Available online at www.sciencedirect.com
SCIENCE DIRECT®
Comput. Methods Appl. Mech. Engrg. 192 (2003) 1495–1513
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Parallel adaptive solution of 3D boundary value problems by Hessian recovery

Konstantin Lipnikov [a], Yuri Vassilevski [b,*]

[a] *Department of Mathematics, University of Houston, Houston, TX 77204, USA*
[b] *Institute of Numerical Mathematics, Russian Academy of Sciences, Gubkina 8, 119991 Moscow, Russia*

Received 21 February 2002; received in revised form 27 August 2002

Abstract

A parallel technique for an adaptive solution of 3D boundary value problems is described. It incorporates a parallel mesh generation and a parallel iterative solution of the corresponding discrete problem. Both generation and solution are problem independent and may be considered as *black-boxes*.

Keywords: Hessian recovery; Quasi-optimal meshes; Inertial bisection; Parallel mesh generator; Parallel iterative solvers

1. Introduction

Robust parallel algorithms for adaptive solution of boundary value problems for PDEs have been of interest to engineers and mathematicians over many years. Significant improvement of the accuracy of approximation through the adaptive distribution of mesh elements rather than an increase of their number enables to solve large problems arising in applications. Parallel techniques for generation of adaptive meshes and solution of the associated discrete problems extend computational capabilities considerably.

It has been shown in a number of papers (see, for example, [1,2]) that simplexes with obtuse and acute angles stretched along the direction of minimal second derivative of a solution may be the best elements for minimizing the interpolation error. As the result, an optimal adaptive mesh may frequently contain anisotropic elements, i.e., elements with obtuse and acute angles. Until recently, the use of adaptive anisotropic 3D meshes has been delayed because of the lack of robust adaptive mesh generators.

Nowadays, a lot of popular adaptation algorithms are based on *a posteriori* error estimators. Besides the discrete solution, the error estimators require additional data of the problem which reduces their flexibility. An alternative approach is based on a *black-box* Hessian recovery technique [3–6]. The objective of this technique is to construct a mesh which is quasi-uniform in a metric field generated by the discrete Hessian

* Corresponding author.
E-mail addresses: lipnikov@math.uh.edu (K. Lipnikov), vasilevs@dodo.inm.ras.ru (Y. Vassilevski).

0045-7825/03/$ - see front matter
doi:10.1016/S0045-7825(02)00654-0

recovered from the discrete solution. If the solution features an anisotropic behavior, the adaptive mesh turns out to be anisotropic in order to fit very well the solution variations. Nowadays, this is the most promising problem independent approach for adaptive mesh generation. The sequential robust version of the technique has been studied in [3,6] and appeared to be relatively slow. In this paper we describe its parallel version.

Any adaptive technique has to be equipped with a solution procedure. To our best knowledge, there is a lack of efficient parallel black-box solvers. The majority of available algorithms are problem dependent, i.e., they have to be tuned up for the particular problem. In this paper, we propose a new parallel black-box solver independent (as much as possible) of the underlying problem. The solver is based on the domain decomposition (DD) technique and an efficient sequential subdomain black-box preconditioner which is assumed to be at hand. In our work, we take advantage of the algebraic multigrid (AMG) [7] preconditioner known as AMG1R5 (J. Ruge, K. Stuben, Release 1.5, 1990). For the discrete operators considered in Section 6, the AMG is known to be a very good preconditioner [8]. In our DD framework a block-diagonal preconditioner based on the AMG solver in subdomains is corrected by a special smoother. The smoother enables us to formulate an iterative solver with the convergence rate independent of the number of processors (subdomains), problem coefficients, and only slightly sensitive to the problem size. The overall preconditioner is very simple to implement and parallelize.

The paper outline is as follows. In Section 2 we introduce the concept of a *quasi-optimal* mesh and describe a sequential algorithm for its generation. In Section 3 we formulate a parallel algorithm for generation of quasi-optimal meshes (QOMs). In Section 4 we propose the DD framework for the iterative solution of discrete problems. A black-box parallel implementation of the solver is discussed in Section 5. Section 6 is devoted to the detailed numerical study of the proposed algorithms. We discuss the accuracy of approximations and scalability and efficiency of parallel algorithms for problems with anisotropic singularities, boundary layers, and jumps in coefficients.

2. Quasi-optimal meshes

Let $\Omega \in \Re^3$ be a polyhedral domain and Ω_h be its conformal partition into tetrahedra,

$$\Omega_h = \bigcup_{i=1}^{\mathcal{N}(\Omega_h)} e_i,$$

where $\mathcal{N}(\Omega_h)$ is the number of elements in Ω_h. Let $C^k(D)$ be a space of functions with continuous in $D \subset \overline{\Omega}$ partial derivatives up to order k. Let $\|\cdot\|_{\infty,D}$ and $\|\cdot\|_{2,D}$ denote the norms on the spaces $L_\infty(D)$ and $C^2(D)$, respectively, and $\|\cdot\|_\infty \equiv \|\cdot\|_{\infty,\Omega}$. In addition, we shall use notation $P_1(\Omega_h)$ for the space of functions continuous in Ω and linear on each element of Ω_h. Furthermore, let $\mathscr{P}^h_{\Omega_h} : C^0(\overline{\Omega}) \to P_1(\Omega_h)$ be a projector on the discrete space $P_1(\Omega_h)$ and $\mathscr{I}^h_{\Omega_h} : C^0(\overline{\Omega}) \to P_1(\Omega_h)$ be the linear interpolation operator.

Most theoretical results formulated in this section are based on the assumption that the solution of a continuous second order boundary value problem belongs to $C^2(\overline{\Omega})$. However, constants in our error estimates are independent of the actual value of C^2-norm of the solution. Since $C^2(\overline{\Omega})$ is dense in $C^0(\overline{\Omega})$, one can try to analyze regularized problems with smooth solutions and obtain error estimates for the original problem by the density arguments. We shall address this challenging problem in the future papers.

Definition 1. Let $u \in C^0(\overline{\Omega})$ and $\mathscr{P}^h_{\Omega_h}$ be given. The mesh $\Omega_h(N_T, u)$ consisting of N_T elements is called optimal if it is a solution of the optimization problem

$$\Omega_h(N_T, u) = \arg \min_{\Omega_h : \mathcal{N}(\Omega_h) \leqslant N_T} \|u - \mathscr{P}^h_{\Omega_h} u\|_\infty. \tag{1}$$

In a general case, the optimization problem (1) may be not well posed. The existence of the optimal mesh is analyzed elsewhere [9]. Here, we assume that the solution to problem (1) exist.

Since the exact solution is unknown, the error $\|u - \mathscr{P}^h_{\Omega_h} u\|_\infty$ cannot be estimated. Therefore, the optimization problem (1) has to be replaced by another optimization problem whose solution at least approaches the solution of (1). To this end, we introduce concepts of a mesh quality and a mesh quasi-optimality.

Let $Q(\Omega_h)$ be an easily computed quantitative characteristic of mesh Ω_h such that $0 < Q(\Omega_h) \leqslant 1$. We shall use the definition of $Q(\Omega_h)$ proposed in [10]. Let a fixed number of elements N_T be given, $\boldsymbol{G}(x) = \{G_{ps}(x)\}_{p,s=1}^3$, $x \in \Re^3$, be a continuous metric in Ω, and $x_e \in e$ be a point in tetrahedron e where $|\det(G(x))|$ attains its maximal value. We set $\boldsymbol{G}_e = \boldsymbol{G}(x_e)$ and define the volume of this tetrahedron and the length of its edge $\vec{l}_e \in \Re^3$ (in metric $\boldsymbol{G}$) by

$$|e|_G = |e|(\det(\boldsymbol{G}_e))^{1/2} \quad \text{and} \quad |\vec{l}_e|_G = (\boldsymbol{G}_e \vec{l}_e, \vec{l}_e)^{1/2},$$

respectively, where $|e|$ is the tetrahedron volume in the Cartesian coordinate system. Denote the sum of lengths of edges of tetrahedron e measured in metric $\boldsymbol{G}$ by $|\partial\partial e|_G$. Let $|\Omega_h|_G$ be the total volume of the computational domain measured in metric $\boldsymbol{G}$,

$$|\Omega_h|_G = \sum_{e \in \Omega_h} |e|_G.$$

Following [10], we define $Q(\Omega_h)$ as follows:

$$Q(\Omega_h) = \min_{e \in \Omega_h} Q(e) \quad \text{with } Q(e) = 6^4\sqrt{2}\frac{|e|_G}{|\partial\partial e|_G^3} F\left(\frac{|\partial\partial e|_G}{6h^*}\right), \tag{2}$$

where function $F(\cdot)$ and the average length of a tetrahedron edge h^* (in metric $\boldsymbol{G}$) are given by

$$F(x) = \left(\min\left\{x, \frac{1}{x}\right\}\left(2 - \min\left\{x, \frac{1}{x}\right\}\right)\right)^3 \quad \text{and} \quad h^* = \sqrt[3]{\frac{12|\Omega_h|_G}{\sqrt{2}N_T}}.$$

Hereafter we shall use notation $Q(\boldsymbol{G}, N_T, \Omega_h)$ instead of $Q(\Omega_h)$ to emphasize its dependence on the metric $\boldsymbol{G}$ and the predefined number of elements N_T. It is easy to check that $0 < Q(\boldsymbol{G}, N_T, \Omega_h) \leqslant 1$ and the maximal value is attained when all mesh elements are equilateral (in metric $\boldsymbol{G}$) tetrahedra with the edge length h^*. We refer to $Q(\boldsymbol{G}, N_T, \Omega_h)$ as the *mesh quality* with respect to the metric $\boldsymbol{G}$ and the number of elements N_T.

Definition 2. Let $\boldsymbol{G}$ be a continuous metric and N_T be a given integer. The mesh Ω_h is called $\boldsymbol{G}$-quasi-optimal if there is a fixed positive constant Q_0 such that $Q_0 = \mathrm{O}(1)$ and

$$Q(\boldsymbol{G}, N_T, \Omega_h) > Q_0.$$

Let function $u \in C^2(\overline{\Omega})$ have a nonsingular Hessian $\boldsymbol{H}(x) = \{H_{ps}(x)\}_{p,s=1}^3$, i.e., $\det \boldsymbol{H}(x) \neq 0$ for $\forall x \in \overline{\Omega}$. Since the Hessian is symmetric, the spectral decomposition of $\boldsymbol{H}$ is possible for any $x \in \overline{\Omega}$,

$$\boldsymbol{H} = W^{\mathrm{t}} \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} W,$$

and the following metric may be defined:

$$|\boldsymbol{H}| = W^{\mathrm{t}} \begin{pmatrix} |\lambda_1| & 0 & 0 \\ 0 & |\lambda_2| & 0 \\ 0 & 0 & |\lambda_3| \end{pmatrix} W.$$

Definition 3. Let $u \in C^2(\overline{\Omega})$ and $|\boldsymbol{H}|$ be a metric generated by the Hessian of u. For the given function u and a given integer N_T, the mesh $\Omega_h(N_T, u)$ is called quasi-optimal if it is $|\boldsymbol{H}|$-quasi-optimal.

We notice that it is impossible to cover 3D space by equilateral tetrahedra. Therefore a quasi-optimal mesh satisfying $Q(|\boldsymbol{H}|, N_T, \Omega_h) = 1$ rather does not exist. Fixing $Q_0 < 1$ relaxes the above constraint as well as restrictions imposed by the boundary of Ω. On the other hand, due to $Q(|\boldsymbol{H}|, N_T, \Omega_h) < 1$, the number of mesh elements $\mathcal{N}(\Omega_h)$ in the $|\boldsymbol{H}|$-quasi-optimal mesh may differ from N_T but approaches it when $Q_0 \to 1$.

The QOMs have been studied in [3,11]. It turns out that in certain cases the QOM is an approximate solution of optimization problem (1).

Theorem 4 (Vassilevski, Agouzal, Lipnikov, 1999). *Let $N_T > 0$, $u \in C^2(\overline{\Omega})$ and $|\boldsymbol{H}|$ be a metric generated by the Hessian of u. Furthermore, let $\Omega_h(N_T, u)$ and $\widetilde{\Omega}_h(N_T, u)$ be the quasi-optimal and optimal meshes, respectively, and $e^* \in \Omega_h$ be the element where $\|u - \mathscr{I}^h_{\Omega_h} u\|_\infty$ is attained. Let for both any element $\tilde{e} \in \widetilde{\Omega}_h$ and the element $e^* \in \Omega_h$ the following estimate holds*:

$$\|H_{ps} - H_{e,ps}\|_{\infty,e} < q|\lambda_1(\boldsymbol{H}_e)|/2, \quad 0 < q < 1, \ \ p,s = 1,2,3, \tag{3}$$

where q is a constant, $\boldsymbol{H}_e = \boldsymbol{H}(x_e)$, $\lambda_1(\boldsymbol{H}_e)$ is the closest to zero eigenvalue of $\boldsymbol{H}_e$, and $x_e = \operatorname{argmax}_{x \in e} |\det(\boldsymbol{H}(x))|$. Then

$$\|u - \mathscr{I}^h_{\Omega_h} u\|_\infty \leqslant C(Q_0, q)\|u - \mathscr{I}^h_{\widetilde{\Omega}_h} u\|_\infty, \tag{4}$$

where $C(Q_0, q)$ is a constant depending only on q and Q_0 from Definition 2. Moreover,

$$C_1(q)\left(\frac{|\Omega|_{|\boldsymbol{H}|}}{N_T}\right)^{2/3} \leqslant \|u - \mathscr{I}^h_{\widetilde{\Omega}_h} u\|_\infty \leqslant C_2(q)\left(\frac{|\Omega|_{|\boldsymbol{H}|}}{N_T}\right)^{2/3}, \tag{5}$$

where $C_1(q), C_2(q)$ depend only on q.

We notice that for projectors $\mathscr{P}^h_{\Omega_h}$ satisfying

$$\|u - \mathscr{P}^h_{\Omega_h} u\|_\infty \leqslant \widehat{C}\|u - \mathscr{I}^h_{\Omega_h} u\|_\infty$$

formulae (4) and (5) transform to

$$\|u - \mathscr{P}^h_{\Omega_h} u\|_\infty \leqslant \widehat{C} C_2(q) C(Q_0, q)\left(\frac{|\Omega|_{|\boldsymbol{H}|}}{N_T}\right)^{2/3}. \tag{6}$$

Obviously, the Hessian $\boldsymbol{H}(x)$ is an unknown function. In computations we use its approximation $\boldsymbol{H}^h$ recovered from the discrete solution $\mathscr{P}^h_{\Omega_h} u$. In the following, we briefly describe a Hessian recovery algorithm [5,6] and advocate the replacement of $\boldsymbol{H}(x)$ by its discrete counterpart $\boldsymbol{H}^h$.

Let $u^h = \mathscr{P}^h_{\Omega_h} u$ be a discrete function from $P_1(\Omega_h)$. The discrete Hessian $\boldsymbol{H}^h = \{H^h_{ps}\}^3_{p,s=1}$, $H^h_{ps} \in P_1(\Omega_h)$, is defined as follows. In interior mesh node a_i, the Hessian entries $H^h_{ps}(a_i)$, $p,s = 1,2,3$, are defined by

$$\int_{\sigma_i} H^h_{ps}(a_i) v^h \,\mathrm{d}x = -\int_{\sigma_i} \frac{\partial u^h}{\partial x_p}\frac{\partial v^h}{\partial x_s}\,\mathrm{d}x \quad \forall v^h \in P_1(\sigma_i), \ \ v^h = 0 \ \text{ on } \partial\sigma_i, \tag{7}$$

where σ_i is the union of tetrahedra sharing the node a_i. At boundary node a_i, values of $H^h_{ps}(a_i)$, $p,s = 1,2,3$, are weighted extrapolations from the neighboring interior values [12]:

$$H^h_{ps}(a_i) = \frac{\int_{\sigma_i} \varphi(a_i) \overset{\circ}{H}{}^h_{ps}\,\mathrm{d}x}{\int_{\sigma_i} \varphi(a_i)\Big(\sum_{a_j \notin \partial\Omega_h} \varphi(a_j)\Big)\mathrm{d}x}, \tag{8}$$

where $\varphi(a_i)$ denotes the nodal basis function from $P_1(\Omega_h)$ and $\overset{\circ}{H}{}^h_{ps}$ stands for the finite elements function defined by (7) and vanishing on $\partial\Omega_h$.

Theorem 5 (Vassilevski, Agouzal, Lipnikov, 1999). *Let $N_T > 0$, $u \in C^2(\overline{\Omega})$, $u^h = \mathscr{P}^h_{\Omega_h} u$, $\boldsymbol{H}$ be the Hessian of u, and $\boldsymbol{H}^h$ be the discrete Hessian recovered from u^h. Furthermore, let for any superelement $\sigma \in \Omega_h$ associated with a mesh node a the following estimates hold*:

$$\|H_{ps} - H_{\sigma,ps}\|_{\infty,\sigma} < \delta, \tag{9}$$

$$|H^h_{ps}(a) - H_{\sigma,ps}| < \varepsilon, \tag{10}$$

where $\boldsymbol{H}_\sigma = \boldsymbol{H}(x_\sigma)$ and $x_\sigma = \arg\max_{x\in\sigma} |\det(\boldsymbol{H}(x))|$. Then for ε and δ sufficiently small with respect the minimal eigenvalue of $|\boldsymbol{H}_\sigma|$, the $|\boldsymbol{H}^h|$-quasi-optimal mesh Ω_h ($Q(|\boldsymbol{H}^h|, N_T, \Omega_h) \geqslant Q_0$) is also $|\boldsymbol{H}|$-quasi-optimal:

$$Q(|\boldsymbol{H}|, N_T, \Omega_h) \geqslant CQ_0$$

with constant C independent of N_T and $\|u\|_{2,\Omega}$.

The assumption (9) implies small variations of the Hessian on any superelement σ and (10) is the requirement of nodal-wise approximation for the Hessian. The latter assumption does not always hold true in practice, since it is implicative of a small gradient error for u^h. A theoretical generalization uses another definition of the discrete Hessian and weaker norms [13]. However, in practical computations the adapted meshes look very feasible regardless possible violation of (10).

Assumptions (9) and (10) imply that the discrete Hessian approaches the continuous one. They also mean that the computational mesh and the discrete solution correspond in some way. In computations it is achieved via the following iterative algorithm.

Algorithm 1 (*Generation of a QOM*)

Initialization step. Generate an initial triangulation Ω_h. Choose the final mesh quality Q_0, $Q_0 < 1$, and the final number N_T of mesh elements.

Iterative step.

(1) Compute the approximation $u^h = \mathscr{P}^h_{\Omega_h} u$.
(2) Recover the discrete Hessian $\boldsymbol{H}^h$ from u^h. If $Q(|\boldsymbol{H}^h|, N_T, \Omega_h) > Q_0$, then stop.
(3) Generate next mesh $\widetilde{\Omega}_h$ such that $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h) > Q_0$.
(4) Set $\Omega_h := \widetilde{\Omega}_h$ and go to 1.

There are several heuristic methods for generation of QOMs [4–6]. We follow the approach proposed in [6,10] for generation triangular and simplicial meshes. The basic strategy is a modification of the mesh by local operations which increase the mesh quality. This is achieved by changing the mesh element e with the lowest value $Q(e)$ together with its neighborhood. It requires to support an ordered list of element qualities. In our implementation, each modification of the ordered list is proportional to $\mathscr{N}(\Omega_h)^{1/2}$ albeit a theoretically optimal implementation may require $O(\log \mathscr{N}(\Omega_h))$ operations. The complete list of the local operations is presented below (see [3] for more details).

Add a new node. Try to insert a new point in the middle of a mesh edge and split all tetrahedra sharing the edge by connecting the new point and the vertices of the tetrahedra. In the case of curvilinear surfaces, new boundary point is shifted on the boundary.

Swap face to edge. Try to remove the common face of two tetrahedra and connect opposite vertices by the edge.

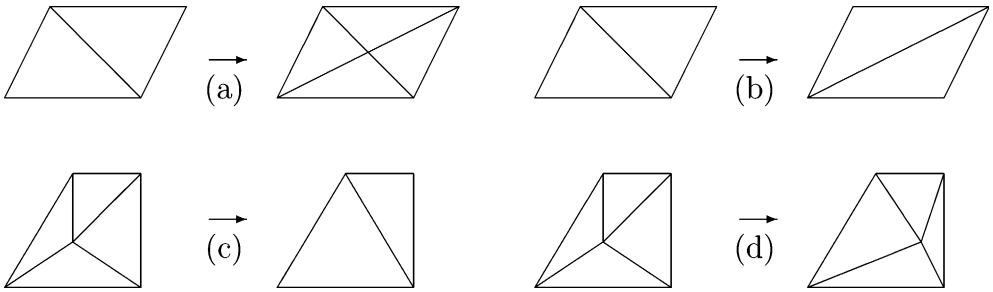

Fig. 1. Local operations: add node (a), swap edge (b), delete node (c), and move node (d).

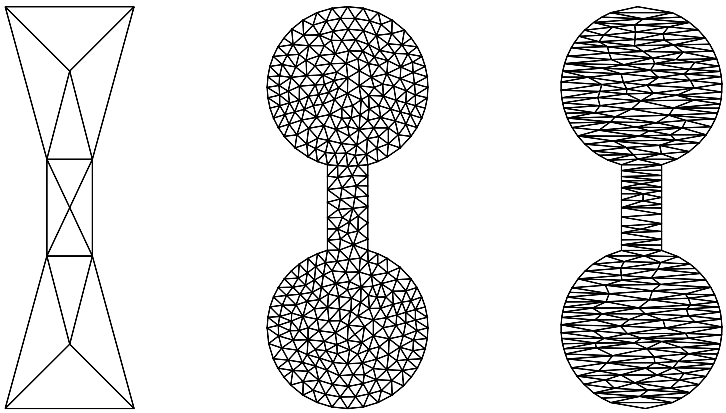

Fig. 2. 2D example: initial, isotropic and anisotropic meshes.

Swap edge to a face. Try to remove a mesh edge and split thus appearing polyhedron into a new set of tetrahedra.

Delete a node. Try to delete a mesh node together with all mesh edges ending in it and split the resulted polyhedron into a new set of tetrahedra.

Move a node. Try to move a mesh node inside a superelement consisting of tetrahedra sharing the node.

Analogs of local operations for triangular meshes are shown on Fig. 1.

We notice that Step 3 of the Algorithm 1 can be used beyond the algorithm. For instance, it can be applied for generation of meshes with given properties. Moreover, the initial grid may contain just a few tetrahedra. We illustrate this again by the 2D example shown on Fig. 2. The initial mesh (on the left) contains a few triangles. Let the metric field be isotropic, i.e., $\boldsymbol{H} = \mathrm{diag}\{1, 1\}$ and $N_T = 600$. After a few local modifications we get an isotropic mesh shown in the middle. Let us replace the isotropic metric by the anisotropic one given by $\underset{\sim}{\boldsymbol{H}} = \mathrm{diag}\{1, 50\}$. The mesh generator produces the QOM shown on the right.

We recall that mesh Ω_h obtained at Step 3 of Algorithm 1 may not contain exactly N_T elements due to some topological restrictions and difference between Q_0 and 1.

3. Parallelization of adaptive mesh generation

The local nature of Algorithm 1 is very flexible for different conceptions to its parallelization. Hereafter, we consider one of them which is based on the following set of assumptions.

First, we can afford to have the whole mesh on each processor. Simple calculations show that a mesh containing 10^6 tetrahedra requires about 34 Mbytes of processor's memory which is far below the memory limit of modern computers.

Second, our parallel computer has a few processors and the mesh is divided evenly among them. We believe that the generation of adaptive unstructured conforming meshes cannot be realized effectively on a massive parallel computer due to a big data flow between processors and difficulties with load-balancing control.

Third, we assume that the mesh can be easily distributed among processors and gathered back, i.e., the time for necessary computations and communications is much less than the time for the mesh generation. The arithmetical complexity of the scattering/gathering algorithm described below is proportional to the total number of mesh elements and is negligibly small in computations.

Let P be the number of processors. There are several ways to divide mesh Ω_h into P approximately equal parts, and two popular algorithms are: spectral bisection, and geometric methods such as inertial bisection. The spectral method [14,15] uses spectral properties of the mesh graph, such as the second eigenvector of the Laplacian matrix of the graph. Necessity to solve an eigenvalue problem for the Laplacian matrix makes this method very expensive. In contrast, the inertial bisection algorithm [16] ignores the connectivity information of the mesh graph. It is based on coordinate sorting and partitioning along inertial axes of the graph. The arithmetical complexity of this algorithm is proportional to $\mathcal{N}(\Omega_h)$.

In the inertial bisection algorithm, we first compute the principle directions (axes) of the inertia tensor for a body consisting of unit masses centered at mesh elements. Then, we split the mesh into P approximately equal disjoint submeshes by $P-1$ parallel planes orthogonal to one of the axes. We shall call these planes as *splitting planes*. Similarly, the boundaries between submeshes will be called as mesh splitting planes. The axes are obtained by computing eigenvectors of the inertia tensor

$$\boldsymbol{T}(\Omega_h) = \sum_{i=1}^{\mathcal{N}(\Omega_h)} (|p_i|^2 I_3 - p_i p_i^{\mathrm{t}}), \quad p_i = x_i - \bar{x}, \tag{11}$$

where $x_i \in \Re^3$ is the barycenter of ith element and $\bar{x}$ is the mesh mass center,

$$\bar{x} = \frac{1}{\mathcal{N}(\Omega_h)} \sum_{i=1}^{\mathcal{N}(\Omega_h)} x_i.$$

Let the mesh Ω_h and the discrete Hessian $|\boldsymbol{H}^h|$ be given on one processor with rank `root`. Then, the parallel generation of a $|\boldsymbol{H}^h|$-quasi-optimal mesh $\widetilde{\Omega}_h$ such that $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h) > Q_0$ (see Step 3 of Algorithm 1) includes the following steps.

Algorithm 2 (*Parallel generation of a $|\boldsymbol{H}^h|$-QOM*)

Initialization step. Processor with rank `root` computes the inertia tensor $\boldsymbol{T}(\Omega_h)$ and broadcasts the discrete Hessian $\boldsymbol{H}^h$ to all processors. Set $k = 1$.

Decomposition step ($k < 4$). Processor with rank `root` extracts the mesh elements whose quality is less than Q_0 and their neighbors having at least one common point. Then, it colors the extracted mesh elements using the inertial bisection algorithm for the kth principle direction of $\boldsymbol{T}(\Omega_h)$ and broadcasts the extracted mesh and the colors to all processors.

Decomposition step ($k = 4$). Processor with rank `root` extracts the mesh elements whose vertices belonged to one of the mesh splitting planes during each of three previous steps. The extracted mesh elements are colored by the root's color.

Generation step. Processor with rank `p` extracts the pth subgrid from the received data and tries to construct a $|\boldsymbol{H}^h|$-quasi-optimal mesh. In order to keep conformity of the global grid, the boundary triangles shared by any two subgrids are not modified. Note that there is only one subgrid when $k = 4$.

Gathering step. Processor with rank `root` gathers the subgrids from other processors and builds a conforming global grid $\widetilde{\Omega}_h$. If $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h) > Q_0$, the algorithm stops. Otherwise, we set $k := k + 1$ (if $k = 5$, we set $k = 1$), $\Omega_h := \widetilde{\Omega}_h$ and go to decomposition step.

The arithmetical complexity of the decomposition and gathering steps is proportional to $\mathcal{N}(\Omega_h)$. The numerical experiments show that these steps are not time consuming.

We recall that the interior boundary triangles are frozen during the generation step. This leads to a bad quality of near-boundary mesh elements and as the result to a small value for $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h)$. Therefore, the alternation of decomposition directions (the principle directions of $\boldsymbol{T}(\Omega_h)$) is important for convergence of Algorithm 2. However, after three iterations of Algorithm 2 (for $k = 1, 2, 3$), we may still have tetrahedra whose vertices (or some of them) were all the time on one of the mesh splitting planes. The definition of the inertial bisection algorithm implies that the number of such vertices is less or equal to $(P-1)^3$. Indeed, they are intersections of three sets of the mesh splitting planes. By one of the assumptions, the number of processors is small. Therefore, the generation step for $k = 4$ is not time consuming.

Another interesting observation is that the dynamics of grid modifications is completely different in the parallel and sequential algorithms, because the parallel algorithm changes the global grid in P different places simultaneously. As we shall see in Section 6, this may significantly affect the performance of the algorithm.

4. Iterative solution with a two-stage preconditioner

In the next two sections, we focus on the iterative solution of a discrete problem. In this paper, it is the Step 1 of Algorithm 1. More specifically, we propose and study a new two-stage black-box preconditioner based on the DD technique: a block diagonal preconditioner is combined with a special smoother. The blocks are the black-box AMG solvers [7]. The smoother removes dependence (of the convergence rate for an iterative solver) on the number of subdomains and leaves minor sensitivity to the problem size. For higher robustness of our algorithm, we minimize as much as possible its dependence on the underlying problem. In fact, the only additional information we shall exploit is the mesh graph.

We assume that the projector $\mathscr{P}^h_{\Omega_h}$ results in a linear system

$$Au = f, \tag{12}$$

with a sparse nonsingular matrix $A \in \Re^{n\times n}$. The order of A may be very large so that only iterative methods are applicable for the solution of (12). For simplicity of presentation, we assume that the solution entries are associated with the mesh nodes.

Our preconditioner is based on a two-stage DD-method proposed in [17]. At the first stage, the conventional additive Schwarz preconditioner is applied. We partition the entries of vector u into P disjoint subsets using the inertia bisection algorithm for the mesh nodes. This is equivalent to an overlapping DD with the minimal (one element) intersection of subdomains (see Fig. 3). Matrix A admits a block representation associated with the above partitioning:

$$A = \begin{pmatrix} A_{11} & \dots & A_{1P} \\ \vdots & \ddots & \vdots \\ A_{P1} & \dots & A_{PP} \end{pmatrix}.$$

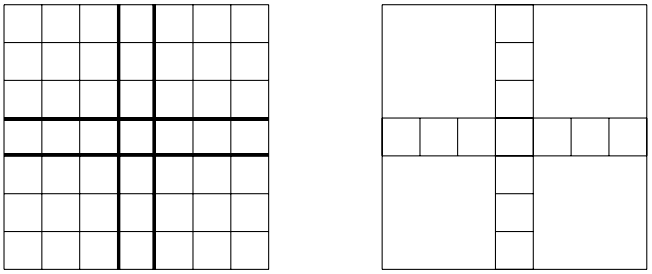

Fig. 3. Mesh associated with the additive Schwarz (left) and the correction (right).

The diagonal blocks $A_{ii} \in \Re^{n_i \times n_i}$, $i = 1, \ldots, P$, correspond to boundary value problems in subdomains with the Dirichlet boundary conditions on interior boundaries.

Let B_{ii} be the AMG preconditioner for A_{ii}, $i = 1, \ldots, P$. The complexity of both initialization and evaluation of the AMG preconditioner is linear with respect to n_i, as well as the memory requirements. For the discrete operators considered in the numerical part of the paper, the AMG is known to be a very good preconditioner [8]. Define a block diagonal matrix

$$B_1 = \begin{pmatrix} B_{11} & & \\ & \ddots & \\ & & B_{PP} \end{pmatrix}. \tag{13}$$

The matrix B_1 is the additive Schwarz preconditioner for A with the minimal overlap of subdomains. It is a very simple preconditioner for A and can be easily parallelized. However, its efficiency is affected by the number of subdomains P and the relative width of the overlapping. Indeed, consider the simplest elliptic operator, $-\Delta + 1$, and a regular shaped DD. If the subdomain diameters and the intersection areas are of orders D and δD, $\delta < 1$, respectively, we get two independent lower estimates on the condition number of matrix $B_1 A$ [18,19]:

$$\mathrm{cond}(B_1 A) \geqslant C_D \frac{1}{\delta} \tag{14}$$

and

$$\mathrm{cond}(B_1 A) \geqslant C_\delta \frac{1}{D^2}. \tag{15}$$

The constants C_δ and C_D depend on δ and D, respectively, and do not depend on the other parameter as well as on the mesh size. Estimate (14) implies that for the fixed number of subdomains, the convergence of an iterative method will depend on the relative overlap. Similarly, estimate (15) implies that for the fixed relative overlap, the condition number will be very sensitive to the number of subdomains. In order to decrease the effect of the overlapping and eliminate dependence on the number of subdomains, we apply a correction step. The resulting two-stage preconditioner B is implicitly described by its action on a vector $u \in \Re^n$. Denoting $v = Bu$, we have

$$\begin{aligned} &w = B_1 u, \\ &r = (u - Aw), \quad v = w + B_2 r, \end{aligned} \tag{16}$$

where B_2 is a preconditioner described below. Formally, B may be presented as follows:

$$B = B_1 + B_2(I - AB_1). \tag{17}$$

Clearly, it is the nonsymmetric matrix even when $A = A^{\mathrm{t}}$ and $B_i = B_i^{\mathrm{t}}$, $i = 1, 2$. Therefore, for symmetric positive definite matrices the preconditioned conjugate gradient (PCG) method with preconditioner (17) may not be directly applied. However, if the initial vector u_0 in the PCG method is corrected as follows:

$$u_0 := u_0 + B_2(f - Au_0),$$

the method will converge provided B_2 has a general form (18). The reason for that is that the evaluation of B is equivalent to a symmetric three-stage algorithm (see, for example, [20,21]).

We choose the preconditioner B_2 as follows. Let the number of nontrivial rows in matrix $A_i = [A_{i1}, \ldots, A_{i,i-1}, A_{i,i+1}, \ldots, A_{i,P}]$ (without the diagonal block) be $\tilde{n}_i$. We define

$$\tilde{\boldsymbol{n}} = \sum_{i=1}^{P} \tilde{\boldsymbol{n}}_i + P,$$

and assume that rows of matrix A are ordered in such a way that in each A_i the nontrivial rows go first. Then the local *aggregation matrix* $T_{ii} \in \Re^{n_i \times (\tilde{n}_i + 1)}$ is given by

$$T_{ii} = \begin{pmatrix} I_i & 0 \\ 0 & e_i \end{pmatrix}, \quad e_i = (1, \ldots, 1)^{\mathrm{t}} \in \Re^{n_i - \tilde{n}_i},$$

where I_i is the identity matrix. We define the global block diagonal aggregation matrix T by

$$T = \begin{pmatrix} T_{11} & & \\ & \ddots & \\ & & T_{PP} \end{pmatrix},$$

and the aggregated stiffness matrix $\widetilde{A}$ by

$$\widetilde{A} = T^{\mathrm{t}} A T, \quad \widetilde{A} \in \Re^{\tilde{n} \times \tilde{n}}.$$

Let $\widetilde{B}$ be the conventional BSOR smoother [22,23] for $\widetilde{A}$. Then, the preconditioner B_2 is defined implicitly by

$$B_2 = T \widetilde{B} T^{\mathrm{t}}. \tag{18}$$

Note that B_2 is a smoother in a subspace of aggregated vectors $\{v \in \Re^n : v = T\tilde{v}, \tilde{v} \in \Re^{\tilde{n}}\}$. We motivate its construction as follows. The drawback of the additive Schwarz preconditioner B_1 is that it damps the error locally in subdomains but it does not control the error propagation on the global scale (15) and does not coordinate the error damping between neighboring subdomains (14). The approximate BSOR inversion of aggregated matrix $\widetilde{A}$ coordinates the mean subdomain values and matches the local errors in overlapping strips (see Fig. 3).

It may be shown that the preconditioner B is not worse than the additive Schwarz preconditioner B_1. Numerical experiments exhibit that the two-stage preconditioner is insensitive to the overlap regularity and the number of subdomains (in the case of shape regular DD). However, the associated condition number is affected by the relative width δ of the overlap (reciprocally) but to a much smaller extent than B_1. We remark that an alternative aggregation-based DD [24] exhibits δ^{-2}-growth of the condition number.

5. Parallelization of the iterative solution

The above two-stage preconditioner allows us to use a very simple scheme for its parallelization. We begin with assumptions used hereinafter.

First, we cannot keep the global stiffness matrix A and build up the corresponding AMG preconditioner on one processor. For an unstructured mesh with 10^6 tetrahedra, the amount of memory needed to keep A and the AMG preconditioner is estimated by 100 and 400 Mbytes, respectively. However, we assume that the aggregated matrix $\widetilde{A}$ may be stored on one processor. In our example, the storage for $\widetilde{A}$ does not exceed the doubled storage for the interface block of A which is estimated by 4 Mbytes for $P = 2$.

Second, the number of processors in our parallel computer is assumed to be not large, since the size $\tilde{n}$ of the aggregated matrix may approach n as the number of processors grows. It is prohibited by the first assumption.

Third, each iterative vector $v \in \Re^n$ is partitioned into P approximately equal disjoint subsets. The ith subset as well as the corresponding block matrix row $[A_{i1}, \ldots, A_{iP}]$ are treated by the only processor with rank i. The matrix–vector multiplication

$$A_{ii}v_i + \sum_{j \neq i}^{P} A_{ij}v_j$$

requires vectors v_j which are not known on the processor with rank i. These data are provided by the conventional "ghost" node technique which attaches to the ith processor copies of v_j-entries, $i \neq j$, needed for computation of $A_{ij}v_j$. Before any matrix evaluation, the values in the "ghost" nodes must be updated by the corresponding values from neighboring processors (procedure `Update`). In other words, `Update` is *the only* procedure used in communications between processors. It is implemented via asynchronous send/receive MPI routines.

Fourth, the arithmetical complexity of the subdomain preconditioner B_{ii} is assumed to be linear dependent on n_i. This implies the necessity of the even distribution of degrees of freedom among processors.

We consider two popular Krylov subspace methods, preconditioned GMRES and PCG. Both of them require three parallel operations: scalar product calculation, matrix–vector and preconditioner–vector multiplications. Since the number of "ghost" nodes participating in the matrix–vector multiplication is small compared to the size of A_{ii}, the amount of interprocessor communications is small as well.

The preconditioner incorporates three matrix–vector multiplications, with matrices B_1, A and B_2. The matrix B_1 is block diagonal and no communications are needed in its evaluation. The matrix B_2 is a result of a few BSOR iterations for the aggregated system. Implementation of BSOR sweeps presumes that the aggregated matrix is generated and partitioned (after a permutation) into blocks with the diagonal blocks being diagonal matrices. These operations are performed at the initialization step: the processor with rank i computes the local aggregation matrix T_i and the $(\tilde{n}_i + 1) \times \tilde{n}$ block row of $\widetilde{A}$ corresponding to ith submesh. The latter is not large and may be assembled on the processor with rank *root* to form the global matrix $\widetilde{A}$. Matrix $\widetilde{A}$ is sparse except P rows which have many entries. Hence, it may be effectively partitioned into blocks by a conventional sequential multi-coloring technique [25]. Multi-coloring virtually implies the permutation of $\widetilde{A}$ such that a diagonal block corresponding to a color is the diagonal matrix. Therefore, broadcasting multi-color data specifies implicitly the block partitioning of $\widetilde{A}$ on any processor. With the optimal relaxation parameter ω computed adaptively at the initialization step [23,26], the BSOR iterations are known to be a very efficient smoother [22,23]. Moreover, the parallel BSOR sweeps use only `Update` procedure for communications between processors [22].

The usage of the correction step with the aggregated system is three-fold. First, we can apply a very simple and efficient sequential method for multi-coloring the aggregated matrix, since both its size and the number of nonzero entries are very small compared to the size of the original matrix. Second, we essentially reduce the complexity of BSOR sweeps due to smaller size of $\widetilde{A}$. Third, for stiff problems, our experience shows that the aggregated matrix $\widetilde{A}$ is much *less stiff* than the original matrix A and the BSOR sweep turns out to be very efficient smoother.

6. Numerical experiments

Numerical experiments have been performed on a COMPAQ Tru64 Cluster with processors cadenced at 667 MHz.

6.1. A problem with anisotropic singularities

We consider the Poisson equation in a domain Ω with one reentrant corner, $\Omega = (0,1)^3 \setminus [0,0.5]^3$, and a singular right hand side $f(x) = 1/|x - x_0|$ where $x_0 = (0.5, 0.5, 0.5)$:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{19}$$

Properties of the solution to (19) are investigated in [27]. The solution possesses anisotropic edge singularities and a strong singularity at the reentrant corner point. The equation is discretized with the piecewise linear FEs. In order to estimate the discretization error, we replace the exact unknown solution u by a discrete solution $\bar{u}$ computed on a very fine adapted mesh. The fine mesh with approximately 1.5×10^6 tetrahedra was generated after 20 iterations of Algorithm 1. We shall not discuss here the stopping criteria for the algorithm and shall use 20 adaptive iterations in this subsection. As we shall see later, 10–14 steps is enough to get a saturation in the error reduction. In Table 1 we illustrate asymptotic result (6) with $\bar{u}$ instead of u. Proportionality of the error to $\mathcal{N}(\Omega_h)^{-2/3}$ (asymptotically optimal result) is clearly observed. Therefore, the meshes obtained in Algorithm 1 are quasi-optimal. A slightly smaller value of $\|\bar{u} - u_h\|_\infty \cdot \mathcal{N}(\Omega_h)^{2/3}$ in the last column is probably attributable to a discrepancy between $\|\bar{u} - u_h\|_\infty$ and $\|u - u_h\|_\infty$.

In Fig. 4 we study Algorithm 1 for $N_T = 100\,000$. The graphs represent the behavior of $\|\bar{u} - u_h\|_\infty$ and the mesh quality $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h)$ in the course of adaptive iterations. It is easy to see that the L_∞-error is about the same for all parallel runs ($P > 1$) but is slightly smaller for $P = 1$. This is explained by the higher quality of the final mesh. Note that the discrepancy between $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h)$ and 1 leads to the adaptive mesh with the number of elements bigger than N_T. Actually, on all adaptive iterations we have $\mathcal{N}(\widetilde{\Omega}_h) \sim 160\,000$. Another important remark is that on all iterations the mesh generator was capable of constructing the $|\boldsymbol{H}^h|$-quasi-optimal mesh with $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h) \gtrsim 0.1$.

In the next three tables, the arithmetical complexity of the mesh generation is studied. Since the QOM mesh is generated by a sequence of local modifications, their number can be one of characteristics for the arithmetical complexity. In Table 2 we exhibit the CPU time and the number of local modifications, #mod, performed at the last ($L = 20$) adaptive iteration. The number of local modifications is proportional to $\mathcal{N}(\Omega_h)$ while the complexity of each modification (in terms of CPU time per modification) is roughly proportional to $\mathcal{N}(\Omega_h)^{1/2}$. The latter dependence is probably attributable to our nonoptimal implementation of algorithms working with the ordered list of element qualities (see Section 2).

In Table 3, we show the number of local modifications for parallel runs and their execution time measured on processor with rank `root`. It is pertinent to note that only when the mesh is settled down (after 10th adaptive iteration), the processor time is proportional to the number of modifications. Before that, the number of modifications may not correlate with the arithmetical complexity (CPU time). We explain this by a different distribution of the local operations (Fig. 1).

Table 1
L_∞-norm of the error after 20 adaptive iterations

$\mathcal{N} \equiv \mathcal{N}(\Omega_h)$	9735	19 359	28 151	36 134	52 079	100 075	160 944
$\|\bar{u} - u_h\|_\infty$	0.025	0.017	0.013	0.0095	0.0066	0.0046	0.0024
$\|\bar{u} - u_h\|_\infty \cdot \mathcal{N}^{2/3}$	11.3	12.2	11.9	10.3	9.1	9.8	7.0

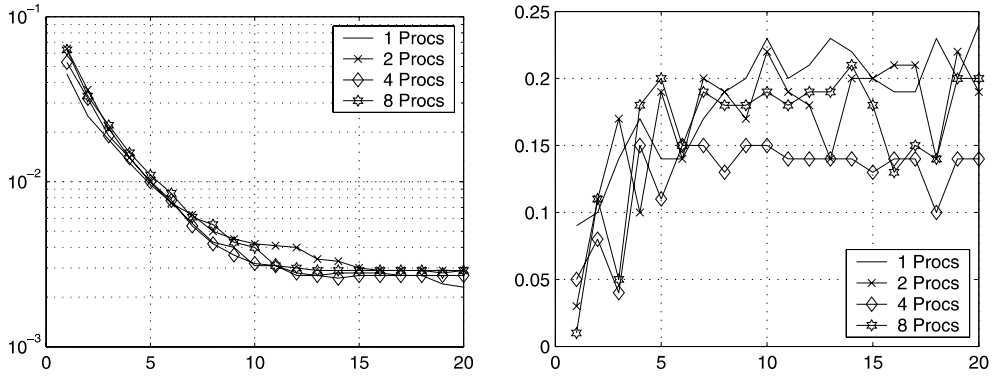

Fig. 4. The error $\|\bar{u} - u_h\|_\infty$ (on the left) and the mesh quality $Q(\boldsymbol{H}^h, N_T, \widetilde{\Omega}_h)$ (on the right) for $\mathcal{N}(\Omega_h) \sim 160000$.

Table 2
Arithmetical complexity of the mesh generation for $P = 1$

$\mathcal{N}(\Omega_h)$	9735	19 359	28 151	36 134	52 079	100 075	160 944
#mod	731	1155	3747	3097	6075	11 147	18 904
CPU time	1.2	1.9	4.7	4.6	11.2	25.3	58.6
$\frac{\text{CPU time}}{\#\text{mod}} 10^3$	1.6	1.6	1.3	1.5	1.8	2.3	3.1

Table 3
Number of mesh modifications and `root`'s CPU time for $\mathcal{N}(\Omega_h) \sim 160000$

L	$P = 1$		$P = 2$		$P = 4$		$P = 6$		$P = 8$	
	#mod	s	#mod	s	#mod	s	#mod	s	#mod	s
1	30 403	27.2	16 204	15.2	8752	7.9	6448	7.2	5484	6.5
2	30 850	39.6	18 273	27.1	12 340	14.5	8231	11.6	6937	10.8
4	28 027	48.7	22 585	38.4	9804	16.6	6570	12.8	5294	11.8
6	33 018	74.6	19 332	38.8	7643	15.0	4841	11.2	3061	9.4
10	32 986	87.9	13 776	30.2	4996	11.7	2602	8.7	1445	7.2
14	23 878	65.9	10 336	21.5	3741	9.8	1885	7.6	911	6.3
18	25 056	75.8	9802	20.1	2842	8.6	1536	6.3	434	5.7
20	18 904	58.6	7902	15.6	3265	9.7	1384	6.0	785	6.1

Another interesting observation is that the number of modifications per processor is not reciprocal to the number of processors. On the first adaptive iterations, the total arithmetical complexity, #mod · P, grows when P is increased. However, when the mesh is settled down, the total arithmetical complexity is decreased which results in the super linear speed-up (see Table 4). This is attributable to different order of local modifications. We recall that in a parallel run, we modify simultaneously P elements (and their neighbors). It turns out that if the mesh is not well adapted, the most efficient order is one in the sequential algorithm: repeatedly take the worst element in the global mesh. If the mesh is adapted, the more efficient order is that where the worst elements are taken from subgrids. Being separated in space, the local modifications turn to improve qualities of more elements in this case. Another consequence of that is the faster reduction of #mod per processor in the course of the parallel adaptation on 6 and 8 processors. We emphasize that the super linear speed-up of the mesh generation is an unexpected interesting feature of the parallel mesh generator.

Table 4
Speed-up of the mesh generation for $\mathcal{N}(\Omega_h) \sim 160000$

P	Adaptive iterations											
	1	2	3	4	5	6	7	8	9	10	11	12
2	1.8	1.5	1.3	1.3	1.4	1.9	2.2	2.6	2.8	2.9	3.1	3.4
3	2.4	1.9	2.0	2.1	2.9	3.4	4.4	5.3	4.9	5.4	5.1	6.1
4	3.4	2.7	2.3	2.9	3.7	5.0	5.4	6.1	6.5	7.5	7.0	7.5
6	3.8	3.4	3.3	3.8	4.6	6.7	7.5	9.3	10	10.1	9.7	10.1
8	4.2	3.7	3.5	4.1	5.4	7.9	9.4	10.5	11.8	12.2	12.5	12.8

In Tables 5 and 6 we show the arithmetical complexity for the sequential and parallel PCG methods, respectively. For the sequential method, the preconditioner is the V-cycle of AMG. For the parallel method, the preconditioner is given by (17) with four BSOR sweeps. Iterations are terminated when the initial residual is reduced by a factor of 10^6. We note that in both cases the number of iterations grows as $\mathcal{N}(\Omega_h)$ is increased. However, for the parallel solver, dependence on $\mathcal{N}(\Omega_h)$ is stronger albeit still very moderate: 10-fold increase in $\mathcal{N}(\Omega_h)$ only doubles the number of PCG iterations (#CG). It is attributable to the two-fold feature of method (17). From one side, the smaller order of B_{ii}, the better $\mathrm{cond}(B_{ii}A_{ii})$, $i = 1, \ldots, P$ is. On the other hand, for uniform meshes $\mathrm{cond}(BA)$ depends on the mesh size h, i.e., #CG is proportional to $h^{-1/2}$. In our experiments, the meshes are not uniform but a weak dependence of #CG on $\mathcal{N}(\Omega_h)$ is observed as well.

Another interesting observation for the sequential method is that the arithmetical complexity per iteration per element is increased as $\mathcal{N}(\Omega_h)$ grows (see Table 5). In contrast, for the parallel method it is decreased and saturated (see Table 6) resulting in a very good parallel scalability on the fine mesh. This is due to reduction of relative weight of interprocessor communications as $\mathcal{N}(\Omega_h)$ grows.

In Table 7 we exhibit the performance of the parallel solver in the course of adaptation for $\mathcal{N}(\Omega_h) \sim 160000$. The most important observation is that the number of iterations is insensitive to both the structure of the mesh (on the first steps it is more uniform) and the number of processors.

In Table 8 we exhibit speed-ups per PCG iteration for several adaptive steps. It is pertinent to note that at any adaptive iteration, $\mathcal{N}(\Omega_h)$ is not exactly the same for different number of processors. This and, possibly, cash effects may explain the super linear speed-up for $P = 2$. On the other hand, the speed-up for $P = 8$ is not very good because of small subdomain problems ($n_i \sim 4000$) resulting in domination of communications over computations.

Table 5
Number of PCG iterations and CPU time for $P = 1$ and 20th adaptive iteration

$\mathcal{N} = \mathcal{N}(\Omega_h)$	9735	19 359	28 151	36 134	52 079	100 075	160 944
#CG	8	11	10	13	12	12	13
CPU time	0.03	0.08	0.15	0.23	0.35	0.88	1.9
$\frac{\text{CPU time}}{\#\text{CG}\ \mathcal{N}} 10^7$	3.8	3.8	5.3	4.9	5.6	7.3	9

Table 6
Number of PCG iterations and CPU time for $P = 4$ and 20th adaptive iteration

$\mathcal{N} = \mathcal{N}(\Omega_h)$	9452	19 802	28 754	36 810	52 893	101 344	164 184
#CG	8	11	12	13	14	17	20
CPU time	0.05	0.10	0.12	0.15	0.22	0.53	0.88
$\frac{\text{CPU time}}{\#\text{CG}\ \mathcal{N}} 10^7$	6.6	4.6	3.5	3.1	3.0	3.1	2.7

Table 7
Number of PCG iterations and `root`'s CPU time for $\mathcal{N}(\Omega_h) \sim 160000$

L	$P=1$		$P=2$		$P=4$		$P=6$		$P=8$	
	#CG	s	#CG	s	#CG	s	#CG	s	#CG	s
1	11	0.65	15	0.38	16	0.28	17	0.28	17	0.25
2	14	1.40	17	0.75	18	0.58	20	0.52	19	0.45
4	14	1.92	18	1.38	20	0.88	19	0.65	20	0.60
6	12	1.68	17	1.37	19	0.83	19	0.68	19	0.63
10	13	1.90	18	1.37	19	0.85	19	0.72	19	0.67
14	14	2.08	17	1.37	20	0.93	18	0.73	20	0.70
18	14	2.38	18	1.48	20	0.95	19	0.72	19	0.65
20	13	1.88	18	1.38	20	0.88	20	0.73	19	0.67

Table 8
Speed-up of the solver for $\mathcal{N}(\Omega_h) \sim 160000$

P	$L=1$	$L=4$	$L=8$	$L=13$	$L=17$	$L=20$
2	2.3	1.8	1.7	1.7	2.1	1.9
3	2.1	2.4	2.6	2.7	2.7	2.7
4	3.4	3.1	3.2	3.2	3.6	3.3
6	3.6	4.0	3.9	3.9	4.1	4.0
8	4.0	4.6	4.4	4.4	4.9	4.1

6.2. A problem with boundary layers

The second example is the singularly perturbed convection–diffusion equation in the back-step domain $\Omega = (0,1)^3 \setminus [0,0.5]^2 \times (0,1)$:

$$\begin{aligned} -0.01\Delta u + \frac{\partial u}{\partial x_1} &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned} \tag{20}$$

The solution to (20) possesses a severe exponential boundary layer at $x_1 = 1$ and parabolic boundary layers at $x_2 = 0$, $x_2 = 1$, $x_3 = 0$ and $x_3 = 1$, as well as a weak interior layer at $x_2 = 0.5$ (for details we refer to [6]). It is evident that QOMs have to be anisotropic in the boundary layers. Similarly to the previous example, we replace the exact solution u by its piecewise linear FE approximation $\bar{u}$ on a very fine adapted mesh with $\mathcal{N}(\Omega_h) \sim 0.6 \times 10^6$. In Table 9 we show the dependence of error $\|\bar{u} - u_h\|_\infty$ on $\mathcal{N}(\Omega_h)$. Asymptotic result (6) is confirmed as well but with another factor due to the larger value of $|\Omega|_{|\boldsymbol{H}|}$.

In Table 10 the arithmetical complexity of the sequential mesh generation at the 30th adaptive iteration is shown. We also remark a rough proportionality of #mod to $\mathcal{N}(\Omega_h)$ and proportionality of the CPU time per modification to $\mathcal{N}(\Omega_h)^{1/2}$.

In Fig. 5 we compare the performances of Algorithm 1 for different number of processors. Note that the error is stabilized after a larger number of adaptive steps compared to the previous example. The parallel mesh generator is capable of constructing $|\boldsymbol{H}^h|$-QOMs with $Q(|\boldsymbol{H}^h|, N_T, \widetilde{\Omega}_h) \gtrsim 0.1$.

Therefore, it provides the same value of $\|\bar{u} - u_h\|_\infty$ as the sequential algorithm. Moreover, the QOM is generated faster in the parallel algorithm (see Table 11). The reasons for the super linear speed-up have been discussed in the previous section.

In Tables 12 and 13 we exhibit the performance of the preconditioned GMRES method for the reduction of the initial residual by a factor of 10^6. We observe two-fold increases of the number of GMRES iterations (#GMRES) for 15-fold and 8-fold increases of $\mathcal{N}(\Omega_h)$, for the sequential and parallel solvers, respectively.

Table 9
L_∞-norm of the error after 30 adaptive iterations

$\mathcal{N} = \mathcal{N}(\Omega_h)$	9531	18 798	36 175	70 344	140 392
$\|\bar{u} - u_h\|_\infty$	0.057	0.031	0.022	0.016	0.010
$\|\bar{u} - u_h\|_\infty \cdot \mathcal{N}^{2/3}$	25.5	21.8	23.9	27.1	26.8

Table 10
Arithmetical complexity of the mesh generation for $P = 1$

$\mathcal{N}(\Omega_h)$	9531	18 798	36 175	70 344	140 392
#mod	7784	10 346	20 147	35 731	84 931
CPU time	9.0	12.9	28.1	83.6	290
$\frac{\text{CPU time}}{\#\text{mod}} 10^3$	1.1	1.2	1.4	2.3	3.4

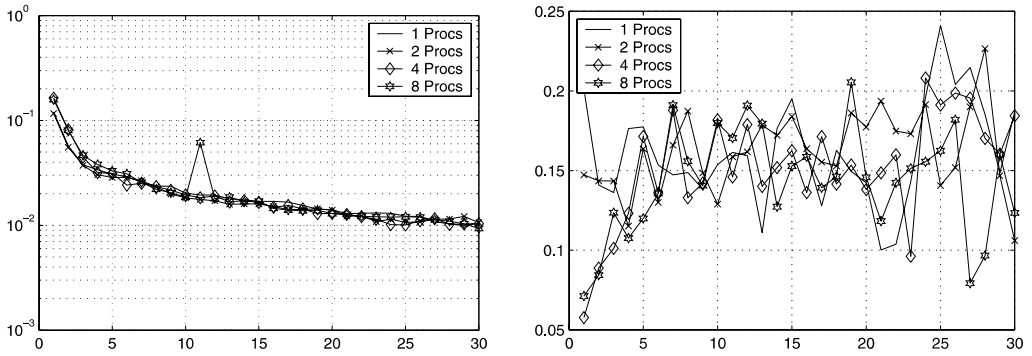

Fig. 5. The error $\|\bar{u} - u_h\|_\infty$ (on the left) and the mesh quality $Q(\boldsymbol{H}^h, N_T, \widetilde{\Omega}_h)$ (on the right) for $\mathcal{N}(\Omega_h) \sim 140\,000$.

Table 11
Speed-up for the mesh generation, $\mathcal{N}(\Omega_h) \sim 140\,000$

P	$L = 1$	$L = 10$	$L = 20$	$L = 30$
2	1.9	3.6	3.4	5.3
4	4.2	9.0	10.7	13.2
8	7.8	15	26.6	31.2

Table 12
Number of GMRES iterations and CPU time for $P = 1$ on 30th adaptive iteration

$\mathcal{N} = \mathcal{N}(\Omega_h)$	9531	18 798	36 175	70 344	140 392
#GMRES	9	9	11	13	16
CPU time	0.03	0.07	0.23	0.8	2.3
$\frac{\text{CPU time}}{\#\text{GMRES}\ \mathcal{N}} 10^7$	3.5	4.1	5.8	8.7	9.8

Both dependencies are very weak albeit the convergence of the parallel solver is more sensitive to $\mathcal{N}(\Omega_h)$. On the other hand, the parallel solver demonstrates the excellent arithmetical scalability.

In Table 14 we show the solver performance for several adaptive iterations and different number of processors. As in the previous example, the number of iterations is slightly dependent on the number of

Table 13
Number of GMRES iterations and root's CPU time for $P = 4$ on 30th adaptive iteration

$\mathcal{N} = \mathcal{N}(\Omega_h)$	17 707	34 061	67 949	134 123
#GMRES	9	12	14	17
CPU time	0.07	0.12	0.25	0.72
$\frac{\text{CPU time}}{\#\text{GMRES}\ \mathcal{N}} 10^7$	4.4	2.9	2.6	3.1

Table 14
Number of GMRES iterations and root's CPU time for $\mathcal{N}(\Omega_h) \sim 140000$

L	$P = 1$		$P = 2$		$P = 4$		$P = 8$	
	#GMRES	s	#GMRES	s	#GMRES	s	#GMRES	s
3	9	1.0	10	0.55	11	0.43	11	0.28
10	12	1.7	12	0.82	13	0.45	14	0.35
20	14	1.8	15	0.98	16	0.65	15	0.42
30	16	2.2	16	1.12	17	0.72	16	0.47

Table 15
Speed-up for the solver for $\mathcal{N}(\Omega_h) \sim 140000$

P	$L = 3$	$L = 10$	$L = 20$	$L = 30$
2	2.0	2.1	1.9	1.9
4	3.0	4.1	3.2	3.2
8	4.4	5.8	4.6	4.7

processors. Good speed-ups per iteration (see Table 15) are observed except for the case $P = 8$ where the size of subproblems becomes too small ($n_i \sim 3500$).

6.3. A problem with jumping diffusion coefficients

The last example is the diffusion equation in the unit cube $\Omega = (0,1)^3$:

$$\begin{aligned} -\mathrm{div}(a(x)\nabla u) &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{21}$$

where $a(x)$ is a positive function bounded from below. We study three choices for $a(x)$. First we take the uniform isotropic case with $a(x) \equiv 1$. Second, we consider the checkerboard-like jumps in the diffusion tensor:

$$a(x) = \begin{cases} 1 & x \in \Omega_1 \cup \Omega_2 \cup \Omega_3 \cup \Omega_4, \\ 1000 & \text{otherwise}, \end{cases}$$

where $\Omega_1 = (0,1/2)^3$, $\Omega_2 = (1/2,1)^2 \times (0,1/2)$, $\Omega_3 = (1/2,1) \times (0,1/2) \times (1/2,1)$ and $\Omega_4 = (0,1/2) \times (1/2,1)^2$. Third, we consider the piecewise constant quasi-random diffusion tensor given by

$$a(x)|_{e_i} = \begin{cases} 1 & \sin(1000x_{i1} + 3000x_{i2} + 5000x_{i3}) > 0, \\ 1000 & \text{otherwise}, \end{cases}$$

where $x_i = (x_{i1}, x_{i2}, x_{i3})$ is the barycenter of tetrahedron e_i. We notice that in the last two cases it is impossible to construct a conformal QOM since the Hessian has very strong jumps. Therefore, in the vicinity

Table 16
Number of PCG iterations and root's CPU time

P	Case 1		Case 2		Case 3	
	#CG	s	#CG	s	#CG	s
2	14	1.22	12	1.02	13	1.13
4	14	0.58	12	0.58	14	0.62
8	14	0.38	13	0.35	13	0.33

of the jumps, the QOM has to have neighboring equilateral elements with very different sizes which is impossible due to mesh conformity. This is the reason why we study only the solver behavior and consider a uniform cubic mesh with $h = 1/32$ and split each cell into six tetrahedra, i.e., $\mathcal{N}(\Omega_h) = 196\,608$. In addition, we artificially split the mesh elements between processors either by plane $x_1 = 0.5$ (for $P = 2$), or by planes $x_1 = 0.5$ and $x_2 = 0.5$ (for $P = 4$), or by planes $x_1 = 0.5$, $x_2 = 0.5$ and $x_3 = 0.5$ (for $P = 8$). Therefore, the corresponding subdomains (in the Schwarz method) overlap along interfaces where $a(x)$ has jumps.

Table 16 shows that the convergence rate of the PCG method and the execution time do not depend on the jumps in the diffusion coefficient and on the number of subdomains. Moreover, in all cases, the solver exhibits the good parallel properties.

7. Conclusions

The parallel technique for the adaptive solution of 3D boundary value problems is considered. It includes two basic parallel algorithms, the tetrahedral mesh generation and the iterative solution. The algorithms are *independent* of the underlying boundary value problem and have good parallel properties.

The input data for the adaptive mesh generator are a mesh and the corresponding discrete solution. The output is a QOM provided the discrete Hessian approaches the differential one. Numerical experiments confirm the theoretically predicted asymptotic error estimates for QOMs. The parallel mesh generation reveals the super linear speed-up on a few last adaptive iterations.

The problem independent parallel iterative technique is based on the sequential black-box (problem independent) AMG preconditioner. The AMG preconditioner enters the first (block Jacobi) part of the suggested two-stage preconditioner. The second (correction) part is a few BSOR sweeps in the space of aggregated vectors. It eliminates the convergence dependence on the number of subdomains, problem coefficients, and dumps its sensitivity to the mesh size. The additional data required by the preconditioner are confined to the mesh graph.

Acknowledgement

The authors are grateful to Prof. M. Sarbey for the assistance in performing numerical experiments.

References

[1] E. D'Azevedo, On optimal triangular meshes for minimizing the gradient error, Numerische Mathematik 59 (1991) 321–348.
[2] S. Rippa, Long and thin triangles can be good for linear interpolation, SIAM J. Numer. Anal. 29 (1992) 257–270.
[3] A. Agouzal, K. Lipnikov, Y. Vassilevski, Adaptive generation of quasi-optimal tetrahedral meshes, East–West J. Numer. Math. 7 (1999) 223–244.
[4] H. Borouchaki, P.-L. George, F. Hecht, P. Laug, B. Mohammadi, E. Saltel, Mailleur bidimensionnel de Delaunay gouverne par une carte de metriques, Technical Report R2760, INRIA, 1995.

[5] J. Dompierre, M.-G. Vallet, M. Fortin, W. Habashi, D. Ait-Ali-Yahia, A. Tam, Edge-based mesh adaptation for CFD, Technical Report R95-73, CERCA, 1995.
[6] G. Buscaglia, E. Dari, Anisotropic mesh optimization and its application in adaptivity, Int. J. Numer. Meth. Engrg. 40 (1997) 4119–4136.
[7] K. Stüben, Algebraic multigrid (AMG): experiences and comparisons, Appl. Math. Comput. 13 (1983) 419–452.
[8] T. Grauschopf, M. Griebel, H. Regler, Additive multilevel preconditioners based on bilinear interpolation, matrix dependent geometric coarsening and algebraic multigrid coarsening for second order elliptic PDEs, Appl. Numer. Math. 23 (1997) 63–96.
[9] K. Lipnikov, Y. Vassilevski, Optimal triangulations: existence, approximation and double differentiation of P_1 finite element functions, in: Electronic Proceedings of the Second Workshop on Grid Generation, Moscow, June 2002. Available from <http://www.ccas.ru/gridgen/ggta02/abstracts.html>, to appear in Comp. Math. Math. Phys. (2003).
[10] P. Zavattieri, E. Dari, G. Buscaglia, Optimization strategies in unstructured mesh generation, Int. J. Numer. Meth. Engrg. 39 (1996) 2055–2071.
[11] Y. Vassilevski, K. Lipnikov, Adaptive algorithm for generation of quasi-optimal meshes, Comp. Math. Math. Phys. 39 (1999) 1532–1551.
[12] G. Buscaglia, A. Agouzal, P. Ramirez, E. Dari, On Hessian recovery and anisotropic adaptivity, in: Proceedings of ECCOMAS 98 (the 4th European CFD Conference), Athens, Greece, 1998, pp. 403–407.
[13] A. Agouzal, Y. Vassilevski, On a discrete Hessian recovery for P_1 finite elements, East–West J. Numer. Math. 10 (1) (2002) 1–12.
[14] I. Foster, Designing and building parallel programs, Addison-Wesley Publishing Company, New York, 1995.
[15] A. Pothen, H. Simon, K. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM J. Math. Anal. Appl. 11 (1990) 430–452.
[16] R. Williams, Performance of dynamic load balancing algorithms for unstructured mesh calculations, Concurrency: Pract. Exp. 3 (1992) 457–481.
[17] Y. Vassilevski, Iterative solvers for the implicit parallel accurate reservoir simulator (IPARS). II. Parallelization issues, Technical Report 00-33, TICAM, The University of Texas at Austin, 2000.
[18] S. Brenner, Lower bounds for two-level additive Schwarz preconditioners with small overlap, SIAM J. Sci. Comp. 21 (2000) 1657–1669.
[19] A. Quarteroni, A. Valli, Domain Decomposition Methods for Partial Differential Equations, Clarendon Press, Oxford, 1999.
[20] J. Mandel, Hybrid domain decomposition method with unstructured subdomains, Contemp. Math. 157 (1994) 103–112.
[21] J. Mandel, M. Bresina, Balancing domain decomposition for problems with large jumps in coefficients, Math. Comp. 65 (1996) 1387–1401.
[22] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H.V. der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA, 1994.
[23] D. Young, Iterative Solution of Large Linear Systems, Academic Press, New York, 1971.
[24] E. Jenkins, C. Kees, C. Kelley, C. Miller, An aggregation-based domain decomposition preconditioner for groundwater flow, SIAM J. Sci. Comput. 23 (2001) 430–441.
[25] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Co, Boston, 1996.
[26] D. Kincaid, J. Respess, D. Young, R. Grimes, Algorithm 586 ITPACK 2C: a fortran package for solving large sparse linear systems by adaptive accelerated iterative methods, ACM Trans. Math. Software 8 (1982) 302–322.
[27] T. Apel, Anisotropic Finite Elements: Local Estimates and Applications, Teubner, Stutgart, 1999.