 CMES, vol.49, no.3, pp.191-214, 2009

An Efficient Parallel MLPG Method for Poroelastic Models

Luca Bergamaschi[1,2], **Ángeles Martínez**[2] **and Giorgio Pini**[2]

Abstract: A meshless model, based on the Meshless Local Petrov-Galerkin (MLPG) approach, is developed and implemented in parallel for the solution of axi-symmetric poroelastic problems. The parallel code is based on a concurrent construction of the stiffness matrix by the processors and on a parallel preconditioned iterative method of Krylov type for the solution of the resulting linear system. The performance of the code is investigated on a realistic application concerning the prediction of land subsidence above a deep compacting reservoir. The overall code is shown to obtain a very high parallel efficiency (larger than 78% for the solution phase) and it is successfully applied to the solution of a poroelastic problem with a fine discretization which produces a linear system with more than 6 million equations using up to 512 processors on the HPCx supercomputer.

Keywords: meshless method, poroelasticity, preconditioners, parallel computations, scalability

1 Introduction

Since the late '70s meshless methods have attracted an increasing attention due to their flexibility in solving several engineering problems. Among these, a special attention has been devoted for example to the Element-Free Galerkin Belytschko et al (1994), the Meshless Galerkin with Radial Basis Functions Wendland (1999), and the Meshless Local Petrov-Galerkin (MLPG) Atluri Shen (2002a,b); Atluri Zhu (1998) method. In particular, the last one has the attractive feature of begin "truly" meshless as it does not need any kind of connection among the nodes selected within the computational domain. For this reason MLPG method turns out to be more flexible and easier to use than other meshless methods or the conventional Finite Element (FE) Methods Pini et al (2008). The MLPG method has been

[1] Corresponding author. Tel.: 0039 049 8271307. E-mail: berga@dmsa.unipd.it
[2] Dipartimento di Metodi e Modelli Matematici per le Scienze Applicate, Università di Padova. Via Trieste 63, 35127 Padova, Italy

successfully employed in several applications Sladek et al (2008b,a); Ma (2008); Jarak Soric (2008); Zheng et al (2009).

However, the main drawback of a meshless method, which currently precludes its use especially in fully 3-D problems, is the large computational cost with respect to low-order (especially linear) FE (see Yuan et al (2007) for an attempt to solve this problem). In particular, the numerical integration of the meshless shape functions, that may be quite complex in nature, can be very expensive and sometimes inaccurate, so that one of the current main challenges for the improvement of meshless techniques relies on the robust and efficient implementation of effective integration rules.

The present paper is concerned with the development of a parallel poroelastic MLPG model for large scale parallel solution of an axi-symmetric poroelastic problem. In several real problems related to the mechanics of pore fluid withdrawal or injection, the assumption of an axi-symmetric geometry with the symmetry axis coinciding with a vertical well is often used, e.g. Gambolati et al (1999, 2000); Ferronato et al (2004). This allows for avoiding the computational burden involved by a fully 3-D meshless simulation and nonetheless obtaining relevant results for the engineering practice. The MLPG accuracy in an axi-symmetric problem has been investigated in a recent work Ferronato et al (2007b) with the aid of available analytical solutions, such as for example the land subsidence occurring over a compacting cylindrical gas/oil reservoir Geertsma (1973), and compared to the outcome of a standard FE poroelastic model. In particular, MLPG is shown to take advantage of a great flexibility in the definition of the local sub-domains and the number of nodal contacts, allowing for a potentially better accuracy with a relatively small number of nodes over the integration domain.

The present paper is concerned with the development of a parallel poroelastic MLPG model for large scale parallel solution of an axi-symmetric poroelastic problem. The parallel implementation of our MLPG model is based on the concurrent construction of the stiffness matrix by the processors without any intercommunication and on an efficient preconditioned iterative solution of the linear system arising from the meshless discretization. To this aim a parallel FSAI (Factorized Sparse Approximate Inverse) preconditioner Kolotilina Yeremin (1993) with pre- and post- filtration Kolotilina et al (1999); Nikishin Yeremin (2003) has been developed in combination with a parallel BiCGSTAB van der Vorst (1992) iterative solver. The resulting preconditioner proves very effective in the acceleration of the iterative solver, revealing at the same time roughly as sparse as the coefficient matrix.

Numerical results on the HPCx supercomputer available at EPCC (Edinburgh Parallel Computing Center, UK) show that our approach is perfectly scalable even on a

very high number of processor (up to 512) and allows for the solution of very large geomechanical problems.

The paper is organized as follows. The equations of elastic equilibrium of an axi-symmetric porous volume are solved with the aid of MLPG in Section §2. In §3, we recall the basic algorithm for constructing FSAI preconditioners, while our parallelization strategies are described in §4. In §5 the parallel numerical results are presented, together with a discussion regarding the optimal choice of the preconditioner. A number of remarks close the paper in §6.

2 MLPG formulation for axi-symmetric poroelastic problems

The problem used to test our parallel MLPG iterative solver consider a cylindrical reference system r, θ, z, with z the vertical axis taken positive in the upward direction. We choose to solve an axi-symmetric poroelastic problem since it is still widely used in real-life applications (see for example Ferronato et al (2007a)). Moreover, an analytic solution can be explicitly computed to guarantee the accuracy of the MLPG approach (see Ferronato et al (2007b) where MLPG has been compared with Galerkin Finite Elements).

The equations of elastic equilibrium of a cylindrical, isotropic and axi-symmetric porous medium with the axis at $r = 0$ read Verruijt (1969):

$$\overline{D}\sigma + \mathbf{b} = \mathbf{0} \tag{1}$$

where:

$$\overline{D} = \begin{bmatrix} \frac{\partial}{\partial r} + \frac{1}{r} & 0 & -\frac{1}{r} & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial z} & 0 & \frac{1}{r} + \frac{\partial}{\partial r} \end{bmatrix} \qquad \sigma = [\sigma_r,\ \sigma_z,\ \sigma_\theta,\ \sigma_{rz}]^T$$

$$\mathbf{b} = [b_r,\ b_z]^T = \left[-\frac{\partial p}{\partial r},\ -\frac{\partial p}{\partial z} \right]^T$$

where σ_r, σ_z, σ_θ, σ_{rz} are the effective stress components, and p is the pore fluid pressure.

Consider the equilibrium of the porous medium at steady state, so that p is a known variable which can be calculated, for example, from the classical flow equation Verruijt (1969); Gambolati (1974). The solution to eq. (1) requires appropriate Dirichlet boundary conditions over Γ_u and Neumann boundary conditions over Γ_t. It is obtained in a weak form by orthogonalizing the residual to a test function v_i, $i = 1,2$, over a number of local sub-domains $\Omega_s^{(k)}$, $k = 1,\ldots,n$, located within the overall cylindrical porous volume Ω. Setting:

$$V = \begin{bmatrix} v_1 & 0 \\ 0 & v_2 \end{bmatrix}$$

the local weak form of (1) with Dirichlet condition reads Ferronato et al (2007b):

$$\int_{\Omega_s^{(k)}} V(\overline{D}\sigma + \mathbf{b})\, d\Omega \;-\; \alpha \int_{\Gamma_{su}^{(k)}} V(\mathbf{u} - \overline{\mathbf{u}})\, dS = 0 \qquad k = 1, \ldots, n \tag{2}$$

where $\Gamma_{su}^{(k)}$ is the portion of $\Omega_s^{(k)}$ where Dirichlet conditions are prescribed, $\mathbf{u} = [u_r,\, u_z]^T$ is the vector of radial and vertical displacements with $\overline{\mathbf{u}}$ the prescribed ones, and α is a penalty parameter. Because of the axi-symmetry of Ω, the domain of influence $\Omega_s^{(k)}$ is chosen an annular volume with a circular cross section $\Omega_s^{'(k)}$ of radius $r_0^{(k)}$ on the plane $\theta = \overline{\theta}$. Hence eq. (2) yields:

$$\int_{\Omega_s^{'(k)}} V(\overline{D}\sigma + \mathbf{b}) r\, dr\, dz \;-\; \alpha \int_{\Gamma_{su}^{'(k)}} V(\mathbf{u} - \overline{\mathbf{u}}) r\, d\Gamma = 0 \qquad k = 1, \ldots, n \tag{3}$$

with the global integration domain thus restricted to the plane $\theta = \overline{\theta}$.

Different choices for v_i provide different MLPG formulations. The most popular one is the so-called MLPG1Atluri Shen (2002a); Ferronato et al (2008). Let $v_1 = v_2 = w^{(k)}$, where $w^{(k)}$ is the weight function used in the Moving Least Square (MLS) approximation Atluri Shen (2002a) defined over an annular support whose section on the plane $\theta = \overline{\theta}$ is a circle with radius $r^{(k)} = r_0^{(k)}$, i.e. $\Omega_s^{'(k)}$. Common weight functions are Gaussian surfaces Belytschko Gu (1994) or splines Belytschko et al (1996). In the axi-symmetric MLPG1 model investigated in the sequel, $w^{(k)}$ is the following spline function:

$$w^{(k)} = \begin{cases} 1 - 6\left(\dfrac{\delta}{r^{(k)}}\right)^2 + 8\left(\dfrac{\delta}{r^{(k)}}\right)^3 - 3\left(\dfrac{\delta}{r^{(k)}}\right)^4 & 0 \le \delta \le r^{(k)} \\ 0 & \delta > r^{(k)} \end{cases} \tag{4}$$

where δ is the distance between the point (r, z) and the center of $w^{(k)}$ local circular support, i.e. node k. The advantage of MLPG1 implementation is that $w^{(k)}$, and so v_i, vanishes over $\Gamma_{s0}^{'(k)}$. Hence, setting:

$$E_v = \begin{bmatrix} r\frac{\partial v_1}{\partial r} & 0 & v_1 & r\frac{\partial v_1}{\partial z} \\ 0 & r\frac{\partial v_2}{\partial z} & 0 & r\frac{\partial v_2}{\partial r} \end{bmatrix}$$

eq. (3) can be written in the following matrix form:

$$\int_{\Omega_s^{'(k)}} E_v \sigma\, dr\, dz + \alpha \int_{\Gamma_{su}^{'(k)}} rV\mathbf{u}\, d\Gamma - \int_{\Gamma_{su}^{'(k)}} rV\mathbf{t}\, d\Gamma =$$

$$= \int_{\Gamma_{st}^{'(k)}} rV\bar{\mathbf{t}}\, d\Gamma + \alpha \int_{\Gamma_{su}^{'(k)}} rV\overline{\mathbf{u}}\, d\Gamma + \int_{\Omega_s^{'(k)}} rV\mathbf{b}\, dr\, dz \qquad k = 1, \ldots, n \tag{5}$$

where $\mathbf{t} = [t_r, t_z]^T$ is the local vector of linear loads over $\Gamma_s^{(k)}$ with $\bar{\mathbf{t}}$ the prescribed ones.

The MLS approximation for the unknown $\mathbf{u}$ reads:

$$\mathbf{u} \simeq \begin{bmatrix} \sum_{j=1}^{n} \phi_j \hat{u}_r^{(j)} \\ \sum_{j=1}^{n} \phi_j \hat{u}_z^{(j)} \end{bmatrix} \tag{6}$$

with ϕ_j the MLS shape functions with linear basis and $\hat{u}_r^{(j)}$ and $\hat{u}_z^{(j)}$ the fictious nodal displacements Belytschko et al (1996); Ferronato et al (2008). Using (6), the Hooke's law becomes:

$$\sigma = \sum_{j=1}^{n} DB_j \hat{\mathbf{u}}_j \tag{7}$$

with:

$$B_j = \begin{bmatrix} \frac{\partial \phi_j}{\partial r} & 0 \\ 0 & \frac{\partial \phi_j}{\partial z} \\ \frac{\phi_j}{r} & 0 \\ \frac{\partial \phi_j}{\partial z} & \frac{\partial \phi_j}{\partial r} \end{bmatrix} \qquad \hat{\mathbf{u}}_j = \left[\hat{u}_r^{(j)}, \hat{u}_z^{(j)} \right]^T$$

$$D = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{(1-\nu)} & \frac{\nu}{(1-\nu)} & 0 \\ \frac{\nu}{(1-\nu)} & 1 & \frac{\nu}{(1-\nu)} & 0 \\ \frac{\nu}{(1-\nu)} & \frac{\nu}{(1-\nu)} & 1 & 0 \\ 0 & 0 & 0 & \frac{(1-2\nu)}{2(1-\nu)} \end{bmatrix}$$

and E and ν are the Young modulus and the Poisson ratio of the porous medium, respectively.

Finally, substituting eq. (7) into eq. (5) provides the following linear algebraic system with $\hat{\mathbf{u}}$ as the unknown vector:

$$\sum_{j=1}^{n} K_{kj} \hat{\mathbf{u}}_j = \mathbf{f}_k, \qquad k = 1, 2, ..., n \qquad \text{shortly} \quad K\mathbf{u} = \mathbf{f} \tag{8}$$

where:

$$K_{kj} = \int_{\Omega_s'^{(k)}} E_v DB_j \, dr \, dz + \alpha \int_{\Gamma_{su}'^{(k)}} rVS\phi_j \, d\Gamma - \int_{\Gamma_{su}'^{(k)}} rVNDB_j S \, d\Gamma \tag{9}$$

$$\mathbf{f}_k = \int_{\Gamma_{st}^{\prime(k)}} rV\bar{\mathbf{t}}\, d\Gamma + \alpha \int_{\Gamma_{su}^{\prime(k)}} rVS\bar{\mathbf{u}}\, d\Gamma + \int_{\Omega_s^{\prime(k)}} rV\mathbf{b}\, dr\, dz \tag{10}$$

with

$$N = \begin{pmatrix} n_r & 0 & 0 & n_z \\ 0 & n_z & 0 & n_r \end{pmatrix}$$

the matrix of the components of the outer normal to Γ_t and S the auxiliary flag matrix:

$$S = \begin{bmatrix} S_r & 0 \\ 0 & S_z \end{bmatrix}$$

$$S_r = \begin{cases} 1 \text{ if } u_r \text{ is prescribed on } \Gamma_{su}^{\prime(k)} \\ 0 \text{ otherwise} \end{cases} \qquad S_z = \begin{cases} 1 \text{ if } u_z \text{ is prescribed on } \Gamma_{su}^{\prime(k)} \\ 0 \text{ otherwise} \end{cases}$$

The stiffness matrix K of system (8) has size $2n$, is generally unsymmetric, and its sparsity is very much dependent on the size of the domains of influence $\Omega_s^{\prime(k)}$ and the local supports for $w^{(k)}$, i.e. ultimately on $r_0^{(k)}$ and $r^{(k)}$.

2.1 The quadrature formula

Following a suggestion by Peirce Peirce (1957) as implemented in De Bathe (2000), a Gauss-Legendre formula for the ρ integral and a midpoint rule for the θ integral are used:

$$\int_0^{r_0} \int_{\theta_1}^{\theta_2} F(\rho,\theta)\rho \, \mathrm{d}\rho \, \mathrm{d}\theta \approx \sum_{i=1}^{n_\rho} \sum_{j=1}^{n_\theta} a_i b_j F(\rho_i, \theta_j) \tag{11}$$

where ρ_i is the square root of the i-th zero of the n_ρ-degree Legendre polynomial, $a_i = \dfrac{r_0^2 w_i}{4}$ the corresponding weight, $\theta_j = \theta_1 + (j - \dfrac{1}{2})b_j$, and $b_j = \dfrac{\theta_2 - \theta_1}{n_\theta}$. It follows straightforwardly that $a_i b_j = \dfrac{w_i A}{2n_\theta}$, with $A = \dfrac{r_0^2}{2}(\theta_2 - \theta_1)$ the circular sector area.

3 Iterative solution of the linear system

System (8) is sparse and can be very large. Preconditioners for the iterative solution via Krylov subspace methods is a common choice for this kind of problems, provided that an efficient preconditioner is available. We choose to work with the BiCGSTAB method accelerated by a preconditioner belonging to the class of *approximate inverse preconditioners*, which have been extensively studied by many

authors. We quote among the others the SPAI preconditioner Grote and Huckle (1997), the AINV preconditioner described in Benzi Tuma (1998); Benzi et al (2000) and the FSAI (Factorized Sparse Approximate Inverse) preconditioner proposed in Kolotilina Yeremin (1993). These preconditioners explicitly compute an approximation to the inverse of the coefficient matrix and their application needs only matrix vector products, which are more effectively parallelized than solving two triangular systems, as in the ILU preconditioner. Factorized sparse approximate inverses are known to preserve the positive definiteness of the problem and provide better approximations to the inverse of the coefficient matrix for the same amount of storage (than non-factorized ones). Both FSAI and AINV compute the approximation to A^{-1} in factorized form. However, the AINV preconditioner is generally more efficient than FSAI as accelerator of Krylov solvers, due to its flexibility in the generation of the pattern of the approximate inverse factor. On the contrary, the FSAI preconditioner requires the *a priori* specification of the sparsity pattern of the triangular factors. We have chosen to use the FSAI preconditioner because in its current formulation AINV offers limited opportunity for parallelization of the preconditioner construction phase, while the construction of FSAI is inherently parallel. For an exhaustive comparative study of sparse approximate inverse preconditioners the reader is referred to Benzi Tuma (1999).

3.1 The FSAI Preconditioner

Let A be a symmetric positive definite (SPD) matrix and $A = L_A L_A^T$ be its Cholesky factorization. The FSAI method gives an approximate inverse of A in the factorized form

$$M = G_L^T G_L, \tag{12}$$

where G_L is a sparse nonsingular lower triangular matrix approximating L_A^{-1}. To construct G_L one must first prescribe a selected sparsity pattern $S_L \subseteq \{(i,j) : 1 \leq i \neq j \leq n\}$, such that $\{(i,j) : i < j\} \subseteq S_L$, then a lower triangular matrix $\hat{G}_L$ is computed by solving the equations

$$(\hat{G}_L A)_{ij} = \delta_{ij}, \qquad (i,j) \notin S_L. \tag{13}$$

The diagonal entries of $\hat{G}_L$ are all positive. Defining $D = [\mathrm{diag}(\hat{G}_L)]^{-1/2}$ and setting $G_L = D\hat{G}_L$, the preconditioned matrix $G_L A G_L^T$ is SPD and has diagonal entries all equal to 1. As it has been described above the matrix $\hat{G}_L$ is computed by rows: each row requires the solution of a small SPD dense linear system of size equal to the number of nonzeros allowed in that row. Each row of $\hat{G}_L$ can be computed

independently of each other. The extension of FSAI to the nonsymmetric case is straightforward; however the solvability of the local linear systems and the non singularity of the approximate inverse is only guaranteed if all the principal submatrices of A are non singular (which is the case, for instance, if A is positive definite, i.e., $A+A^T$ is SPD).

A common choice for the sparsity pattern is to allow nonzeros in G_L only in positions corresponding to nonzeros in the lower triangular part of A. A slightly more sophisticated and more costly choice is to consider the sparsity pattern of the lower triangle of A^2, see Kaporin (1994). While the approximate inverses corresponding to higher powers of A are often better than the one corresponding to $k=1$, they may be too expensive to compute and apply. It is possible to considerably reduce storage and computational costs by working with sparsified matrices. Dropping entries below a prescribed threshold in A produces a new matrix $\tilde{A} \approx A$; using the structure of the lower triangular part of $\tilde{A}^k$ often results in a good pattern for the preconditioner factor.

3.2 Prefiltration

One feature of the coefficient matrices produced by the MLPG discretization method is that they are denser than the matrices arising from FE discretization of the same problem. On the average the number of nonzero elements per row can be about 70–100 entries. This feature makes almost mandatory the use of a technique called *prefiltration*, to reduce the cost of computing the preconditioner. The basic prefiltration technique (pointwise) consists in dropping entries below a prescribed threshold in the coefficient matrix to produce a sparsified matrix, whose nonzero structure will be used as the symbolic pattern for the preconditioner factors.

In Nikishin Yeremin (2003), the authors introduce a prefiltration strategy based on the so-called vector aggregation technique. The prefiltration is performed on a aggregated matrix naturally induced from the original coefficient matrix by the degrees of freedom per mesh node, and its size is n/s being $s=2,3$ the spatial dimension of the discretization. The small entries of the computed "aggregated matrix" FSAI preconditioning matrix are dropped, and the resulting pointwise sparsity pattern is used to construct the low density block sparsity pattern for the original matrix. This approach usually produces a slight worsening of the preconditioner quality together with a sometimes important saving of the CPU time. Experimental results performed with the matrices arising from the MLPG discretization of the axi-symmetric poroelastic problem considered in this work revealed no advantage of using this prefiltration strategy instead of the trivial one (pointwise). We believe the reason of this behavior is the small dimension (2) of blocks which corresponds to the degrees of freedom per node, which makes the saving of the CPU time too

small to pay for the increased number of iterations. In the section describing the numerical results we limit ourselves to the basic prefiltration strategy.

3.3 Postfiltration of FSAI preconditioners

Following Kolotilina et al (1999) we also employed the technique called *postfiltration*, which consists on a posteriori sparsification of an already constructed FSAI preconditioner by using a small drop–tolerance parameter. The aim is to reduce the number of nonzero elements of the preconditioner factors to decrease the arithmetic complexity of the iteration phase. Also, in a parallel environment, a substantial reduction of the communication complexity in the matrix-vector product can be achieved.

We extend the postfiltration technique described in Kolotilina et al (1999) to nonsymmetric matrices as follows. In the nonsymmetric case both preconditioner factors, G_L and G_U, must be sparsified. We limit ourselves to nonsymmetric matrices with a symmetric nonzero pattern (which is the common situation in matrices arising from i.e. discretization of PDEs) and assume $S_L = S_U^T$. We perform a symmetric filtration of factors G_L and G_U by filtering out the same number of small entries (in absolute value) of row i and column i of both preconditioner factors, respectively. The postfiltrated FSAI preconditioner has been successfully employed in accelerating iterative methods in the solution of e.g. FE problems (see Bergamaschi et al (2005a,b, 2007)).

Banded FSAI preconditioners

In this paper we also propose a dropping strategy based on the distance of an entry of the preconditioner from the diagonal. By this variant we keep all the elements of G_L of indices i, j such that $|i-j| < nd$ where nd is an integer input value.

4 Parallelization of the MLPG code

The parallelization of the MLPG code is subdivided into two phases. The first phase consists in the parallel assembly of the stiffness matrix. We choose to partition the computational domain into np subdomains where np is the number of processors employed. Moreover, every processor can access to information regarding every node belonging to its influence domain, so that no communication is required in this phase.

The second phase is represented by the iterative solution of the linear system $A\mathbf{x} = \mathbf{b}$ – equation (8) with $A, \mathbf{x}, \mathbf{b}$ used instead of $K, \mathbf{u}, \mathbf{f}$ – by the above mentioned BiCGSTAB method, accelerated by the FSAI preconditioner. In the parallel implementation of this method, a number of linear algebra kernels require communi-

cations among processors. Among this kernels the crucial one is represented by the matrix-vector product.

The code is written in FORTRAN 90 and exploits the MPI library for exchanging data among the processors.

4.1 Efficient matrix-vector product

Our implementation of the matrix-vector product is tailored for application to sparse matrices and minimizes data communication between processors. Within the BiCGSTAB algorithm, the vector $H\mathbf{u}$ has to be calculated, where $H \in \{A, G_L, G_U\}$ is an $N \times N$ matrix. Each processor exchanges entries of its local components of vector $\mathbf{u}$ with a very small number of other processors (compared to the total number p).

Let us consider a given processor with processor identifier (pid) say $r, 0 \leq r \leq p-1$. Assume for simplicity that N is exactly np, and denote by S the indices of the nonzero entries of matrix H

$$S = \{(i,j) : \ h_{ij} \neq 0\}.$$

The set S is normally referred as the nonzero pattern of H. After distributing the matrix, the subset P^r containing the indices corresponding to nonzero elements of H belonging to processor r can be defined as

$$P^r = \{(i,j) : \ rn+1 \leq i \leq (r+1)n \ \} \cap S.$$

This set can be partitioned into two subsets

$$P^r_{\text{loc}} = \{(i,j) \in P^r, \ rn+1 \leq j \leq (r+1)n\} \qquad P^r_{\text{nonloc}} = P^r \backslash P^r_{\text{loc}}.$$

For every processor r we also define the subsets C^r_k, R^r_k, $k \neq r$ of indices as:

$$R^r_k = \{i : (i,j) \in P^r_{\text{nonloc}}, kn+1 \leq j \leq (k+1)n\}$$

and

$$C^r_k = \{j : (i,j) \in P^r_{\text{nonloc}}, kn+1 \leq j \leq (k+1)n\}$$

Processor r has in its local memory the elements of the vector $\mathbf{u}$ whose indices lie in the interval $[rn+1, (r+1)n]$. Before computing the matrix-vector product, for every k such that $R^r_k \neq \emptyset$ processor r sends to processor k the components of vector $\mathbf{u}$ whose indices belongs to R^r_k; it also gets from every processor k such that $C^r_k \neq \emptyset$, the elements of $\mathbf{u}$ whose indices are in C^r_k. We subdivided the overall communication in *communication phases*. On each *phase* two communications are

completed, one with a processor with pid less than r and another with a processor with pid greater than r. When $H \equiv A$ this communication is symmetric due to the symmetric nonzero pattern of A inherited by the MLPG discretization here adopted, and thus is performed by the `MPI_SendRecv` routine. In the case $H \equiv G_L(G_U)$ in each phase processor r sends data to one processor whose pid is $k > r(k < r)$ and receives data from one processor whose pid is $k < r(k > r)$. This schedule is tuned on our block-banded matrices and guarantees minimization of waiting times.

When all the communication phases have completed processors are able to compute locally their part of the matrix-vector product. A scalability analysis of the parallel sparse matrix–vector product implementation, was performed in Martinez et al (2009) and an experimental study of the communication overhead was accomplished. As a result of this study the current version described here was developed showing good scaling behavior even employing more than one thousand processors.

4.2 Parallel implementation of the FSAI preconditioner

We give in this section the main lines of the parallel implementation of the FSAI preconditioner that has been carried out. For the details of the implementation the reader is referred to the paper Bergamaschi et al (2005b).

We implemented the construction of the FSAI preconditioner for general nonsymmetric matrices. Our code allows the specification of both A or A^2 as sparsity patterns with prefiltration and postfiltration. We used a block row distribution of matrices A, G_L and G_U, that is, with complete rows assigned to different processors. All these matrices are stored in static data structures in CSR format.

Even in the nonsymmetric case, we assumed a symmetric non zero pattern for matrix A and set $S_L = S_U^T$. The preconditioner factor G_L is computed by rows while G_U is computed by columns. In this way no added row exchanges is needed respect to the SPD case. Every processor computes a set of rows of G_L and a set of columns of G_U, completely in parallel. Matrix G_U is stored in CSC format only during the computation phase and it is transposed in parallel to CSR format before the start of the iterative solver.

5 Numerical results for axial-symmetric MLPG discretizations

5.1 Problem description

The MLPG1 axi-symmetric poroelastic model is used to simulate the land subsidence above a compacting disk-shaped reservoir. Consider the depletion of a cylindrical volume with small thickness and vertical axis, embedded in a linear elastic, homogeneous and isotropic porous half-space bounded by a horizontal plane at

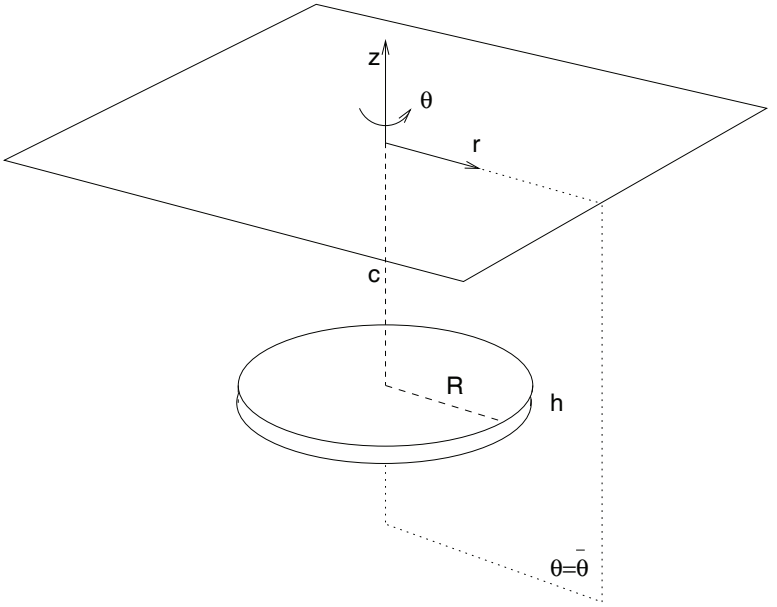

Figure 1: Schematic representation of the land subsidence problem due to the compaction of a deep disk-shaped reservoir in a semi-infinite medium.

$z = 0$ representing the ground surface (see Fig.1). The problem of predicting the land deformation due to a uniform pore pressure variation Δp prescribed within the reservoir has been theoretically solved by Geertsma with the aid of the nucleus of strain concept.

A regular nodal pattern is used, with spacing $\Delta r = \Delta z = d$, over a rectangular cross-section domain $\theta = \overline{\theta}$ extending from the land surface down to 1000 m ($-1000 \leq z \leq 0$ m) and for 3000 m in the radial direction ($0 \leq r \leq 3000$ m) (see Fig.2). Table 1 display the number of nodes and the dimension N of the resulting linear system for different d-values. The porous medium is characterized by $E = 833.33$ MPa and $\nu = 0.25$, corresponding to a uniaxial vertical compressibility equal to $10^{-3}\,\mathrm{MPa}^{-1}$. The values of $r_0^{(k)}$ and $r^{(k)}$ i.e. the size of $\Omega_s^{(k)}$ and the support of $w^{(k)}$, respectively, are $r_0^{(k)} = \alpha d$ and $r_0^{(k)} = \beta d$. The optimal values for $\alpha = 1.425$ ($\alpha = 1.3$ for problem #4) and $\beta = 2.2$ were experimentally found, while the values of the other parameters are shown in Table 1. The reader is referred to Ferronato et al (2007b) for further details on problem description.

All test cases were run on the HPCx supercomputer located at Daresbury (UK). HPCx is available for HPC-Europa visitors at EPCC (Edinburgh Parallel Comput-

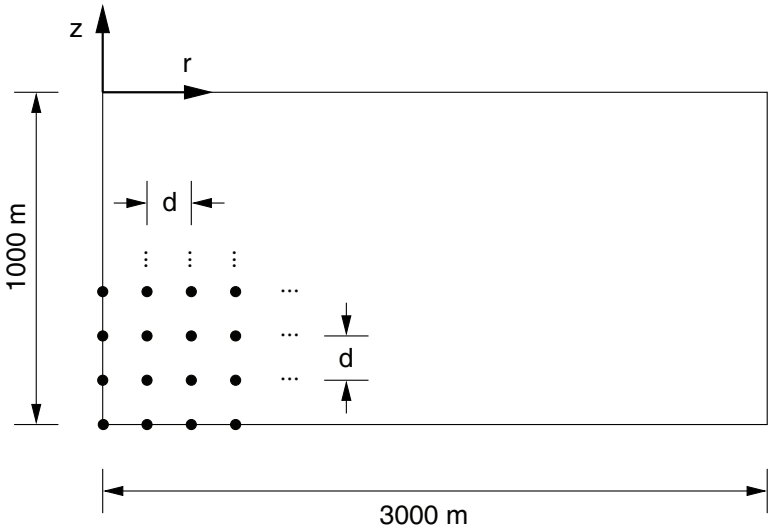

Figure 2: Sketch of the rectangular domain with a regular node pattern.

ing Center). HPCx is at the time of writing 43th in the TOP 500 list of fastest machines in the world.

5.2 HPCx architecture overview

The HPCx system consists out of 160 IBM eServer 575 logical partitions (LPARs) for the compute and 8 IBM eServer 575 LPARs for login and disk I/O. Each eServer LPAR contains 16 processors, the maximum allowable by the hardware. The main HPCx service provides a total of 2560 processors for computation (however the largest CPU count for a single job is 1024).

The eServer 575 compute nodes utilize IBM Power5 processors. The Power5 is a 64-bit RISC processor implementing the PowerPC instruction set architecture. It has a 1.5 GHz clock rate, and has a 5-way super-scalar architecture with a 20 cycle pipeline. There are two floating point multiply-add units each of which can deliver one result per clock cycle, giving a theoretical peak performance of 6.0 Gflop/s.

The level 1 cache is split into a 32 Kbyte data cache and a 64 Kbyte instruction cache. The level 1 data cache has 128-byte lines, is 2-way set associative and write-through. Inter node communication is provided by an IBM's High Performance Switch (HPS), also known as "Federation." Each eServer node has two network adapters and there are two links per adapter, making a total of four links between each of the computing nodes and the switch network. Intra node communication is

Table 1: Problem description. *nr* number of horizontal divisions, *nz* number of vertical divisions, $nodes = (nr+1)(nz+1)$, *nnz* number of nonzero entries of coefficient matrix A

#problem	nr	nz	d	nodes	N	nnz
1	300	100	10	30401	60802	4415972
2	750	250	4	188501	377002	27689972
3	1500	500	2	752001	1504002	110879972
4	3000	1000	1	3004001	6008002	443759972

accomplished via shared memory.

5.3 Results with the FSAI preconditioner using prefiltration

In this section we report the results of our preconditioning strategy for the 4 test problems. We employed the pointwise prefiltration strategy described in the previous section. For each problem, the performance of the FSAI preconditioner is compared to that of the diagonal (JACOBI) preconditioner, which is easier to implement and parallelize. We denote with δ the prefiltration parameter and with ε the postfiltration threshold. T_{prec} is the CPU time to evaluate the preconditioner, T_{sol} the elapsed time for the BiCGSTAB iteration and T is the total time also comprehensive of matrix assembly. We use left preconditioning for the BiCGSTAB method. The initial guess is chosen to be $\mathbf{x}_0 = M\mathbf{b}$, where M is the computed FSAI sparse approximate inverse of A and $\mathbf{b}$ is the right hand side. The iteration is stopped when the residual $\mathbf{r}_k$ satisfies:

$$\|\mathbf{r}_k\| \leq tol_1 + tol_2\|\mathbf{b}\| \tag{14}$$

where we set $tol_1 = tol_2 = 10^{-12}$ (except for problem #4, for which we set $tol_1 = tol_2 = 10^{-10}$ in an attempt to reduce the large number of CPU hours consumed by every run of the code needed to solve this problem). This choice guarantees that the initial residual norm is reduced by a factor 10^{12} if the 2-norm of the right hand side is not too small. However, if $\|\mathbf{b}\| \ll 1$, a common situation in linear systems arising from discretization of PDEs with no source terms, the iteration is stopped when the 2-norm of the residual is less than 10^{-12} (10^{-10} for problem # 4).

In Tables 2–5 we report the performance of the FSAI-BiCGSTAB algorithm with various combination of δ, ε and the initial nonzero pattern (A or A^2). The results presented in the Tables have been produced on the HPCx supercomputer with a fixed number of processors for a given problem. In the majority of the test problems

Table 2: Problem #1, solved using 16 processors.

FSAI preconditioner							
δ	PATT	ε	nnzprec	iter	T_{prec}	T_{sol}	T
0.0	A^2	0.0	15133684	121	5.02	1.12	8.64
0.01	A^2	0.0	8175388	155	1.19	0.90	4.48
0.1	A^2	0.05	2405604	212	0.64	0.76	3.83
0.0	A	0.0	4415972	202	0.34	0.87	3.57
0.05	A	0.0	1267960	288	0.16	0.91	**3.44**
0.05	A	0.05	850538	283	0.17	0.86	3.47
0.1	A	0.1	780598	324	0.16	0.97	3.60
JACOBI preconditioner							
			60802	513	0.0	1.06	**3.47**

Table 3: Problem #2, solved using 32 processors.

FSAI preconditioner							
δ	PATT	ε	nnzprec	iter	T_{prec}	T_{sol}	T
0.0	A^2	0.0	95882884	322	16.3	13.5	38.2
0.01	A^2	0.1	10450762	680	4.2	7.5	19.9
0.1	A^2	0.0	33611662	444	1.9	19.3	29.5
0.1	A^2	0.1	6402086	778	2.1	8.3	18.6
0.0	A	0.0	27689972	566	1.2	10.1	19.4
0.001	A	0.1	6389734	769	1.1	7.6	16.9
0.01	A	0.01	10906790	682	0.7	7.6	16.6
0.05	A	0.05	5276438	713	0.7	7.0	**15.8**
0.1	A	0.0	6370476	941	0.6	9.1	17.7
JACOBI preconditioner							
			377002	1434	0.001	9.50	**17.7**

we see that the best sparsity pattern is represented by A, being that based on A^2 generally too costly to compute. The only exception is provided by the largest test case # 4 where the A^2 pattern with $\delta = 0.1$ and $\varepsilon = 0$ reveals the optimal combination of parameters. Generally, choosing δ and ε parameters is problem dependent. However, the previous results would suggest to choose $\delta \in [0.05, 0.1]$

Table 4: Problem # 3, solved using 128 processors.

FSAI preconditioner							
δ	PATT	ε	nnzprec	iter	T_{prec}	T_{sol}	T
0.0	A^2	0.0	385264884	616	16.6	35.5	64.8
0.001	A^2	0.0	293151928	674	8.6	28.8	50.2
0.01	A^2	0.1	37431220	1692	4.5	20.0	36.3
0.1	A^2	0.0	134723052	936	2.2	23.6	38.3
0.1	A^2	0.05	55603242	1082	2.4	14.8	29.8
0.0	A	0.0	110879972	1159	1.4	29.4	44.7
0.01	A	0.0	49513966	1947	1.1	24.7	28.7
0.01	A	0.01	48010926	1952	1.0	25.9	39.4
0.1	A	0.0	25490982	1493	0.7	15.9	**28.6**
JACOBI preconditioner							
			1504002	2881	0.002	21.2	**33.6**

and $\varepsilon \leq 0.05$.

5.4 Banded FSAI preconditioner

In the following Tables 6 and 7 we report some results of the banded FSAI preconditioner on problems #2 and 4, which we regard as representative of the whole set of results. We selected to use the pattern of A with neither pre nor postfiltration. Note that the two extreme limit cases $nd = 0$ and $nd = N$, where N is the number of rows of the coefficient matrix, corresponds to point JACOBI and FSAI(A) preconditioners, respectively.

From Table 7 we can observe that only in the larger problem there is an optimal nd value, which yields the smallest CPU time. This optimal value is to be related to the mesh size, namely the values of nr and nz. It turns out that 4080 is equal to $\max|i-j|, a_{ij} \neq 0$. This result suggests that a pattern simply based on the nonzero distribution of powers of A may not be the optimal choice. Finding more efficient patterns for the FSAI preconditioner is still an open problem.

5.5 An illustration of parallel performance and scaling

To illustrate the scalability of our code, we report in this section the results obtained on the largest problems # 2,3 and 4 using up to 512 processors on the HPCx machine. Throughout the whole section we will denote with T_p the CPU elapsed

Table 5: Problem # 4, solved using 128 processors.

FSAI preconditioner							
δ	PATT	ε	nnzprec	iter	T_{prec}	T_{sol}	T
0.0	A^2	0.0	1544528884	1009	66.5	256.2	417.6
0.1	A^2	0.0	539415898	1475	8.2	172.8	**277.7**
0.0	A	0.0	443759972	1810	4.8	196.2	294.6
0.001	A	0.01	264183448	2199	4.0	200.2	297.6
JACOBI preconditioner							
			6008002	5204	0.018	355.2	**449.1**

times expressed in seconds (unless otherwise stated) when running the code on p processors. In figure 3 we display the total T_p CPU time vs the number of processors for the above mentioned problems and using either the FSAI or the JACOBI preconditioner to accelerate the parallel BiCGSTAB solver.

It can be observed that the CPU time decreases as the number of processors increases, for both preconditioners. The solid line accounting for the FSAI preconditioner is always under the dot-dashed line representing the use of the JACOBI preconditioner. This shows the superiority (more evident on problem # 4) of the proposed approach with respect to the classical diagonal preconditioner.

Table 6: Problem # 2, banded FSAI preconditioner, with different number of diagonals, *nd* using 32 processors.

nd	nnzprec	iter	T_{prec}	T_{sol}	T
0	377002	1434	0.001	9.60	**17.80**
1	1129504	1347	0.057	11.48	19.77
2	1880504	1339	0.064	11.15	19.30
5	4124492	1009	0.10	9.80	18.15
10	5241980	1063	0.12	10.30	18.64
20	5241980	1063	0.12	10.30	18.64
300	5241980	1063	0.12	10.30	18.64
501	10100480	1190	0.22	12.98	21.34
510	15711980	780	0.50	10.10	18.73
N	27689972	566	1.17	10.13	19.40

Table 7: Problem # 4, banded FSAI preconditioner, with different number of diagonals, *nd* using 256 processors.

nd	nnzprec	iter	T_{prec}	T_{sol}	T
0	6008002	5771	0.004	187.4	236.5
1	17998004	5765	0.78	204.6	254.7
5	65997992	5130	0.54	235.5	285.6
10	83967980	4931	0.66	226.1	274.9
2040	15711980	3720	1.42	208.7	259.7
4080	371174386	1980	2.33	112.4	**164.2**
N	443759972	1935	2.85	120.5	173.2

It can be seen that in each problem the two lines come closer when the number of processor is large, thus accounting for a slight loss of efficiency of the FSAI preconditioner.

We note also that the gap between the two lines increases with the size of the problem, thus indicating that finer discretizations may need more sophisticated preconditioners than the simple JACOBI preconditioner.

In order to further compute the efficiency of the parallel iterative solver in our code, and observing that the number of BiCGSTAB iterations is depending on the number of processor, we first computed an average time per iteration

$$\mathscr{T}_p = \frac{T_{sol}}{\text{iter}},$$

expressed in milliseconds (ms) in Tables 8, 9 and 10. Then we computed a relative measure of the parallel efficiency achieved by the solver defining as $S_p^{(\bar{p})}$ the pseudo speedup computed with respect to the smallest number of processors ($\bar{p}$) used to solve the given problem:

$$S_p^{(\bar{p})} = \frac{\mathscr{T}_{\bar{p}}\bar{p}}{\mathscr{T}_p}.$$

We will denote $E_p^{(\bar{p})}$ the corresponding scaled relative efficiency, obtained according to

$$E_p^{(\bar{p})} = \frac{S_p^{(\bar{p})}}{p} = \frac{\mathscr{T}_{\bar{p}}\bar{p}}{\mathscr{T}_p p}.$$

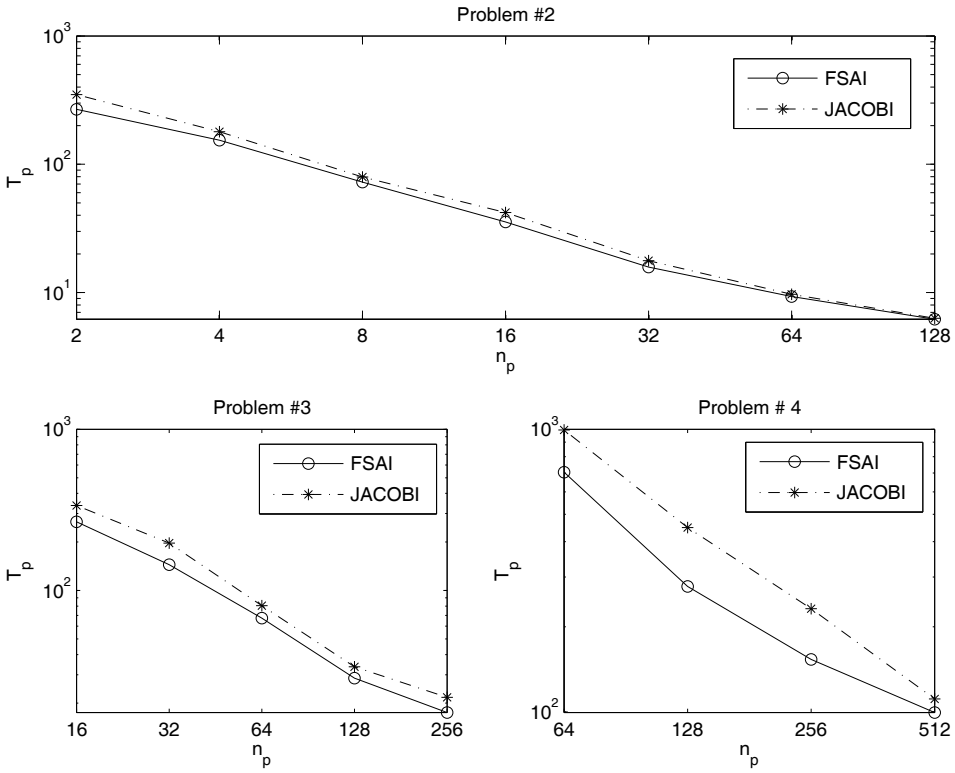

Figure 3: Scaling results: overall CPU time T_p vs number of processors, for problems #2, #3 and #4, using FSAI or JACOBI preconditioners.

Note that we could not solve the largest problems with a smaller number of processors due to memory limitations.

In Tables 8, 9 and 10, we show that the FSAI-BiCGSTAB algorithm is almost perfectly scalable, with the relative efficiency of the 78% in the worst case. In some instances, both the preconditioners (FSAI and JACOBI) display the so called "superspeedup" with an efficiency larger than 100%. This is likely to be ascribed to cache effects. Finally it can also be observed that employing the FSAI preconditioner yields a number of iterations and total CPU time which is smaller than using JACOBI preconditioner, irrespective of the number of processors employed. These results confirm that FSAI preconditioner with pre and post filtration represents an efficient and scalable preconditioner for the iterative solution of MLPG models.

Table 8: Timing results and scaled efficiency for solving problem #2.

Procs	iter	T_{prec}	T_{sol}	T_p	$\mathscr{T}_p$ (ms)	$E_p^{(\bar{p})}$
FSAI(A,$\varepsilon = 0.05$), $\delta = 0.05$ preconditioner						
2	750	7.6	151.6	286.2	202.1	
4	804	3.9	86.2	154.0	107.2	0.94
8	763	2.0	38.8	72.6	50.9	0.99
16	712	1.0	18.4	35.5	25.8	0.98
32	713	0.7	7.0	15.8	9.8	>1.00
64	742	0.4	3.8	9.3	5.1	>1.00
128	725	0.4	2.2	6.2	3.0	>1.00
JACOBI preconditioner						
2	1354	0.031	222.1	349.3	164.0	
4	1447	0.015	115.6	179.1	79.89	>1.00
8	1302	0.007	47.7	79.5	36.6	>1.00
16	1273	0.003	26.0	42.1	20.4	1.00
32	1434	0.001	9.5	17.7	6.6	>1.00
64	1336	0.001	4.8	9.7	3.6	>1.00
128	1274	0.001	2.8	6.3	2.2	>1.00

6 Conclusions

An efficient parallel meshless model, based on the MLPG method, is developed for axi-symmetric poroelastic models. The parallelization strategy allows every processor to construct concurrently its local part of the stiffness matrix without any communication with any other processor. We also developed a preconditioned parallel solver based on the BiCGSTAB method for the solution of the linear system arising from the MLPG discretization. The study carried on in this paper on a class of approximate inverse preconditioner (FSAI) with various strategies to reduce the fill in of the preconditioner together with the use of an efficient parallel matrix vector product, reveal that the FSAI preconditioner can be conveniently used to accelerate the BiCGSTAB method in the solution of MLPG discretization of poroelastic models. In particular, the FSAI preconditioners show their superiority with respect to the classical and "perfectly parallelizable" JACOBI preconditioner, both in terms of iteration number and CPU time, irrespective of the number of processors employed. Therefore, the proposed parallel method reveals a promising tool for solving large-scale realistic geomechanical problems.

Table 9: Timing results and scaled efficiency for solving problem #3.

Procs	iter	T_{prec}	T_{sol}	T_p	$\mathscr{T}_p$ (ms)	$E_p^{(\bar{p})}$
FSAI(A,$\varepsilon = 0.0$), $\delta = 0.1$ preconditioner						
16	1619	3.3	177.6	266.6	109.7	
32	1702	1.9	99.4	144.6	58.4	0.94
64	1491	1.1	44.2	67.4	29.6	0.93
128	1493	0.7	15.9	28.6	10.7	>1.00
256	1363	0.6	8.4	17.5	6.2	>1.00
JACOBI preconditioner						
16	2999	0.015	250.3	336.2	83.5	
32	3303	0.008	153.6	196.8	46.5	0.90
64	2669	0.008	57.9	80.4	21.7	0.96
128	2881	0.002	21.2	33.6	7.4	>1.00
256	2871	0.001	12.7	21.7	4.4	>1.00

Table 10: Timing results and scaled efficiency for solving problem #4.

Procs	iter	T_{prec}	T_{sol}	T_p	$\mathscr{T}_p$ (ms)	$E_p^{(\bar{p})}$
FSAI(A^2,$\varepsilon = 0.0$), $\delta = 0.1$ preconditioner						
64	2051	15.4	502.2	704.7	244.9	
128	1475	8.2	172.8	277.7	117.2	>1.00
256	1546	4.6	96.7	153.3	62.5	0.98
512	1640	2.9	64.2	99.5	39.1	0.78
JACOBI preconditioner						
64	5657	0.02	808.6	994.2	142.9	
128	5204	0.02	355.2	449.1	68.3	>1.00
256	5771	0.02	180.2	231.7	31.2	>1.00
512	5841	0.01	80.2	111.1	13.7	>1.00

Acknowledgement: This study has been supported by the Italian MIUR project "Numerical models for multiphase flow and deformation in porous media". The first two authors were also supported by the HPC-Europa programme, funded under the European Commission's Research Infrastructures activity of the Structuring the European Research Area programme, contract number RII3-CT-2003-506079.

References

Atluri S. N., Shen S. (2002a) The Meshless Local Petrov Galerkin (MLPG) Method, Tech Science Press, Forsyth, GA, USA.

Atluri S. N., Shen S. (2002b) The Meshless Local Petrov Galerkin (MLPG) Method: A simple & less-costly alternative to the finite element and boundary element methods, *CMES: Computer Modeling in Engineering & Sciences*, 3 (1) pp. 11-51.

Atluri S. N., Zhu T. (1998) A New Meshless Local Petrov-Galerkin (MLPG) Approach in Computational Mechanics, *Computational Mechanics*, 22, pp. 117-127.

Belytschko T., Krongauz Y., Organ D., Fleming M., Krysl P. (1996) Meshless methods: an overview and recent developments. *Comput Meth. in Applied Mech. and Engrg.*, 139: pp. 3–47.

Belytschko T., Lu Y. Y., Gu L. (1994) Element-free Galerkin methods. *International Journal of Numerical Methods in Engineering*, 37: pp. 229–256.

Benzi M., Cullum J. K., Tůma M. (2000) Robust approximate inverse preconditioning for the conjugate gradient method, *SIAM J. Sci. Comput.*, 22, pp. 1318–1332.

Benzi M., Tůma M. (1998) A sparse approximate inverse preconditioner for nonsymmetric linear systems, *SIAM J. Sci. Comput.*, 19, pp. 968–994.

Benzi M., Tůma M. (1999) A comparative study of sparse approximate inverse preconditioners, *Applied Numerical Mathematics*, 30, pp. 305–340.

Bergamaschi L., Gambolati G., Pini G. (2007) A numerical study of inverse preconditioning for the parallel iterative solution to 3d finite element flow equations, *J. Comput. Appl. Math.*, 210, pp. 64–70.

Bergamaschi L., Caliari M., Martínez A., Vianello M. (2005a) A parallel exponential integrator for large-scale discretizations of advection-diffusion models. *In Recent Advances in PVM and MPI, 12th European PVM/MPI Users' Group Meeting, Sorrento, Italy, volume 3666 of Lecture Notes in Computer Sciences*, pages 483–492. Springer-Verlag Heidelberg.

Bergamaschi L., Martínez A. (2005b) Parallel acceleration of Krylov solvers by factorized approximate inverse preconditioners, *in VECPAR 2004, M. Daydè et al., ed., vol. 3402 of Lecture Notes in Computer Sciences*, Heidelberg, Springer-Verlag, pp. 623–636.

De S., Bathe K. J. (2000) The method of finite spheres. *Computational Mechanics*, 25: pp. 329–345.

Ferronato M., Gambolati G., Teatini P., Baù D. (2004) Radioactive marker measurements in heterogeneous reservoirs: numerical study. *International Journal of Geomechanics*, 4: pp. 79–92.

Ferronato M., Gambolati G., Teatini P., Janna C. (2007) Casing influence in reservoir compaction measurement by radioactive markers in the Northern Adriatic, Italy *International Journal of Geomechanics* 7, pp. 444-447.

Ferronato M., Mazzia A., Pini G., Gambolati G. (2007) A meshless method for axi-symmetric poroelastic simulations: numerical study, *International Journal for Numerical Methods in Engineering*, 70, pp. 1346–1365.

Ferronato M., Mazzia A., Pini G., Gambolati G. (2008) Accuracy and performance of Meshless Local Petrov-Galerkin methods in 2-D elastostatic problems. *JP Journal of Solids and Structures*, vol. 1, pp. 34–53.

Gambolati G. (1974) Second-order theory of flow in three-dimensional deforming media. *Water Resources Research*, 10: pp. 1217–1228.

Gambolati G., Teatini P., Baù D., Ferronato M. (2000) Importance of poroelastic coupling in dynamically active aquifers of the Po river basin, Italy. *Water Resources Research*, 36: pp. 2443–2459.

Gambolati G., Teatini P., Tomasi L. (1999) Stress-strain analysis in productive gas/oil reservoirs, *Int. J. Numer. Anal. Methods Geom.*, 23, pp. 1495–1519.

Geertsma J. (1973) Land subsidence above compacting oil and gas reservoirs. *Journal of Petroleum Technology*, 25: pp. 734–744.

Grote M. J., Huckle T. (1997) Parallel preconditioning with sparse approximate inverses, *SIAM J. Sci. Comput.*, 18, pp. 838–853.

Jarak T., Soric J. (2008) Analysis of rectangular square plates by the mixed Meshless Local Petrov-Galerkin (MLPG) approach. *CMES: Computer Modeling in Engineering & Sciences*, 38 (3), pp. 231–262.

Kaporin I. E. (1994) New convergence results and preconditioning strategies for the conjugate gradient method, *Numer. Lin. Alg. Appl.*, 1, pp. 179–210.

Kolotilina L., Nikishin A. A., Yeremin A. (1999) Factorized sparse approximate inverse preconditionings IV. Simple approaches to rising efficiency, *Numer. Lin. Alg. Appl.*, 6, pp. 515–531.

Kolotilina L., Yeremin A. (1993) Factorized sparse approximate inverse preconditionings I. Theory, *SIAM J. Matrix Anal.*, 14, pp. 45–58.

Lu Y. Y., Belytschko T., Gu L. (1994) A new implementation of the element free Galerkin method. *Comput Meth. in Applied Mech. and Engrg.*, 113: pp. 397–414.

Ma Q. W. (2008) A New Meshless Interpolation Scheme for MLPG_R Method. *CMES: Computer Modeling in Engineering & Sciences*, 23 (2), pp. 75–90.

Martínez Á., Bergamaschi L., Caliari M., Vianello M. (2009) A massively parallel exponential integrator for advection-diffusion models. *J. Comput. Appl. Math.*, 231 (1), pp. 82–91.

Nikishin A. A., Yeremin A. (2003) Prefiltration technique via aggregation for constructing low density high quality factorized sparse approximate inverse preconditionings., *Numer. Lin. Alg. Appl.*, 10, pp. 235–246.

Peirce W. H. (1957) Numerical integration over the planar annulus. *Journal of the Society of Industrial and Applied Mathematics*, 5: pp. 66–73.

Pini G., Mazzia A., Sartoretto F.. (2008) Accurate MLPG Solution of 3D Potential Problems. *CMES: Computer Modeling in Engineering & Sciences*, 36 (1), pp. 43–64.

Sladek J., Sladek V., Solek P., Wen P. H. (2008a) Thermal Bending of Reissner-Mindlin Plates by the MLPG. *CMES: Computer Modeling in Engineering & Sciences*, 28 (1), pp. 57–74.

Sladek J., Sladek V., Tan C. L., Atluri S. N. (2008b) Analysis of Transient Heat Conduction in 3D Anisotropic Functionally Graded Solids, by the MLPG Method *CMES: Computer Modeling in Engineering & Sciences*, 32 (3), pp. 161–174.

van der Vorst H. A. (1992) Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 13: pp. 631–644.

Verruijt A. (1969) Elastic storage of aquifers. In R. J. M. De Wiest, editor, Flow Through Porous Media, pp. 331–376. Academic Press, New York (NY).

Wendland H. (1999) Meshless Galerkin methods using radial basis function. *Mathematics of Computation*, 68: pp. 1521–1531.

Yuan W., Chen Y., Gagalowicz A., Liu K. (2008) Application of Meshless Local Petrov-Galerkin (MLPG) Method in Cloth Simulation. *CMES: Computer Modeling in Engineering & Sciences*, 35 (2), pp. 133–156.

Yuan W., Chen Y., Liu K. (2007) A New Quasi-Unsymmetric Sparse Linear Systems Solver for Meshless Local Petrov-Galerkin Method (MLPG). *CMES: Computer Modeling in Engineering & Sciences*, 17 (2), pp. 115–134.

Zheng J., Long S., Xiong Y., Li G. (2009) A Finite Volume Meshless Local Petrov-Galerkin Method for Topology Optimization Design of the Continuum Structures. *CMES: Computer Modeling in Engineering & Sciences*, 42 (1), pp. 19–34.