Load and Memory Balanced Mesh Partitioning for a Parallel Envelope Method*

Ondřej Medek, Pavel Tvrdík, and Jaroslav Kruis

Czech Technical University, Prague, Czech Republic,
{xmedeko,tvrdik}@fel.cvut.cz, kruis@fsv.cvut.cz

Abstract. We use a parallel direct solver based on the Schur complement method for solving large sparse linear systems arising from the finite element method. A domain decomposition of a problem is performed using a graph partitioning. It results in sparse submatrices with balanced sizes. An envelope method is used to factorize these submatrices. However, the memory requirements to store them and the computational cost to factorize them depends heavily on their structure. We propose a technique that modifies the multilevel graph partitioning schema to balance real computational load or memory requirements of the solver.

1 Introduction

Many engineering and scientific problems are solved using the finite element method (FEM). We consider a mesh of *finite elements*. One element consists of several *nodes*. Each node has its *degrees of freedom* (DOFs). In other words, the nodes contain *variables*. Globally, these variables form a system of *equations* $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $n \times n$ sparse matrix of coefficients. A mesh is usually represented by a *dual graph* G^{D} and a *nodal graph* G^{N}. The vertices in the dual graph represent the finite elements and 2 vertices are adjacent if and only if the corresponding elements share a common surface in 3D or a common edge in 2D. The vertices in the nodal graph represent the mesh nodes. All vertices that represent mesh nodes belonging to one element form a clique. An example of a FE mesh and its dual and nodal graph is on Fig. 1.

We use solver SIFEL, developed at the Czech Technical University, for solving problems by the FEM. Among others, it can solve problems in parallel by the popular method of the Schur complements, which are computed by an envelope method [1]. A solution of a problem consists of 6 following phases:

1. Domain decomposition.
2. Ordering of nodes.
3. Assembling of submatrices.
4. Factorization of submatrices (computation of the Schur complements).
5. Solution of the reduced problem.
6. Back substitution on subdomains.

* This work was supported by IBS3086102 grant of Czech Academy of Science and by MŠMT under research program #J04/98:212300014.

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 734–741, 2004.

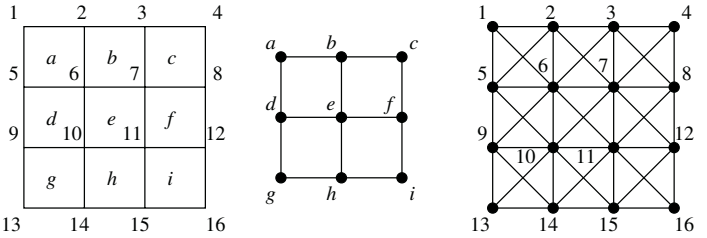

Fig. 1. A quadrilateral mesh (a) with 9 elements $a - i$ and 16 nodes 1 – 16. The dual (b) and the nodal (c) graph derived from the mesh.

The domain decomposition, and optionally ordering of nodes, are done as preprocessing steps. The solver does the rest. Phase 3 is the most memory consuming and Phase 4 is the most computationally intensive part of the whole solution.

Multilevel tools are widely used to solve the problem of domain decomposition. The dual graph is partitioned into k parts, inducing subdomains and corresponding submatrices, so that the sizes of submatrices are roughly equal.

Definition 1. *Consider a graph $G = (V, E)$ and an integer $k \geqq 2$. An* edge cut *(*node 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 $V_1, V_2, \ldots, V_k$ such that $|V_i| \doteq |V|/k$ and the size of the edge cut is minimized. A partitioning of a graph by a* node cut *is similar.*

Even though the sizes of submatrices are roughly equal, their memory requirements or their factorization time in Phase 4 are not equal. To define this formally, we use the term *quality* to denote the memory or computational complexity.

Definition 2. *Given an* unbalancing threshold $\delta \geqq 1$, *we say that a partitioning $V_1, V_2, \ldots, V_k$ with a set of qualities $\{q_1, q_2, \ldots, q_k\}$ is* balanced *if*

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

A partition V_i is overbalanced *if $\delta < q_i k / \sum_i q_i$. The partitioning is* disbalanced *if at least one partition is overbalanced.*

In general, the qualities of submatrices are influenced by the ordering of variables, as was already mentioned in [2, 3].

In this paper, we describe a novel approach to the domain decomposition that results into partitioning with balanced memory requirements or balanced factorization time estimations. This in fact leads to shorter execution time of the parallel solver. The idea is to integrate an ordering algorithm into a multilevel graph partitioning schema.

Section 2 describes the previous work. In Section 3, the new refinement heuristics that allows to balance better memory or computing complexity is explained. In Section 4, the results of experiments are presented and Section 5 concludes the paper.

2 Previous Work

2.1 Domain Decomposition (Phase 1)

Common methods for domain decomposition are based on graph partitioning, typically of dual graphs. A subdomain is made of elements from the same partition. The nodes belonging to more than one subdomain are called *boundary* and the remaining nodes are *internal*. Variables of the internal (boundary) nodes are called correspondingly. An example on Fig. 2 (a) shows a 2-way partitioning of G^{D} of the quadrilateral mesh from Fig. 1. The corresponding domain decomposition is shown on Fig. 2 (b). The boundary nodes are 3, 7, 10, 11, and 14. Note that this way of domain decomposition produces partitioning of the nodal graph G^{N} by a node cut, as shown on Fig. 2 (c).

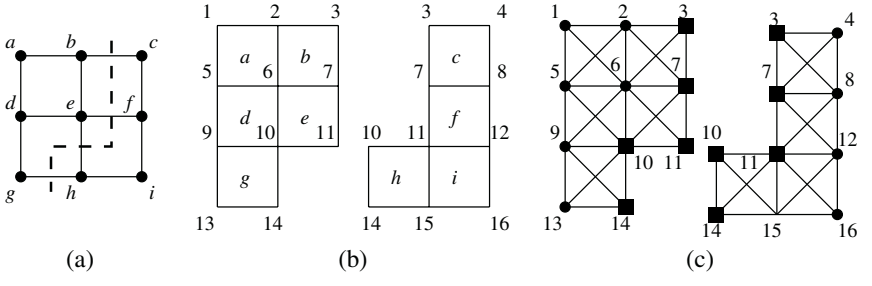

Fig. 2. Partitioning of the mesh from Fig. 1 using its dual graph (a) into 2 partitions (b) and corresponding partitioning of the nodal graph (c). The edge cut (a) is indicated by a dashed line and nodes in the node cut (c) as filled squares.

Multilevel Graph Partitioning. Popular multilevel graph partitioning software is METIS [4, 5], CHACO [6], JOSTLE [7], and SCOTCH [8]. Our work is based on the multilevel k-way graph partitioning implemented in METIS [5]. This schema consists of the following 3 phases, shown on Fig. 3.

Coarsening. A sequence of smaller graphs $G_l = (V_l, E_l)$ is constructed from the original graph $G = G_0 = (V_0, E_0)$ so that $|V_l| > |V_{l+1}|$. The sequence of coarser graphs creates *levels*. A matching is often used to collapse 2 vertices into a *multivertex*. The SHEM heuristics for matching, introduced in [5], works well.

Initial partitioning. When the coarsest graph is sufficiently small, it can be partitioned by any graph partitioning technique.

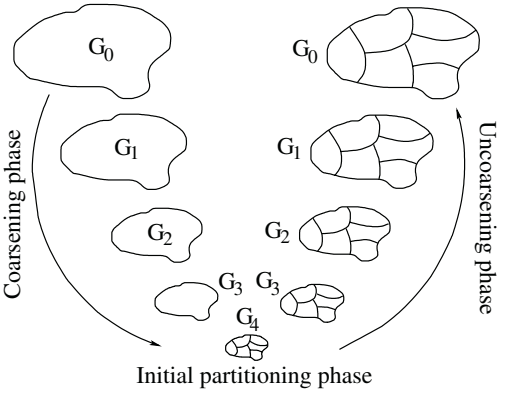

Fig. 3. The schema of multilevel *k-way* partitioning.

Uncoarsening. A coarser graph G_l is uncoarsed to G_{l-1} and the partitioning of G_l is projected to G_{l-1} and then refined. The Fiduccia-Mattheyses (FM) heuristics is a simple, fast, and sufficiently good option for the refinement. It searches candidate vertices in the set of *boundary vertices*, i.e., vertices adjacent to a vertex in another partition. Then it tries to move a 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 heuristics, but the conditions for accepting a move are different:

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

2.2 Reordering and Assembling of Submatrices (Phases 2 + 3)

After the domain decomposition, the internal nodes in all partitions are reordered to minimize the size of the *envelope*.

Definition 3. *In the i-th step of the factorization of a symmetric matrix* $\boldsymbol{A}$*, the* wavefront *is set of pairs*

$$w_i(\boldsymbol{A}) = \{\{r, i\} : \exists a_{rs} \neq 0, r \geqq i, s \leqq i\} \ . \tag{2}$$

The envelope *of* $\boldsymbol{A}$ *is then (symbol* $\setminus$ *is the set difference)*

$$Env(\boldsymbol{A}) = \bigcup_{i=1}^{n} w_i(\boldsymbol{A}) \setminus \{i, i\} \ . \tag{3}$$

The envelope method stores the parts of $\boldsymbol{A}$ inside envelopes, which represents the main portion of the memory consumption of the solver. The SIFEL solver uses the Sloan reordering algorithm [9], but other algorithms (RCM, hybrid algorithm) can be used as well.

Every node i generates d_i equations, where d_i is the number of DOFs. Let n_i and n_b denote the number of internal and boundary, respectively, variables. Clearly, $n_i + n_b = n$. The SIFEL solver reads the subdomains and assembles submatrices with nodes in a given order. The boundary nodes come last. We estimate memory requirements $W(\boldsymbol{A})$ of the envelope solver and its computational complexity $OP(\boldsymbol{A})$ by

$$W(\boldsymbol{A}) = \sum_{i=1}^{n} |w_i(\boldsymbol{A})| = |Env(\boldsymbol{A})| + n \tag{4}$$

$$OP(\boldsymbol{A}) = \sum_{i=1}^{n_i} |w_i(\boldsymbol{A})|^2 \ . \tag{5}$$

3 The New Refinement Heuristics with Quality Balancing

In this section, we explain the main ideas of load and memory balancing for the parallel envelope method. At the beginning, it must be decided which quality will be balanced: the computational load or the memory complexity. Then the partitioning is started. To estimate the quality, the partitioner must have the information about the structure of the matrix $\boldsymbol{A}$. Since the dual graph G^{D} does not suffice, the partitioner must have at disposal also the weighted nodal graph G^{N}, in which each vertex is labelled with the number of DOFs of the node. We partition G^{D} and project the partitioning of G^{D} to G^{N} to estimate the quality. So, the partitioner needs also the information about the relation between G^{D} and G^{N}.

We use a multilevel k-way graph partitioning as described in Sect. 2.1. The first two phases are performed unchanged as in METIS. The SHEM heuristics is used for the coarsening and the multilevel graph bisection is used for the initial partitioning. We have modified the conditions of move acceptance of the refinement heuristics in the uncoarsening phase and extended it with a quality estimation process. This new heuristics is called *QB*. Assume that the current level is l. Similar to FM, QB chooses a vertex as a candidate for moving from a *source* partition $G^{\mathrm{D}}_{l,s}$ to a *target* partition $G^{\mathrm{D}}_{l,t}$. To decide whether the move will be accepted, QB starts for both partitions $G^{\mathrm{D}}_{l,s}$ and $G^{\mathrm{D}}_{l,t}$ the estimation process, sketched on Fig. 4, to obtain information about balancing of the qualities.

First, partition $G^{\mathrm{D}}_{l,p}, p \in \{s,t\}$, is projected to $G^{\mathrm{D}}_{0,p}$, the original partition. (This is skipped in the final level, where $l = 0$). After that, all nodes belonging to elements from $G^{\mathrm{D}}_{0,p}$ correspond to a partition G^{N}_{p} of G^{N}. Then the internal vertices of G^{N}_{p} are reordered by the Sloan algorithm. Finally, the quality of G^{N}_{p} is estimated and returned to the refinement heuristics.

In the current implementation of the QB heuristics, nodes have either constant number of DOFs $d > 0$ or are *constrained*, i.e., the number of DOFs is 0.

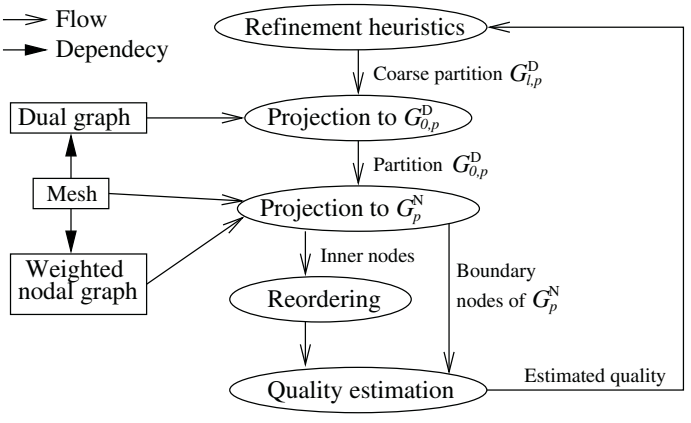

Fig. 4. Data flow of the QB heuristics.

All constrained nodes are omitted in the step of projection of $G^{D}_{0,p}$ to G^{N}_{p}, i.e., the reordering is performed only with nodes with the number of DOFs $d > 0$. After that, node i generates equations numbered $di, di + 1, \ldots di + d - 1$ and wavefronts $w_{di}(\boldsymbol{A}), w_{di+1}(\boldsymbol{A}), \ldots, w_{di+d-1}(\boldsymbol{A})$.

The original FM heuristics computes sums of weights of vertices in the source and target partitions for every candidate move. In fact, the weight of the candidate vertex is subtracted from the weight of the source partition and added to the weight of the target partition. However, in the QB heuristics, this would imply the reordering and estimation computing for every candidate move and this would extremely slow down the refinement. Thus, we had to modify the conditions of move acceptance as follows:

1. The size of the edge cut is decreased and the target partition is not overbalanced.
2. The quality q_s of the source partition is greater than the quality q_t of the target partition, but the size of the edge cut is not increased.

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

1. $q_s > q_t$.
2. The size of the edge cut is decreased and $q_s \geqq q_t$.

Only if a move is accepted, the qualities q_s and q_t are recomputed. Note that the new conditions may lead to overbalancing of the target, or even the source, partitions. Therefore, if the new value q_s is greater than its previous value, the vertex move is nullified.

4 Experimental Results

The SIFEL solver was modified to perform just the assembling and factorization of submatrices (Phases 3 and 4 in Sect. 1). All experiments were performed on

Table 1. Description of test problems. # stands for number of problem.

#	problem name	$\lvert V(G^{\mathrm{D}})\rvert$	$\lvert V(G^{\mathrm{N}})\rvert$	#	problem name	$\lvert V(G^{\mathrm{D}})\rvert$	$\lvert V(G^{\mathrm{N}})\rvert$
1	floor	49245	25065	4	jete	84123	22860
2	sieger	41012	21140	5	wheel	157529	34369
3	block	89196	16789				

a PC with Intel Pentium III, 1GHz, under GNU/Linux 2.4. The benchmarks are models of real problems of structural mechanics: "floor" and "sieger" are 2D problems discretized by triangles and "block", "jete", and "wheel" are 3D problems discretized by tetrahedrons. Their description is in Table 1.

Table 2 shows the results. # denotes the problem number as is quoted in Table 1. All problems were partitioned to $k = 4, 8, 16, 32$ partitions by METIS, called with its default parameters, and by the QB heuristics called with unbalancing threshold $\delta = 1.1$. First, QB was used to balance memory requirements of the solver and second, to balance computational complexity of the factorization phase. The size of the edge cut of G^{D} is denoted by $|E_c|$. Let $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_k$ be the corresponding submatrices of $\boldsymbol{A}$. Then $W_{max} = \max_{i=1}^{k} W(\boldsymbol{A}_i)$, $\overline{W} = \frac{1}{k}\sum_{i=1}^{k} W(\boldsymbol{A}_i)$, and $\Delta W = W_{max}/\overline{W}$. Similarly, let t_i be the factorization time of submatrix $\boldsymbol{A}_i$. Then $t_{max} = \max_{i=1}^{k} t_i$, $\overline{t} = \frac{1}{k}\sum_{i=1}^{k} t_i$, and $\Delta t = t_{max}/\overline{t}$. The time for the domain decomposition of the QB heuristics is denoted by t_p. The values of t_{max} and t_p in Table 2 are given in seconds.

Table 2. Comparison of the FM and QB heuristics.

		METIS					Memory Balancing						Load Balancing			
#	k	$\lvert E_c\rvert$	W_{max}	ΔW	t_{max}	Δt	$\lvert E_c\rvert$	W_{max}	ΔW	t_{max}	Δt	t_p	$\lvert E_c\rvert$	t_{max}	Δt	t_p
1	4	317	9751224	1.34	52.3	1.68	343	7312581	1.09	31.0	1.27	49	358	30.3	1.03	34
	8	524	3889227	1.35	15.5	1.72	547	2798742	1.04	10.2	1.33	38	629	8.9	1.09	66
	16	894	1634397	1.40	5.6	1.99	944	1175811	1.07	3.2	1.33	49	1011	2.9	1.18	69
	32	1405	623400	1.36	1.5	1.79	1424	456873	1.08	0.9	1.36	75	1447	0.8	1.11	125
2	4	160	3540777	1.08	7.4	1.14	159	3294747	1.03	6.5	1.06	8	157	6.5	1.10	10
	8	298	2094126	1.30	5.3	1.72	288	1523460	1.06	3.2	1.33	16	301	2.7	1.11	34
	16	533	971229	1.35	2.3	1.91	576	700158	1.08	1.3	1.35	23	663	1.6	1.61	102
	32	1048	428196	1.40	0.8	1.75	1072	308319	1.09	0.5	1.35	46	1068	0.4	1.21	53
3	4	2561	18490254	1.05	236.4	1.07	2753	17972640	1.03	233.7	1.07	219	2741	218.3	1.01	257
	8	4255	7336176	1.12	65.1	1.20	4231	7005948	1.10	55.8	1.09	198	4303	51.9	1.03	222
	16	5896	3388941	1.34	25.3	1.72	6045	2747472	1.22	15.3	1.39	282	6026	12.5	1.08	244
	32	8744	1288023	1.33	5.1	1.51	8749	1008285	1.14	3.7	1.38	287	8986	3.0	1.12	350
4	4	915	17489307	1.24	167.9	1.39	922	14000742	1.08	113.4	1.14	64	930	97.6	1.03	76
	8	1560	6281361	1.18	40.7	1.26	1628	5662314	1.09	34.2	1.13	59	1674	34.4	1.08	85
	16	2398	2987406	1.48	14.6	1.67	2537	2148342	1.09	10.1	1.23	76	2460	9.0	1.13	92
	32	3591	1224027	1.65	4.7	2.16	3471	749844	1.09	2.2	1.21	104	3583	2.0	1.09	134
5	4	914	21716046	1.15	196.2	1.27	999	20269455	1.07	178.0	1.17	117	993	172.6	1.09	146
	8	1747	11810322	1.31	106.1	1.58	1823	9354273	1.08	77.9	1.26	120	1802	67.9	1.08	156
	16	3253	4625520	1.24	31.8	1.42	3173	3557904	1.06	21.3	1.15	157	3438	21.3	1.08	298
	32	5917	1838763	1.29	10.0	1.70	5826	1550547	1.18	6.4	1.32	504	6420	6.5	1.33	577

5 Evaluations of Results and Conclusions

The results demonstrate that the QB heuristics always produces partitionings with better ΔW and Δt than the FM heuristics. Also $\Delta W \leq \delta$ in nearly all cases of memory balancing, whereas $\Delta t > \delta$ in about 30% of cases of load balancing, for the sake of minimizing the edge cut size $|E_c|$. The balancing of the memory requirements always improves Δt as a side effect. On the other hand, the QB heuristics sometimes produces partitionings with slightly greater $|E_c|$.

The memory balancing is beneficial for the distributed systems with limited amount of main memory per processor. The balancing of the computational load leads to the shorter time of the Phase 4 of the solver. Of course, the QB heuristics slows down the domain decomposition phase. Whereas the standard METIS takes times of order of seconds, the time t_p of the QB heuristics is of order of tens or hundreds of seconds. Therefore, it should be used if the same decomposition can be reused several times, e.g., in nonlinear systems or when the same problem is solved with different materials, etc.

References

1. George, A., Liu, J.: Computer Solution of Large Sparse Positive Definite Systems. Prentice Hall, Englewood Cliffs, NJ (1981)
2. Hendricson, B.: Graph partitioning and parallel solvers: Has emperor no clothes? Irregular'98, Lecture Notes in Computer Science **1457** (1998) 218–225
3. Hendricson, B.: Load balancing fictions, falsehoods and fallacies. Applied Mathematical Modelling **25** (2000) 99–108
4. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20** (1998) 359–392
5. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. J. of Parallel and Distrib. Comput. **48** (1998) 96–129
6. Hendrickson, B., Leland, R.: A multilevel algorithm for partitioning graphs. In: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), ACM Press (1995) 28
7. Walshaw, C., Cross, M.: Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. SIAM J. Sci. Comput. **22** (2000) 63–80 (originally published as Univ. Greenwich Tech. Rep. 98/IM/35)
8. Pellegrini, F.: Static mapping by dual recursive bipartitioning of process and architecture graphs. SHPCC'94 (1994) 486–493
9. Kumfert, G., Pothen, A.: Two improved algorithms for envelope and wavefront reduction. BIT **37** (1997) 559–590