On the Embedding of Refinements of 2-dimensional Grids*

F. d'Amore[1], L. Becchetti[1], S.L. Bezrukov[2], A. Marchetti-Spaccamela[1], M. Ottaviani[1], R. Preis[2], M. Röttger[2], and U.-P. Schroeder[2]

[1] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Salaria 113, I-00198 Roma, Italy
[2] Fachbereich Mathematik/Informatik, Universität-GH Paderborn, Fürstenallee 11, D-33102 Paderborn, Germany

Abstract. We consider the problem of constructing embeddings of 2-dimensional FEM graphs into grids. Our goal is to minimize the edge-congestion and dilation and optimize the load. We introduce some heuristics, analyze their performance, and present experimental results comparing the heuristics with the methods based on the usage of standard graph partitioning libraries.

1 Introduction

We consider the problem to embed large scale FEM graphs for the solution of partial differential equations into massively parallel computing systems. Roughly speaking, solving such equations with respect to a function F, say in two dimensions, requires to partition the domain of F into simple polygons (e.g. triangles or rectangles). Afterwards the value of the function F is computed in the nodes of the obtained partition. It turns out that accuracy requirements are not constant in the considered region but might vary considerably. This may lead to a partition of the area into polygons where the polygons sizes can be essentially different.

Such a partition can be viewed as a planar graph G whose nodes and edges correspond to the nodes and the sides of the polygons respectively. Each node represents a task for a processing element of the multiprocessor computing system. In order to minimize the running time the tasks have to be uniformly distributed among the processing elements. Furthermore, since the FEM requires to solve for each node x a difference equation involving x and its adjacent nodes, an information flow between the processing elements is caused which should also be minimized. These demands on the mapping can be expressed in the terms of graph embedding.

* This work was supported by the DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen", the EC ESPRIT Long Term Research Project 20244 "ALCOM-IT" and the DFG Graduate Center "Parallele Rechnernetze in der Produktionstechnik", GRK 124/2-96.

Let $G = (V, E)$ and $H = (V', E')$ be finite graphs. An *embedding* of the *guest* graph G into the *host* graph H is a function $f : V \mapsto V'$ together with a routing scheme R_f which assigns to each edge $e = \{v_1, v_2\} \in E$ a path in H from $f(v_1)$ to $f(v_2)$. The *congestion of an edge* $e' \in E'$ is the number of paths in $\{R_f(e) \mid e \in E\}$ containing e'. The *edge-congestion of an embedding* is the maximum congestion of the edges of E'. The *dilation of an edge* $e \in E$ is the length of the path $R_f(e)$, the *dilation of an embedding* is the maximum length of the paths in $\{R_f(e) \mid e \in E\}$.

Many papers in the literature study the embedding of graphs with the goal to minimize both load and communication costs. However, most papers assume that either G is given or it belongs to a rather restricted class of graphs whose structural properties are exploited (see [MS] for an overview). A large amount of literature deals with the problem of finding a partitioning of a graph into k clusters of approximately the same size. Here the aim is to minimize the number of cut edges connecting nodes that belong to different clusters [DH,PSL]. In [DMT] the authors analyze the cost of implementing multigrid methods using parallel architectures. The multigrid methods define a hierarchy of graphs that need to be embedded which consist of the original fine grid and successively coarser grids. However, the authors did not consider the communication/load tradeoff of the embedding.

In this paper we examine the case where the host graph H representing the computing system is a grid (e.g. Intel Paragon and Parsytec GC are commercial grid-based systems), and study the embedding of quasi grids into H with the aim of minimizing the load, the dilation, and the edge-congestion. The *quasi grid* is defined as follows. Let R be a rectangular area on the 2-dimensional plane with sides parallel to the coordinate axes. By splitting this area with $x-2$ horizontal and $y-2$ vertical lines we get a $x \times y$ grid with $(x-1) \cdot (y-1)$ rectangular cells. Now for a cell C we define a *cell refinement* operation. The operation consists of splitting C into 4 subcells with one vertical and one horizontal line passing through the center of C. This results in a graph which has 5 new nodes and 4 new edges as shown with the thin lines in Fig. 1(a). Note that each edge of the original cell is now partitioned into 2 new edges. If two cells of the original graph which have to be refined have a common edge, we create the new node in their common edge just once (cf. Fig. 1(b)). The new subcells obtained after the cell refinement operation are allowed to be further refined. Applying the cell refinement operation a number of times we obtain a quasi grid S (cf. Fig. 2).

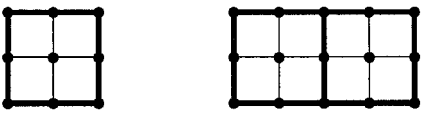


Fig. 1. (a) Refinement of a single cell and (b) of two neighboring cells.

Since the embedding problem is computationally hard [BU], we are interested in approximation algorithms. In Sect. 2, we introduce several heuristics and analyze them from the worst case point of view. We also present the results of preliminary experiments with the heuristics. In order to obtain fair results we compare the performance of the proposed heuristics with respect to the solution obtained by means of libraries for partitioning; these libraries exploit sophisticated algorithms that partition the nodes of a graph into clusters in order to minimize the load and the number of cut edges.

2 Algorithms

In this section we briefly present five heuristics for embedding a quasi grid S with m nodes into a $n_h \times n_v$ grid P and analyze their performance. Let $L^H(S,P)$ denote the load provided by heuristic H. A lower bound on the load is given by $L^{avg}(S,P) = \lceil m/(n_h \cdot n_v) \rceil$ and we define $R^H(S,P) := L^H(S,P)/L^{avg}(S,P)$.

Heuristic ***Tile1***: This and the next two heuristics are based on partitioning the quasi grid S into boxes that correspond to the structure of the grid P. To describe *Tile1* we first introduce two orderings ϕ_h and ϕ_v of the nodes of S. We consider the nodes of S as points (x,y) on the plane, assuming that the origin of the coordinate system is the leftmost and bottommost node of S and its axes are parallel to the segments of S. For the nodes $(x_1,y_1),(x_2,y_2)$ of S we say that $(x_1,y_1) <_{\phi_h} (x_2,y_2)$ iff $y_1 < y_2$, or if $y_1 = y_2$ then $x_1 < x_2$. Similarly, we say that $(x_1,y_1) <_{\phi_v} (x_2,y_2)$ iff $x_1 < x_2$, or if $x_1 = x_2$ then $y_1 < y_2$. Now we partition the nodes into n_h sets $A_1,...,A_{n_h}$. A_i consists of m'_i w.r.t ϕ_v consecutive nodes of S, with $\lfloor m/n_h \rfloor \leq m'_i \leq \lceil m/n_h \rceil$, $i = 1,...,n_h$. Moreover, the nodes are partitioned into n_v sets $B_1,...,B_{n_v}$ where B_j consists of m''_j w.r.t. ϕ_h consecutive nodes of S, with $\lfloor m/n_v \rfloor \leq m''_j \leq \lceil m/n_v \rceil$, $j = 1,...,n_v$. The embedding defined by *Tile1* is the following: The nodes of $C_{ij} = A_i \cap B_j$ are mapped onto the node p_{ij} of P, with $i = 1,...,n_h$ and $j = 1,...,n_v$. (cf. Fig. 2(a)).
This heuristic is first of all designed to provide a small dilation and edge-congestion. It guarantees that the total load of each column (row) of the grid P is the same up to one. However, the loads of single processors can be essentially different.

Proposition 1. $R^{Tile1}(S,P) \leq \min\{n_h, n_v\}$.

The first step of ***Tile2*** partitions the nodes in the same way as *Tile1*. Then each set A_i, $i = 1,...,n_h$, is partitioned into sets C_{ij} of m^j_i consecutive nodes of A_i (w.r.t. ϕ_h), where $\lfloor m'_i/n_v \rfloor \leq m^j_i \leq \lceil m'_i/n_v \rceil$ and $j = 1,...,n_v$ (cf. Fig. 2(b)).

Proposition 2. $R^{Tile2}(S,P) = 1$.

The heuristic ***Tile3*** involves an integer parameter d and uses the heuristic *Tile2* as a subroutine. First of all we partition the nodes of P into clusters C_{kl}:

$$C_{kl} = \{p_{ij} \mid (k-1)d+1 \leq i \leq kd,\ (l-1)d+1 \leq j \leq ld\},$$

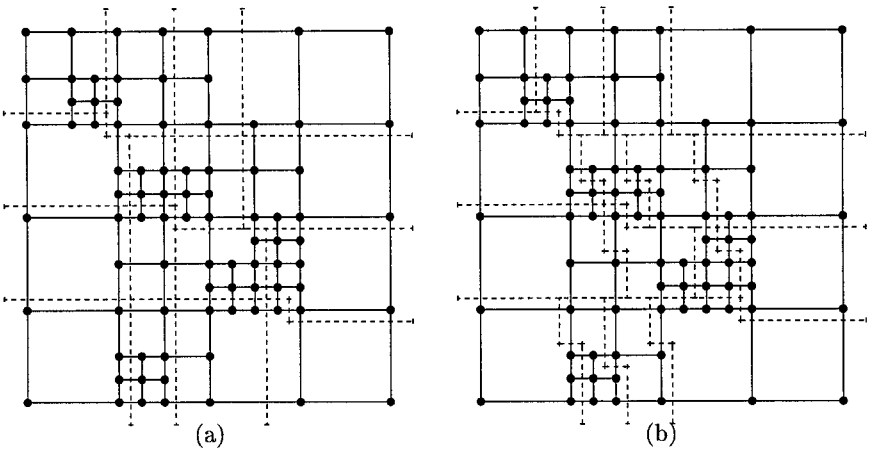


Fig. 2. The exemplified heuristics *Tile1* (a) and *Tile2* (b), where the guest graph is the 4 × 4 grid.

with $k = 1, ..., \lceil n_h/d \rceil$ and $l = 1, ..., \lceil n_v/d \rceil$. Furthermore, we partition the nodes of the quasi grid S into $\lceil n_h/d \rceil \cdot \lceil n_v/d \rceil$ blocks B_{kl} and map the nodes of B_{kl} onto the processors of the cluster C_{kl} using heuristic *Tile2*. The use of heuristic *Tile3* supposes that the unrefined quasi grid (that is, the grid obtained from S by considering its coarsest subgrid) coincides with, or is a subgraph of, P; if this is not the case a pre-embedding step is carried out. Details will be given in the full version of the paper.

The aim of the ***Pac-Man*** heuristic is to group nodes of S into $n_h \cdot n_v$ clusters in such a way that nodes of the same cluster lie as close to each other as possible. Under this approach we measure the nearness as the length of the shortest path between the corresponding nodes in S. We start by choosing randomly $n_h \cdot n_v$ seed-points of S. Then each seed-point tries in parallel to occupy nodes in its neighborhood; it stops when there is no node so that the already occupied area remains connected. Depending on the accrued clustering a new seed-point for the next iteration is computed. Namely, the center of each cluster is chosen as the seed-point for the next iteration. The algorithm terminates when all seed-points remain unchanged. Since there is no guarantee that this heuristic produces a balanced load, we have integrated a global load balancing step as post-processing. This step guarantees a totally balanced load without loss of the compactness of the previously computed clusters.

Heuristic ***Kohonen***: We adapt Kohonen's self-organizing maps [K] to compute an embedding that preserves topological relations between the nodes of the refinement. The general Kohonen process maps points of an euclidean space to adaptive elements called neurons in such a way that points of the space which are close to each other are mapped onto neurons which are close to each other. We represent the nodes of S as points of the 2-dimensional euclidean plane

E^2 and the grid P as the neural network. The Kohonen heuristic first assigns to each node v of P a point $p(v) \in E^2$ uniformly distributed in the rectangle $R = \{(x,y) \in E^2 \mid 1 \leq x \leq n_h, 1 \leq y \leq n_v\}$; let M denote the set of the chosen points. The heuristic proceeds by repeatedly relocating points as follows: randomly choose a node w of S, compute a point $n(w) \in M$ that is closest to w and then move all points $p(v) \in M$ in the direction of w. At-this the movement of each point $p(v)$ is inversely proportional to the euclidean distance between $p(v)$ and $n(w)$; the intensity of the movements decreases in time in order to guarantee the convergence. The process terminates after a certain number of iterations. The final embedding is given by the partition of the rectangle R into clusters $\{C(u) \mid u \in M\}$ induced by the Voronoi-diagram of the points of M. Since this algorithm generates an embedding which minimizes the communication costs (see [RMS]) but does not care about a balanced load, we additionally apply a partial load balancing procedure after an initial convergence phase every 300 iteration steps. In this procedure we compute the local load gradient of every node of the grid P by comparing the loads of all its neighbors. After that, for each node v of P, we move the point $p(v)$ in the direction of the local load gradient.

3 Partitioning Tools

Graph partitioning problems arise in many different applications, which leads to many heuristics based on different ideas. In general, graph partitioning can be viewed as an embedding of the guest graph into a complete graph, i.e., the load balance and the cut size are the major cost measures. Most applications require the load balance to be optimal, i.e., the number of nodes in each part differ at most by one. The goal is to minimize the cut size of the partition. The problem of constructing such a partition is known to be NP-complete even in the case of partitioning into two parts of the same size. Efficient heuristics have been designed in the last decades to construct partitions with very low cut sizes. Although they are based on some reasonable arguments for a low cut size, there is no guarantee that a method works well for all kinds of graphs.

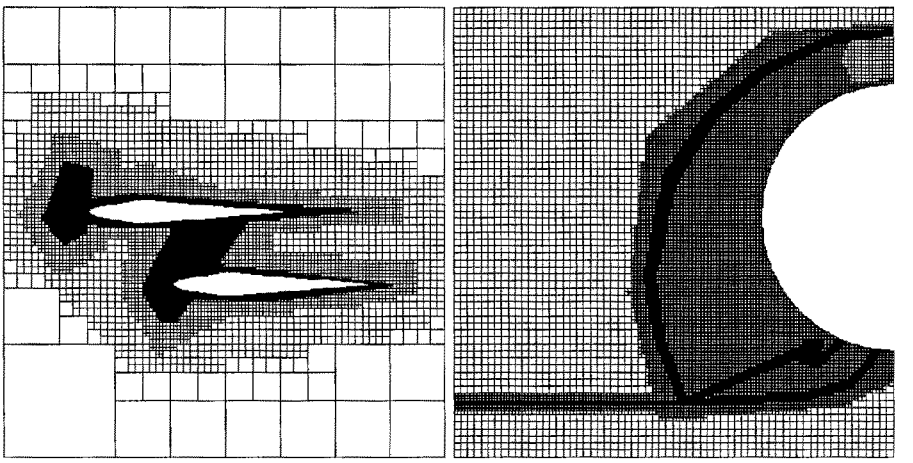
Partitions with low cut sizes might be very useful for our embedding problem. In several previous studies (e.g. [BB,DMM]), graph partitioning was used in a first step to partition the graph in as many clusters as there are nodes in the host graph. A second step then performs the one-to-one embedding of the cluster-graph into the host graph. This two step strategy integrates the powerful partitioning methods into the embedding problem.

The efficiency of heuristics strongly depends on the implementation details. Several libraries like *Jostle* ([WCE]) by Walshaw, *Metis* ([KK]) by Karypis and Kumar, *Scotch* ([PR]) by Pellegrini or *Party* ([PD]) by Preis and Diekmann exist to solve the partitioning problem. A library like *Scotch* also encounters the embedding problem, but the cost function takes into account only the total sum of the dilation of all edges and not the maximum dilation or maximum edge-congestion. In a first approach, we use these libraries to compute good partitions

of our graphs and compare the resulting loads and cut sizes to those heuristics described above.

4 Tests

In this section we will present some experimental results of the mentioned methods on two test graphs shown in Fig. 3. Both of them are subgraphs of the previously defined quasi grids.



(a) (b)

Fig. 3. (a) Graph *biplane* (21,701 nodes and 42,038 edges) and (b) graph *shock* (36,476 nodes and 71,290 edges).

The embedding is performed on an 8×8 grid as the host graph and the values for dilation, edge-congestion, load, and cut size are presented. A routing scheme for the edge-congestion is calculated in a sequential order for each path. To decide on a single path, the X/Y and the Y/X paths are considered and we choose the one with the lowest occuring edge-congestion along the path.

Our main cost criteria are the dilation and the edge-congestion which are shown in Fig. 4, (a) and (b). The results show that *Tile1* and *Tile3* have a very low dilation and that all *Tile*-heuristics have a low edge-congestion. Please note that the partitioning methods *Pac-Man*, *Party*, *Jostle*, and *Metis* are performing an embedding into a complete host graph. At this stage, we used the identical embedding on the grid to see how the dilation and congestion will be without optimization of the embedding. The maximum load, the cut edges and the CPU running times (sec) on a SUN Sparc20 workstation of the methods are shown in Fig. 4, (c), (d) and (e), respectively. A value of 100% refers to an optimal load balance. This or a just slightly unbalanced load is only guaranteed by the

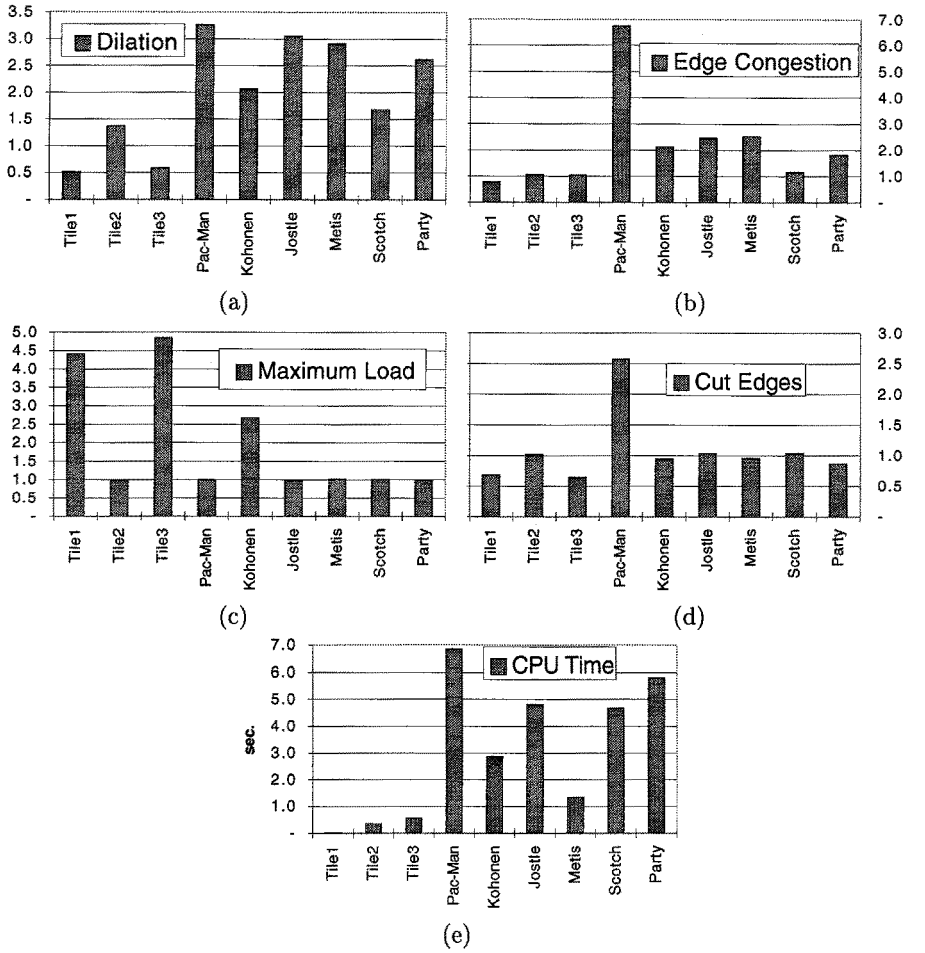


Fig. 4. Performances of tested algorithms: (a) Dilation, (b) edge-congestion, (c) maximum load, (d) cut size and (e) CPU time.

methods *Tile2*, *Jostle*, *Metis*, *Scotch*, *Party*, and *Pac-Man*. In case of *Tile1* and *Tile3*, the small values of the dilation imply high unbalance of the load.

The high values for the load for the heuristics *Tile1*, *Tile3*, and *Kohonen* are compensated by low cut sizes. In comparison with the other methods, *Party* seems to perform slightly better than the others. In general, the partitioning libraries produce a balanced load and very low cut sizes. This is a strong argument to integrate them in a two-step strategy for solving the embedding problem.

5 Conclusion

In this paper several heuristics for the off-line mapping of refinements of 2-dimensional grids are presented. They were all tested with two benchmark graphs

considering a wide set of parameters. According to the results of the experiments, the *Tile*-heuristics seem to guarantee a good trade-off between load and dilation/edge-congestion and, in comparison with other partitioning tools, PARTY seems to provide good results with respect to cut-size and load.

Although the Kohonen method does not provide very promising results, being applied for the static embedding considered so far, it seems to be more suitable for dynamic embeddings than the other considered methods. This method can explore the already computed embedding to react on local changes of the guest graph without completely recomputing the whole embedding.

As to future work, we are going to modify and improve some of the presented heuristics and also to study the dynamic version of the problem. As to more theoretical aspects, we are completing the analytical study of our heuristics. One of the most interesting aspects is the trade-off existing between the different metrics considered, in particular between load, dilation, and edge-congestion, for which we already gained some preliminary results.

References

[BB] M.J. Berger and S.H. Bokhari, *A Partitioning Strategy for Nonuniform Problems on Multiprocessors*, IEEE Trans. on Comp., C-36 (5), 570–580, 1987.

[BU] S.L. Bezrukov and W. Unger, *On Refinement of 2-Dimensional Grids*, Preprint, 1995.

[DMM] R. Diekmann, D. Meyer and B. Monien, *Parallel Decomposition of Unstructured FEM-Meshes*, Proc. of Irregular 95, LNCS 980, 199–215, 1995.

[DH] W.E. Donath and A.J. Hoffman, *Lower bounds for the partitioning of graphs*, IBM J. Res. Develop. 17, 1973, 420–425.

[DMT] S.E. Dorward, L.R. Matheson and R.E. Tarjan, *Toward efficient unstructured multigrid preprocessing*, Proc. of Irregular 96, LNCS 1117, 1996, 105–118.

[K] T. Kohonen, *Self-Organization and Associative Memory*, 3rd edition, Springer Verlag, Berlin 1989.

[KK] G. Karypis and V. Kumar, *A fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs*, Tech. Rep. 95-035, Dept. of Computer Science, U. of Minnesota, 1995.

[MS] B. Monien and I.H. Sudborough, *Embedding one Interconnection Network in Another*, Computing Suppl., 7, 1990, 257–282.

[PD] R. Preis and R. Diekmann, *The PARTY Partitioning-Library User Guide - Version 1.1*, Tech. Rep. TR-RSFB-96-024, U. Paderborn, 1996.

[PSL] A. Pothen, H.D. Simon and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl.,11, 1990, 430–452.

[PR] F. Pellegrini and J. Roman, *SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs*, Proc. of HPCN, 1996, 493–498.

[RMS] H. Ritter, T. Martinetz and K. Schulten, *Neural Computation and Self-Organizing Maps*, Addison Wesley, 1991.

[WCE] C. Walshaw, M. Cross and M.G. Everett, *A Localized Algorithm for Optimizing Unstructured Mesh Partitions*, Int. J. Supercomputer Appl., 9, 1995, 280–295.