DELFT UNIVERSITY OF TECHNOLOGY

REPORT 11-05

Comparison of the Deflated Preconditioned Conjugate Gradient method and parallel direct solver for composite materials.

T.B. Jönsthövel, M.B. van Gijzen, S. MacLachlan, C. Vuik, A. Scarpas

ISSN 1389-6520

Reports of the Department of Applied Mathematical Analysis

Delft 2011

COMPARISON OF THE DEFLATED PRECONDITIONED CONJUGATE GRADIENT METHOD AND PARALLEL DIRECT SOLVER FOR COMPOSITE MATERIALS

T.B. JÖNSTHÖVEL[†], M.B. VAN GIJZEN[‡], S. MACLACHLAN[⋆], C. VUIK[‡], AND A. SCARPAS[†]

Abstract. The demand for large FE meshes increases as parallel computing becomes the standard in FE simulations. Direct and iterative solution methods are used to solve the resulting linear systems. Many applications concern composite materials, which are characterized by large discontinuities in the material properties. An example of such a material is asphalt concrete, which is a mixture of components with large differences in material stiffness. Such discontinuities give rise to small eigenvalues that negatively affect the convergence of iterative solution methods such as the Preconditioned Conjugate Gradient (PCG) method. This paper considers the Deflated Preconditioned Conjugate Gradient (DPCG) method for solving such systems within reasonable time using the rigid body modes of sets of elements with homogeneous material properties. We compare the performance of the parallel direct solver MUMPS, the PCG method and the DPCG method for the FE mesh of a real asphalt core sample. The mesh is obtained using a CT scan. We show that the DPCG method is the method of choice for large linear systems with respect to the wall clock time, storage and accuracy of the solution.

1. Introduction

Finite element computations are indispensable for the simulation of material behavior. Recent developments in visualization and meshing software give rise to high-quality but very large meshes. As a result, large systems with millions of degrees of freedom need to be solved. When choosing a solver we distinguish between direct solution methods and iterative methods. In recent years parallel computing has become the standard in FE software packages, therefore only parallel algorithms are considered. In our application, the finite element stiffness matrix is symmetric, positive definite and therefore the Preconditioned Conjugate Gradient (PCG) method is the iterative method of choice. Furthermore the PCG method is well suited for parallel computing.

Many finite element computations involve simulations of *inhomogenous* materials. The difference in properties of materials lead to large differences in the entries of the stiffness matrix. We have shown in [10] that these jumps slow down the convergence of the PCG method. By decoupling regions with homogeneous material properties with a deflation technique a more robust PCG method has been constructed: the Deflated Preconditioned Conjugate Gradient (DPCG) method. The

2000 *Mathematics Subject Classification.* 65F10, 65F08, 65Z05.

Key words and phrases. deflation, preconditioners, conjugate gradients, rigid body modes, CT scan, structural mechanics.

[†]Delft University of Technology, Faculty of Civil Engineering, Department of Structural Mechanics, 2628CN Delft, the Netherlands (t.b.jonsthovel@tudelft.nl, a.scarpas@tudelft.nl).

[‡]Delft University of Technology, Faculty Electrical Engineering, Mathematics and Computer Science, Department of Applied Mathematical Analysis, 2628CN Delft, the Netherlands (m.b.vangijzen@tudelft.nl, c.vuik@tudelft.nl).

[⋆]Tufts University, Department of Mathematics, Bromfield-Pearson Building, 503 Boston Avenue, Medford, MA 02155, USA (scott.macLachlan@tufts.edu).

DPCG method proposed in [10] is an extension of the technique of subdomain deflation, introduced in [13]. There is a correlation between the number of rigid body modes of sub-bodies of materials contained within the FE mesh and the number of small eigenvalues of the scaled stiffness matrix. We used rigid body modes combined with existing deflation techniques to remove those small eigenvalues from the spectrum of the scaled stiffness matrix yielding a stable and robust adaptation of the PCG method. Like the PCG method, the DPCG method is well suited for parallel computing.

The alternative for iterative methods are direct methods. An important advantage of direct solution methods is their robustness: they can to a large extend be used as a black box for solving a wide range of problems. For this reason they are still popular for use in general finite element codes. Several high quality, well parallelisable public domain direct solvers exist. Of these we mention for example SuperLU [12], SPIKE [14], PARDISO [16], ILUPACK [3], MUMPS [1] and those solvers contained in PETSc. For our comparisons we have selected MUMPS as direct solver because it has support for element-based data structures which coincides with the assembly of the stiffness matrix in our FE code. Moreover it is easy to embed into existing software due to its interface and it is known for good performance on parallel, distributed memory hardware.

In this paper we will compare the performance of MUMPS, PCG and DPCG within a parallel environment on the solution of the large systems that come from FE meshes. We will provide an overview of the DPCG method proposed in [10] discuss the parallel implementation of the DPCG method into an existing FE software package. Finally, we present numerical experiments on FE meshes from real life cores of asphalt c oncrete as case studies for this comparison.

2. Problem definition: composite materials

Until recently, because of the extremely long execution time, memory and storage space demands, the majority of FE simulations of composite materials were performed by means of homogenization techniques [6]. Unfortunately these techniques do not provide an understanding of the actual interaction between the components of the material. Nevertheless, it is known that component interaction is the most critical factor in determining the overall mechanical response of the composite material.

In this paper, we consider asphalt concrete as an example of a composite material. It consists of a mixture of bitumen, aggregates and air voids. Obviously the difference between the stiffness of bitumen and the aggregates is significant, especially at high temperatures. The surge in recent studies on wheel-pavement interaction show the importance of understanding the component interaction within asphalt concrete, demanding high quality FE meshes.

We obtain accurate finite element meshes of the asphalt concrete materials by means of Computed Tomography (CT) X-ray scans and additional, specialized software tools like Simpleware ScanFE [17].

We use the computational framework described in [6] to simulate the response of a composite material that is subjected to external forces by means of small load steps. By using the FE method we obtain the corresponding stiffness matrix. Solving linear system (1),

$$Ku = f \tag{1}$$

is the most time consuming computation of the FE simulation. In this equation u represents the change of displacement of the nodes in the FE meshes and f the force unbalance in the system, which is determined by the difference of the internal forces within the system and the external forces exerted on the system. The internal forces are computed by solving non-linear equations for each finite element. The computing time and costs are negligible compared to solving linear system

(1). The stiffness matrix K is symmetric positive definite for elastic, constrained systems, hence $\forall u \neq 0 : u^T K u > 0$ and all eigenvalues of K are positive. Within the context of mechanics, $\frac{1}{2} u^T K u$ is the strain energy stored within the system for displacement vector u, [2]. Energy is defined as a non-negative entity, hence the strain energy must be non-negative also.

3. Solvers

3.1. **Direct solution method.** Solving system (1) can be done by computing the LU-decomposition, or more specific in the case of a symmetric system, the $K = R^T R$ decomposition of the stiffness matrix. The well known algorithm for finding $R^T R$ is the Cholesky algorithm, a modified version of the Gaussian elimination. For singular matrices the Cholesky decomposition cannot be determined due to zero pivots. Many adaptions of the Cholesky algorithm and new methods have been developed to obtain more robust and faster algorithms for the determination of the decomposition. In general the conditioning as well as the bandwidth of the matrix are the most important factors with respect to work and stability for any direct solution method. We refer to [8] for an extensive overview of direct solution methods.

We consider direct solution methods as black-box solution methods. Direct solution methods are guaranteed to find the decomposition for well-conditioned, non-singular matrices without requiring any prior knowledge of the linear system. For this reason direct solution methods are widely used within the field of engineering. Moreover, when the decomposition of the stiffness matrix has been computed, solving system (1) for multiple right sides is cheap in terms of work and fast in time. Hence, many different linear systems can be solved within a small amount of time if the stiffness matrix remains unchanged and its decomposition has been computed. An obvious application is the use of direct solution methods where the tangent stiffness matrix is kept constant during the non-linear solution process. The main disadvantage of direct solution methods is the high demand of storage, potentially the full bandwidth of the matrix. Therefore direct solution methods are less favorable when solving large systems of equations resulting from 3D FE meshes containing high connectivity of the elements.

3.2. **Preconditioned Conjugate Gradient method.** Another class of solvers are the Krylov methods. Those methods find a solution for system (1) within a given accuracy.

Because K is SPD, CG [9] will be used to solve (1) iteratively. The CG method is based on minimizing the energy error of the $i-th$ solution over the Krylov subspace,

$$\mathcal{K}^{i-1}(K; r_0) = \text{span}\{r_0, Kr_0, ..., K^{i-1} r_0\}. \tag{2}$$

The energy norm is defined as $\|u\|_K = \left(u^T K u\right)^{\frac{1}{2}}$. We note that minimizing the error in the K-norm is in fact minimizing the strain energy over the Krylov subspace $\mathcal{K}^{i-1}(K; r_0)$. This implies that for a given distributed static load we construct a displacement vector that has an optimal distribution of the force over the material.

Theorem 10.2.6 in [8] provides a bound on the error of CG. Let us denote the ith eigenvalue of K in nondecreasing order by $\lambda_i(K)$ or simply by λ_i. After k iterations of the CG method, the error is bounded by,

$$\|u - u_k\|_K \leq 2\|u - u_0\|_K \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \tag{3}$$

where $\kappa = \kappa(K) = \dfrac{\lambda_n}{\lambda_1}$ is the spectral condition number of K, and the K-norm of u is given by $\|u\|_K = \sqrt{u^T K u}$. The error reduction capability of CG is limited when the condition number is large. The condition number of K will increase when the number of elements increases or when the stiffness of the materials changes. For plastic and viscous behavior this can result in a series of increasing number of iterations as the stiffness changes every load or time step. However, this is out of the scope of this paper but will need future research as plasticity and viscosity are key to realistic simulations.

The convergence of CG is not only affected by the condition number but also by the number and distribution of very small eigenvalues, which has been shown in [19]. The eigenvectors corresponding to the smallest eigenvalues do have a significant contribution to the global solution but may need a significant number of iterations to convergence locally. Hence, very small eigenvalues can increase the number of iterations. We will see that the number of aggregates has a direct correlation with the number of smallest eigenvalues of K. Increasing the number of aggregates may therefore result in more very small eigenvalues and deterioration of the convergence rates.

To improve the performance of CG we change the linear system resulting in a problem with more favorable extreme eigenvalues and/or clustering. The most efficient way to do this is by preconditioning of the linear system. Preconditioners are essential for the performance of iterative solvers and no Krylov iterative solver can perform well without one [15].

The preconditioned stiffness matrix reads

$$M^{-1} K u = M^{-1} f, \tag{4}$$

where matrix M is the left preconditioner which is assumed to be symmetric, positive definite too. The CG iteration bound of equation (3) also applies to the preconditioned matrix. The preconditioning matrix must satisfy the requirements that it is cheap to construct and it is inexpensive to solve the linear system $Mv = w$. This is because preconditioned algorithms need to solve the linear system $Mv = w$ every iteration step. A rule of thumb is that M must resemble the original matrix K to obtain eigenvalues that cluster around 1. Obviously $M = K$ would be the best but most expensive choice and is equivalent to solving the original system. Common choices of M are the diagonal of K, which is known as diagonal scaling, and the Incomplete Cholesky factorization using a drop tolerance for the fill-in.

We consider the PCG method method of choice when solving a large linear system which is well-conditioned. PCG iterations are cheap, and the storage demands are modest and fixed. However, the condition number of the matrix, and therefore the eigenvalues, determine the performance of the PCG method. The amount of iterations needed for convergence depends on the condition number. Moreover, the residual can be small, but the corresponding approximate solution may be far from the true solution. Hence, stability and robustness are important aspects of the PCG method that need extra attention. Those can be improved by using the right preconditioners, but those may be expensive in terms of work and storage. Within the field of engineering the PCG method remains widely used for it is easy to implement. The PCG method uses the right hand side for determining the solution of (1), hence it is less favorable compared to direct solution methods when using the Modified Newton Method with initial stiffness. However, for highly non-linear materials it may be advantage to use the PCG method as the stiffness matrix may be changed within every iteration step of the full Newton Method, yielding less iterations and thus evaluation of the internal forces.

This reduces computation time as it is not always required to find an accurate solution, for example when solving the tangent for the non-linear solution process.

3.3. **Deflated Preconditioned Conjugate Gradient method.** We have shown in [10] that the number of iterations to convergence for preconditioned CG is highly dependent on the number of aggregates in a mixture as well as the ratio of the E moduli. Increasing the number of aggregates introduces correspondingly more (clustered) small eigenvalues in stiffness matrix K. The jumps in the E moduli are related to the size of the small eigenvalues. We know from [19] that the smallest eigenvalues correspond to the slow converging components of the solution.

When a matrix K_{unc} represents a rigid body, i.e. an unconstrained mechanical problem (with no essential boundary conditions) the strain energy equals zero for the rigid body displacements as the system remains undeformed and the matrix is positive semi-definite, $\forall u : u^T K_{unc} u \geq 0$. More specifically, the number of rigid body modes of any unconstrained volume equals the number of zero-valued eigenvalues of its corresponding stiffness matrix. When a matrix has zero-valued eigenvalues the kernel $\mathcal{N}(A)$ is non-trivial. Moreover the basis vectors of the kernel of a stiffness matrix represent the principal directions of the rigid body modes. In general, two types of rigid body modes exist: translations and rotations. In three dimensions this implies six possible rigid body modes and hence six kernel vectors can be associated with the rigid body modes.

For any finite element computation we consider subsets of unconstrained elements as rigid bodies. Their corresponding (sub) stiffness matrices are assemblies of the element stiffness matrices. In the context of asphalt concrete the aggregates are sub-sets of elements, with their E modulus as a shared property, as well as the bitumen and the air voids.

In [10] we conclude that the number of aggregates times the number of rigid body modes per aggregate (6 in three dimensions) is equal to the number of small eigenvalues of stiffness matrix K. By using the deflation technique we augment the Krylov subspace with pre-computed rigid body modes of the aggregates and remove all corresponding small eigenvalues from the system. As a result the number of iterations of the Deflated Preconditioned Conjugated Gradient method is nearly not affected by jumps in material stiffness or by the number of aggregates.

For the description of deflation we split the solution of (1) into two parts [7]

$$u = \left(I - P^T\right) u + P^T u, \tag{5}$$

where P is a projection matrix that is defined by,

$$P = I - KZ(Z^T KZ)^{-1} Z^T, \quad Z \in \mathbb{R}^{n \times m} \tag{6}$$

where Z is the deflation subspace, i.e., the space to be projected out of the system, and I is the identity matrix of appropriate size. We assume that $m \ll n$ and Z has rank m. Under this assumption $K_c \equiv Z^T KZ$ is symmetric positive definite and may be easily computed and factored. Hence,

$$\left(I - P^T\right) u = ZK_c^{-1} Z^T Ku = ZK_c^{-1} Z^T f \tag{7}$$

can be computed immediately. We only need to compute $P^T u$. Because KP^T is symmetric,

$$KP^T = PK, \tag{8}$$

we solve the deflated system,

$$PK\hat{u} = Pf \tag{9}$$

for $\hat{u}$ using the CG method and multiply the result by P^T. We should note that (9) is singular. However, the projected solution $P^T\hat{u}$ is unique, it has no components in the null space, $\mathcal{N}(PK) =$

span$\{Z\}$. Moreover, from [11], [19] we learn that the null space of PK never enters the iteration process and the corresponding zero-eigenvalues do not influence the solution.

To obtain a useful bound for the error of CG for positive semi-definite matrices we define the effective condition number of a semi-definite matrix $D \in \mathbb{R}^{n\times n}$ with corank m to be the ratio of the largest and smallest positive eigenvalue analogue to equation (3),

$$\kappa_{\text{eff}}(D) = \frac{\lambda_n}{\lambda_{m+1}}. \tag{10}$$

Theorem 2.2 from [7] here repeated as Theorem 3.1 implies that a bound on the condition number of PK can be obtained.

Theorem 3.1. *Let P as defined in (6) and suppose there exists a splitting $K = C + R$ such that C and R are symmetric positive semi-definite with $\mathcal{N}(C) = span\{Z_k\}$ the null space of C. Then for ordered eigenvalues λ_i,*

$$\lambda_i(C) \leq \lambda_i(PK) \leq \lambda_i(C) + \lambda_{max}(PR). \tag{11}$$

Moverover, the effective condition number of PK is bounded by,

$$\kappa_{eff}(PK) \leq \frac{\lambda_n(K)}{\lambda_{m+1}(C)}. \tag{12}$$

Proof. See [7] (p445). □

The large discontinuities in matrix entries due to strongly varying material properties in the FE discretization induce unfavorable eigenvalues (either large or small) in the spectrum of stiffness matrix K. The effective condition number of PK is bounded by the smallest eigenvalue of C and the largest eigenvalue of K. To remove the discontinuities and thus eliminating those unfavorable eigenvalues we decouple the sub-matrices of stiffness matrix K that correspond to different materials by finding the correct splitting. The eigenvalues of the decoupled sub-matrices determine the spectrum of PK. However, due to the large differences in stiffness the value of the eigenvalues for different sub-matrices can vary over several order of magnitudes. We use a preconditioner to map the spectra of the sub-matrices onto the same region, around 1. The deflation technique can be used in conjunction with ordinary preconditioning techniques such as diagonal scaling or Incomplete Cholesky factorization. This is a two-level approach, treating the smallest eigenvalues and largest eigenvalues by deflation and preconditioning respectively. By choosing a smart combination of deflation and preconditioning a more favorable spectrum is obtained, yielding a smaller condition number and less iterations. For a symmetric preconditioner $M = LL^T$, e.g. diagonal scaling, we extend the result of Theorem 3.1 to

$$\kappa_{\text{eff}}(L^{-1}PKL^{-T}) \leq \frac{\lambda_n(L^{-1}KL^{-T})}{\lambda_{m+1}(L^{-1}CL^{-T})}. \tag{13}$$

We introduce a strategy to construct the deflation space Z to obtain decoupled problems using Theorem 3.1. We observe that null spaces of sets of elements are represented by the rigid body modes of those sets of elements. By choosing sets of elements we define C and the null space of C is our deflation space, which is by definition spanned by the rigid body modes. In Appendix A an algorithm is given for computing rigid body modes of sets of elements. The matrix C consists of the assembly of all finite elements that belong to a body of material. The matrix R consists of the assembly of all finite elements that share nodes with the elements on the boundary of a

body of material but that are not contained within the sub-mesh. We note that if some elements of a less stiff material are assigned to the element set of a stiffer material, the material stiffness matrices are not decoupled. So for instance, when a node belongs to two elements and two different materials and is assigned to the wrong (least stiff) element with respect to the splitting of K, then the preconditioning step will reintroduce the coupling.

The DPCG method [18] is given as Algorithm 1.

Algorithm 1 Deflated preconditioned CG solving $K\mathbf{u} = \mathbf{f}$

Select $\mathbf{u}_0$. Compute $\mathbf{r}_0 = (\mathbf{f} - K\mathbf{u}_0)$, set $\hat{\mathbf{r}}_0 = P\mathbf{r}_0$ and $\mathbf{p}_0 = \hat{\mathbf{r}}_0$
Solve $M\mathbf{y}_0 = \hat{\mathbf{r}}_0$ and set $\mathbf{p}_0 = \mathbf{y}_0$
for $j = 0, 1, \ldots$ until convergence **do**
  $\hat{\mathbf{w}}_j = PK\mathbf{p}_j$
  $\alpha_j = \frac{\left(\hat{\mathbf{r}}_j, \mathbf{y}_j\right)}{\left(\hat{\mathbf{w}}_j, \mathbf{p}_j\right)}$
  $\hat{\mathbf{u}}_{j+1} = \hat{\mathbf{u}}_j + \alpha_j \mathbf{p}_j$
  $\hat{\mathbf{r}}_{j+1} = \hat{\mathbf{r}}_j - \alpha_j \hat{\mathbf{w}}_j$
  Solve $M\mathbf{y}_{j+1} = \hat{\mathbf{r}}_{j+1}$
  $\beta_j = \frac{\left(\hat{\mathbf{r}}_{j+1}, \mathbf{y}_{j+1}\right)}{\left(\hat{\mathbf{r}}_j, \mathbf{y}_j\right)}$
  $\mathbf{p}_{j+1} = \mathbf{y}_{j+1} + \beta_j \mathbf{p}_j$
end for
$\mathbf{u} = ZK_c^{-1}Z^T\mathbf{f} + P^T\hat{\mathbf{u}}_{j+1}$

We consider the DPCG method as an extension to the PCG enhancing stability and robustness when solving for a symmetric, positive definite matrix. The DPCG method yields extra storage for the deflation matrix Z. Moreover PKu in Algorithm 1 needs to be computed in every iteration. However, the unfavorable eigenvalues due to the discontinuities in the stiffness matrix are treated by the deflation method. Therefore the convergence of the DPCG method is assured for even highly ill-conditioned problems. Moreover, the accuracy of the acquired solution is better compared to the PCG method.

4. Parallel computing

4.1. **Parallel paradigm: domain decomposition.** We have implemented the computational framework described in [6] in the FE software package CAPA-3D [4]. In the scope of this research we have parallelized CAPA-3D on basis of domain decomposition. This section describes the basic principles behind parallelism applied to FE meshes based on domain decomposition. We disregard all issues related to implementation. Introduce domain Ω which is divided into D subdomains yielding $\Omega = \sum_{d=1}^{D} \Omega_d$. Domain Ω holds E elements, each subdomain holds E_d elements, hence $E = \sum_{d=1}^{D} E_d$. Elements can share nodes - degrees of freedom - that lie in multiple subdomains, but no element is contained in more than one subdomain. Element wise operations can be done independently for each subdomain as long as the values of any quantity at shared nodes are updated after finishing the operation. Examples of elementwise operations are numerical integration, matrix-vector multiplications, dot products etc. The iterative solution methods PCG and DPCG consist of matrix-vector multiplications, dot products and the preconditioning operator. The direct solution method is provided as a black box and hence we do not have to define any special parallel operations for implementation.

4.1.1. *Subdomain mapping operators.* We define two operators for mapping vectors and matrices on subdomains onto the global domain and scaling of vectors for shared nodes in multiple subdomains. The mapping operator M_d is essentially identical to the finite element connectivity matrix N_e for

assembling stiffness matrix K_e into K. The operator M_d has dimension $N^d \times N$ and consists of one and zero entries. We can map vector u_d from subdomain Ω_d onto domain Ω by $u = M_d^T u_d$. The averaging operator W_d is diagonal and has dimension $N_d \times N_d$. It contains ones on the main diagonal when the corresponding degree of freedom lies only in the subdomain Ω_d. It contains 1 over the number of subdomains it is contained in when multiple subdomains are involved.

4.1.2. *Parallel matrix-vector product.* We define the global matrix-vector product as $Ku = v$ where K and u have dimension $N \times N$ and $N \times 1$ respectively. The parallel matrix-vector product yields the same result v but is computed on the subdomains separately by computing $\{K^{\Omega_1}u^{\Omega_1}, ..., K^{\Omega_D}u^{\Omega_D}\}$ and combining $\{v^{\Omega_1}, ..., v^{\Omega_D}\}$ where K^{Ω_i} and u^{Ω_i} have dimension $N^d \times N^d$ and $N^d \times 1$ at subdomain Ω_d respectively. We have $v = \sum_{d=1}^{D} M_d^T v_d$. We emphasize that in this formulation the entries of the shared degrees of freedom in the vectors u_d should be identical for each domain it is defined on.

4.1.3. *Parallel dot product.* We compute the global dot product as $\lambda = u^T u$. The parallel dot product yields the same result λ but is computed on the subdomains separately by computing $\lambda_d = u_d^T W_d u_d$ where W_d and u_d and have dimension $N^d \times N^d$ and $N^d \times 1$ at subdomain Ω_d respectively. We have $\lambda = \sum_{d=1}^{D} \lambda_d$. We emphasize that in this formulation the entries of the shared degrees of freedom in the vectors u_d should be identical for each domain it is defined on.

4.2. **MUMPS: parallel direct solver.** The parallel direct solution method of choice is MUMPS, a parallel sparse direct solver [5]. The solver is based on a multifrontal approach and we refer to [1] for any details on the theoretical background. The solver is implemented and available as a open source code and can be easily embedded within existing FE codes based on parallelization by subdomains. The solver can be used within a shared memory as well as distributed memory cluster architecture.

4.3. **Parallel implementation PCG.** The algorithm of PCG can be found in [8]. We observe that the method is constructed from basic linear algebraic operations. As described in previous Section only the matrix-vector operation and inner product require communication. All other linear algebraic operations can be done locally, i.e. there is no communication with other subdomains. This makes the PCG method easy to parallelize. The other operation that needs to be taken care of explicitly is the preconditioner. In this research we consider diagonal scaling and Incomplete Cholesky decomposition. We note that diagonal scaling is in fact a matrix vector operation. The Incomplete Cholesky decomposition of the stiffness matrix is only computed locally on each subdomain, although losing global accuracy but avoiding communication with other subdomains. We note that the stiffness matrix on a subdomain can be singular, this can be avoided when the values of the main diagonals of the local stiffness matrices equal the values of the global stiffness matrix. This will ensure non-singularity and robustness of the Incomplete Choleskey decomposition. In our implementation we have used ILUPACK [3] for computing the Incomplete Cholesky decomposition on each subdomain.

4.4. **Parallel implementation DPCG.** The DPCG method given by Algorithm 1 is almost similar to the standard PCG algorithm, but the parallelization of the DPCG method involves two steps. First the construction of the deflation matrix Z on each subdomain and second the evaluation of PKx for each iteration of DPCG.

By using domain decomposition we do not store any vector or matrix globally, hence, we do not assemble the global deflation matrix Z. However, because materials may lie in multiple subdomains all the local deflation matrices Z must have the same number of columns, i.e. deflation vectors.

For each domain the values of entries of the deflation vectors that belong to subdomain boundary nodes are communicated to neighboring subdomains. We only store the non-zero elements of local Z, hence this approach will have a small memory overhead.

The evaluation of PKx can be optimized. Consider,

$$PKx = Kx - ZKE^{-1}Z^TKx$$

where $K \in \mathbb{R}^{n\times n}$, $Z \in \mathbb{R}^{n\times k}$. Here $Kx = y$ is computed as usual $ZK = \tilde{Z} \in \mathbb{R}^{n\times k}$ and $E^{-1} = (Z^TKZ)^{-1}$ are computed only once, before entering the Krylov process (iteration loop). Hence, for each iteration of DPCG we have three extra operations compared to PCG,

$$\begin{aligned} Z^Ty &= \tilde{y}, \quad \tilde{y} \in \mathbb{R}^{k\times 1} \\ E^{-1}\tilde{y} &= \hat{y}, \quad \hat{y} \in \mathbb{R}^{k\times 1} \\ \tilde{Z}\hat{y} &= \bar{y}, \quad \bar{y} \in \mathbb{R}^{n\times 1}. \end{aligned}$$

Communication between subdomains is needed for the computation of KZ, E and Z^T. The coarse matrix E is equal on each subdomain with dimension $k \times k$ and its inverse is determined on each subdomain simultaneously. On iteration level only Z^T involves a parallel communication at the cost of k parallel inner products of $k\times 1$ sized vectors. The weakness of the (parallel) DPCG method lies within the evaluation of KZ which is in fact k parallel matrix vector multiplications. Considering that matrix vector multiplications are the most time consuming part of the PCG algorithm we prefer to have k relatively small.

5. Numerical experiments

The case given in Figure 1 is a FE mesh of a real life sample of asphaltic material obtained from CT scan. Both experiments in this section concern the analysis of the same sample of material but different mesh sizes; 2,9 and 4,9 million degrees of freedom respectively. We compare MUMPS, DPCG and PCG in combination with Incomplete Cholesky with drop tolerance 10^{-2} and diagonal scaling. The case involves a mixture of materials that is subjected to an external force applied to the upper boundary of the volume. Zero displacement boundary conditions are imposed on three sides of the volume, this is homogenous Dirichlet boundary conditions to all degrees of freedom in the x, z-, x, y- and y, z- planes for $y = 0$, $z = 0$ and $x = 0$ respectively. These materials give rise to coupled partial differential equations [6]. The experiments make use of the same set of material parameters. We distinguish between three materials: aggregates, bitumen, and air voids. The corresponding stiffness coefficients (E modulus) are given in Table 1 and are the dominating contributions to the entries of the stiffness matrix. We have implemented MUMPS, PCG and DPCG into the existing parallel FE software package CAPA-3D [4]. All experiments were done on a cluster of Dell workstations containing 8 CPUs Intel Xeon E5450, running at 3.00GHz and connected by Infiniband.

Table 1. E modulus for different materials

aggregate	bitumen	air voids
69000	5000	100

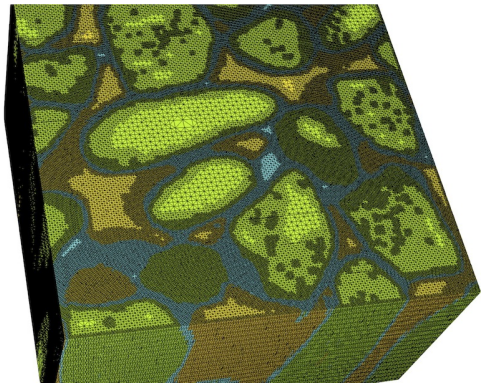


FIGURE 1. FE mesh that consists of 7,977,448 elements representing cube of asphaltic material containing aggregates (light green), bitumen (dark green) and air voids (blue).

5.1. **Experiment 1.** The results of the computation of the solution for equation (1) are given in Table 2. The deflation space of the DPCG method consists of the rigid body modes of the bodies corresponding to the three different materials. Moreover, we have also appended the deflation space with the rigid body modes of the subdomains to increase stability for the Incomplete Cholesky preconditioning. Therefore we have 438 and 342 deflation vectors for the DPCG method with and without the subdomain rigid body modes respectively. We consider two tolerances TOL=10^{-2}, 10^{-6} due to utilization of the iterative solver. In many engineering applications the solution to equation (1) does not need to be accurate because it is computed when using the Newton Method for solving non-linear equations.

The wall clock time for MUMPS has been highlighted as it is the fastest solution method for solving with higher accuracy. Comparing the iterative solution methods in terms of wall clock time, DPCG in combination with the Incomplete Choleskey preconditioning without subdomain deflation is the fastest solution method for tolerance TOL=10^{-2}. and therefore printed in italics. We also observe from the table what can be expected of the preconditioners and the deflation in terms of number of iterations. This is also illustrated by the plots of the convergence of PCG in Figure 3 and DPCG for diagonal scaling and Incomplete Cholesky preconditioning in Figures 4(a) and 4(b) respectively. The convergence curve of the PCG method shows clear plateaus due to the unfavorable eigenvalues of the stiffness matrix related to the discontinuous coefficients for both preconditioners. The convergence curve of the DPCG method contains no plateaus as the unfavorable eigenvalues have been removed from the spectrum of the projected stiffness matrix. The difference in effectiveness of the preconditioners is clearly visible as the reduction of iterations using Incomplete Cholesky preconditioning instead of diagonal scaling is roughly a factor of 3 for both the PCG and the DPCG method. The addition of the subdomain deflation vectors has no real (positive) influence on the convergence of DPCG.

In Figure 2 we have provided the wall clock time for the different operations of the PCG and DPCG method. Also the wall clock time of MUMPS is plotted by the dashed line. We compare the operations for each of the solution methods. What stands out is the trade-off between the quality of the preconditioner and its wall clock time. The work for the diagonal scaling is negligible compared to the other operations, but yields much more iterations. The Incomplete Cholesky preconditioning

TABLE 2. Experiment 1: 2,976,627 DOF, wall clock time MUMPS, PCG and DPCG for different preconditioners and TOL=10^{-2},10^{-6} and 10^{-13}

		PCG		DPCG (342)		DPCG (438)	
	MUMPS	diag	ilu	diag	ilu	diag	ilu
TOL=10^{-2}							
iterations	-	4105	1481	1431	424	1214	383
cpu(s)	-	130	173	104	*86*	112	93
TOL=10^{-6}							
iterations	-	7855	2283	4106	1283	3428	1160
cpu(s)	-	253	263	253	203	257	207
TOL=10^{-13}							
cpu(s)	**182**	-	-	-	-	-	-

tends to be roughly 3 times as expensive as the evaluation of the matrix-vector products but reduces the number of iterations with the same factor. The deflation operation Px is as expensive as the matrix-vector operation. However for higher accuracy, hence yielding more iterations, the deflation method needs as much time for the computation of the invariant matrices KZ and E as for the computation of Px. From Figure 2 it is difficult to judge the effect of adding extra deflation vectors but we know from the convergence curve that the influence on the performance of DPCG in terms of iterations is negligible.

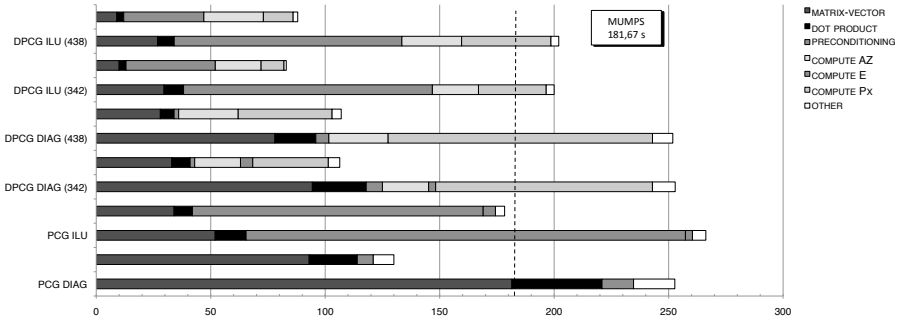


FIGURE 2. Experiment 1: Wall clock time for different stages of PCG and DPCG for TOL=10^{-2},10^{-6}.

5.2. **Experiment 2.** The results of the computation of the solution for equation (1) are given in Table 3. The deflation space of the DPCG method consists of the rigid body modes of the bodies corresponding to the three different materials. Moreover, we have also appended the deflation space with the rigid body modes of the subdomains to increase stability for the Incomplete Cholesky preconditioning. Therefore we have 1168 and 1068 deflation vectors for the DPCG method with and without the subdomain rigid body modes respectively. We again consider two tolerances TOL=10^{-2}, 10^{-6}.

The wall clock time for the PCG method in combination with diagonal scaling has been highlighted as it is the fastest solution method for both tolerances. The MUMPS solver is significantly

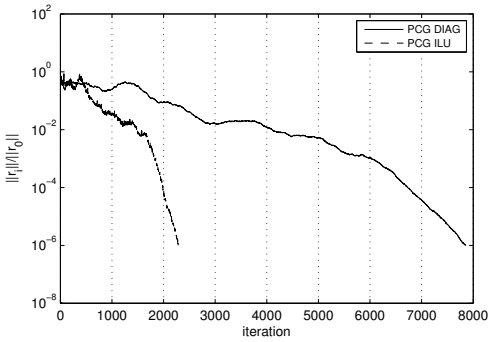
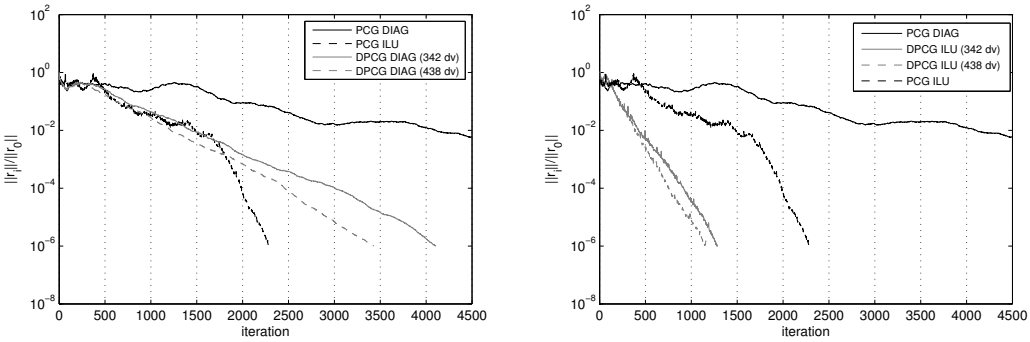


FIGURE 3. Experiment 1: convergence curve of the PCG method for diagonal scaling and Incomplete Cholesky preconditioners



(a) (b)

FIGURE 4. Experiment 1: convergence curve of the DPCG method for diagonal scaling (a) and Incomplete Cholesky (b) preconditioners compared to the PCG method (diagonal scaling and IC)

slower. Comparing the iterative solution methods in terms of wall clock time, DPCG in combination with the Incomplete Choleskey preconditioning without subdomain deflation is almost as fast as the PCG method with diagonal scaling and therefore the preferable solution methods due to the accuracy of the PCG method in general. Again we observe from the table what can be expected of the preconditioners and the deflation in terms of number of iterations. This is also illustrated by the plots of the convergence of PCG in Figure 7 and DPCG for diagonal scaling and Incomplete Cholesky preconditioning in Figures 8(a) and 8(b) respectively. In this case the addition of the subdomain deflation vectors has also no real (positive) influence on the convergence of DPCG.

In Figure 5 and 6 we have provided the wall clock time and memory occupation for the different operations of MUMPS and the PCG and DPCG method. The wall clock time of MUMPS is plotted by the dashed line. We can validate our conclusions made in the previous experiment regarding the performance PCG and the DPCG method. However, although the number of dof of experiment 2 is almost 1,5 times bigger than experiment 1 and the number of deflation vectors has increased by a

TABLE 3. Experiment 2: 4,991,679 DOF, wall clock time MUMPS, PCG and DPCG for different preconditioners and TOL=10^{-2},10^{-6} and 10^{-13}

		PCG		DPCG (1068)		DPCG (1164)	
	MUMPS	diag	ilu	diag	ilu	diag	ilu
TOL=10^{-2}							
iterations	-	4033	1512	1286	467	1182	444
cpu(s)	-	**259**	352	267	*262*	300	287
TOL=10^{-6}							
iterations	-	8070	2951	3910	1438	3516	1372
cpu(s)	-	**513**	660	547	*527*	598	556
TOL=10^{-13}							
cpu(s)	589	-	-	-	-	-	-

factor of 3 the ratios between the deflation operations and the standard PCG operations are almost equal for both experiments. This means that the deflation method scales well under increasing mesh size and the number of deflation vectors. Clearly MUMPS utilizes significantly more memory for the computation of the decomposition of the stiffness matrix when compared to amount of memory needed for the storage of Z, E and AZ. In this case the ratio is $1:11$ in favor of the DPCG method. We observe a difference in memory occupation for DPCG (1068) and DPCG(1164) which use 1068 and 1164 deflation vectors respectively. Clearly, adding more deflation vectors to the deflation subspace does increase the occupation of memory and more than one would expect by the increase of the number of deflation vectors. However, the added deflation vectors are related to sub domain deflation and hence densely populated deflation vectors which require significantly more memory than the sparse aggregate deflation vectors.

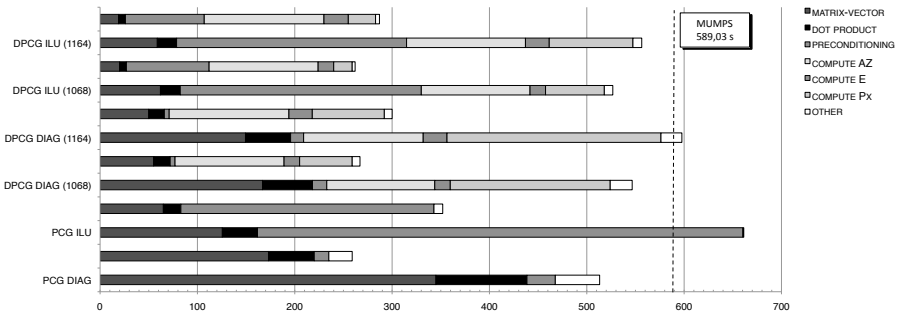


FIGURE 5. Experiment 2: Wall time for different stages of PCG and DPCG for TOL=10^{-2},10^{-6}

6. CONCLUSION

We compared a parallel direct solver, the Preconditioned Conjugate Gradient method and the Deflated Preconditioned Conjugate Gradient method for the solution of large linear systems from mechanical problems with strongly varying stiffness of materials. The DPCG method is favorable for large systems as it outperforms the direct solver for larger tolerances in time. Also the DPCG

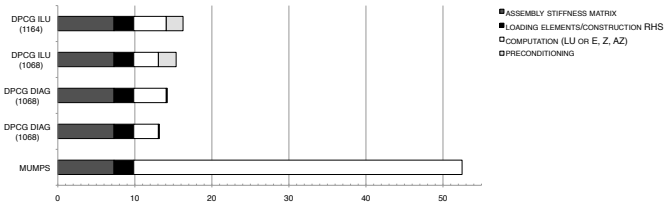


FIGURE 6. Experiment 2: memory occupation in GigaBytes for different stages of MUMPS, PCG and DPCG.

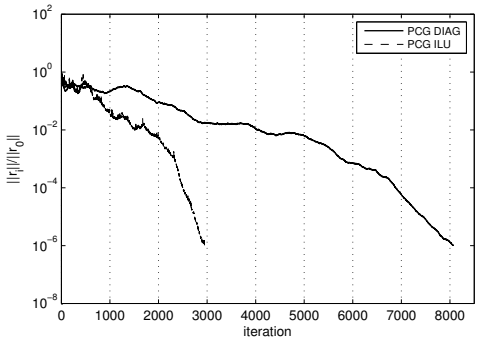
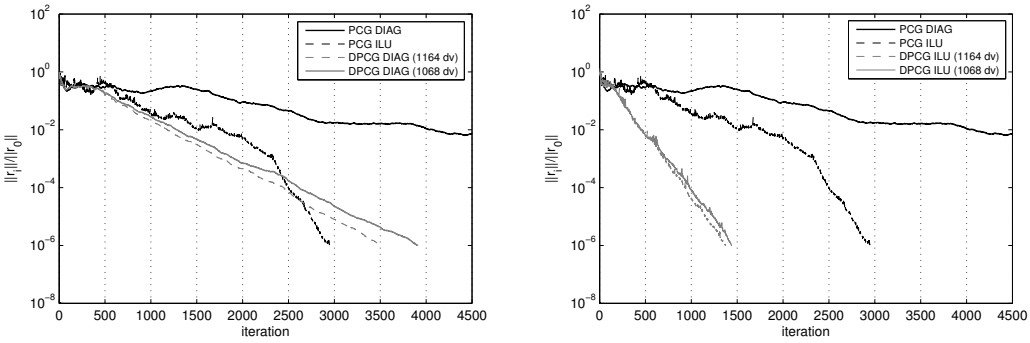


FIGURE 7. Experiment 2: convergence curve of the PCG method for diagonal scaling and Incomplete Cholesky preconditioners



(a) (b)

FIGURE 8. Experiment 2: convergence curve of the DPCG method for diagonal scaling (a) and Incomplete Cholesky (b) preconditioners compared to the PCG method (diagonal scaling and IC)

method has a relatively low and predictable occupation of memory compared to the direct solver. Moreover, the DPCG method is well suited for parallel computing and can be implemented into any existing FE software package by using basic parallel linear algebraic operations.

Appendix A. Computing rigid body modes of a finite element

We know from [2] that the rigid body modes of a finite element are spanned by the kernel base vectors of the corresponding element stiffness matrix. We will show a fast and cheap solution for the computation of the rigid body modes. The same principle can be easily extended to sets of finite elements of arbitrary shape and order. We note that the rigid body modes are only defined by the geometric properties of the element.

In three dimensions a finite element has 6 rigid body motions; three translations and three rotations. For simplicity we consider a 4 noded tetrahedral element, however all derivations can be extended to N noded elements without loss of generality. The coordinate vector of the element is given by,

$$\left\{ \begin{array}{cccccccccccc} x_1 & y_1 & z_1 & x_2 & y_2 & z_2 & x_3 & y_3 & z_3 & x_4 & y_4 & z_4 \end{array} \right\}^T$$

A translation can be considered as a uniform displacement of every node in a given direction. To obtain three orthogonal translations we choose the x,y and z direction respectively. The three translation vectors are given by,

$$\left\{ \begin{array}{cccccccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{array} \right\}^T$$
$$\left\{ \begin{array}{cccccccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array} \right\}^T$$
$$\left\{ \begin{array}{cccccccccccc} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right\}^T$$

The rotations can be easily described using the spherical coordinate system,

$$x = r\cos(\theta)\sin(\phi), \quad y = r\sin(\theta)\sin(\phi), \quad z = r\cos(\phi)$$

where

$$r = \sqrt{x^2 + y^2 + z^2}, \quad \theta = \tan^{-1}\left(\frac{y}{x}\right), \quad \phi = \cos^{-1}\left(\frac{x}{r}\right)$$

and θ and ϕ as in Figure 9(a).

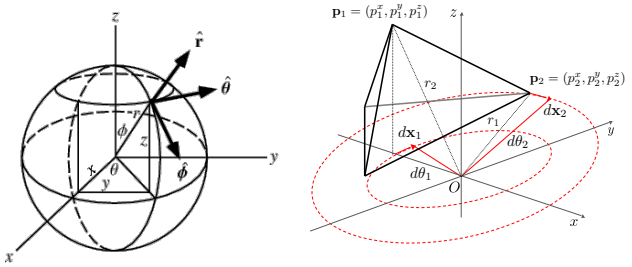
We derive a rotation $d\theta$ in the x, y-plane, hence $d\phi = 0$ and $dr = 0$. The x-y, x-z and y-z planes contain unique rotations. The corresponding vectors can be found by swapping axis. For an arbitrary point in space which has spherical coordinates (r, θ, ϕ) a change $d\theta$ in the x, y-plane yields a displacement in cartesian coordinates of,

$$dx = -r\sin(\theta)\sin(\phi)d\theta, \quad dy = r\cos(\theta)\sin(\phi)d\theta, \quad dz = 0.$$

Figure 9(b) shows the rotation for one element with respect to the origin over angle $d\theta$. By using above expressions we obtain all three rotation vectors,

rotation x-y plane,

$$\theta_j = \tan^{-1}\left(\frac{y_j}{x_j}\right), \quad \phi_j = \cos^{-1}\left(\frac{z_j}{r_j}\right), \quad \left\{ \begin{array}{c} -r_1\sin(\theta_1)\sin(\phi_1) \\ r_1\cos(\theta_1)\sin(\phi_1) \\ 0 \\ -r_2\sin(\theta_2)\sin(\phi_2) \\ r_2\cos(\theta_2)\sin(\phi_2) \\ 0 \\ -r_3\sin(\theta_3)\sin(\phi_3) \\ r_3\cos(\theta_3)\sin(\phi_3) \\ 0 \\ -r_4\sin(\theta_4)\sin(\phi_4) \\ r_4\cos(\theta_4)\sin(\phi_4) \\ 0 \end{array} \right\}$$



(a) (b)

FIGURE 9. (a) spherical coordinates, (b) rotation around origin of tetrahedral element in x, y-plane

rotation y-z plane,

$$\theta_j = \tan^{-1}\left(\frac{z_j}{x_j}\right), \quad \phi_j = \cos^{-1}\left(\frac{y_j}{r_j}\right), \quad \begin{Bmatrix} -r_1\sin(\theta_1)\sin(\phi_1) \\ 0 \\ r_1\cos(\theta_1)\sin(\phi_1) \\ -r_2\sin(\theta_2)\sin(\phi_2) \\ 0 \\ r_2\cos(\theta_2)\sin(\phi_2) \\ -r_3\sin(\theta_3)\sin(\phi_3) \\ 0 \\ r_3\cos(\theta_3)\sin(\phi_3) \\ -r_4\sin(\theta_4)\sin(\phi_4) \\ 0 \\ r_4\cos(\theta_4)\sin(\phi_4) \end{Bmatrix}$$

rotation x-z plane,

$$\theta_j = \tan^{-1}\left(\frac{z_j}{y_j}\right), \quad \phi_j = \cos^{-1}\left(\frac{x_j}{r_j}\right), \quad \begin{Bmatrix} 0 \\ r_1\cos(\theta_1)\sin(\phi_1) \\ -r_1\sin(\theta_1)\sin(\phi_1) \\ 0 \\ r_2\cos(\theta_2)\sin(\phi_2) \\ -r_2\sin(\theta_2)\sin(\phi_2) \\ 0 \\ r_3\cos(\theta_3)\sin(\phi_3) \\ -r_3\sin(\theta_3)\sin(\phi_3) \\ 0 \\ r_4\cos(\theta_4)\sin(\phi_4) \\ -r_4\sin(\theta_4)\sin(\phi_4) \end{Bmatrix}$$

We compute the null space of each element matrix. Sets of elements make up the bodies of materials, as a collection of elements share a certain property and are neighbors. The rigid body modes of a collection of elements is equal to the assembly of the rigid body modes of the individual elements taking into account the multiplicity of those degrees of freedom that lie in multiple neighboring elements. In the case of asphaltic materials we choose the element stiffness as the property for discrimination between elements. We can think of stones, bitumen and air voids. We should note that we compute the rigid body modes of each independent body of material. Hence, two bodies of the same material imply 12 deflation vectors. This has a physical meaning also, two bodies will not rotate and translate at the same time and at the same rate. Therefore these movements need to be taken care of independently.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers, 1998.

[2] K. J. Bathe. *Finite Element Procedures*. Prentice Hall, 2 revised edition, June 1995.

[3] Matthias Bollhöfer and Yousef Saad. Multilevel preconditioners constructed from inverse-based ilus. *SIAM J. Sci. Comput.*, 27(5):1627–1650, 2006.
[4] CAPA3D. Capa-3d computer aided pavement analysis. http://www.capa-3d.org, 2009.
[5] CERFACS. Mumps: a parallel sparse direct solver. http://graal.ens-lyon.fr/mumps/, 2010.
[6] Andrew Drescher, Niki Kringos, and Tom Scarpas. On the behavior of a parallel elasto-visco-plastic model for asphaltic materials. *Mechanics of Materials*, October 2009.
[7] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM J. Sci. Comput.*, 23(2):442–462, 2001.
[8] G. H. Golub and C. F. Van Loan. *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, Baltimore, October 1996.
[9] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, Dec 1952.
[10] T.B. Jönsthövel, M.B. van Gijzen, C.Vuik, C. Kasbergen, and A. Scarpas. Preconditioned conjugate gradient method enhanced by deflation of rigid body modes applied to composite materials. *Computer Modeling in Engineering and Sciences*, 47:97–118, 2009.
[11] E. F. Kaasschieter. Preconditioned conjugate gradients for solving singular systems. *J. Comput. Appl. Math.*, 24(1-2):265–275, 1988.
[12] Xiaoyes. Li and James W. Demmel. Superlu dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29:110–140, 2003.
[13] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM J. Numer. Anal.*, 24(2):355–365, 1987.
[14] Eric Polizzi and Ahmed H. Sameh. A parallel hybrid banded system solver: the spike algorithm abstract, 2005.
[15] Y. Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Mathematics, Philadelphia, April 2003.
[16] Olaf Schenk, Klaus Gärtner, Wolfgang Fichtner, and Andreas Stricker. Pardiso: a high-performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Generation Computer Systems*, 18(1):69–78, 2001.
[17] Simpleware. http://www.simpleware.com, 2009.
[18] J.M. Tang, R. Nabben, C. Vuik, and Y.A. Erlangga. Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods. *Journal of Scientific Computing*, 39:340–370, 2009.
[19] A. Van der Sluis and H.A. Van der Vorst. The rate of convergence of conjugate gradients. *Numer. Math.*, 48(5):543–560, 1986.