Available online at www.sciencedirect.com

ScienceDirect

Computers and Structures 85 (2007) 489–498

Computers & Structures

www.elsevier.com/locate/compstruc

Static load balancing applied to Schur complement method

Ondřej Medek [a,*], Jaroslav Kruis [b], Zdeněk Bittnar [b], Pavel Tvrdík [a]

[a] *Department of Computer Science and Engineering, Faculty of Electrical Engineering, 16100 Prague, Czech Republic*
[b] *Department of Structural Mechanics, Faculty of Civil Engineering, Czech Technical University, Prague, Czech Republic*

Received 12 September 2005; accepted 2 August 2006
Available online 25 October 2006

Abstract

A finite element method often leads to large sparse symmetric and positive definite systems of linear equations. We consider parallel solvers based on the Schur complement method on homogeneous parallel machines with distributed memory. A finite element mesh is partitioned by graph partitioning. Such partitioning results in submeshes with similar numbers of elements and, consequently, submatrices of similar sizes. The submatrices are partially factorised. The time spent on the partial factorisation can be different, i.e., disbalanced, because methods exploiting the sparsity of submatrices are used. This paper proposes a Quality Balancing heuristic that modifies classic mesh partitioning so that the partial factorisation times are balanced, which saves overall computation time, especially for time dependent mechanical and nonstationary transport problems.
© 2006 Elsevier Ltd. All rights reserved.

Keywords: Domain decomposition; Finite element methods; Mesh partitioning; Multilevel graph partitioning; Parallel solvers; Static load balancing; Schur complement method; Time-dependent problems

1. Introduction

Parallel computers have become a popular and widespread tool for solving large scientific and engineering problems. Parallelisation of sequential algorithms may involve considerable changes. Algorithms for solving systems of linear equations are an important example. In this paper, only the finite element (FE) method [1,2], and large sparse symmetric and positive definite linear systems are considered. Parallelisation of classic direct methods for solving linear systems, such as the $\boldsymbol{LDL}^{\mathrm{T}}$ factorisation, is not easy and only partial success has been achieved [3]. The parallelisation of iterative methods, such as the conjugate gradient method, is easier. However, convergence properties are not always optimal. Methods based on at least two level approaches have significantly better properties. Domain decomposition methods are an example of such methods [4–7].

The parallel solvers should be faster than sequential ones. Ideal time reductions cannot be obtained if the processor loads are not balanced. Load balancing describes the fact that all used processors execute an identical or nearly identical number of operations. A slightly disbalanced load of processors is acceptable in problems where the linear system is solved only once. A typical example of such problems is the static linear problem. On the other hand, there are problems such as creep analysis or nonstationary heat transfer where numerous time steps are needed. Each time step involves the solving of a linear system in the case of the implicit method. Every slightly disbalanced load in such cases is significantly amplified and the best possible load balancing is desirable.

In this paper, only the Schur complement method is considered for domain decomposition. Typically, the FE mesh is represented by a graph that is partitioned by graph partitioning. It produces submeshes with similar numbers of elements and nodes. Consequently, submatrices of similar sizes are assembled on each submesh. The Schur complements are computed by partial factorisation from the submatrices

* Corresponding author.
E-mail addresses: xmedeko@fel.cvut.cz (O. Medek), jk@cml.fsv.cvut.cz (J. Kruis), bittnar@fsv.cvut.cz (Z. Bittnar), tvrdik@fel.cvut.cz (P. Tvrdík).

0045-7949/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compstruc.2006.08.025

and the reduced problem is usually solved with a suitable iterative method. Common methods for partial factorisation exploit the structure of submatrices, i.e., the number and positions of nonzero matrix entries [8,9]. Therefore, the computational complexity of the partial factorisation depends more on the structure than on the size of a submatrix and classic mesh partitioning may not result in good load balancing. This problem has already been observed [10,11], but has not yet been resolved satisfactorily.

This paper deals with a static load balancing technique for the Schur complement method and homogeneous parallel computers with distributed memory. This technique is called a Quality Balancing (QB) heuristic and its preliminary version appeared in [12]. As the QB heuristic prolongs the time spent on mesh partitioning, its advantages appear in problems where several linear systems with the same structure need to be solved.

This paper is organised as follows: Sections 2 and 3 describe time dependent mechanical and nonstationary transport problems. Sections 4 and 5 explain Schur complement methods for a parallel solution of a linear system. Classic mesh partitioning is described in Section 6 and the QB heuristic is proposed in Section 7. Some illustrative results of solutions to practical problems are presented in Section 8. Finally, Section 9 concludes the paper.

2. A time dependent mechanical problem

For the purposes of this paper, a *time dependent mechanical problem* denotes a problem that depends on time, but the inertial forces are negligible. A typical example of such a problem is creep analysis [1].

Time dependent mechanical problems are usually formulated in the rate form

$$\boldsymbol{K}\dot{\boldsymbol{r}} = \dot{\boldsymbol{f}} + \int_V \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\dot{\boldsymbol{\varepsilon}}^{ir}\,\mathrm{d}V, \tag{1}$$

where $\boldsymbol{K}$ denotes the stiffness matrix of the problem (domain), $\boldsymbol{r}$ denotes the vector of nodal displacements, $\boldsymbol{f}$ denotes the vector of prescribed nodal forces, $\dot{\boldsymbol{\varepsilon}}_{ir}$ denotes the irreversible strains, $\boldsymbol{D}$ denotes the stiffness matrix of the material, $\boldsymbol{B}$ denotes the strain–displacement matrix, and V denotes the volume of the domain considered. The superimposed dot denotes the time derivative. Eq. (1) is solved by a numerical method that discretises time. The number of time steps is denoted by N_s. The basic steps are summarised in Table 1. The method described in Table 1 is explicit and the particular expression for irreversible strain increments depends on the material model used. The algorithm can be applied to visco-plastic problems as well as creep analysis. The increments of irreversible strains are not specified in more detail in Table 1 because they are not the focus of this paper.

The most time consuming part of the algorithm is the computation of displacement increments that consist of solving a system of linear equations

$$\boldsymbol{K}\Delta\boldsymbol{r}_{i+1} = \Delta\boldsymbol{f}_{i+1} + \Delta\boldsymbol{f}^{ir}_{i+1}. \tag{2}$$

Table 1
An algorithm for time-dependent mechanical problems

For $i = 0$ until $i \leqslant N_s$ compute	
Increments of irreversible strains	$\Delta\boldsymbol{\varepsilon}^{ir}_i$
Increments of internal nodal forces	$\Delta\boldsymbol{f}^{\,ir}_{i+1} = \int_V \boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\Delta\boldsymbol{\varepsilon}^{ir}_i\,\mathrm{d}V$
Increments of external (prescribed) nodal forces	$\Delta\boldsymbol{f}_{i+1} = \boldsymbol{f}(t_{i+1}) - \boldsymbol{f}(t_i)$
Increments of displacements	$\Delta\boldsymbol{r}_{i+1} = \boldsymbol{K}^{-1}(\Delta\boldsymbol{f}_{i+1} + \Delta\boldsymbol{f}^{ir}_{i+1})$
New vector of displacements	$\boldsymbol{r}_{i+1} = \boldsymbol{r}_i + \Delta\boldsymbol{r}_{i+1}$
Total strain increments (previous total strain $\boldsymbol{\varepsilon}_i$ is stored)	$\Delta\boldsymbol{\varepsilon}_{i+1} = \boldsymbol{B}\boldsymbol{r}_{i+1} - \boldsymbol{\varepsilon}_i$
Stress increments	$\Delta\boldsymbol{\sigma}_{i+1} = \boldsymbol{D}(\Delta\boldsymbol{\varepsilon}_{i+1} - \Delta\boldsymbol{\varepsilon}^{ir}_i)$
New stresses	$\boldsymbol{\sigma}_{i+1} = \boldsymbol{\sigma}_i + \Delta\boldsymbol{\sigma}_{i+1}$

3. Nonstationary transport problems

Nonstationary transport problems are also considered in this paper. They are similar to time dependent mechanical problems in the sense that several time steps are used to solve them [2].

Basic facts can be shown on an example of a heat transfer with constant coefficients, which is described by the equation

$$k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) = \rho c\frac{\partial T}{\partial t}, \tag{3}$$

where T denotes the temperature, k denotes the coefficient of conductivity, ρ denotes the density of the material, and c denotes the thermal capacity. After space discretisation, a system of ordinary differential equations is obtained in the form

$$\boldsymbol{C}\frac{\mathrm{d}\boldsymbol{d}}{\mathrm{d}t} + \boldsymbol{K}\boldsymbol{d} = \boldsymbol{f}, \tag{4}$$

where $\boldsymbol{C}$ denotes the capacity matrix, $\boldsymbol{K}$ denotes the conductivity matrix, $\boldsymbol{d}$ denotes the vector of nodal unknowns, and $\boldsymbol{f}$ denotes the vector of prescribed fluxes. This system of equation (4) is then discretised in time using

$$\boldsymbol{d}_{i+1} = \boldsymbol{d}_i + \Delta t\boldsymbol{v}_{i+\theta}, \tag{5}$$

$$\boldsymbol{v}_{i+\theta} = (1-\theta)\boldsymbol{v}_i + \theta\boldsymbol{v}_{i+1}, \tag{6}$$

where the first time derivative of nodal values is denoted by $\boldsymbol{v}$. Substitution of (5) and (6) into (4) results in the system of linear equations

$$(\boldsymbol{C} + \Delta t\theta\boldsymbol{K})\boldsymbol{v}_{i+1} = \boldsymbol{f}_{i+1} - \boldsymbol{K}(\boldsymbol{d}_i + \Delta t(1-\theta)\boldsymbol{v}_i) \tag{7}$$

with unknown vector $\boldsymbol{v}_{i+1}$. Nodal values are obtained from Eqs. (5) and (6).

4. The Schur complement method

This section summarises only the basic facts about the Schur complement method [4,6,7]. It is based on a special form of the linear system of equations

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y} \tag{8}$$

that has to be written in the form

$$\begin{pmatrix} \boldsymbol{A}_{II}^{(1)} & & \boldsymbol{0} & & \boldsymbol{A}_{IB}^{(1)} \\ & \boldsymbol{A}_{II}^{(2)} & & & \boldsymbol{A}_{IB}^{(2)} \\ \boldsymbol{0} & & \boldsymbol{A}_{II}^{(3)} & & \boldsymbol{A}_{IB}^{(3)} \\ \vdots & & & \ddots & \vdots \\ & & & \boldsymbol{A}_{II}^{(k)} & \boldsymbol{A}_{IB}^{(k)} \\ \boldsymbol{A}_{BI}^{(1)} & \boldsymbol{A}_{BI}^{(2)} & \boldsymbol{A}_{BI}^{(3)} & \cdots \quad \boldsymbol{A}_{BI}^{(k)} & \boldsymbol{A}_{BB} \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_I^{(1)} \\ \boldsymbol{x}_I^{(2)} \\ \boldsymbol{x}_I^{(3)} \\ \vdots \\ \boldsymbol{x}_I^{(k)} \\ \boldsymbol{x}_B \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_I^{(1)} \\ \boldsymbol{y}_I^{(2)} \\ \boldsymbol{y}_I^{(3)} \\ \vdots \\ \boldsymbol{y}_I^{(k)} \\ \boldsymbol{y}_B \end{pmatrix}, \tag{9}$$

where $\boldsymbol{A}_{II}^{(j)}$, $\boldsymbol{A}_{IB}^{(j)}$ and $\boldsymbol{A}_{BI}^{(j)}$ denote the submatrices of the jth subdomain, $\boldsymbol{x}_I^{(j)}$ denotes the vector of unknowns defined at the inner nodes of the jth subdomain, $\boldsymbol{y}_I^{(j)}$ denotes the right-hand-side vector defined at the inner nodes of the jth subdomain, $\boldsymbol{x}_B$ denotes the vector of all unknowns defined the on boundaries, and $\boldsymbol{y}_B$ denotes the right-hand-side vector defined at the boundary nodes. The system of equation (9) can be obtained by the FE method applied to the problem decomposed into k subdomains. The subscript I denotes an inner quantity, while B denotes a boundary one. Clearly, IB and BI denote coupling between inner and boundary quantities.

If all diagonal submatrices $\boldsymbol{A}_{II}^{(1)}$ to $\boldsymbol{A}_{II}^{(k)}$ are regular, a particular subvector of unknowns, $\boldsymbol{x}$, can be expressed in the form

$$\boldsymbol{x}_I^{(j)} = (\boldsymbol{A}_{II}^{(j)})^{-1}(\boldsymbol{y}_I^{(j)} - \boldsymbol{A}_{IB}^{(j)}\boldsymbol{x}_B). \tag{10}$$

The existence of the inverse matrices $(\boldsymbol{A}_{II}^{(j)})^{-1}$ is guaranteed by a special ordering of unknowns, which is used in order to eliminate the inner unknowns. After repeating this process for each row of (9) except the last one, the reduced system of equations becomes

$$\left(\boldsymbol{A}_{BB} - \sum_{j=1}^{k}\boldsymbol{A}_{BI}^{(j)}(\boldsymbol{A}_{II}^{(j)})^{-1}\boldsymbol{A}_{IB}^{(j)}\right)\boldsymbol{x}_B = \boldsymbol{y}_B - \sum_{j=1}^{k}\boldsymbol{A}_{BI}^{(j)}(\boldsymbol{A}_{II}^{(j)})^{-1}\boldsymbol{y}_I^{(j)} \tag{11}$$

where $\boldsymbol{x}_B$ denotes the vector of unknowns in the reduced system. The original system of equations contains a significantly larger number of unknowns than the reduced system (11). This reduction of unknowns is an important feature of the Schur complement method. When the vector $\boldsymbol{x}_B$ is computed, all vectors of inner unknowns $\boldsymbol{x}_I^{(j)}$ are established from Eq. (10).

The system of equation (11) is also called the coarse grid problem, because only unknowns defined on the boundaries are used. Each subdomain can be considered as one super-element and the coarse grid is assumed to consist of those super-elements. The system of equation (11) is usually solved by a suitable iterative method [4,7].

Systems of linear equations (2) and (7) described in the previous two sections can be solved by the Schur complement method. The original FE mesh is partitioned into several smaller submeshes. Each submesh is dealt with as one independent problem. This means that submatrices and subvectors are assembled independently on each submesh. Continuity is enforced during the solution by the Schur complement method, because special ordering of unknowns is used on subdomains. Unknowns defined at the inner nodes are ordered first, while unknowns defined at the boundary nodes are ordered at the end. The special ordering results in the required form of the system of equations.

Each subdomain contains a part of the matrix of the original system (8) and can be expressed in the form

$$\boldsymbol{A}^{(j)} = \begin{pmatrix} \boldsymbol{A}_{II}^{(j)} & \boldsymbol{A}_{IB}^{(j)} \\ \boldsymbol{A}_{BI}^{(j)} & \boldsymbol{A}_{BB}^{(j)} \end{pmatrix} \tag{12}$$

Inverse matrix $(\boldsymbol{A}_{II}^{(j)})^{-1}$ in Eq. (10) is not computed explicitly, because it is a dense matrix in general. Instead, a partial $\boldsymbol{LDL}^{\mathrm{T}}$ factorisation is computed, because the linear systems (2) and (7) are symmetric and positive definite. This leads to the factorised submatrix $\boldsymbol{A}_{II}^{(j)}$, the eliminated submatrices $\boldsymbol{A}_{BI}^{(j)}$ and $\boldsymbol{A}_{IB}^{(j)}$, and the modified submatrix $\widetilde{\boldsymbol{A}}_{BB}^{(j)}$ which can be expressed in the form

$$\widetilde{\boldsymbol{A}}_{BB}^{(j)} = \boldsymbol{A}_{BB}^{(j)} - \boldsymbol{A}_{BI}^{(j)}(\boldsymbol{A}_{II}^{(j)})^{-1}\boldsymbol{A}_{IB}^{(j)} \tag{13}$$

The matrix $\widetilde{\boldsymbol{A}}_{BB}^{(j)}$ is not computed with respect to Eq. (13) but it is obtained by partial factorisation. The factorisation is partial, because matrix entries from the lower triangular part of the submatrix $\boldsymbol{A}_{BB}^{(j)}$ are not eliminated at the subdomain level. The partial factorisation can be based on the envelope (skyline) storage scheme or on the sparse storage scheme [8]. If the partial factorisations on particular subdomains have similar computational complexity, good load balancing is obtained. On the other hand, if the computational complexity of the partial factorisations on individual subdomains differs, load balancing is poor, as is the time of the whole computation. Repeated use of this disbalanced partial factorisation leads to the even poorer performance of the method.

A significant reduction of the computational complexity of the partial factorisation can be achieved by the proper reordering of unknowns. Of course, such a reordering subsequently has a strong impact on load balancing. It must be noted that boundary unknowns have to be ordered last. Since a reordering of nodes implies a reordering of unknowns, only the reordering of nodes is considered in the following text. The reordering methods used in this paper are described in Section 6.2.

5. Estimation of computational complexity

Let n_I and n_B denote the number of inner and boundary unknowns, respectively, in a subdomain. The boundary unknowns come last. The envelope or sparse storage scheme [8] stores nonzero and possibly some zero entries in a submatrix (12). These are the submatrix entries inside an envelope in the case of the envelope storage scheme, or submatrix entries computed by a symbolical factorisation [8,9], in the case of the sparse storage scheme. Due to the symmetry, only the lower part of the submatrices and the main diagonals are stored.

If $\eta(A_{*i}^{(j)})$ denotes the number of entries stored by a storage scheme in the column i under the main diagonal, then the number of floating-point operations (FLOPs) during the partial factorisation of $A^{(j)}$ is

$$OP(A^{(j)}) = \sum_{i=1}^{n_I} \frac{1}{2}(\eta(A_{*i}^{(j)}) - 1)(\eta(A_{*i}^{(j)}) + 2), \tag{14}$$

see [8]. One FLOP can be a multiplication or division or a "multiply–add" operation. $OP(A^{(j)})$ is taken as the estimate of the computational complexity of the partial factorisation of $A^{(j)}$.

6. Classic mesh partitioning

One element of an FE mesh consists of several nodes. One node may belong to several elements. Each node contains unknowns.

Definition. A graph $G = (V, E)$ consists of a finite set of vertices V and a finite set of edges E, which are unordered pairs of vertices $(u, v), u, v \in V$. The number of vertices in the graph is $|V|$ and the number of edges is $|E|$.

An FE mesh is usually represented by a *dual graph* $G^{\mathrm{D}} = (V^{\mathrm{D}}, E^{\mathrm{D}})$ and a *nodal graph* $G^{\mathrm{N}} = (V^{\mathrm{N}}, E^{\mathrm{N}})$. The vertices in the dual graph represent the finite elements and two vertices are adjacent if and only if the corresponding elements share a common boundary, i.e., a surface in 3D or an edge in 2D. The vertices in the nodal graph represent the nodes. Two nodes are adjacent if and only if they belong to the same element. An example of a quadrilateral mesh and its dual and nodal graph is shown in Fig. 1.

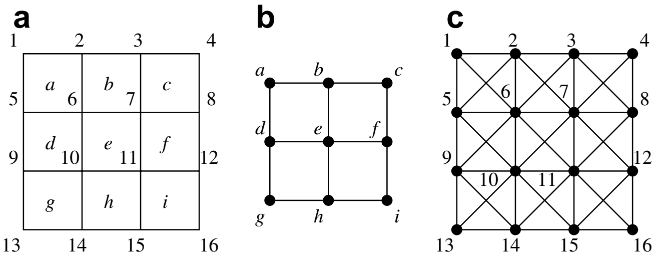

Fig. 1. A quadrilateral mesh (a) with 9 elements $a, \ldots, i$ and 16 nodes $1, \ldots, 16$. The dual graph (b) and the nodal (c) graph derived from the mesh.

Definition. A graph $G = (V, E)$ with an integer $k \geqslant 2$ is considered. An *edge cut* (a *vertex cut*) is a set of edges (vertices, respectively) whose removal divides the graph into at least k partitions. The *k-way* graph partitioning problem is to partition V into k pairwise disjoint subsets $V_1, V_2, \ldots, V_k$ by an edge cut such that $|V_i| \cong |V|/k$ and the size of the edge cut is minimised. The subsets $V_1, V_2, \ldots, V_k$ induce subgraphs $G_1, G_2, \ldots, G_k$. A partitioning of a graph by a vertex cut is similar, except that the pending edges are also removed.

Common methods for mesh partitioning are based on graph partitioning, typically by edge cut of dual graphs. A submesh is made of elements from the same partition. The boundary nodes belong to more than one submesh. The remaining ones are the inner nodes. The example Fig. 2(a) shows a 2-way partitioning of G^{D} of the quadrilateral mesh from Fig. 1. The corresponding mesh partitioning is shown in Fig. 2(b). The boundary nodes are 3, 7, 10, 11, and 14. Note that this way of mesh partitioning produces partitioning of the nodal graph G^{N} by a vertex cut, as shown in Fig. 2(c).

The dual graph G^{D} is partitioned into k parts, inducing submeshes and corresponding submatrices $A^{(j)}$, so that the sizes of submatrices are roughly equal.

6.1. Multilevel graph partitioning

Multilevel graph partitioning tools are widely used to perform the partitioning by edge cut. Our work is based on the multilevel k-way graph partitioning implemented in METIS [13]. This scheme consists of the following three phases, shown in Fig. 3.

6.1.1. Coarsening

A sequence of smaller coarser graphs $G_l^{\mathrm{D}} = (V_l^{\mathrm{D}}, E_l^{\mathrm{D}})$ is constructed from the original graph $G^{\mathrm{D}} = G_0^{\mathrm{D}} = (V_0^{\mathrm{D}}, E_0^{\mathrm{D}})$ so that $|V_l^{\mathrm{D}}| > |V_{l+1}^{\mathrm{D}}|$. The sequence of coarser graphs induces levels. The graphs have integer vertex weights $\sigma(v_l)$, $v_l \in V_l^{\mathrm{D}}$, and integer edge weights $\sigma(e_l)$, $e_l \in E_l^{\mathrm{D}}$. If

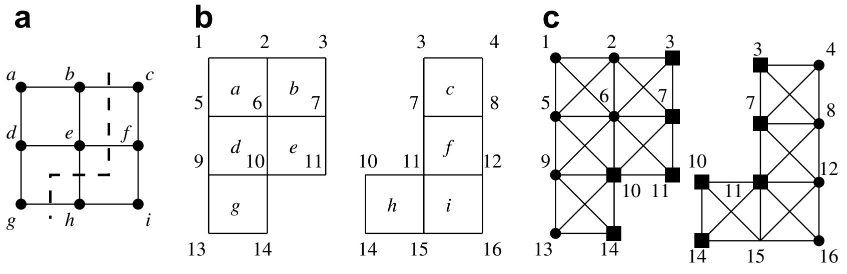

Fig. 2. Partitioning of the quadrilateral mesh from Fig. 1 using its dual graph (a) into two partitions (b) and corresponding partitioning of the nodal graph (c). The edge cut (a) is indicated by a dashed line and vertices in the vertex cut (c) as filled squares.

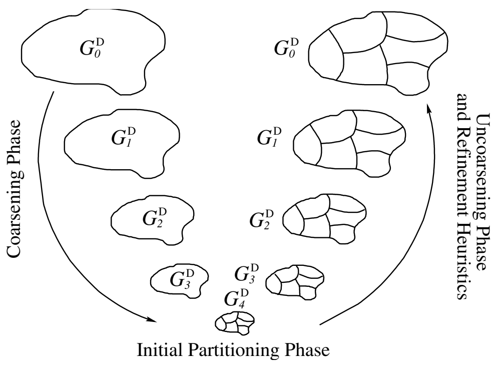
G_0^D
G_1^D
G_2^D
G_3^D
G_4^D
Coarsening Phase
Uncoarsening Phase and Refinement Heuristics
Initial Partitioning Phase

Fig. 3. The schema of multilevel *k-way* partitioning.

the original G_0^D is unweighted, then the weight 1 is assigned to its every vertex and edge.

A matching is used to collapse 2 adjacent vertices into a so-called multivertex. In level l, two adjacent vertices or multivertices $v_l, v'_l \in V_l^D$ can be collapsed into a new multivertex $v_{l+1} \in V_{l+1}^D$. After the matching, $\sigma(v_{l+1}) = \sigma(v_l) + \sigma(v'_l)$. The edge (v_l, v'_l) is not included in E_{l+1}^D. If v_l is adjacent to some other vertex $u_l \in V_l^D$, then there is an edge $(v_{l+1}, u_{l+1}) \in E_{l+1}^D$, $u_{l+1} \in V_{l+1}^D$ such that either $u_{l+1} \equiv u_l$ or u_{l+1} is a multivertex created from u_l and some other vertex, and $\sigma((v_{l+1}, u_{l+1})) = \sigma((v_l, u_l))$. A similar rule applies if v'_l is adjacent to u_l. If both v_l and v'_l are adjacent to u_l, then $(v_{l+1}, u_{l+1}) \in E_{l+1}^D$ and $\sigma((v_{l+1}, u_{l+1})) = \sigma((v_l, u_l)) + \sigma((v'_l, u_l))$. The METIS uses a variation of matching called SHEM [13].

6.1.2. Initial partitioning

When the coarsest graph is sufficiently small, it can be partitioned by any other graph partitioning technique. The METIS partitions the coarsest graph by the multilevel graph bisection [14].

6.1.3. Uncoarsening and refinement

A coarser graph G_l^D is uncoarsed to G_{l-1}^D, the partitioning of G_l^D is projected to G_{l-1}^D, and then refined by the Fiduccia–Mattheyses (FM) heuristic. It searches candidate vertices in the set of vertices adjacent to a vertex in another partition. Then it tries to move the selected vertex into other partitions. The move is accepted if one of the following conditions is fulfilled:

1. The size of the edge cut is decreased and the partitioning remains balanced.
2. The size of the edge cut is not increased, but the balancing is improved.

To improve the balancing of a strongly disbalanced partitioning, a balancing step may by added. It works like the FM heuristic, but the conditions for accepting a move are different:

1. The balancing is improved.
2. The balancing is not worsened, but the size of the edge cut is decreased.

Stopping criteria for FM heuristic can be found in [13].

6.2. Reordering of nodes

After the mesh partitioning, the inner nodes in all partitions are reordered to minimise the computational complexity of the partial factorisation of submatrices $\boldsymbol{A}^{(j)}$. The reordering algorithm works with the nodal graph G^N. The boundary (inner) nodes are represented by boundary (inner, respectively) vertices.

There are two common algorithms for the reordering of nodes for the sequential factorisation of matrices. The Sloan algorithm [15,9] and the minimum degree algorithm [8,9]. These algorithms, if applied within the partial factorisation of submatrices, should be modified in order to distinguish the inner nodes from the boundary ones. Paper [16] describes a modification of the Sloan algorithm, called the boundary Sloan algorithm (BSA). It reorders the nodes to minimise the envelopes of submatrices.

The minimum degree algorithm reorders the nodes for the sparse storage scheme. It can be described in terms of elimination graphs [8] as follows:

Step 1. Initialise the current elimination graph by G^N.
Step 2. In the current elimination graph, choose a vertex u with the minimum degree.
Step 3. Form a new elimination graph by eliminating u and updating the degrees of its neighbours.
Step 4. If any vertex remains, go to Step 2.

A modification of this algorithm, called the boundary minimum degree algorithm, distinguishes between inner and boundary vertices.

Step 1. Initialise the current elimination graph by G^N.
Step 2. In the current elimination graph, choose an inner vertex u of the minimum degree.
Step 3. Form a new elimination graph by eliminating u and updating the degrees of its neighbours.
Step 4. If any inner vertex remains, go to Step 2.
Step 5. Order the boundary vertices randomly after the inner ones.

In this paper, the boundary multiple minimum degree algorithm (BMMDA) is used. It differs from the previous modification in Steps 2 and 3, where it performs a multiple elimination as proposed in [17].

7. Mesh partitioning with load balancing

This section describes a new approach to mesh partitioning based on the Quality Balancing heuristic. As stated in Section 1, a decomposition into submatrices $\boldsymbol{A}^{(j)}$ of similar sizes does not necessarily lead to a balanced computation. The load is significantly influenced by the reordering of the nodes. In the classic approach, the reordering comes after the mesh partitioning and, therefore mesh partitioning cannot balance the load. The main idea of the QB heuristic is to integrate a reordering algorithm into the mesh partitioning, specifically into the multilevel k-way graph partitioning of the dual graph G^{D}. The QB heuristic can balance various qualities computed over subdomains. In this paper, the number of FLOPs given by the formula (14) is taken as an estimation of the load.

Definition. Given an unbalancing threshold $\delta \geqslant 1$, a set of qualities $\{q_1, q_2, \ldots, q_k\}$ is called balanced if

$$\delta \geqslant (\max_{i=1}^{k} q_i)k \Big/ \sum_{i=1}^{k} q_i. \tag{15}$$

A quality q_i is overbalanced if $\delta < q_i k / \sum_i q_i$. The set of qualities is disbalanced if at least one quality is overbalanced.

The input to the QB heuristic is an FE mesh, its dual G^{D} and nodal G^{N} graphs. Every node generates either a constant number of unknowns (in the case of unconstrained nodes) or no unknowns (in the case of constrained nodes). The constrained nodes are not included in G^{N}. The first two phases, the coarsening and the initial partitioning, are performed almost unchanged as in Section 6.1. In addition, the weight of lost edges $\varsigma(v)$ for every multivertex v is computed during the coarsening. If two adjacent vertices or multivertices v_l, v'_l are collapsed into a multivertex v_{l+1}, the edge (v_l, v'_l) is called "lost". Let $\varsigma(v)$ denote the total weight of "lost" edges of a multivertex v. Then, $\varsigma(v_{l+1}) = \varsigma(v_l) + \varsigma(v'_l) + \sigma((v_l, v'_l))$. Initially, $\varsigma(v_0) = 0$ for every $v_0 \in V_0^{\mathrm{D}}$.

The QB heuristic modifies the conditions of move acceptance of the FM heuristic and extends it with a quality estimation process, as illustrated in Fig. 4. Assume that the current level is l. Similar to FM, QB chooses a candidate vertex v for moving from its source partition $G_{l,s}^{\mathrm{D}} = (V_{l,s}^{\mathrm{D}}, E_{l,s}^{\mathrm{D}})$. Next, it chooses a target partition $G_{l,t}^{\mathrm{D}} = (V_{l,t}^{\mathrm{D}}, E_{l,t}^{\mathrm{D}})$, $t \in \{1, \ldots, k\}$, $t \neq s$, where v is intended to be moved. Note that the subscript s denotes the number of the source partition and the subscript t denotes the number of the target partition. The values of qualities $q_p, p \in \{s, t\}$ are saved to $\tilde{q}_p$ (see Step **a** in Fig. 4). Then the process continues for both partitions $G_{l,p}^{\mathrm{D}}$.

To prevent the unnecessary nullifying of moves (Step **i**), the changes in the qualities q_p are predicted (Step **b**). If the predicted values q'_p are not accepted (Step **c**), the whole process is returned to the beginning. The prediction will be explained in more detail at the end of this section.

The estimation process then starts by the uncoarsening of the partition $G_{l,p}^{\mathrm{D}}$ to $G_{0,p}^{\mathrm{D}}$, the original partition of G^{D} (Step **d**). This is skipped in the final level $l = 0$. After that, the partition G_p^{N} is constructed from all nodes belonging to the elements from $G_{0,p}^{\mathrm{D}}$ (Step **e**). If the relation between the

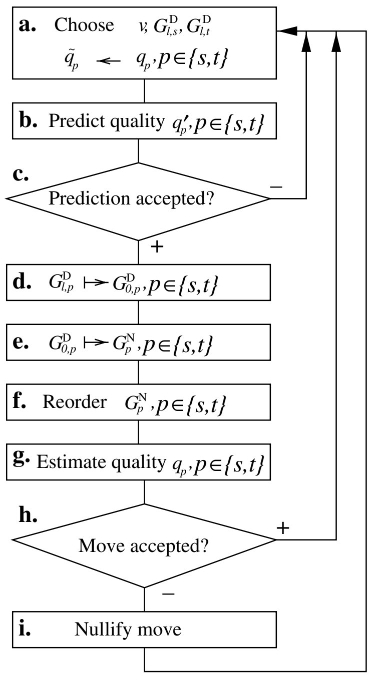
a. Choose $v, G_{l,s}^{\mathrm{D}}, G_{l,t}^{\mathrm{D}}$
$\tilde{q}_p \leftarrow q_p, p \in \{s,t\}$
b. Predict quality $q'_p, p \in \{s,t\}$
c. Prediction accepted?
−
+
d. $G_{l,p}^{\mathrm{D}} \mapsto G_{0,p}^{\mathrm{D}}, p \in \{s,t\}$
e. $G_{0,p}^{\mathrm{D}} \mapsto G_p^{\mathrm{N}}, p \in \{s,t\}$
f. Reorder $G_p^{\mathrm{N}}, p \in \{s,t\}$
g. Estimate quality $q_p, p \in \{s,t\}$
h. Move accepted?
+
−
i. Nullify move

Fig. 4. QB heuristic.

G_l^{D} and G^{N} is constructed for every level l, then the previous two steps (**d** and **e**) can be efficiently implemented as one step. Next, the vertices of G_p^{N} are reordered by BSA or BMMDA (Step **f**), depending on the storage scheme, see Section 6.2. The estimation process finishes by a computation of the qualities (Step **g**) using Eq. (14).

After the estimation of the qualities q_p, the conditions of move acceptance are evaluated (Step **h**). In contrast to FM, they are modified as follows:

1. The size of the edge cut of G_l^{D} is decreased. Neither q_s nor q_t is overbalanced, or else the balancing was not worsened by the move, i.e., $\max\{q_s, q_t\} \leqslant \max\{\tilde{q}_s, \tilde{q}_t\}$.
2. The balancing is improved, i.e., $\max\{q_t, q_s\} < \max\{\tilde{q}_t, \tilde{q}_s\}$, but the size of the edge cut of G_l^{D} is not increased.

The conditions of move acceptance for the balancing step are also modified:

1. The balancing is improved.
2. The size of the edge cut of G_l^{D} is decreased and neither q_s nor q_t is overbalanced.

Only if the conditions of move acceptance fail, is the move nullified (Step **i**), i.e., v remains in $G_{l,s}^{\mathrm{D}}$ and the qualities are returned to the saved values $q_p \leftarrow \tilde{q}_p$. Finally, QB returns to the beginning (Step **a**).

The reordering of nodes is a time consuming operation. It significantly slows down the uncoarsening and refinement phase, because it is performed for every move of a multivertex. Therefore, the prediction of qualities (Step **b**) has been inserted before the estimation process. The quality of the source partition is predicted by the following formula:

$$q'_s = q_s \cdot \frac{\sigma(V^{\mathrm{D}}_{l,s}) - \sigma(v)}{\sigma(V^{\mathrm{D}}_{l,s})} \cdot \frac{\sigma(E^{\mathrm{D}}_{l,s}) - \sum_{(u,v)\in E^{\mathrm{D}}_l, u\in V^{\mathrm{D}}_{l,s}} \sigma((u,v)) - \varsigma(v)}{\sigma(E^{\mathrm{D}}_{l,s})} \tag{16}$$

and the quality of the target partition is predicted by the following formula:

$$q'_t = q_t \cdot \frac{\sigma(V^{\mathrm{D}}_{l,t}) + \sigma(v)}{\sigma(V^{\mathrm{D}}_{l,t})} \cdot \frac{\sigma(E^{\mathrm{D}}_{l,t}) + \sum_{(u,v)\in E^{\mathrm{D}}_l, u\in V^{\mathrm{D}}_{l,t}} \sigma((u,v)) + \varsigma(v)}{\sigma(E^{\mathrm{D}}_{l,t})}, \tag{17}$$

where

$$\sigma(V^{\mathrm{D}}_{l,p}) = \sum_{u\in V^{\mathrm{D}}_{l,p}} \sigma(u), \quad p \in \{s,t\} \tag{18}$$

and

$$\sigma(E^{\mathrm{D}}_{l,p}) = \sum_{e\in E^{\mathrm{D}}_{l,p}} \sigma(e) + \sum_{u\in V^{\mathrm{D}}_{l,p}} \varsigma(u), \quad p \in \{s,t\}. \tag{19}$$

The second coefficients in (16) and (17) represent ratios of vertex weights of the updated and original partitions. The third coefficients represent the ratios of edge weights and lost edge weights of the updated and original partitions. The conditions of move acceptance are evaluated for q'_p in the same way as for q_p (Step **c**).

The estimation process is performed for every move that satisfies the prediction conditions. The reordering is heuristic and thus, the change of qualities after a move of a subset of vertices is not deterministic. Thus, some moves must be nullified if they worsen the balancing. The nondeterministic changes in qualities and the move nullifications increase the time complexity of the mesh partitioning by the QB heuristic. Hence, the QB heuristic is suitable for problems where the same decomposition is used several times, like the time dependent mechanical problems or the nonstationary transport problems, described in Sections 2 and 3.

8. Experimental results

The proposed QB heuristic was tested on four 3D FE models: a "vessel" (see Fig. 5(a)), a "block" (just a plain block), a "wheel" (see Fig. 5(b)), and a "dam" (see Fig. 5(c)). They were discretised by tetrahedrons with various mesh sizes by the software described in [18]. The names of FE meshes are the names of the problems and the number relates to the mesh size. A description of the meshes is given in Table 2. $|V^{\mathrm{D}}|$ denotes the number of elements and $|V^{\mathrm{N}}|$ denotes the number of nodes.

The dual graphs of meshes from Table 2 were partitioned first by METIS and then by the QB heuristic into k partitions. METIS was used with its default parameters. The unbalancing threshold for the QB heuristic was set to $\delta = 1.10$. All times are in seconds. The reduced system of equations was solved with the parallel conjugate gradient method.

The names of the columns in the tables have the form $\square_X$, where $X = M$ denotes parameters related to partitioning by METIS and $X = \mathrm{QB}$ denotes parameters related to partitioning done by the QB heuristic. Two types of tables follow. The tables for the first type (Tables 3, 5, 7 and 9) summarise the time characteristics of the results. The wall clock time for solving a problem is denoted by t^{R}_X, i.e., $t^{\mathrm{R}}_{\mathrm{M}}$ is the solution time for problems partitioned by the METIS, and $t^{\mathrm{R}}_{\mathrm{QB}}$ is the solution time of problems partitioned by the QB heuristic. The parameter

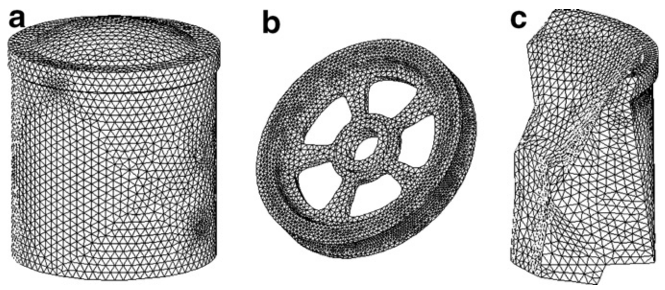

Fig. 5. An example of FE models of the problems (a) "vessel", (b) "wheel", (c) "dam".

Table 2
Description of the test FE meshes

Name	$\|V^{\mathrm{D}}\|$	$\|V^{\mathrm{N}}\|$
vessel09	227,559	53,670
vessel06	726,128	157,148
block05	153,051	28,339
block03	698,742	124,842
wheel5	241,955	51,376
wheel3	1,043,149	204,586
dam15	264,038	48,865
dam11	620,054	112,060

Table 3
The timing results of experiments on the PC cluster with a solver using the envelope storage scheme

Mesh	k	$t^{\mathrm{R}}_{\mathrm{M}}$	$t^{\mathrm{R}}_{\mathrm{QB}}$	$\Delta t^{\mathrm{R}}[\%]$	$t^{\mathrm{D}}_{\mathrm{M}}$	$t^{\mathrm{D}}_{\mathrm{QB}}$	λ
vessel09	4	1190	1104	7	0.34	116.46	14
vessel09	6	619	565	9	0.33	92.07	17
vessel09	8	605	386	36	0.34	87.66	4
vessel09	10	402	276	31	0.34	85.67	7
block05	4	2348	1210	48	0.24	146.4	2
block05	6	810	663	18	0.25	282.67	20
block05	8	590	322	45	0.26	139.55	6
block05	10	297	226	24	0.26	191.17	27
wheel5	4	1814	982	46	0.36	115.03	2
wheel5	6	552	538	3	0.37	68.45	49
wheel5	8	404	382	5	0.38	92.16	42
wheel5	10	319	291	9	0.36	81.33	29
dam15	4	3644	2455	33	0.43	433.76	4
dam15	6	2127	1490	30	0.46	350.58	6
dam15	8	1920	928	52	0.44	412.76	5
dam15	10	1094	921	16	0.46	434.88	26

$$\Delta t^{\mathrm{R}} = 100\left(1 - t^{\mathrm{R}}_{\mathrm{QB}}/t^{\mathrm{R}}_{\mathrm{M}}\right)$$

represents the percentage of the solution time saved by the QB heuristic. The mesh partitioning time is denoted by t^{D}_X. The parameter λ denotes how many times the partitioning produced by the QB heuristic must be used in order to compensate for its time requirements, i.e.,

$$\lambda = \left\lceil \frac{t^{\mathrm{D}}_{\mathrm{QB}} - t^{\mathrm{D}}_{\mathrm{M}}}{\left(t^{\mathrm{R}}_{\mathrm{M}} - t^{\mathrm{R}}_{\mathrm{QB}}\right)/\kappa} \right\rceil,$$

where κ denotes the number of time steps, which is the number of calls of a linear system solver. The tables of the second type (Tables 4, 6, 8 and 10) contain the sizes of edge cuts $|C|$ of the dual graphs of the meshes and the time spent on the partial factorisation. The maximum partial factorisation time over subdomains is denoted by

$$\max\left(t^{\mathrm{F}}_X\right) = \max_{j=1}^{k}\left(t^{\mathrm{F}}_X\right)^{(j)},$$

where $\left(t^{\mathrm{F}}_X\right)^{(j)}$ denotes the time spent on the partial factorisation of $\boldsymbol{A}^{(j)}$. The disbalancing in partial factorisation times is

$$\delta\left(t^{\mathrm{F}}_X\right) = \max\left(t^{\mathrm{F}}_X\right)/\overline{t^{\mathrm{F}}_X} = \frac{\max\left(t^{\mathrm{F}}_X\right)k}{\sum_{j=1}^{k}\left(t^{\mathrm{F}}_X\right)^{(j)}}.$$

The experiments were carried out on two parallel machines. The first one was a cluster of 10 PCs with 3.20 GHz Intel Pentium 4 processors and 3GB of memory. The cluster operated under Linux 2.6. The software was compiled using gcc 3.3 with optimisation `-O3`. A total of only $\kappa = 10$ time steps were performed. The meshes were partitioned into $k = 4, 6, 8, 10$ submeshes. The timing characteristics for the envelope storage scheme are in Table 3 and the other characteristics are in Table 4. Likewise, the sparse storage scheme time characteristics are in Table 5 and the others are in Table 6.

The second parallel machine was a Sun Fire 15K Server with 900MHz UltraSPARC III processors with 1GB of memory associated with each processor. The machine is located in the EPCC in Edinburgh, Scotland. It was oper-

Table 4
The characteristics of results of experiments on the PC cluster with a solver using the envelope storage scheme

Mesh	k	$\lvert C\rvert_{\mathrm{M}}$	$\lvert C\rvert_{\mathrm{QB}}$	$\max\left(t^{\mathrm{F}}_{\mathrm{M}}\right)$	$\max\left(t^{\mathrm{F}}_{\mathrm{QB}}\right)$	$\delta\left(t^{\mathrm{F}}_{\mathrm{M}}\right)$	$\delta\left(t^{\mathrm{F}}_{\mathrm{QB}}\right)$
vessel09	4	1991	2025	110	102	1.17	1.10
vessel09	6	2622	2726	55	49	1.18	1.08
vessel09	8	3276	3382	53	33	1.72	1.10
vessel09	10	3847	3913	34	22	1.60	1.10
block05	4	3674	4089	229	114	1.47	1.05
block05	6	5207	5559	75	59	1.37	1.02
block05	8	5980	6462	55	28	1.79	1.03
block05	10	6827	7466	26	19	1.38	1.09
wheel5	4	1275	1547	175	90	1.74	1.09
wheel5	6	1947	2045	49	47	1.08	1.09
wheel5	8	2421	2734	36	33	1.14	1.09
wheel5	10	2871	3060	27	25	1.21	1.10
dam15	4	4415	5122	350	231	1.48	1.06
dam15	6	6504	7534	196	131	1.46	1.06
dam15	8	7877	8759	176	79	1.93	1.09
dam15	10	8893	10,162	95	56	1.67	1.10

Table 5
The timing results of experiments on the PC cluster with a solver using the sparse storage scheme

Mesh	k	$t^{\mathrm{R}}_{\mathrm{M}}$	$t^{\mathrm{R}}_{\mathrm{QB}}$	Δt^{R}[%]	$t^{\mathrm{D}}_{\mathrm{M}}$	$t^{\mathrm{D}}_{\mathrm{QB}}$	λ
vessel09	4	762	719	6	0.33	143.26	34
vessel09	6	442	410	7	0.33	169.18	53
vessel09	8	302	279	8	0.33	216.23	94
vessel09	10	240	207	14	0.35	219.64	67
block05	4	2193	1746	20	0.24	522.77	12
block05	6	923	836	9	0.25	627.91	73
block05	8	615	445	28	0.26	475.37	28
block05	10	381	312	18	0.28	497.64	73
wheel5	4	747	686	8	0.36	155.93	26
wheel5	6	508	447	12	0.37	221.54	37
wheel5	8	336	308	8	0.38	214.85	77
wheel5	10	274	249	9	0.37	280.81	113
dam15	4	4957	4703	5	0.45	1857.14	74
dam15	6	2741	2439	11	0.45	4375.82	145
dam15	8	1823	1405	23	0.45	1437.18	35
dam15	10	1499	972	35	0.46	1527.12	29

Table 6
The characteristics of results of experiments on the PC cluster with a solver using the sparse storage scheme

Mesh	k	$\lvert C\rvert_{\mathrm{M}}$	$\lvert C\rvert_{\mathrm{QB}}$	$\max\left(t^{\mathrm{F}}_{\mathrm{M}}\right)$	$\max\left(t^{\mathrm{F}}_{\mathrm{QB}}\right)$	$\delta\left(t^{\mathrm{F}}_{\mathrm{M}}\right)$	$\delta\left(t^{\mathrm{F}}_{\mathrm{QB}}\right)$
vessel09	4	1991	1936	67	63	1.16	1.09
vessel09	6	2622	2728	37	34	1.19	1.08
vessel09	8	3276	3582	23	21	1.20	1.10
vessel09	10	3847	3941	18	15	1.26	1.11
block05	4	3674	3945	213	168	1.39	1.11
block05	6	5207	5709	85	76	1.19	1.10
block05	8	5980	6378	57	40	1.44	1.10
block05	10	6827	7304	35	28	1.34	1.08
wheel5	4	1275	1422	67	61	1.19	1.09
wheel5	6	1947	2235	44	37	1.25	1.10
wheel5	8	2421	2682	29	26	1.20	1.09
wheel5	10	2871	3055	23	20	1.26	1.10
dam15	4	4415	5006	482	454	1.25	1.11
dam15	6	6504	8315	258	220	1.33	1.11
dam15	8	7877	8263	166	126	1.44	1.09
dam15	10	8893	9787	135	82	1.70	1.11

Table 7
The timing results of experiments on the Sun Server with a solver using the envelope storage scheme

Mesh	k	$t^{\mathrm{R}}_{\mathrm{M}}$	$t^{\mathrm{R}}_{\mathrm{QB}}$	Δt^{R}[%]	$t^{\mathrm{D}}_{\mathrm{M}}$	$t^{\mathrm{D}}_{\mathrm{QB}}$	λ
vessel06	16	6087	4231	30	1.35	690	1
vessel06	32	2055	1773	14	1.35	807.92	6
block03	16	16,368	14,020	14	1.45	2299.34	2
block03	32	4861	3078	37	1.56	2412.76	3
wheel3	16	16,476	13,692	17	2.12	2361.2	2
wheel3	32	5263	3600	32	2.17	1942.43	3
dam11	16	15,025	9212	39	1.27	1934.38	1
dam11	32	4284	2890	33	1.34	1836.76	3

Table 8
The characteristics of results of experiments on the Sun Server with a solver using the envelope storage scheme

Mesh	k	$\lvert C\rvert_{\mathrm{M}}$	$\lvert C\rvert_{\mathrm{QB}}$	$\max\left(t_{\mathrm{M}}^{\mathrm{F}}\right)$	$\max\left(t_{\mathrm{QB}}^{\mathrm{F}}\right)$	$\delta\left(t_{\mathrm{M}}^{\mathrm{F}}\right)$	$\delta\left(t_{\mathrm{QB}}^{\mathrm{F}}\right)$
vessel06	16	9680	10,482	2801	1849	1.75	1.16
vessel06	32	15,148	16,281	775	588	1.91	1.37
block03	16	23,738	26,402	7775	6197	1.76	1.18
block03	32	34,536	36,684	2065	1301	1.83	1.24
wheel3	16	12,033	13,994	7734	6275	1.38	1.13
wheel3	32	22,171	24,284	2158	1358	1.71	1.23
dam11	16	20,765	23,256	6689	3808	2.00	1.14
dam11	32	30,460	33,420	1565	877	1.89	1.17

ated under SunOS 5.9 and the software was compiled by Forte Developer 7 C 5.4 compiler with optimisation -`fast`. Only $\kappa = 2$ time steps were performed. The meshes were partitioned into $k = 16$ submeshes. The mesh partitioning was performed on a PC from the above described cluster. This fact influences the parameters t_X^{D} and λ. It is common in parallel computing for preprocessing to be done on a different machine from the one used for the main computation. The timing characteristics for the envelope storage scheme are in Table 7 and the others are in Table 8. Likewise, the sparse storage scheme time characteristics are in Table 9 and the others are in Table 10.

The balancing of the estimates of the computational complexity by the QB heuristic always leads to a better balancing of the real computational load, $\delta\left(t_{\mathrm{QB}}^{\mathrm{F}}\right) < \delta\left(t_{\mathrm{M}}^{\mathrm{F}}\right)$. On the Sun Fire 15K Server the disbalancing in partial factorisation times exceeds the given unbalancing threshold of 1.10. The reason is that the partial factorisation times are also influenced by CPU architecture, particularly by the caches, and not only by the number of FLOPs.

Table 9
The timing results of experiments on the Sun Server with a solver using the sparse storage scheme

Mesh	k	$t_{\mathrm{M}}^{\mathrm{R}}$	$t_{\mathrm{QB}}^{\mathrm{R}}$	$\Delta t^{\mathrm{R}}[\%]$	$t_{\mathrm{M}}^{\mathrm{D}}$	$t_{\mathrm{QB}}^{\mathrm{D}}$	λ
vessel06	16	2702	2349	13	1.34	1734.38	10
vessel06	32	1213	1275	−5	1.38	2055.84	–
block03	16	14,624	12,893	12	1.47	9701.83	12
block03	32	4255	3174	25	1.57	7522.78	14
wheel3	16	8887	8075	9	2.12	6162.03	16
wheel3	32	3377	3226	4	2.17	6312.9	84
dam11	16	12,497	10,036	20	1.27	9295.9	8
dam11	32	3887	3008	23	1.34	5780.12	14

In all but one case, better load balancing led to a reduction in the solution time. When the mesh "vessel06" was solved on 32 processors on the Sun Fire 15K Server with sparse storage scheme for submatrices, the $t_{\mathrm{QB}}^{\mathrm{R}}$ is slightly greater than $t_{\mathrm{M}}^{\mathrm{R}}$. This is caused by the longer solution time for the reduced problem.

The parameter λ clearly indicates whether the QB heuristic is useful for a given problem. In general, if a problem needs more than one hundred time steps, then QB is almost always profitable. Otherwise, the user can continue to use METIS or other common graph partitioning software.

9. Conclusion

Time dependent mechanical problems and heat transfer problems requiring many time steps, solved by domain decomposition methods, particularly by mesh partitioning, are very sensitive to good load balancing. Classic mesh partitioning, which balances the number of elements in each subdomain, may have random results in balancing the computational load. The new proposed QB heuristic demonstrates that for such problems it is beneficial to spend more time on mesh partitioning in order to improve the balancing, since the solution time savings pays off significantly as the number of time steps required to solve a given problem increases. It should be noted that practical problems typically require hundreds or even thousands of time steps.

Acknowledgements

This research has been supported by the Czech Academy of Sciences under grant IBS3086102, by MŠMT under

Table 10
The characteristics of results of experiments on the Sun Server with a solver using the sparse storage scheme

Mesh	k	$\lvert C\rvert_{\mathrm{M}}$	$\lvert C\rvert_{\mathrm{QB}}$	$\max\left(t_{\mathrm{M}}^{\mathrm{F}}\right)$	$\max\left(t_{\mathrm{QB}}^{\mathrm{F}}\right)$	$\delta\left(t_{\mathrm{M}}^{\mathrm{F}}\right)$	$\delta\left(t_{\mathrm{QB}}^{\mathrm{F}}\right)$
vessel06	16	9680	10,405	1116	929	1.44	1.20
vessel06	32	15,148	16,373	388	253	1.75	1.18
block03	16	23,738	25,886	7026	5988	1.39	1.16
block03	32	34,536	36,510	1818	1245	1.59	1.10
wheel3	16	12,033	13,892	4012	3263	1.38	1.11
wheel3	32	22,171	24,126	1225	977	1.51	1.20
dam11	16	20,765	23,750	5359	3877	1.50	1.12
dam11	32	30,460	32,790	1361	863	1.58	1.12

research programme 6840770014 "Research in the Area of the Perspective Information and Navigation Technologies", and by the European Commission's Research Infrastructures activity of the Structuring the European Research Area programme, contract number RII3-CT-2003-506079 (HPC-Europa). The authors thank the anonymous referees for their valuable remarks on the submitted version of this paper.

References

[1] Bittnar Z, Šejnoha J. Numerical methods in structural mechanics. New York, USA: ASCE Press; 1996.
[2] Hughes TJR. The finite element method, linear static and dynamic finite element analysis. Englewood Cliffs, New Jersey, USA: Prentice-Hall; 1987.
[3] Farhat C, Roux FX. Implicit parallel processing in structural mechanics. Comput Mech Adv 1994;2:1–124.
[4] Toselli A, Widlund O. Domain decomposition methods – algorithms and theory. Springer series in computational mathematics, vol. 34. Berlin, Germany: Springer-Verlag; 2005.
[5] Farhat C, Roux FX. A method of finite element tearing and interconnecting and its parallel solution algorithm. Int J Numer Methods Eng 1991;32:1205–27.
[6] Quarteroni A, Valli A. Domain decomposition methods for partial differential equations. Numerical mathematics and scientific computation. New York, USA: Oxford University Press Inc.; 1999.
[7] Smith B, Bjørstad P, Gropp W. Domain decomposition. Parallel multilevel methods for elliptic partial differential equations. Cambridge, UK: Cambridge University Press; 1996.
[8] George A, Liu J. Computer solution of large sparse positive definite systems. Englewood Cliffs, NJ: Prentice Hall; 1981.
[9] Meurant G. Computer solution of large linear systems. Elsevier Science B.V.; 1999.
[10] Hendricson Bruce. Graph partitioning and parallel solvers: has the emperor no clothes? Irregular'98. Lecture Notes in Computer Science 1998;1457:218–25.
[11] Hendricson B. Load balancing fictions, falsehoods and fallacies. Appl Math Modell 2000;25:99–108.
[12] Medek O, Tvrdík P, Kruis J. Load and memory balanced mesh partitioning for a parallel envelope method. In: Danelluto M, Laforenza D, Vanneschi M, editors. EuroPar 2004 parallel processing. Lecture notes in computer science, vol. 3149. Springer; 2004. p. 734–41.
[13] Karypis G, Kumar V. Multilevel *k*-way partitioning scheme for irregular graphs. J Parallel Distrib Comput 1998;48(2):96–129.
[14] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 1998;20:359–92.
[15] Kumfert G, Pothen A. Two improved algorithms for envelope and wavefront reduction. BIT 1997;37(3):559–90.
[16] Medek O, Tvrdík P. Variable reordering for a parallel envelope method. In: Yang Yuanyuan, editor. Proceedings of the 2004 international conference on parallel processing workshops. IEEE Computer Society Press; 2004. p. 254–61.
[17] Liu Joseph WH. Modification of the minimum-degree algorithm by multiple elimination. ACM Trans Math Software 1985;11(2):141–53.
[18] Daniel Rypl. T3D Mesh Generator, 2004. URL: http://ksm.fsv.cvut.cz/~dr/t3d.html.