Comput Geosci (2013) 17:723–738
DOI 10.1007/s10596-013-9352-4

ORIGINAL PAPER

GPU-accelerated iterative solutions for finite element analysis of soil–structure interaction problems

Xi Chen · Yuxin Jie · Yuzhen Yu

Received: 21 May 2012 / Accepted: 16 April 2013 / Published online: 15 May 2013

Abstract Soil–structure interaction problems are commonly encountered in engineering practice, and the resulting linear systems of equations are difficult to solve due to the significant material stiffness contrast. In this study, a novel partitioned block preconditioner in conjunction with the Krylov subspace iterative method symmetric quasi-minimal residual is proposed to solve such linear equations. The performance of these investigated preconditioners is evaluated and compared on both the CPU architecture and the hybrid CPU–graphics processing units (GPU) computing environment. On the hybrid CPU–GPU computing platform, the capability of GPU in parallel implementation and high-intensity floating point operations is exploited to accelerate the iterative solutions, and particular attention is paid to the matrix–vector multiplications involved in the iterative process. Based on a pile-group foundation example and a tunneling example, numerical results show that the partitioned block preconditioners investigated are very efficient for the soil–structure interaction problems. However, their comparative performances may apparently depend on the computer architecture. When the CPU computer architecture is used, the novel partitioned block symmetric successive over-relaxation preconditioner appears to be the most efficient, but when the hybrid CPU–GPU computer architecture is adopted, it is shown that the inexact block diagonal preconditioners embedded with simple diagonal approximation to the soil block outperform the others.

Keywords Soil–structure interaction · Iterative solution · Partitioned block preconditioner · GPU acceleration · Hybrid CPU–GPU architecture · Matrix–vector multiplication

X. Chen (✉)
Department of Geotechnical and Geoenvironmental Engineering, School of Civil Engineering, Beijing Jiaotong University, Beijing, 100044, China
e-mail: xichen.geo@gmail.com

Y. Jie · Y. Yu (✉)
State Key Laboratory of Hydroscience and Engineering, Tsinghua University, Beijing, 100084, China
e-mail: yuyuzhen@tsinghua.edu.cn

Y. Jie
e-mail: jieyx@tsinghua.edu.cn

1 Introduction

Over the past few decades, more and more scientific and engineering problems have been modeled in 3D space in order to accurately capture the 3D effects, which may be neglected by 2D numerical simulations (e.g., [1]). With the increase of 3D numerical modeling, the demands on high-performance computing grow rapidly in various fields. The impact of high-performance computing on geosciences and geotechnical engineering is also impressive, and, by using the parallel architecture, the previously unimaginable computational work may be accomplished in a short period of time. For example, to simulate the time-dependent ground settlement using the Biot's consolidation theory, the resulting total degrees of freedom (DOFs) is as high as 1.64495 million if the solution domain is discretized into the $50 \times 50 \times 50$ finite element mesh. Solving such a linear system of equation by the generalized Jacobi (GJ) [2] preconditioned symmetric quasi-minimal residual (SQMR) method [3] may require 1.28 h given that an ordinary desktop computer (equipped with an i7−920 quad core processor and 6G physical memory) is used. Consequently, solving a large nonlinear consolidation problem, which may produce

hundreds of or even more such linear equations, may overwhelm the patience and the efforts. Moreover, given the 1 m × 1 m × 1 m cube as the minimum allowable mesh size, the 50 × 50 × 50 mesh discretization can only simulate a domain of 125,000 m^3, which may be still far from the practical geological scale.

To accomplish a large-scale finite element computation in an acceptable period of time, one may follow two paths: utilizing more powerful computer resources such as high-performance computer architecture or adopting robust numerical algorithms such as fast linear solvers. Along with the former path, the distributed parallel architecture has attracted much attention (e.g., [4]). For example, the element-by-element (EBE) preconditioned conjugate gradient (PCG) method was employed by Liu et al. [5] to solve the dam ground system and the underground cavity problem, while the domain decomposition method (DDM) was used by Zhao et al. [6] for the large slope problems in millions of DOFs. Ferronato et al. [7] and Janna et al. [8] extended their constraint block preconditioners from the sequential computers to the parallel environment to suit for the ill-conditioned geomechanical models with the size in kilometers, and attention was paid to the approximate inverse-based preconditioners. The distributed parallel computing architecture has great potentials in large-scale computations, but its high cost in construction and maintenance may limit its wide use. In the past decade, it has been witnessed that the graphics processing units (GPU) technology has greatly changed the complexion of parallel computing due to its programmable streams, powerful capability in floating point operations (FLOPs) and high price/performance ratio (e.g., [9]). For instance, GPU has been applied to expedite numerical computations, such as 2D/3D Monte Carlo simulations (e.g., [10]), neurosurgical simulation (e.g., [11]), 3D incompressible Navier–Stokes equations (e.g., [12]), and others (e.g., [13–16]). Some researchers expressed their positive views on the applications of GPUs. Nickolls and Dally [17] thought that "Today GPU computing enables applications that we previously thought infeasible because of long execution times". Papadrakakis et al. [18] even believed that the hybrid CPU–GPU architectures are opening a new era for scientific computing and especially studied the DDM for the hybrid computing platform. In the finite element (FE) realm, GPU is increasingly being used as a major computing engine for the linear solutions (e.g., [19]) or the assembly of finite elements (e.g., [20]). As a result, it is expected that with the advent of GPU computing, more FE computations may resort to the GPUs.

Compared to the adoption of high-performance computer architecture, the evolution in fundamental algorithms is another driving force. For large 3D geotechnical finite element analyses, it seems that iterative methods may offer the only way towards a solution (e.g., [21]). Mroueh and Shahrour [22] employed the sparse Krylov subspace iterative methods in conjunction with symmetric successive over-relaxation (SSOR) preconditioning to analyze the pile-group foundation, and furthermore, they applied the methods to the tunneling simulations (e.g., [23]). However, the soil–structure interaction problems are generally difficult to solve due to the significant material stiffness contrast, and the available preconditioned iterative methods may converge slowly or even fail (e.g., [24–27]). Recently, an advance was made by Chaudhary et al. [28] who proposed the inexact block diagonal preconditioners by partitioning the solid stiffness matrix in terms of the structure elements and soil elements. Later, the partition strategy was extended to the coupled consolidation problem (e.g., [29]). It is interesting to notice that, to some extent, the above partition strategy is similar to that used by Gambolati et al. [30], in which the unknown displacements associated with the linear FE nodes are numbered first, and those associated with the nonlinear FE nodes, second. According to their numerical results, the partition strategies appear to be appealing for the soil–structure interaction problems.

There has been an increasing demand for large 3D finite element simulations of soil–structure interaction problems, but the iterative solution technologies are still far from reaching a high degree of maturity. The paper is intended to assist practitioners to greatly expedite the iterative solutions of soil–structure interaction problems in virtue of two different computer architectures. Hence, it is organized as follows: Section 2 introduces the partition strategy and the resulting finite element formulation for soil–structure interaction problems. Section 3 firstly presents the recently proposed inexact block diagonal preconditioners, which have been illustrated effective for soil–structure interactions. To further exploit the capacity of partitioned block strategy, a novel partitioned block SSOR preconditioner is proposed. In Section 4, a GPU device is utilized to accelerate the iterative solutions, and attention is devoted to the involved matrix–vector multiplications. Based on two numerical examples in Section 5, the computer execution time on the hybrid CPU–GPU architecture is compared with that on the pure CPU architecture, and the speedup is evaluated. Finally, a summary and some concluding remarks are given in Section 6.

2 Finite element formulation for soil–structure interactions

The time-dependent settlement of a soil may be modeled by the general consolidation theory which was firstly proposed by Biot [31]. In the Biot's theory, soil is assumed as the fully saturated porous media, and the

deformation of soil skeleton is solely related to the effective stress. Hence, the equilibrium equation is given as follows:

$$\nabla \cdot \left[\boldsymbol{\sigma}'(\mathbf{x},t) + p(\mathbf{x},t)\,\mathbf{1}\right] - \gamma \mathbf{b} = 0,\ (\mathbf{x},\ t) \in \Omega \times (0,T]. \tag{1}$$

Here, $\mathbf{1} = [1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0]^T$, $\mathbf{b} = [0,0,1]^T$ is the unit body force vector; γ is the total unit weight; Ω is the solution space domain, and T is the total time domain. Based on the assumption of incompressible soil particles and pore fluid, the mass conservation equation (that is, the inflow is equal to the outflow for a fully saturated soil media) leads to

$$\nabla \cdot \dot{\mathbf{u}}(\mathbf{x},t) + \nabla \cdot \mathbf{v}_f(\mathbf{x},t) = 0 \tag{2}$$

in which $\nabla \cdot \dot{\mathbf{u}}$ represents the volumetric strain (i.e., ε_v), and $\mathbf{v}_f$ is the fluid velocity governed by Darcy's law. After discretizing the equilibrium equation and the mass conservation equation in space domain and applying finite difference in time domain, the linear equation with the 2×2 matrix form is derived (e.g., [32]):

$$\begin{bmatrix} K & B \\ B^T & -H \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} f \\ g \end{Bmatrix} \text{ with } A = \begin{bmatrix} K & B \\ B^T & -H \end{bmatrix}_{N \times N} \tag{3}$$

where the global matrix A is symmetric indefinite; $K \in \Re^{(m+n)\times(m+n)}$ is symmetric positive definite (SPD) and corresponds to the solid stiffness matrix; $H \in \Re^{l \times l}$ is the symmetric positive semi-definite matrix for fluid stiffness which corresponds to the fluid stiffness matrix. As a result, the block K is governed by the solid constitutive laws, while the block C depends on the time increment Δt. Many factors, such as the time step, size of the spatial domain, and choice of physical units, may influence the ill conditions of stiffness matrix. In addition, the significant material stiffness contrast and various material constitutive laws may also contribute a lot to the ill conditions. Based on the above observations, Gambolati et al. [30] partitioned the solid stiffness according to the linear constitutive laws and the nonlinear ones, while Chaudhary et al. [28, 29] proposed to partition the solid stiffness according to the structural elements and the soil elements. Here, the latter partition strategy is followed, so that Eq. (3) may be expressed as follows:

$$\begin{bmatrix} P & C & B_1 \\ C^T & G & B_2 \\ B_1^T & B_2^T & -H \end{bmatrix} \begin{Bmatrix} x_P \\ x_G \\ y \end{Bmatrix} = \begin{Bmatrix} f_P \\ f_G \\ g \end{Bmatrix} \text{ with } A = \begin{bmatrix} P_{m\times m} & C & B_1 \\ C^T & G_{n\times n} & B_2 \\ B_1^T & B_2^T & -H_{l\times l} \end{bmatrix}_{N \times N} \tag{4}$$

in which $P \in \Re^{m\times m}$ and $G \in \Re^{n\times n}$ are SPD matrices and correspond to the structure stiffness matrix and the soil stiffness matrix, respectively; C is the soil-structure coupling matrix; B_1 and B_2 are the structure–fluid coupling matrix and the soil–fluid coupling matrix, respectively; x_P and x_G represent the structural and soil displacement, respectively, while y corresponds to the excess pore pressure. It is noteworthy that different partition scheme may suit for different applications. For instance, Gambolati et al. [30] and Ferronato et al. [7] proposed a multi-level partition scheme according to the material constitutive laws. Generally, they tended to partition the solid block matrix according to the invariant part and the changing part, so that the factorizations of the invariant part may be repeatedly utilized during the overall simulations. The idea conforms to that of Chaudhary et al. [28, 29] to some extent. However, the latter partition strategy obviously appears to be more appealing for those geotechnical applications involving soil–structure interactions.

For ease of presentation, the matrix A in Eq. (4) is reformulated as follows:

$$A = \begin{bmatrix} P & E \\ E^T & W \end{bmatrix}. \tag{5}$$

Here, $W = \left[G\ B_2;\ B_2^T - H\right]$ for the consolidation analysis, while it reduces to G for the drained analysis. Hence, for a general symmetric (probably indefinite) linear system of equation, the SQMR solver is employed, although the PCG method is often the first choice for the SPD linear system. In the past decades, the success received by the Krylov subspace iterative methods may be largely attributed to the adoption of preconditioning techniques, accounting for the considerable efforts devoted to the research in robust preconditioners. Nevertheless, available preconditioners may not suit well for the large realistic geotechnical problems, especially when strong soil–structure interactions are involved.

3 Partitioned block preconditioners

For the linear system of Eq. (4) stemming from the soil–structure interactions, the preconditioners proposed here, termed as the partitioned block preconditioners, include the inexact block diagonal (signified by the subscript "IBD") preconditioners (e.g., [28, 29])

$$M_{\text{IBD}}\left(\hat{W}\right) = \begin{bmatrix} P & 0 \\ 0 & \hat{W} \end{bmatrix} \text{ or } M_{\text{IBD}}\left(\hat{G}\right) = \begin{bmatrix} P & 0 \\ 0 & \hat{G} \end{bmatrix} \tag{6}$$

for consolidation analysis and drained analysis, respectively, and in Eq. (6), $\hat{W}$ and $\hat{G}$ are an approximation to the

matrix W and G, respectively. For realistic applications, the approximation is recommended as follows:

$$\hat{W}_1 = \mathrm{GJ}(W) \text{ or } \hat{W}_2 = \mathrm{MSSOR}(W) \quad \text{for consolidation analysis} \tag{7}$$

$$\hat{G}_1 = \mathrm{diag}(G) \text{ or } \hat{G}_2 = \mathrm{SSOR}(G) \quad \text{for drained analysis} \tag{8}$$

Here, GJ(·) and MSSOR(·) represents the generalized Jacobi diagonal (e.g., [2]) and the modified SSOR factorization (e.g., [33]) of an indefinite matrix, that is

$$\mathrm{GJ}(W) = \begin{bmatrix} \mathrm{diag}(G) & 0 \\ 0 & \alpha\,\mathrm{diag}\left(\hat{S}_H\right) \end{bmatrix} \tag{9}$$

$$\mathrm{MSSOR}(W) = \left(L_W + \hat{D}_W\right)\hat{D}_W^{-1}\left(U_W + \hat{D}_W\right) \tag{10}$$

in which α is a scalar, and the approximated Schur complement matrix is defined as $\hat{S}_H = H + B_1^T diag(P)^{-1} B_1 + B_2^T diag(G)^{-1} B_2$. L_W and U_W are the strictly lower and upper parts of W, respectively, and $\hat{D}_W = \mathrm{GJ}(W)$. As mentioned in the previous study (e.g., [34]), the ILU(0) preconditioner with adequate stabilization may be faster than the MSSOR preconditioner; nevertheless, the ILU(0) preconditioner may fail in some cases, so that it is not recommended for large solid–fluid coupling analysis. Hence, the better ILU(0) approximation to W is not investigated. Additionally, in practical finite element analyses, the size of (structure) block P is usually small compared to the size of (soil) block G, that is $m \ll n$, implying that a better approximation to P and a cheap approximation to G could be more feasible. It has been demonstrated that the exact block P plays an essential role for the success of the inexact block diagonal preconditioners (e.g., [28, 29]). When applying the inexact block diagonal preconditioners, the preconditioning step is simply implemented as follows:

Preconditioning step $q = M_{\mathrm{IBD}}^{-1} d$:

$$\begin{aligned} &\text{Given } q = [q_1;\ q_2] \text{ and } d = [d_1;\ d_2]; \\ &\text{Compute } q = M_{\mathrm{IBD}}^{-1} d = \left[q_1 = P^{-1} d_1;\right. \\ &\qquad \left. q_2 = \hat{W}^{-1} d_2 \left(\text{or } q_2 = \hat{G}^{-1} d_2\right)\right]. \end{aligned} \tag{11}$$

The advantage of embedding the exact block P in the partitioned block preconditioners may be more evident when the structural members are modeled in linear elasticity. In that case, the exact factorization of block P can be repeatedly used in the nonlinear finite element analyses until it is updated (e.g., [35]), for example, when some available structural members are removed or some additional structural members are added. While for the block G dominated by soil models, a cheap approximation should be more promising. For the spectral analysis of the block diagonal

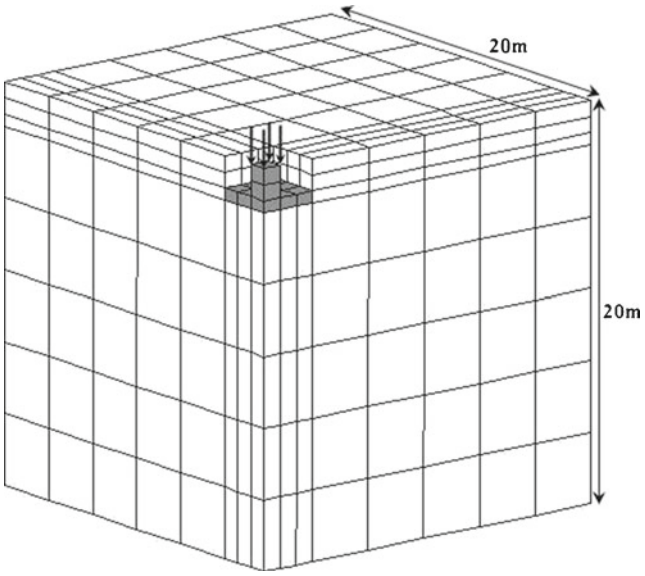

Fig. 1 An 8 × 8 × 8 finite element mesh of an extended pile foundation

preconditioned matrices, the 2 × 2 block matrix in Eq. (3) and the 3 × 3 block matrix in Eq. (4) refer to Toh et al. [36] and Chaudhary et al. [29], respectively.

The strategy of partitioning the solid block into structure part and soil part is appealing, but it has not been exploited to the full capability. Instead of using the diagonal block, a novel block SSOR form is proposed here:

$$M_{\mathrm{PBSSOR}}\left(\hat{W}\right) = \begin{bmatrix} P & 0 \\ E^T & \hat{W} \end{bmatrix}\begin{bmatrix} P & 0 \\ 0 & \hat{W} \end{bmatrix}^{-1}\begin{bmatrix} P & E \\ 0 & \hat{W} \end{bmatrix} \quad \text{for consolidation analysis.} \tag{12}$$

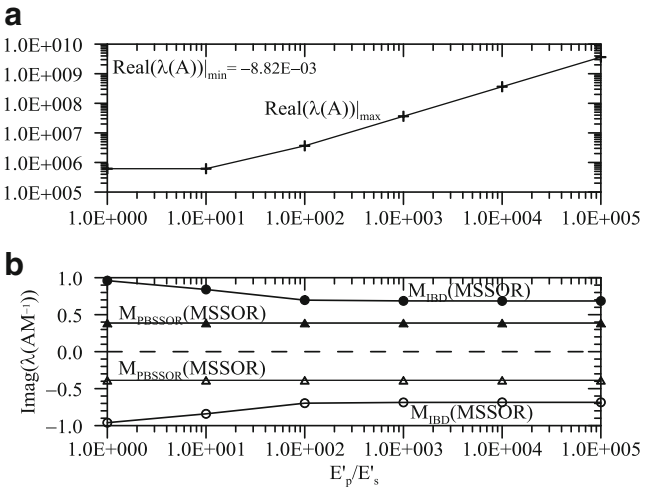

Fig. 2 Spectral property of matrix A and preconditioned matrices AM^{-1} varied with the stiffness ratio E'_p/E'_s **a** $\mathrm{real}(A)|_{max}$ versus E'_p/E'_s and **b** $\mathrm{Imag}(\lambda(AM^{-1}))$ versus E'_p/E'_s

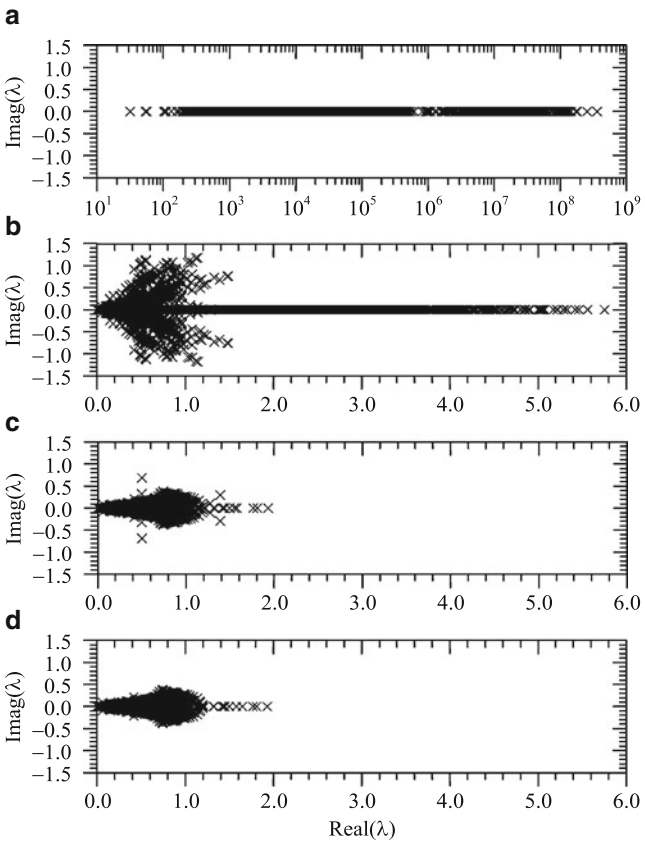

Fig. 3 Eigenvalue distributions in complex plane for $E'_p/E'_s = 1.E+4$ **a** A, **b** AM^{-1}_{IBD}(GJ) or $AM^{-1}_{\mathrm{PBSSOR}}$(GJ), **c** AM^{-1}_{IBD}(MSSOR), **d** $AM^{-1}_{\mathrm{PBSSOR}}$(MSSOR)

For the block SSOR preconditioner in drained analysis, $\hat{W}$ is replaced by $\hat{G}$ in Eq. (12). Obviously, in Eq. (12), M_{PBSSOR} may be rewritten as $\left[P\ E;\ E^T\ \hat{W} + E^T P^{-1} E\right]$. $\hat{W}$ is an approximation to the Schur complement matrix $S_W = W - E^T P^{-1} E$, but the simple approximation $\hat{W} \approx W$ may be employed due to $\left\| E^T P^{-1} E \right\| \ll \|W\|$, so that $M_{PBSSOR} \approx A$ and the computation/construction associated with $E^T P^{-1} E$ may be obviated. Depending on the analysis type, the approximation $\hat{W}$ is defined according to Eqs. (7) and (8), respectively. When applying the M_{PBSSOR} preconditioners, the preconditioning step is implemented as follows:

Preconditioning step $q = M^{-1}_{\mathrm{PBSSOR}} d$:

$$\begin{aligned}
&\text{Given } q = [q_1;\ q_2] \text{ and } d = [d_1;\ d_2];\\
&\text{Compute } h_1 = P^{-1} d_1;\\
&\text{Compute } g_2 = d_2 - E^T h_1;\\
&\text{Compute } q_2 = \hat{W}^{-1} g_2;\\
&\text{Compute } q = M^{-1}_{\mathrm{PBSSOR}} r = \left[q_1 = P^{-1}(d_1 - E q_2);\quad q_2\right].
\end{aligned} \tag{13}$$

Compared to the preconditioning step of M_{IBD} preconditioners, the partitioned block SSOR preconditioners involve one additional P solver and multiplications with off-diagonal parts but without imposing penalty in the memory usage. In the SQMR iterative solver, the right preconditioning format is adopted with the right preconditioned matrix denoted as follows:

$$AM^{-1}_{\mathrm{PBSSOR}} = \begin{bmatrix} I_{m\times m} & 0 \\ \left(I - S_W \hat{W}^{-1}\right) E^T P^{-1} & S_W \hat{W}^{-1} \end{bmatrix} \quad \text{for consolidation analysis} \tag{14}$$

$$AM^{-1}_{\mathrm{PBSSOR}} = \begin{bmatrix} I_{m\times m} & 0 \\ \left(I - S_G \hat{G}^{-1}\right) C^T P^{-1} & S_G \hat{G}^{-1} \end{bmatrix} \quad \text{for drained analysis.} \tag{15}$$

It is apparent that the preconditioned matrix $AM^{-1}_{\mathrm{PBSSOR}}$ has 1 as an eigenvalue with multiplicity m, and the remaining n or $n + l$ eigenvalues are determined by $S_W \hat{W}^{-1} = \left(W - E^T P^{-1} E\right) \hat{W}^{-1} \approx W \hat{W}^{-1}$ or $S_G \hat{G}^{-1} = \left(G - C^T P^{-1} C\right) \hat{G}^{-1} \approx G \hat{G}^{-1}$. Compared to the theoretical analysis of eigenvalues, numerical illustrations could be more impressive. Figure 1 gives the $8 \times 8 \times 8$ finite element mesh of an extended pile foundation example in the consolidation analysis. Figure 2 shows the eigenvalue distribution of matrix A and preconditioned matrices AM^{-1} varied with the stiffness ratio $E'_{\mathrm{p}}/E'_{\mathrm{s}}$ (in which E'_{s} is kept constant as 3 MPa). As we see, the eigenvalues of matrix A are real—most of the eigenvalues are positive

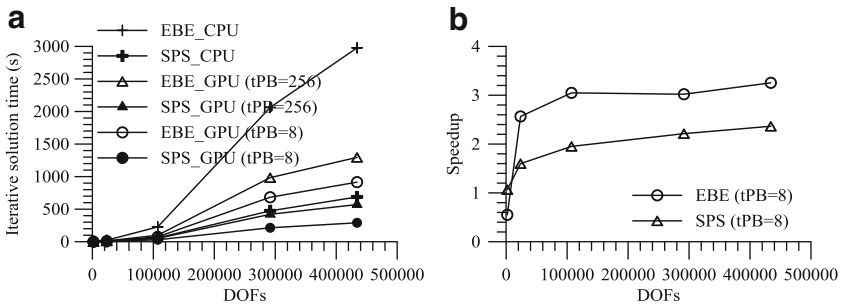

Fig. 4 Performances of GJ preconditioned SQMR iterative solutions of 3D Biot's consolidation problems versus DOFs with and without GPU acceleration, respectively **a** iterative solution time versus DOFs and **b** acceleration ratio versus DOFs

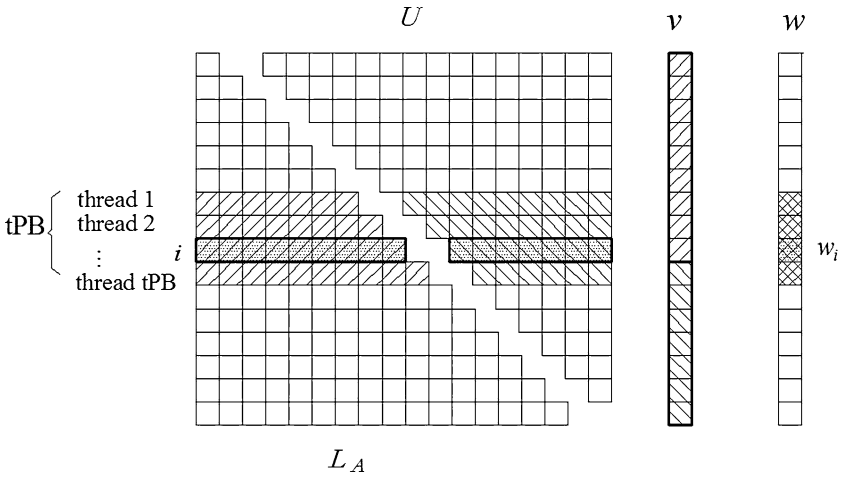

Fig. 5 Schematic representation of the matrix–vector multiplication $w = Av$ in GPU device

and the others are negative. With the stiffness ratio E'_p/E'_s, the smallest eigenvalue remains constant as $-8.82\text{E}-03$, but the largest eigenvalue $\text{real}(A)|_{max}$ increases rapidly as shown in Fig. 2a. The eigenvalues of AM^{-1}_{IBD}(GJ) and AM^{-1}_{PBSSOR} (GJ) are quite similar and remain insensitive to the stiffness ratio. The maximum and minimum real values of the two preconditioned matrices are 5.75 and $2.88\text{E}-03$, while the maximum and minimum imaginary part is 1.18 and -1.18, respectively. As we noticed in Fig. 2b when assessing the eigenvalue distributions of AM^{-1}_{IBD} (MSSOR) and AM^{-1}_{PBSSOR} (MSSOR), the partitioned block SSOR preconditioners shrink the eigenvalues towards the real axis better than the inexact block diagonal preconditioners, implying that the partitioned block SSOR preconditioners may result in faster convergence rates for an iterative solver. However, it is noteworthy that the real part of the eigenvalues has no difference between the two preconditioners, and the maximum (or minimum) real value which is also insensitive to the stiffness ratio is 1.93 (or $2.51\text{E}-03$). To further illustrate the spectral property, the eigenvalue distributions of matrix A and those preconditioned matrices corresponding to $E'_p/E'_s = 1.\text{E}+4$ are plotted in the complex plane as shown in Fig. 3.

4 GPU-accelerated sparse matrix operations

The finite element computation generally consists of six steps: (1) finite element meshing, (2) nonlinear finite element analysis, (3) assembly of finite elements, (4) solution of finite element linear equation, (5) system equilibrium

Table 1 Refined finite element meshes and matrices of the pile-group foundation for the consolidation analysis ("c") and the drained analysis ("d")

FE mesh	Ratio of structure elements	$N = m + n + l$ (c) $N = m + n$ (d)	Nnz
$8 \times 8 \times 8$	25/512	7,159 = 683 + 5,829 + 647 (c)	567,185 (c)
		6,512 = 683 + 5,829 (d)	452,090 (d)
$12 \times 12 \times 12$	73/1,728	23,603 = 1,916 + 19,660 + 2,027 (c)	2,031,253 (c)
		21,576 = 1,916 + 19,660 (d)	1,624,126 (d)
$16 \times 16 \times 16$	296/4,096	55,280 = 4,341 + 46,315 + 4,624 (c)	4,956,037 (c)
		50,656 = 4,341 + 46,315 (d)	3,969,026 (d)
$20 \times 20 \times 20$	448/8,000	107,179 = 8,421 + 89,939 + 8,819 (c)	9,846,741 (c)
		98,360 = 8,421 + 89,939 (d)	7,893,446 (d)
$24 \times 24 \times 24$	520/13,824	184,296 = 9,545 + 159,751 + 15,000 (c)	17,208,965 (c)
		169,296 = 9,545 + 159,751 (d)	13,804,042 (d)
$28 \times 28 \times 28$	672/21,952	291,619 = 12,245 + 255,827 + 23,547 (c)	27,547,797 (c)
		268,072 = 12,245 + 255,827 (d)	22,107,470 (d)
$32 \times 32 \times 32$	840/32,768	434,143 = 15,209 + 384,087 + 34,847 (c)	41,368,773 (c)
		399,296 = 15,209 + 384,087 (d)	33,210,386 (d)

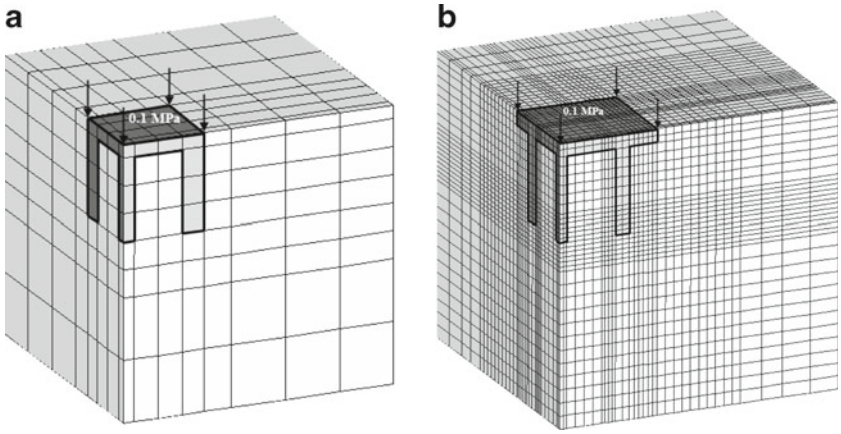

Fig. 6 Finite element mesh of the pile-group examples **a** $8 \times 8 \times 8$ mesh and **b** $32 \times 32 \times 32$ mesh

check, and (6) visualization of output results. As noticed in the recent literatures, apart from the visualization of output results and the assembly of finite elements (e.g., [20]), much attention has been focused on the acceleration of linear solvers (e.g., [37]). Compared to the direct linear solvers, the Krylov subspace iterative methods obviously possess the advantage to combine with GPU technology. For example, the matrix–vector multiplications required in the construction of Krylov subspace may be executed by the GPU device. For the symmetric iterative methods such as PCG and SQMR, there is only one matrix–vector product within each iteration, as shown in the construction of k-th Krylov subspace of $\Re^n$ generated by A with respect to r_0:

$$\mathcal{K}_k(A,\ r_0) = \text{span}\left\{r_0,\ Ar_0,\ \ldots,\ A^{k-1}r_0\right\},\ k = 1, 2, \ldots \tag{16}$$

For the nonsymmetric iterative methods such as GMRES [38], QMR [39], and BiCGSTAB [40], two matrix–vector products are often involved in each iteration. Apart from the matrix–vector multiplication in SQMR solver, the matrix–vector products involved in the preconditioning procedure of M_{PBSSOR} (i.e., Eq. (13)) may also resort to the GPU acceleration.

Since the matrix storage scheme plays a central role in the efficiency of iterative solutions, especially when GPU is used to accelerate the iterative process, it is of practical value to investigate some commonly used matrix storage schemes for GPU-accelerated iterative methods. Nowadays, the most widely applied finite element matrix storages are the EBE matrix storage scheme (e.g., [41, 42]) and the compressed matrix storage scheme (e.g., [43]). The EBE matrix storage scheme was proposed to remove the requirement for assembling global matrix, and it has been demonstrated that the EBE matrix storage is not only attractive for the parallel architecture (e.g., [5]) but also suitable for the ordinary desktop computers (e.g., [2]). The disadvantage of some assembled storage schemes (such as the banded or skyline storage scheme) is that they store some unnecessary zeros for computations. To squeeze out the zeros, the compressed sparse storage schemes, including the coordinate storage, the compressed sparse column (CSC) storage, and the compressed sparse row (CSR) storage, were developed. To choose an appropriate matrix storage scheme, however, one has to consider the following factors: (1) computer architecture (e.g., serial or parallel), (2) available physical memory (e.g., limited or unlimited), and (3) involved FLOPs (i.e., huge or not).

Figure 4 illustrates the performance of GJ-preconditioned SQMR solver for the homogenous consolidation problems, in which the DOFs vary from 1,820 to 434,144. In this

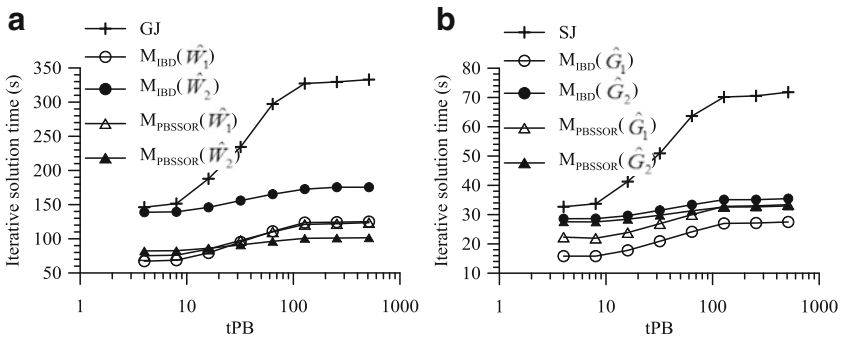

Fig. 7 Iterative solution time varied with tPB based on $24 \times 24 \times 24$ mesh **a** consolidation analysis and **b** drained analysis

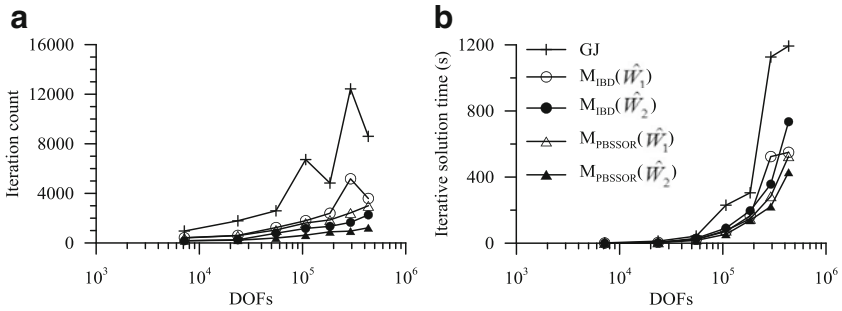

Fig. 8 Iteration count and corresponding iterative solution time versus DOFs for the consolidation analysis based on CPU platform

example, two matrix storage schemes are evaluated based on two different computer architectures. When applying the EBE storage scheme on the GPU environment, each elemental matrix–vector product is conducted by one thread within the thread block. The preliminary study shows that with the aid of GPU device, the SQMR iterative process may be accelerated, and the acceleration effect (or the speedup which is defined as the ratio between the iterative solution time obtained on the CPU platform to that on the hybrid CPU–GPU platform) is varied depending on the employed matrix storage scheme. In Fig. 4, "SPS" denotes the compressed sparse storage, "CPU" or "GPU" after the underscore indicates the adopted computing architecture, and "tPB" represents the number of threads per block which is taken as 256 and 8, respectively. For large-sized finite element problems, the achieved speedup is around 3.0 and 2.0, respectively, for the EBE storage scheme and the sparse storage scheme. The more evident acceleration effect achieved by the EBE storage scheme may be attributed to more FLOPs involved in the elemental level matrix–vector products, reflecting that the GPU device has the great potential in high-intensity computations. Although the GPU acceleration is more remarkable for the EBE storage scheme given that a small tPB is used, the least solution time is attained by the compressed sparse storage. Hence, in the following study, only the compressed sparse storages (i.e., CSR storage and CSC storage) are employed and evaluated based on the serial architecture (i.e., CPU platform) and the parallel architecture (i.e., GPU platform), respectively. Since the stiffness matrix A is symmetric, the upper triangular part of A (i.e., U_A) is stored into the CSC format. The adjacency structure of the full block P has to be constructed in order that the sparse Cholesky subroutine in SparseM package may be applied [44]. On the CPU computer architecture, the matrix–vector multiplication $w = Av$ is conducted in terms of Algorithm 1 in the Appendix. When performing the matrix–vector product in GPU device, the CSR format of U_A is also provided apart from the CSC storage of U_A, as the CSR format is more suitable for GPU parallel threads, and the CSC format of U_A may be used as the CSR format of the lower triangular part of A (i.e., L_A). Before accessing the GPU device, both the CSC storage and the CSR storage of U_A are copied into the GPU (or video) memory. Consequently, in GPU device, the sparse matrix–vector multiplication Av is decomposed into two separate parts, that is

$$w = Av = \big[(L + D) + U\big]v = w_L + w_U \tag{17}$$

in which $A = L + D + U$ with D, L, and U as the diagonal, strictly lower, and upper triangular part of A, respectively. For each triangular matrix–vector multiplication, each row is treated by a single thread of GPU. Once w_L and w_U are both computed, they are summed into w. For the implementation details, see Algorithm 2 in the Appendix. In Algorithm 2, each entry in the resulting vector w_L or w_U is computed by a thread, while the threads per block conduct the dot products concurrently as shown in Fig. 5. For the GTX 580 graphics card, the wrap size is 32, and

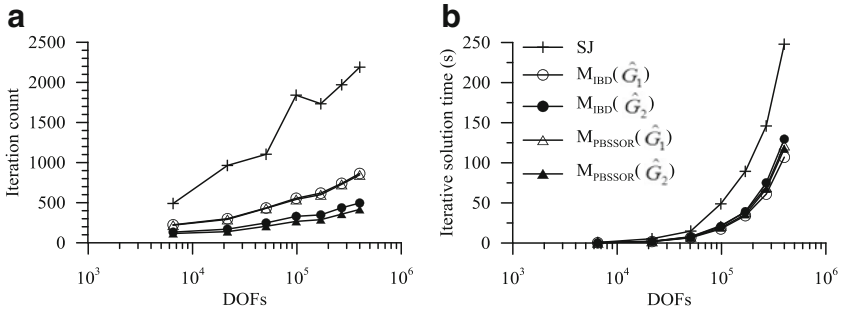

Fig. 9 Iteration count and corresponding iterative solution time versus DOFs for the drained analysis based on CPU platform

Table 2 Consolidation analysis for the pile-group example based on CPU–GUP platform

Mesh size	GJ	$M_{\text{IBD}}\left(\hat{W}_1\right)$	$M_{\text{IBD}}\left(\hat{W}_2\right)$	$M_{\text{PBSSOR}}\left(\hat{W}_1\right)$	$M_{\text{PBSSOR}}\left(\hat{W}_2\right)$
$8 \times 8 \times 8$	973 (1.00 s)	480 (0.59 s)	181 (0.64 s)	410 (0.97 s)	167 (0.75 s)
$12 \times 12 \times 12$	2,336 (6.60 s)	633 (2.25 s)	265 (3.06 s)	583 (3.17 s)	220 (3.01 s)
$16 \times 16 \times 16$	2,631 (17.6 s)	1,290 (11.5 s)	739 (20.8 s)	1,060 (13.9 s)	406 (13.3 s)
$20 \times 20 \times 20$	5,516 (70.7 s)	1,829 (34.3 s)	1,185 (68.5 s)	1,593 (45.4 s)	615 (41.4 s)
$24 \times 24 \times 24$	6,473 (151.2 s)	2,362 (71.8 s)	1,345 (138.3 s)	1,806 (76.2 s)	754 (87.5 s)
$28 \times 28 \times 28$	6,332 (230.4 s)	2,718 (127.8 s)	1,718 (274.6 s)	2,406 (154.8 s)	951 (170.6 s)
$32 \times 32 \times 32$	17,176 (942.4 s)	3,624 (251.4 s)	2,181 (524.8 s)	2,917 (271.7 s)	1, 252 (333.2 s)

each streaming multiprocessor can handle 16 (i.e., 512/32) wraps. Since the stiffness matrix A is generated according to the partitioned block ordering of Eq. (5), the off-diagonal part $\left(\text{i.e., } E \text{ and } E^T\right)$ of block matrix A may be readily retrieved so that the matrix–vector multiplications involving E and E^T (as shown by Eq. (13)) can also be implemented in the GPU device.

5 Performance benchmark

In the following soil–structure interaction examples, both soil and concrete materials are modeled by 20-node hexahedral consolidation element or 20-node hexahedral solid element depending on the analysis type. The 20-node consolidation element is the 20-node solid element coupled with 8-node fluid element, and hence, each consolidation element comprises 60 displacement DOFs (3 spatial DOFs per node for 8 corner nodes and 12 mid-side nodes) and 8 excess pore pressure DOFs (1 DOF per node for 8 corner nodes). Thus, the total number of displacement DOFs is much larger than that of excess pore pressure, and the ratio, $(m+n)/l$, is often larger than 10. In addition, the ratio, m/n, is small because in soil–structure interaction problems, the number of structure elements is usually much smaller than that of the soil elements. In the FE model, the groundwater table is assumed to be at the ground surface and in hydrostatic condition at the initial stage. The base of the mesh is constrained in all directions, side face boundaries are fixed in the transverse direction, but free in in-plane directions. The top surface is free in all directions and is assumed as free-draining surface. For the same FE mesh, both the consolidation analysis and the drained analysis are carried out.

To terminate the iterative process of SQMR solver, the relative residual convergence criterion is applied (e.g., [45]). Accordingly, the maximum allowable iteration count and the tolerance is defined as $\text{maxit} = 50,000$ and $\text{tol} = 1.e-6$, respectively. The present program is coded in double precision with CUDA FORTRAN (e.g., [46]), which is based on the heterogeneous programming model. Hence, the host code runs on the CPU, and the device code runs on the GPU. Host code is used to handle the data management for both the host and device as well as to launch kernels which are subroutines executed on GPU, and the device code is executed by many threads in parallel. For the performance tests, a computer equipped with an Intel i7−920 2.67 GHz CPU, three 2 GB (i.e., 6 GB) DDR5 physical memories and a Geforce GTX 580 graphics card is used. The GTX 580 GPU has 512 CUDA cores and 1,536 MB GDDR5 video memory. For GTX 580 GPU, the maximum number of threads per block (i.e., tPB) is 512, the peak single-precision performance is about 1,580 GFLOPs, while its peak double-precision performance may be only around 210 GFLOPs.

Table 3 Drained analysis for the pile-group example based on CPU–GUP platform

Mesh size	SJ	$M_{\text{IBD}}\left(\hat{G}_1\right)$	$M_{\text{IBD}}\left(\hat{G}_2\right)$	$M_{\text{PBSSOR}}\left(\hat{G}_1\right)$	$M_{\text{PBSSOR}}\left(\hat{G}_2\right)$
$8 \times 8 \times 8$	490 (0.47 s)	224 (0.30 s)	134 (0.39 s)	219 (0.50 s)	116 (0.48 s)
$12 \times 12 \times 12$	964 (2.34 s)	300 (0.94 s)	167 (1.59 s)	294 (1.44 s)	139 (1.59 s)
$16 \times 16 \times 16$	1,105 (6.2 s)	434 (3.3 s)	243 (5.6 s)	428 (4.9 s)	206 (5.6 s)
$20 \times 20 \times 20$	1,840 (19.5 s)	552 (8.9 s)	326 (15.3 s)	538 (13.5 s)	267 (14.9 s)
$24 \times 24 \times 24$	1,734 (33.7 s)	622 (15.8 s)	348 (28.9 s)	601 (21.9 s)	288 (27.1 s)
$28 \times 28 \times 28$	1,971 (59.7 s)	741 (29.0 s)	431 (55.9 s)	730 (40.7 s)	357 (53.1 s)
$32 \times 32 \times 32$	2,189 (99.6 s)	864 (49.6 s)	497 (95.7 s)	844 (68.6 s)	416 (92.8 s)

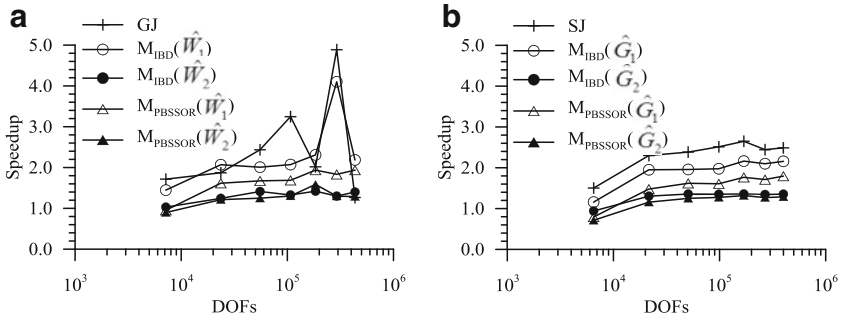

Fig. 10 GPU speedup versus DOFs **a** consolidation analysis and **b** drained analysis

5.1 Pile-group foundation

Based on a pile-group foundation example, the performances of the partitioned block preconditioners are assessed on the CPU platform and the CPU–GPU platform, respectively. Taking advantage of symmetry, only the quadrant is modeled, and the mesh size varies from $8 \times 8 \times 8$ to $32 \times 32 \times 32$. Table 1 gives the details on the finite element meshes, including DOFs, the number of nonzero entries (i.e., Nnz), and the ratio of structure elements. Figure 6 shows the coarsest $8 \times 8 \times 8$ and the finest $32 \times 32 \times 32$ finite element meshes of the pile-group example, in which both soil and structural concrete are modeled in linear elasticity. The ground is assumed to be homogeneous with the effective Young's modulus $E'_s = 1$ MPa, the Poisson's ratio $\nu'_s = 0.3$, and the hydraulic conductivity coefficient $k_s = 1.e-7$ m/s, while for the concrete material of pile-group foundation, $\nu'_p = 0.2$ and $k_p = 1.e-12$ m/s, but E'_p varies in a wide range. The 0.1-MPa uniform pressure is applied at the first time step with $\Delta t = 1$ s. Since tPB is essential for the efficiency of matrix–vector multiplications, its impact on the performance of preconditioned SQMR solver is also examined. Based on the $24 \times 24 \times 24$ mesh with the material stiffness contrast ratio kept as $E'_p/E'_s = 3e4$, Fig. 7a–b illustrates the iterative solution time versus tPB (which varies from 4 to 512) for consolidation analysis and drained analysis, respectively. It is observed that the impact of tPB on each preconditioner is quite different, as it has the most remarkable influence on the simple diagonal preconditioner (that is, GJ or standard Jacobi (SJ)), and the impact of tPB on those preconditioners embedded with simple diagonal approximation to W or G (such as $\hat{W}_1$ or $\hat{G}_1$) is also pronounced. No matter which preconditioner is employed, however, the iterative solution time required by SQMR solver may decrease with tPB, but with the required iteration counts kept unchanged, implying that opening up more threads may probably influence the implementing efficiency. That phenomenon on tPB may also be interpreted in Fig. 5, that is, a large tPB indicates a big difference in the length between the first row and the tPB-th row, so that the other threads have to wait more or less idly for the completion of the thread associated with the longest row. Since a small tPB is often preferred from the perspective of solution time, $tPB = 8$ is adopted in the following GPU-accelerated computations.

Figures 8 and 9 illustrate the iteration count and iterative solution time varying with finite element mesh refining based on the CPU computer architecture. Figure 8 corresponds to the consolidation analysis, while Fig. 9 corresponds to the drained analysis. From the two figures, it can be seen that, compared to the other preconditioners, the iteration count required by GJ (or SJ) preconditioned SQMR solver increases rapidly, leading to the longest iterative solution time. For the consolidation analysis, $M_{\mathrm{PBSSOR}}(\hat{W}_2)$ is the most robust preconditioner, while for the drained analysis $M_{\mathrm{IBD}}(\hat{G}_1)$ is competitive with $M_{\mathrm{PBSSOR}}(\hat{G}_2)$, even though $M_{\mathrm{PBSSOR}}(\hat{G}_2)$ still results in the least iteration count.

The corresponding results obtained from the CPU–GPU platform are provided in Tables 2 and 3 for the consolidation analysis and the drained analysis, respectively. For certain mesh size and a given preconditioning method, the

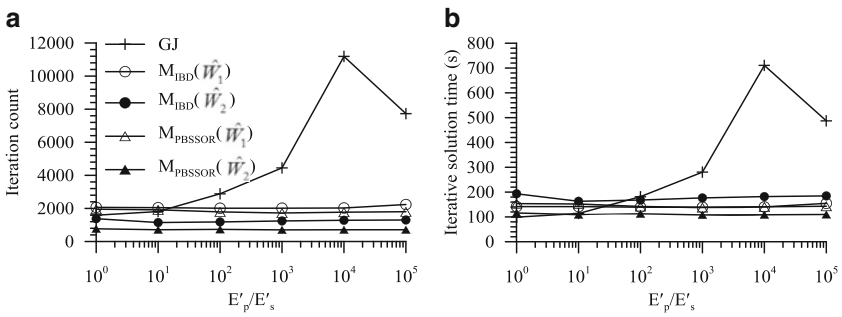

Fig. 11 Iteration count and solution time versus E'_p/E'_s based on CPU platform for the $24 \times 24 \times 24$ mesh of consolidation analysis

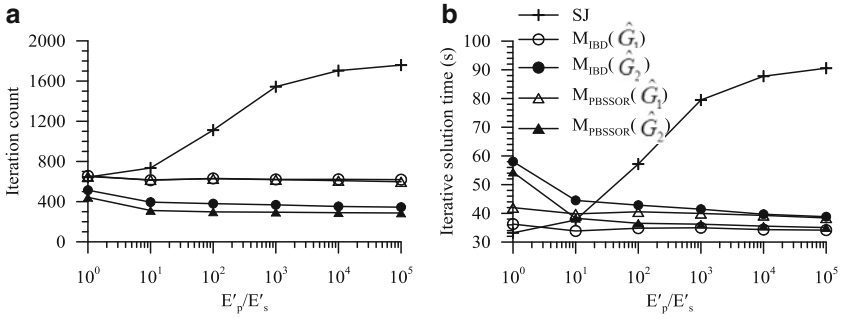

Fig. 12 Iteration count and solution time versus E'_p/E'_s based on CPU platform for the $24 \times 24 \times 24$ mesh of drained analysis

iteration count of SQMR solver and the corresponding solution time (as shown in bracket) are tabulated. Similar to the results on CPU platform, the iteration count for GJ- or SJ-preconditioned SQMR method increases rapidly with the mesh refining, while $M_{\mathrm{PBSSOR}}(\hat{W}_2)$ or $M_{\mathrm{PBSSOR}}(\hat{G}_2)$ preconditioned SQMR has a comparatively mild increase rate of iteration count. As opposed to the results on CPU platform, however, the least iterative solution time is attained by $M_{\mathrm{IBD}}(\hat{W}_1)$ or $M_{\mathrm{IBD}}(\hat{G}_1)$ preconditioned SQMR methods. For the largest finite element mesh and even compared to the $M_{\mathrm{PBSSOR}}(\hat{W}_2)$ and $M_{\mathrm{PBSSOR}}(\hat{G}_2)$ preconditioners, $M_{\mathrm{IBD}}(\hat{W}_1)$ and $M_{\mathrm{IBD}}(\hat{G}_1)$ may lead to a reduction of 24.5 and 46.6 %, respectively, in the iterative solution time. A further observation discloses that the partitioned block preconditioners embedded with simple diagonal approximation to W may outperform the preconditioners embedded with MSSOR (or SSOR) approximation to W, indicating that simple diagonal preconditioner is more suitable for parallel computing. The conclusion may be validated by observing the speedup. Figure 10 shows the GPU speedup versus mesh refining for the consolidation analysis and the drained analysis, respectively. It is obvious that with the mesh refining the GPU speedup increases but stabilizes at different point of DOFs for different preconditioning method. For example, the stabilized point for the partitioned block preconditioners embedded with MSSOR (or SSOR) approximation to W is slightly over 10^4, and the ratio is around 1.3∼1.5 for the consolidation analysis and 1.27∼1.35 for the drained analysis, respectively. In the investigated preconditioners, the diagonal preconditioner (i.e., GJ or SJ) may lead to the most remarkable speedup, although the performance of GJ may fluctuate for the consolidation analysis, while for the drained analysis, the speedup is comparatively regular with the mesh refining for all preconditioners. The irregularity in speedup for GJ preconditioner may be interpreted as follows. For the $28 \times 28 \times 28$ mesh (in which DOFs $= 291,619$) of the consolidation analysis, the iteration count required by GJ-preconditioned SQMR solver on CPU platform is 12,433, while on the hybrid CPU–GPU platform, the iteration count is 6,332—the obvious difference in the required iteration count results in a large speedup (i.e., 4.89). On the contrary, for the $32 \times 32 \times 32$ mesh (in which DOFs $= 434,143$), the iteration count of SQMR solver on CPU platform is 8,597, while the iteration count on the hybrid CPU–GPU platform is 17,176, and hence a small speedup (i.e., 1.27) is observed. In all studied cases in evaluating the speedup, the above two cases may be regarded as the special cases.

The performance of each preconditioner versus E'_p/E'_s is examined on the CPU platform. Figures 11 and 12 illustrate the trend in the iteration count and corresponding solution time for the consolidation analysis and the drained analysis, respectively. The results show that the performance of GJ may deteriorate quickly with E'_p/E'_s increasing, but the performances of partitioned block preconditioners are insensitive to the ratio of $E'_{\mathrm{p}}/E'_{\mathrm{s}}$. The performance of each preconditioner varying with $E'_{\mathrm{p}}/E'_{\mathrm{s}}$ is also assessed on the hybrid CPU–GPU platform. Figures 13 and 14 show their performances with varied $E'_{\mathrm{p}}/E'_{\mathrm{s}}$. Although the iteration count of GJ preconditioned SQMR may increase

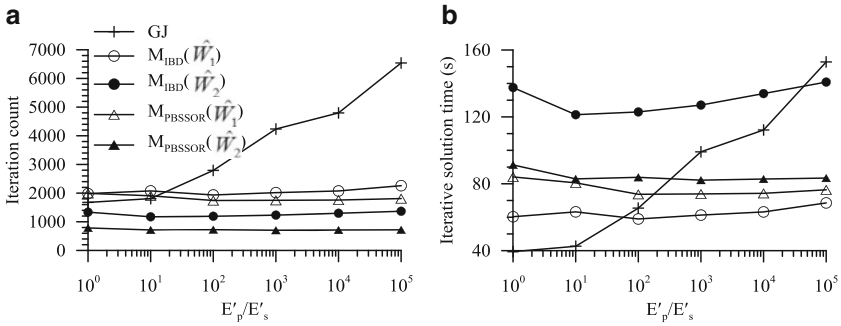

Fig. 13 Iteration count and solution time versus E'_p/E'_s based on the $24 \times 24 \times 24$ mesh and CPU–GPU platform for consolidation analysis

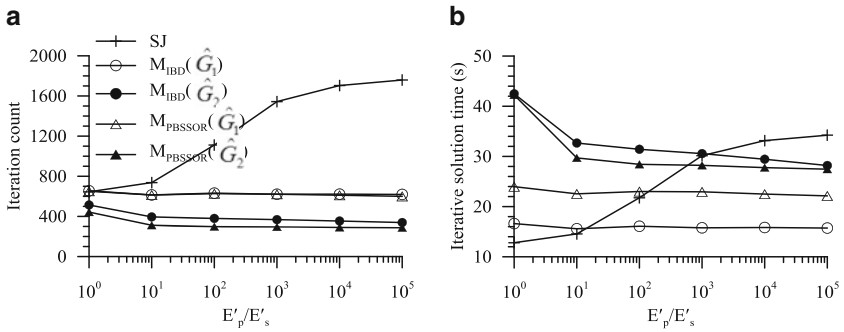

Fig. 14 Iteration count and solution time versus E'_p/E'_s based on the 24 × 24 × 24 mesh and CPU–GPU platform for drained analysis

rapidly with E'_p/E'_s, it may outperform the partitioned block preconditioners when $E'_p/E'_s < 10^2$ and outperform $M_{IBD}(\hat{W}_2)$ in a wider range of E'_p/E'_s, as shown in Fig. 13. For the drained analysis illustrated in Fig. 14, SJ preconditioner may exhibit a better performance for $E'_p/E'_s < 10^3$ compared to the other counterparts.

Based on Figs. 8, 9, 10, 11, 12, 13 and 14 and Tables 2 and 3 for the pile-group foundation, it is concluded that when only the CPU computer architecture is available, $M_{PBSSOR}(\hat{W}_2)$ is recommended for the consolidation analysis, while both $M_{IBD}(\hat{G}_1)$ and $M_{PBSSOR}(\hat{G}_2)$ may be used for the drained analysis provided that the computational complexity is not considered. Nevertheless, on the hybrid CPU–GPU computing platform, the diagonal preconditioner has the potential to be exploited in parallelism, and the partitioned block diagonal preconditioner M_{IBD} embedded with diagonal approximation to W is preferred from the perspective of solution time.

5.2 Tunneling example with existing buildings

Tunnel is another typical soil–structure interaction problem, particularly when the existing buildings are simulated. The significant material stiffness contrast between the structural concrete and the soil media greatly contributes to the ill conditions of the resulting stiffness matrices. In the present study, the concrete material properties for the piles and the tunnel liner remain constant as $E'_p = 30$ GPa, $\nu_p = 0.2$, and $k_p = 1.e-14$ m/s. The two soil profiles are defined as follows:

> Soil profile 1 (homogeneous soil): $E'_s = 5$ MPa, $\nu_s = 0.3$, $k_s = 1.e-7$ m/s (i.e., $E'_p/E'_s = 6e3$);
>
> Soil profile 2 (heterogeneous soil): $E'_{s1} = 2$ MPa, $\nu_{s1} = 0.3$, $k_{s1} = 1.e-7$ m/s for the upper soil layer; $E'_{s2} = 50$ MPa, $\nu_{s2} = 0.3$, $k_{s2} = 1.e-3$ m/s for the lower soil layer (i.e., $E'_p/E'_s = 1.5e4$).

Soil profile 1 is employed to model the tunneling process in the homogeneous soft clay, while soil profile 2 is used to simulate the tunnel crossing two soil layers. In Fig. 15, the pressure applied to the pile-group platform and to the tunnel face is 50 and 10 kPa, respectively, and the entire finite element mesh comprises 16,433 20-node elements in which 328 are structure elements. For consolidation analysis and drained analysis, the resulting DOFs are 221,463 and 203,548, respectively.

In the tunnel example, the performances of partitioned block preconditioners are assessed, and the results are provided in Tables 4 and 5, respectively, for the consolidation analysis and the drained analysis. For the consolidation analysis performed on the CPU platform, the $M_{PBSSOR}(\hat{W}_2)$ preconditioner is ranked as the best preconditioner from the perspective of solution time, and the acceleration ratio of $M_{PBSSOR}(\hat{W}_2)$ preconditioner to GJ preconditioner is about 8.5 and 13.1 for the soil profile 1 and soil profile 2, respectively. When the consolidation

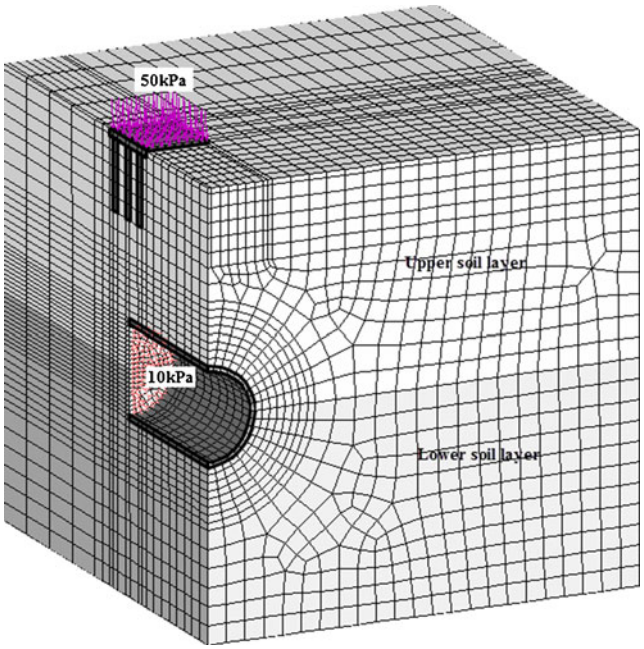

Fig. 15 Finite element mesh of the tunneling example

Table 4 Performances of partitioned block preconditioners for consolidation analysis of the tunnel example

Computer architecture	GJ	$M_{\mathrm{IBD}}\left(\hat{W}_1\right)$	$M_{\mathrm{IBD}}\left(\hat{W}_2\right)$	$M_{\mathrm{PBSSOR}}\left(\hat{W}_1\right)$	$M_{\mathrm{PBSSOR}}\left(\hat{W}_2\right)$
Soil profile 1					
CPU	19,728 (1,427.7 s)	3,751 (282.9 s)	1,412 (227.5 s)	3,713 (309.5 s)	991 (168.7 s)
CPU–GPU	22,126 (622.9 s)	3,699 (118.5 s)	1,408 (165.7 s)	3,478 (145.8 s)	959 (123.9 s)
Speedup	2.29	2.39	1.37	2.12	1.36
Soil profile 2					
CPU	24,644 (1,765.6 s)	3,066 (231.0 s)	1,172 (188.8 s)	2,890 (241.0 s)	790 (134.6 s)
CPU–GPU	19,559 (534.0 s)	3,089 (99.3 s)	1,184 (139.4 s)	2,855 (119.9 s)	783 (101.1 s)
Speedup	3.31	2.33	1.35	2.01	1.33

analyses are carried out on the hybrid CPU–GPU platform, $M_{\mathrm{IBD}}\left(\hat{W}_1\right)$ may outperform $M_{\mathrm{PBSSOR}}\left(\hat{W}_2\right)$, and the acceleration ratio of $M_{\mathrm{IBD}}\left(\hat{W}_1\right)$ preconditioner to GJ preconditioner is about 5.3 and 5.4 for the soil profile 1 and soil profile 2, respectively. The speedup for all preconditioners embedded with diagonal approximation to W is larger than 2.0, while the speedup for all preconditioners embedded with MSSOR (or SSOR) approximation to W is larger than 1.0. It is noteworthy that the iteration number of the preconditioned SQMR solver may vary due to the difference in round-off errors and precision issues between the computing architectures (i.e., CPU and hybrid CPU–GPU), and in our experience, the iterative computations should be operated in double-precision arithmetics to ensure the convergence of an iterative solver. As long as the same convergence criterion and tolerance is utilized, the accuracy of iterative solutions may be guaranteed. Hence, the present study focuses on the computer execution time. Figure 9 gives the results for the drained analysis of the tunnel example. In terms of solution time, it is seen that $M_{\mathrm{PBSSOR}}\left(\hat{G}_2\right)$ is still the best preconditioner on the CPU computer architecture for both soil profiles. Compared to the SJ preconditioner, the acceleration ratio of $M_{\mathrm{PBSSOR}}\left(\hat{G}_2\right)$ is around 7.95 and 8.66 for soil profile 1 and soil profile 2, respectively. Similar to the consolidation analysis, the fastest solutions are achieved by $M_{\mathrm{IBD}}\left(\hat{G}_1\right)$ if the hybrid CPU–GPU computer architecture is used. Compared to the SJ preconditioner, the acceleration ratio of $M_{\mathrm{IBD}}\left(\hat{G}_1\right)$ is about 6.05 and 6.77 for the soil profile 1 and soil profile 2, respectively. Consequently, it may be concluded that $M_{\mathrm{PBSSOR}}\left(\hat{W}_2\right)$ or $M_{\mathrm{PBSSOR}}\left(\hat{G}_2\right)$ is preferred over the others given the CPU computer platform, while either $M_{\mathrm{IBD}}\left(\hat{W}_1\right)$ or $M_{\mathrm{IBD}}\left(\hat{G}_1\right)$ may be used when the hybrid CPU–GPU computer architecture is adopted. The tunnel example also confirms the observation in the pile-group example that simple diagonal preconditioning appears to be especially well suited for the parallel computing. This study evidently answers the question raised by Pini and Gambolati [47], that is, "Is a simple diagonal scaling the best preconditioner for conjugate gradients on supercomputers?" According to our investigations, the preconditioners which may effectively combine the parallel implementation with the physical implication appear to be more attractive, although the simple diagonal scaling is very efficient and suitable for vector/parallel architectures. In this study, the promising preconditioners are $M_{\mathrm{IBD}}\left(\hat{W}_1\right)$ or $M_{\mathrm{IBD}}\left(\hat{G}_1\right)$.

Table 5 Performances of partitioned block preconditioners for drained analysis of the tunnel example

Computer architecture	SJ	$M_{\mathrm{IBD}}\left(\hat{G}_1\right)$	$M_{\mathrm{IBD}}\left(\hat{G}_2\right)$	$M_{\mathrm{PBSSOR}}\left(\hat{G}_1\right)$	$M_{\mathrm{PBSSOR}}\left(\hat{G}_2\right)$
Soil profile 1					
CPU	6,734 (391.1 s)	956 (57.6 s)	473 (60.8 s)	944 (63.6 s)	360 (49.2 s)
CPU–GPU	6,721 (151.3 s)	953 (25.0 s)	474 (44.8 s)	945 (33.7 s)	360 (38.1 s)
Speedup	2.58	2.30	1.36	1.89	1.29
Soil profile 2					
CPU	6,881 (400.1 s)	879 (52.9 s)	438 (56.3 s)	870 (58.6 s)	338 (46.2 s)
CPU–GPU	6,898 (155.6 s)	879 (23.0 s)	437 (41.4 s)	869 (31.1 s)	338 (35.7 s)
Speedup	2.57	2.30	1.36	1.88	1.29

6 Conclusions

The soil–structure interaction problems are commonly encountered in engineering practice, and the resulting linear systems of equations present the difficult task to the practitioners. To solve the large-sized linear systems of equations arising from soil–structure interactions, the Krylov subspace iterative method SQMR, in conjunction with some efficient preconditioners, is utilized. However, the performance of an iterative solution strategy may be significantly influenced by the adopted computer architecture. In view of the rapid advances in GPU technology and ease in access to GPU hardware, the partitioned block preconditioners including the novel partitioned block SSOR preconditioner are evaluated and compared on both the CPU and the hybrid CPU–GPU computer architectures. Based on the numerical results, some observations and concluding remarks are summarized as follows:

1. In finite element computations, the most time-consuming part is the solutions of linear equations, while in the Krylov subspace iterative process, the computationally high-intensity part is the matrix–vector multiplications in the construction of Krylov subspace. Hence, the capability of GPU in intensive FLOPs is exploited to cope with the rapidly increased computational load due to the matrix–vector products in the iterative process.
2. The GPU acceleration effect also depends on the employed matrix storage scheme. A preliminary investigation on two common matrix storage schemes (i.e., the EBE storage and the compressed sparse storage) shows that GPU acceleration on the EBE storage scheme may be more remarkable, which is probably attributed to the more FLOPs involved in the elemental level matrix–vector products, reflecting the capability of GPU in high-intensity computations. However, the compressed sparse storage is still employed in the present study as it usually leads to the faster iterative solutions.
3. The number of threads per block (i.e., tPB) is essential to the efficiency of GPU computing. Numerical investigations reveal that the performance of GPU computing generally increases with the decrease of tPB, perhaps contrary to our intuitions. That phenomenon on tPB may be appropriately interpreted. That is, a large tPB indicates that the other threads have to wait more or less idly for the completion of the thread associated with the longest row. Hence, a small tPB (e.g., 8) is usually good enough for a wide range of problem size.
4. Some preconditioners including one simple diagonal preconditioner and four partitioned block preconditioners are investigated. It is found that their performances may be significantly influenced by the computer architecture. On the CPU platform, $M_{\mathrm{PBSSOR}}(\hat{W}_2)$ or $M_{\mathrm{PBSSOR}}(\hat{G}_2)$ may outperform the other counterparts from the perspective of solution time, while $M_{\mathrm{IBD}}(\hat{W}_1)$ or $M_{\mathrm{IBD}}(\hat{G}_1)$ should be more efficient on the hybrid CPU–GPU platform, indicating that a brute force in the convergence rate may be counterproductive. It should be emphasized, however, that the present numerical experiments are conducted on certain CPU (i.e., Intel i7−920 2.67 GHz CPU) and GPU card (i.e., Geforce GTX 580). The foregoing conclusions may definitely depend on the relative performance of CPU and GPU hardware.
5. As for the speedup achieved, simple diagonal preconditioners appear to be especially well suited for parallel computing. Nevertheless, numerical comparisons disclose that in the design of a good preconditioner, exploiting the physical implications combined with the utilization of parallelism is an effective way. In this study, the $M_{\mathrm{IBD}}(\hat{W}_1)$ or $M_{\mathrm{IBD}}(\hat{G}_1)$ preconditioner provides a good example.

Acknowledgments The research is supported in part by the Research Fund Program of State Key Laboratory of Hydroscience and Engineering Project, no. 2013-KY-4; the Scientific Research Foundation for the 973 Program of China, no. 2012CB026104; and the Fundamental Research Funds for the Central Universities, no. 2013JBM059.

Appendix

Algorithm 1 Matrix–vector multiplication $w = Av$ in the CPU environment *Given* v, $w \in \Re^N$ and $A = A^T \in \Re^{N\times N}$ in symmetric CSC storage (i.e., only upper triangular part of A is stored), $icsc(nnz)$, $jcsc(N+1)$, $vcsc(nnz)$ (nnz is the number of nonzero entries in L_A or U_A).

(1) $w = \mathrm{zeros}(N, 1)$;
(2) **for** $j = 1$ to N **do**:
(3) **if** $v(j) \neq 0$ **then**:
(4) **for** $k = jcsc(j)\mathrm{to}jcsc(j+1) - 1$ **do**:
(5) $r = icsc(k)$;
(6) $w(r) = w(r) + v(j) * vcsc(k)$;
(7) **for** $k = jcsc(j)\mathrm{to}jcsc(j+1) - 2$ **do**:
(8) $r = icsc(k)$;
(9) **if** $v(r) \neq 0$ **then**: $w(j) = w(j) + v(r) * vcsc(k)$;

Algorithm 2 Matrix–vector multiplication $w = Av$ in the GPU device *Given* v, $w \in \Re^N$ and $A = A^T \in \Re^{N\times N}$ in both the CSR (i.e., $icsrDev(N+1)$, $jcsrDev(nnz)$, $vcsrDev(nnz)$) and CSC (i.e., $icscDev(nnz)$, $jcscDev(N+1)$, $vcscDev(nnz)$) storages of U_A, (nnz is the number of nonzero entries in U_A).

(1) Retrieve the current row i for the thread tx;
(2) **if** $i <= N$ **then**:
(3) $val = 0.0$;
(4) **for** $k = jcscDev(i)$to$jcscDev(i+1) - 1$ **do**:
(5) $r = icscDev(k)$;
(6) $val = val + v(r) * vcscDev(k)$;
(7) **for** $k = icsrDev(i) + 1$to$icsrDev(i+1) - 1$ **do**:
(8) $r = jcsrDev(k)$;
(9) $val = val + v(r) * vcsrDev(k)$;
(10) $w(i) = val$;

References

1. Lee, F.H., Hong, S.H., Gu, Q., Zhao, P.: Application of large three-dimensional finite-element analyses to practical problems. Int. J. Geomech. **11**, 529–539 (2011)
2. Phoon, K.K., Toh, K.C., Chan, S.H., Lee, F.H.: An efficient diagonal preconditioner for finite element solution of Biot's consolidation equations. Int. J. Numer. Anal. Methods **55**, 377–400 (2002)
3. Freund, R.W., Nachtigal, N.M.: A new Krylov-subspace method for symmetric indefinite linear systems. In: Ames, W.F. (ed.) Proceedings of the 14th IMACS World Congress on Computational and Applied Mathematics, pp. 1253–1256. Atlanta, U.S.A. (1994)
4. Smith, I.M.: A general purpose system for finite element analyses in parallel. Eng. Comput. **17**, 75–91 (2000)
5. Liu, Y., Zhou, W., Yang, Q.: A distributed memory parallel element-by-element scheme based on Jacobi-conditioned conjugate gradient for 3D finite element analysis. Finite Elem. Anal. Des. **43**, 494–503 (2007)
6. Zhao, B., Goh, S.H., Lee, F.H.: Implicit domain decomposition scheme for parallel dynamic finite element geotechnical analysis, Proceeding SEPADS'10 Proceedings of the 9th WSEAS international conference on Software engineering, parallel and distributed systems (2010)
7. Ferronato, M., Janna, C., Pini, G.: Parallel solution to ill-conditioned FE geomechanical problems. Int. J. Numer. Anal. Methods **36**, 422–437 (2012)
8. Janna, C., Ferronato, M., Gambolati, G.: Parallel inexact constraint preconditioning for ill-conditioned consolidation problems. Comput. Geosci. **16**, 661–675 (2012). doi:10.1007/s10596-012-9276-4
9. Buatois, L., Caumon, G., Lévy, B.: Concurrent number cruncher: a GPU implementation of a general sparse linear solver. Int. J. Parallel Emergent Distrib. Syst. **24**, 205–223 (2009)
10. Preis, T., Virnau, P., Paul, W., Schneider, J.J.: GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. J. Comput. Phys. **228**, 4468–4477 (2009)
11. Joldes, G.R., Wittek, A., Miller, K.: Real-time nonlinear finite element computations on GPU-Application to neurosurgical simulation. Comput. Methods Appl. Mech. Eng. **199**, 3305–3314 (2010)
12. Griebel, M., Zaspel, P.: A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations. Comput. Sci. Res. Dev. **25**, 65–73 (2010)
13. Fatahalian, K., Sugerman, J., Hanrahan, P.: Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. HWWS '04 Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, 133–138 (2004)
14. Baskaran, M.M., Bordawekar, R.: Optimizing sparse matrix-vector multiplication on GPUs. IBM Technical Report RC24704 (2008)
15. Bell, N., Garland, M.: Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004 (2008)
16. Brandon, L.D., Boyd, C., Govindaraju, N.: Fast computation of general Fourier transforms on GPUs. 2008 IEEE international conference on multimedia and expo, 5–8 (2008)
17. Nickolls, J., Dally, W.J.: The GPU computing era. Micro IEEE **30**, 56–69 (2010)
18. Papadrakakis, M., Stavroulakis, G., Karatarakis, A.: A new era in scientific computing: domain decomposition methods in hybrid CPU–GPU architectures. Comput. Methods Appl. Mech. Eng. **200**, 1490–1508 (2011)
19. Göddeke, D., Strzodka, R., Turek, S.: Accelerating double precision FEM simulations with GPUs. In: Proc. of ASIM 2005, 18th symposium on simulation technique, 139–144 (2005)
20. Cecka, C., Lew, A.J., Darve, E.: Assembly of finite element methods on graphics processors. Int. J. Numer. Methods Eng. **85**, 640–669 (2011)
21. van der Vorst, H.A.: Iterative Krylov methods for large linear systems. Cambridge University Press, Cambridge (2003)
22. Mroueh, H., Shahrour, I.: Use of sparse iterative methods for the resolution of three-dimensional soil/structure interaction problems. Int. J. Numer. Anal. Methods **23**, 1961–1975 (1999)
23. Mroueh, H., Shahrour, I.: A full 3-D finite element analysis of tunneling-adjacent structures interaction. Comput. Geotech. **30**, 245–253 (2003)
24. Chen, X., Phoon, K.K., Toh, K.C.: Partitioned versus global Krylov subspace iterative methods for FE solution of 3-D Biot's problem. Comput. Methods Appl. Mech. Eng. **196**, 2737–2750 (2007)
25. Augarde, C.E., Crouch, R.S., Li, T., Ramage, A.: The effects of geotechnical material properties on the convergence of iterative solvers. International Association for Computer Methods and Advances in Geomechanics (IACMAG), 12th IACMAG, Goa, India (2008)
26. Jeremi, B., Jie, G.: Parallel finite element computations for soil-foundation-structure interaction problems, Report UCD–CompGeoMech–02–07, Report version: 1, Computational Geomechanics Group, Department of Civil and Environmental Engineering, University of California, Davis (2008)
27. Haga, J.B., Osnes, H., Langtangen, H.P.: Efficient block preconditioners for the coupled equations of pressure and deformation in highly discontinuous media. Int. J. Numer. Anal. Methods **35**, 1466–1482 (2011)
28. Chaudhary, K.B., Phoon, K.K., Toh, K.C.: Inexact block diagonal preconditioners to mitigate the effects of relative differences in material stiffnesses. Int. J. Geomech. (2012). doi:10.1061/(ASCE)GM.1943-5622.0000197
29. Chaudhary, K.B., Phoon, K.K., Toh, K.C.: Effective block diagonal preconditioners for Biot's consolidation equations in piled-raft foundations. Int. J. Numer. Anal. Methods (2012). doi:10.1002/nag.1127

30. Gambolati, G., Ferronato, M., Janna, C.: Preconditioners in computational geomechanics: a survey. Int. J. Numer. Anal. Methods **35**, 980–996 (2011)
31. Biot, M.A.: General theory of three-dimensional consolidation. J. Appl. Phys. **12**, 155–164 (1941)
32. Smith, I.M., Griffiths, D.V., 4th ed.: Programming the Finite Element Method. Wiley, New York (2004)
33. Chen, X., Toh, K.C., Phoon, K.K.: A modified SSOR preconditioner for sparse symmetric indefinite linear systems of equations. Int. J. Numer. Methods Eng. **65**, 785–807 (2006)
34. Chen, X., Phoon, K.K., Toh, K.C.: Performance of zero-level fill-in preconditioning techniques for iterative solutions with geotechnical applications. ASCE Int. J. Geomech. **12**, 596–605 (2012)
35. Chen, X., Cheng, Y.G.: On Accelerated symmetric stiffness techniques for non-associated plasticity with application to soil problems. Eng. Comput. **28**, 1044–1063 (2011)
36. Toh, K.C., Phoon, K.K., Chan, S.H.: Block preconditioners for symmetric indefinite linear systems. Int. J. Numer. Methods Eng. **60**, 1361–1381 (2006)
37. Wang, M., Klie, H., Parashar, M., Sudan, H.: Solving sparse linear systems on NVIDIA Tesla GPUs. Computational Science-ICCS 2009. Lect. Notes Comput. Sci. **5544**, 864–873 (2009)
38. Saad, Y., Schultz, M.H.: GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **7**, 856–869 (1986)
39. Freund, R.W., Nachtigal, N.M.: QMR: A quasi-minimal residual method for non-Hermitian linear systems. Numer. Math. **60**, 315–339 (1991)
40. van der Vorst, H.A.: BI-CGSTAB: A fast and smoothly converging variant of bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **13**, 631–644 (1992)
41. Hughes, T.J.R., Levit, I., Winget, J.: An element-by-element solution algorithm for problems of structural and solid mechanics. Comput. Methods Appl. Mech. Eng. **36**, 241–254 (1983)
42. Augarde, C.E., Ramage, A., Staudacher, J.: Element-based preconditioners for elasto-plastic problems in geotechnical engineering. Int. J. Numer. Methods Eng. **71**, 757–779 (2007)
43. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing Company, Boston (1996)
44. Ng, E.G., Peyton, B.W.: Block sparse Cholesky algorithms on advanced uniprocessor computers. SIAM J. Sci. Comput. **14**, 1034–1056 (1993)
45. Chen, X., Phoon, K.K.: Some numerical experiences on convergence criteria for iterative finite element solvers. Comput. Geotech. **36**, 1272–1284 (2009)
46. Wolfe, M.: CUDA FORTRAN Programming Guide and Reference, The Portland Group, Release (2012)
47. Pini, G., Gambolati, G.: Is a simple diagonal scaling the best preconditioner for conjugate gradients on supercomputers? Adv. Water Resour. **13**, 147–153 (1990)