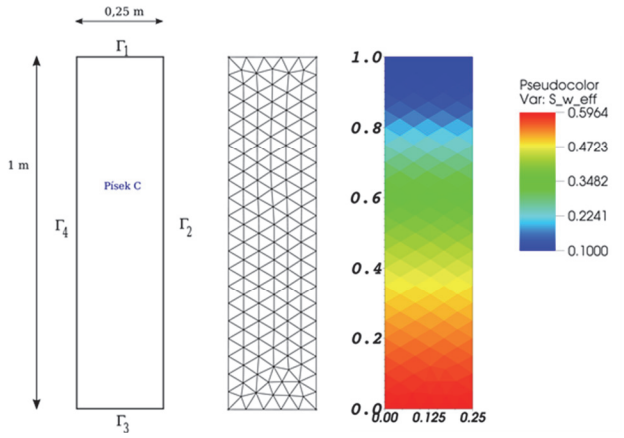


Fig. 4. *Computational domain, unstructured mesh generated by Gmsh (246 elements, 394 edges) and numerical solution of wetting saturation in time $t = 60\ 000\ s$.*

The benchmark problem is first solved using the direct solver UMFPACK, numerical errors are shown in table 4. Then, the knowledge of the exact solution and the numerical errors obtained using UMFPACK are used to find the stopping criterion ε for iterative solvers. We test several values of ε and try to obtain the same results as for the direct solver.

For the BICSTAB method, values of $\varepsilon = 10^{-10}$, $\varepsilon = 10^{-11}$, $\varepsilon = 10^{-12}$, $\varepsilon = 10^{-13}$, $\varepsilon = 10^{-14}$, and $\varepsilon = 10^{-15}$ are tested. Comparison of the errors in L_1 norm is shown in figure 5. For larger ε, the errors for fine meshes increase and the method is not convergent. A stopping criterion of $\varepsilon = 10^{-15}$ must be used to obtain similar results as for UMFPACK, numerical errors for this ε are given in table 4. The same value is used also for the parallel implementation of the BICGSTAB method, numerical errors are given in table 4 and it can be seen that the errors are almost the same as for the serial implementation.

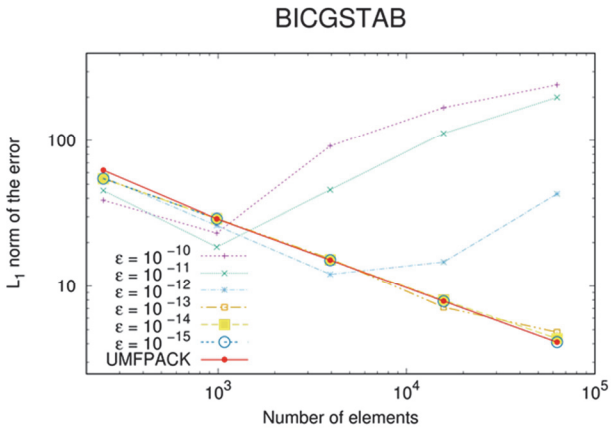


Fig. 5. *Comparison of the numerical errors for the BICGSTAB method using various stopping criteria ε on different meshes.*

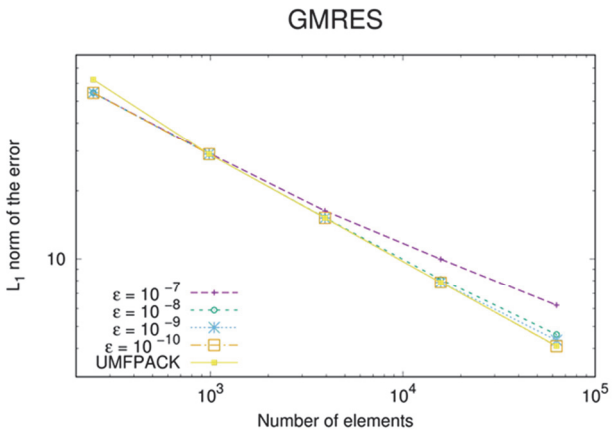


Fig. 6. *Comparison of the numerical errors for the GMRES method using various stopping criteria ε on different meshes.*

Table 4. *Numerical errors for both serial and parallel implementations.*

Number of elements	UMFPACK		BICGSTAB		GMRES		Parallel – 6 cores	
	L_1	L_2	L_1	L_2	L_1	L_2	L_1	L_2
246	62.33	234.93	54.66	208.04	54.55	208.23	54.54	208.22
984	29.06	116.93	29.05	116.92	29.06	116.93	29.04	116.88
3 396	15.17	62.20	15.14	62.12	15.17	62.19	15.17	62.23
15 744	7.86	62.36	7.84	32.31	7.86	32.34	7.85	32.33
62 976	4.10	16.96	4.11	17.04	4.09	16.96	4.08	17.07
251 904	2.20	9.51	2.21	9.52	2.20	9.53	2.21	9.51

For the GMRES method, values of $\varepsilon = 10^{-7}$, $\varepsilon = 10^{-8}$, $\varepsilon = 10^{-9}$, and $\varepsilon = 10^{-10}$ are tested. Comparison of the errors in L_1 norm is shown in figure 6. For larger ε, the errors for fine meshes increase. In this case the errors are not as large as for the BICGSTAB method, but the stopping criterion $\varepsilon = 10^{-10}$ must be used to obtain similar results as for UMFPACK, numerical errors for this ε are given in table 4.

Results show that the numerical solutions converge and the experimental order of convergence is slightly less than one which is typical for methods that use the first order upwind stabilization approach (LeVeque, 2002). For a given stopping criterion, errors for the iterative solvers are almost the same as for the direct solver UMFPACK.

Furthermore, the results for both parallel and serial implementations differs only very slightly. Difference in the solutions obtained using serial and parallel implementation is caused by the usage of different preconditioner in the parallel implementation than in the serial one. The parallel implementation uses ILU(0) decompositions of diagonal blocks only (this approach allows us to efficiently apply preconditioner in parallel), and therefore the resulting preconditioner is slightly different than the ILU(0) decomposition of the whole matrix used in the serial implementation.

Computational times for the serial and parallel implementation using various number of CPU cores are shown in table 6 and table 7, respectively. The speed–up of the computation for p used CPU cores U_p is defined by:

$$U_p = \frac{T_1}{T_p}, \tag{13}$$

where T_1 is the computational time using one CPU core and T_p is the computational time using p CPU cores, is shown in table 8. Efficiency E_p for computations using p CPU cores, defined by:

$$E_p = \frac{U_p}{p} \cdot 100\%, \tag{14}$$

is shown in table 8.

Table 5. *Experimental order of convergence for both serial and parallel implementations.*

Mesh refinement	UMFPACK		BICGSTAB		GMRES		Parallel	
	eoc_1	eoc_2	eoc_1	eoc_2	eoc_1	eoc_2	eoc_1	eoc_2
246 → 984	1.10	1.01	0.91	0.83	0.91	0.84	0.91	0.83
984 → 3 936	0.94	0.91	0.94	0.91	0.94	0.91	0.94	0.91
3 936 → 15 744	0.95	0.94	0.95	0.94	0.95	0.94	0.95	0.94
15 744 → 62 976	0.94	0.93	0.93	0.92	0.94	0.93	0.93	0.92
62 976 → 251 904	0.90	0.83	0.89	0.83	0.89	0.83	0.88	0.83

Table 6. *Computational times for the serial implementation in seconds.*

Number of elements	Size of matrix **M**	UMFPACK	BICGSTAB	GMRES
246	788	1.3	1.4	1.7
984	3 052	7.7	5.2	36.4
3 936	12 008	67	45	800
15 744	47 632	756	373	999
62 976	189 728	9 160	3 968	7 272
251 094	757 312	186 435	34 401	84 900

Based on table 6, BICGSTAB is the fastest linear solver for the serial implementation. On finer meshes, GMRES becomes faster than UMFPACK. In general, tables 7 and 8 demonstrate the advantages of parallelism as expected, the effect of the parallelism is more significant on finer meshes.

Table 7. *Computational times for the parallel implementation of the method in seconds.*

Number of elements	Matrix **M** size	Number of cores						
		1	2	4	6	8	16	32
246	788	1.4	0.5	0.4	0.4	0.4	-	-
948	3 052	5.2	2.9	1.8	1.3	1.2	-	-
3 936	12 008	45	28	11	7	7	6	-
15 744	47 632	373	294	107	79	63	37	30
62 976	189 728	3 968	2 947	1 393	921	820	387	305
251 094	757 312	34 401	21 147	12 718	8 093	7 578	6 222	3 203

Table 8. *Speed up* (U_p) *and efficiency* (E_p) *for various number of cores.*

Number of elements	Number of cores											
	2		4		6		8		16		32	
	U_2	E_2	U_4	E_4	U_6	E_6	U_8	E_8	U_{16}	E_{16}	U_{32}	E_{32}
246	2.8	140 %	3.5	88 %	3.5	58 %	3.5	44 %	-	-	-	-
984	1.8	90%	2.9	72 %	4.0	67 %	4.3	54 %	-	-	-	-
3 936	1.6	80 %	4.1	102 %	6.4	107 %	6.4	80 %	7.5	47 %	-	-
15 744	1.3	63 %	3.5	87 %	4.7	78 %	6.0	74 %	10.1	63 %	12.4	39 %
62 976	1.4	67 %	2.9	71 %	4.3	71 %	4.8	60 %	10.3	64 %	13.0	41 %
251 904	1.6	81 %	2.7	67 %	4.3	70 %	4.6	57 %	5.5	35 %	10.7	34 %

6. CONCLUSION

In this work we dealt with a numerical solution of two–phase flow problems in porous media. We proposed a numerical method for solving this type of problems based on the mixed–hybrid finite element method. The method is implemented for two dimensional problems. Serial implementation using a direct solver from UMFPACK package and iterative solvers BICGSTAB and GMRES together with ILU(0) preconditioning and parallel implementation based on the domain decomposition method using MPI are described.

The method is verified using a problem with known exact solution. We showed that for given stopping criteria for iterative solvers the numerical scheme is convergent and the values of experimental order of convergence are slightly less than one. Errors of the numerical solution are almost the same for all cases considered but there is a significant difference in computational times. BICGSTAB is the fastest of the serial implementations and on finer meshes, the parallelism is advantageous.

ACKNOWLEDGMENT

This work was supported by Grants No. SGS17/194/OHK4/3T/14 and SGS14/206/OHK4/3T/14 of the Grant Agency of the Czech Technical University in Prague.

REFERENCES

Brezzi, F. Fortin, M., 1991, *Mixed and Hybrid Finite Element Methods*, Springer-Verlag.

Brooks, R., Corey, A., 1964, Hydraulic Properties of Porous Media, Colorado State University, *Hydrology Paper*, 3, 27.

Burdine, N., 1953, Relative Permeability Calculations From Pore Size Distribution Data, *Journal of Petroleum Technology*, 5, 71-78.

Davis, T. A., 2004, Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method, ACM Transactions on Mathematical Software, 30, 196–199.

Fučík, R., Illangasekare, T. H., Beneš, M., 2016, Multidimensional self-similar analytical solutions of two-phase flow in porous media, *Advances in Water Resources*, 90, 51-56.

Fučík, R., Mikyška, J., Solovský, J., Klinkovský, J., Oberhuber, T., 2017, Multidimensional Mixed-Hybrid Finite Element Method for Compositional Two-Phase Flow in Heterogeneous Porous Media and its Massively Parallel Implementation on GPU, in review in *Computer Physics Communications*.

Fučík, R., Mikyška, J., 2011, Mixed-hybrid finite element method for modelling two-phase flow in porous media, *Journal of Math for Industry*, 3, 9-19.

Fučík, R., Mikyška, J., Beneš, M., Illangasekare, T. H., 2007, An Improved Semi-Analytical Solution for Verification of Numerical Models of Two Phase Flow in Porous Media, *Vadose Zone Journal*, 6(1), 93-104.

Geuzaine, C., Remacle, J. F., 2009, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, *International Journal for Numerical Methods in Engineering*, 79, 1309-1331.

LeVeque, R. J., 2002, *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press.

McWhorter, D. B., Sunada, D. K., 1990, Exact integral solutions for two-phase flow, *Watter Resources Research*, 26, 399-413.

Mualem, Y., 1976, A new model for predicting the hydraulic conductivity of unsaturated porous media, *Water Resources Research*, 12, 513-522.

Saad, Y., 2003, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics.

The Open MPI Project, 2016, Open MPI: Open Source High Performance Computing, available online at https://www.open-mpi.org/, Accessed: 10. 3. 2016.

Toselli, A., Windlund, O., 2005, Domain Decomposition Methods – Algorithms and Theory, Springer-Verlag.

van Genuchten, M., 1980, A Closed-form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils, *Soil Science Society of America Journal*, 40, 892-898.

RÓWNOLEGŁY MIESZANY HYBRYDOWY MODEL ELEMENTÓW SKOŃCZONYCH DLA PROBLEMÓW DWUFAZOWEGO PRZEPŁYWU W POROWATYCH OŚRODKACH

Streszczenie

W artykule opisano numeryczne rozwiązanie dla dwufazowego przepływu w porowatych ośrodkach. Dla rozwiązania tego typu problemu zaproponowano numeryczną metodę wykorzystującą mieszany hybrydowy model elementów skończonych. Model opisano w skrócie dla problemu 2D. W pracy zaimplementowano kilka wariantów metody stosując różne podejścia do rozwiązania wynikowego układu równań algebraicznych. Zastosowano bezpośredni i iteracyjny solwer oraz równoległą implementację metodą wykorzystującą dekompozycję domeny z zastosowaniem interfejsu transmisji wiadomości (ang. Message Passing Interface MPI).

Dokładność i wydajność obliczeniowa metody zostały zweryfikowane dla problemu o znanym rozwiązaniu dokładnym. Przeprowadzone eksperymenty numeryczne wykazały, że błędy dla różnych wariantów metody są zbliżone. Metoda jest zbieżna ze stopniem zbieżności nieco mniejszym od jeden. Różne były natomiast czasy obliczeń na korzyść solwera iteracyjnego, szczególnie dla gęstych siatek. Przy zastosowaniu takich siatek korzystnie jest zrównoleglić obliczenia, co znacznie skraca czas realizacji programu.

Received: December 12, 2016
Received in a revised form: March 26, 2017
Accepted: April 23, 2017