MULTILEVEL MESH PARTITIONING FOR OPTIMIZING DOMAIN SHAPE

C. Walshaw[1]
M. Cross[1]
R. Diekmann[2]
F. Schlimbach[1]

Summary

Multilevel algorithms are a successful class of optimization techniques that address the mesh partitioning problem for mapping meshes onto parallel computers. They usually combine a graph contraction algorithm together with a local optimization method that refines the partition at each graph level. To date, these algorithms have been used almost exclusively to minimize the cut-edge weight in the graph with the aim of minimizing the parallel communication overhead. However, it has been shown that for certain classes of problems, the convergence of the underlying solution algorithm is strongly influenced by the shape or aspect ratio of the subdomains. Therefore, in this paper, the authors modify the multilevel algorithms to optimize a cost function based on the aspect ratio. Several variants of the algorithms are tested and shown to provide excellent results.

Address reprint requests to Chris Walshaw, School of Computing and Mathematical Sciences, University of Greenwich, 30 Park Row, Greenwich, London, SE10 9LS, U.K.; e-mail: C.Walshaw@gre.ac.uk.

The International Journal of High Performance Computing Applications, Volume 13, No. 4, Winter 1999, pp. 334-353

1 Introduction

The need for mesh partitioning arises naturally in many finite element (FE) and finite volume (FV) applications. Meshes composed of elements such as triangles or tetrahedra are often better suited than regularly structured grids for representing completely general geometries and resolving wide variations in behavior via variable mesh densities. Meanwhile, the modeling of complex behavior patterns means that the problems are often too large to fit onto serial computers, either because of memory limitations or computational demands or both. Distributing the mesh across a parallel computer so that the computational load is evenly balanced and the data locality maximized is known as mesh partitioning. It is well known that this problem is NP-complete, so in recent years much attention has been focused on developing suitable heuristics, and some powerful methods, many based on a graph corresponding to the communication requirements of the mesh, have been devised (e.g., Hendrickson and Leland, 1995).

A particularly popular and successful class of algorithms that address this mesh-partitioning problem is known as multilevel algorithms.

They usually combine a graph contraction algorithm, which creates a series of progressively smaller and coarser graphs together with a local optimization method that, starting with the coarsest graph, refines the partition at each graph level. To date, these algorithms have been used almost exclusively to minimize the cut-edge weight, a cost that approximates the total communications volume in the underlying solver. This is an important goal in any parallel application in order to minimize the communications overhead, but it has been shown (Vanderstraeten, Keunings, and Farhat, 1995) that for certain classes of solution algorithm, the convergence of the solver is actually heavily influenced by the shape or aspect ratio (AR) of the subdomains. In this case, the overall solution time can be more dependent on the number of iterations than on the parallel communications overhead (Vanderstraeten et al., 1996). In this paper, therefore, we modify the multilevel algorithms (the matching and local optimization) to optimize a cost function based on AR. We also abstract the process of modification to suggest how the mul-

[1]SCHOOL OF COMPUTING AND MATHEMATICAL SCIENCES, UNIVERSITY OF GREENWICH, LONDON, U.K.

[2]HILTI AG, CORPORATE RESEARCH, FL-9494 SCHAAN, PRINCIPALITY OF LIECHTENSTEIN

tilevel strategy can be modified into a generic technique that can optimize arbitrary cost functions.

1.1 DOMAIN DECOMPOSITION PRECONDITIONERS AND ASPECT RATIO

To motivate the need for aspect ratio optimization, we consider the requirements of a class of solution techniques. A natural parallel solution strategy for the underlying problem is to use an iterative solver such as the conjugate gradient (CG) algorithm together with domain decomposition (DD) preconditioning (e.g., Blazy, Borchers, and Dralle, 1995). DD methods take advantage of the partition of the mesh into subdomains by imposing artificial boundary conditions on the subdomain boundaries and solving the original problem on these subdomains (Bramble, Pasciac, and Schatz, 1986). The subdomain solutions are independent of each other and thus can be determined in parallel without any communication between processors. In a second step, an "interface" problem is solved on the inner boundaries, which depends on the jump of the subdomain solutions over the boundaries. This interface problem gives new conditions on the inner boundaries for the next step of the subdomain solution. Adding the results of the third step to the first gives the new conjugate search direction in the CG algorithm.

The time needed by such a preconditioned CG solver is determined by two factors, the maximum time needed by any of the subdomain solutions and the number of iterations of the global CG. Both are at least partially determined by the shape of the subdomains. While an algorithm such as the multigrid method as the solver on the subdomains is relatively robust against shape, the number of global iterations is heavily influenced by the AR of subdomains (Vanderstraeten et al., 1996). Essentially, the subdomains can be viewed as elements of the interface problem (Farhat, Maman, and Brown, 1995; Farhat, Mandel, and Roux, 1994), and just as with the normal finite element method, in which the condition of the matrix system is determined by the AR of elements, the condition of the preconditioning matrix is here dependent on the AR of subdomains.

"The time needed by such a preconditioned CG solver is determined by two factors, the maximum time needed by any of the subdomain solutions and the number of iterations of the global CG."

1.2 RELATED WORK

The idea of optimizing AR to maintain scalability in the solver was first developed by Farhat, Maman, and Brown (1995) and Farhat, Mandel, and Roux (1994). This was backed up by Vanderstraeten, Keunings, and Farhat (1995) and Vanderstraeten et al. (1996), who showed that partitioning for cut-edge weight was not necessarily the

most appropriate optimization for every solver. However, the field of mesh partitioning has changed somewhat since this work was carried out, and although other more recent work exists that takes AR into account (e.g., Diekmann, Meyer, and Monien, 1998; Diekmann, Schlimbach, and Walshaw, 1998; Schlimbach, 1998), our aim in this paper is to extend the ideas in light of recent developments in multilevel mesh-partitioning technology.

1.3 OVERVIEW

Below, in Section 2, we introduce the mesh partitioning problem and establish some terminology. We then discuss the mesh-partitioning problem as applied to AR optimization and describe how the graph needs to be modified to carry this out. Next, in Section 3, we describe the multilevel paradigm and present and compare three possible matching algorithms that take account of AR. In Section 4, we then describe a Kernighan-Lin (KL) (Kernighan and Lin, 1970) type iterative local optimization algorithm and describe four possible modifications that aim to optimize AR. Finally, in Section 5, we compare the results with a cut-edge partitioner, suggest how the multilevel strategy can be modified into a generic technique, and present some ideas for further investigation.

The principal innovations described in this paper are the following:

- in Section 2.3, where we describe how the graph can be modified to take AR into account;
- in Section 3.2, where we describe three matching algorithms based on AR;
- in Section 4.3, where we describe four ways of using the cost function to optimize for AR;
- in Section 4.5, where we describe how the bucket sort can be modified to take into account non-integer gains.

2 Mesh Partitioning

2.1 THE MESH-PARTITIONING PROBLEM

To define the mesh-partitioning problem, let $G = G(V, E)$ be an undirected graph of vertices V, with edges E that represent the data dependencies in the mesh. For the purposes of this paper, we assume that each graph vertex represents a mesh element and that graph edges arise from elements that are adjacent in the sense of sharing an element face. We assume that both vertices and edges can be weighted (with positive integer values) and that $|v|$ denotes the weight of a vertex v and similarly for edges and sets of vertices and edges. Given that the mesh needs to be distributed to P processors, define a partition π to be a mapping of V into P disjoint subdomains S_p such that $\cup_p S_p = V$. To evenly balance the load, the optimal subdomain weight is given by $\bar{S} := \lceil |V|/P \rceil$ (where the ceiling function $\lceil x \rceil$ returns the smallest integer $\geq x$), and the imbalance is then defined as the maximum subdomain weight divided by the optimal (since the computational speed of the underlying application is determined by the most heavily weighted processor).

Note that in the context of this paper, we use the term *load balancing* or *balancing* to mean that each subdomain is assigned an equal share of the total graph vertex weight. Indeed, in all the test meshes, every vertex has a weight of 1, and so we can simplify this further to say that perfect balance is attained if each subdomain is assigned an equal number of vertices. However, this should only be considered as balancing the underlying application if it is known that each graph vertex represents an equal amount of work in the application independent of the mesh partition. In fact, for the domain decomposition methods with which we concern ourselves here, this is usually not true, and the computational work is dominated by the cost of the local subdomain solutions. These are not normally independent of the mesh partition and indeed may be impossible to determine a priori, although for certain solution methods, it may be possible to optimize a related cost function (Vanderstraeten et al., 1996). However, it goes beyond the scope of this paper to address such issues, and we assume that a partition is balanced solely by sharing the vertices equally among the processors in the hope that this will approximately balance the underlying application.

The definition of the mesh-partitioning problem, then, is to find a partition that evenly balances the load or vertex weight in each subdomain while minimizing some cost function Γ. Typically, this cost function is simply the total weight of cut edges, but in this paper we describe a cost function based on AR. A more precise definition of the mesh-partitioning problem is therefore to find π such that $S_p \leq \bar{S}$ and such that Γ is minimized.

2.2 THE ASPECT RATIO AND COST FUNCTION

We seek to modify the methods by optimizing the partition on the basis of AR rather than cut-edge weight. To do this, it is necessary to define a cost function that we seek

to minimize, and a logical choice would be $\max_p \mathrm{AR}(S_p)$, where $\mathrm{AR}(S_p)$ is the AR of the subdomain S_p. However, maximum functions are notoriously difficult to optimize (indeed, it is for this reason that most mesh-partitioning algorithms attempt to minimize the total cut-edge weight rather than the maximum between any two subdomains), and so instead we choose to minimize the average AR:

$$\Gamma_{\mathrm{AR}} = \sum_p \frac{\mathrm{AR}(S_p)}{P}. \quad (1)$$

There are several definitions of AR, however. For example, for a given polygon S, a typical definition (Mitchell and Vasavis, 1992) is the ratio of the largest circle that can be contained entirely within S (inscribed circle) to the smallest circle that entirely contains S (circumcircle). However, these circles are not easy to calculate for arbitrary polygons, and in an optimization code in which ARs may need to be calculated very frequently, we do not believe this to be a practical metric. It may also fail to express certain irregularities of shape. A careful discussion of the relative merits of different ways of measuring AR may be found in Schlimbach (1998), and for the purposes of this paper, we follow the ideas therein and define the AR of a given shape by measuring the ratio of its perimeter length (surface area in 3-D) over that of some ideal shape with identical area (volume in 3-D).

Suppose then that in 2-D, the ideal shape is chosen to be a square. Given a polygon S with area ΩS and perimeter length ∂S, the ideal perimeter length (the perimeter length of a square with area ΩS) is $4\sqrt{\Omega S}$, and so the AR is defined as $\partial S/4\sqrt{\Omega S}$. Alternatively, if the ideal shape is chosen to be a circle, then the same argument gives the AR of $\partial S/2\sqrt{\pi\Omega S}$. In fact, given the definition of the cost function (1), it can be seen that these two definitions will produce the same optimization problem (and hence the same results) with the cost just modified by a constant C (where $C = 1/4$ for the square and $1/2\sqrt{\pi}$ for the circle). These definitions of AR are easily extendible to 3-D, and given a polyhedron S with volume ΩS and surface area ∂S, the AR can be calculated as $C\partial S/(\Omega S)^{2/3}$, where $C = 1/4$ if the cube is chosen as the optimal shape and $C = 1/\pi^{1/3}6^{2/3}$ for the sphere. Note that henceforth, in order to talk in general terms for both 2-D and 3-D, given an object S, we shall use the terms ∂S or *surface* for the surface area (3-D) or perimeter length (2-D) of the object and ΩS or *volume* for the volume (3-D) or area (2-D).

Of the above definitions of AR, we choose to use the circle/sphere-based formulas since they guarantee that the aspect ratios of any shape are ≥ 1.

"The definition of the mesh-partitioning problem, then, is to find a partition that evenly balances the load or vertex weight in each subdomain while minimizing some cost function Γ."

This gives a convenient formula for the cost function of

$$\Gamma_{\text{template}} = \frac{1}{C} \sum_p \frac{\partial S_p}{(\Omega S_p)^{\frac{d-1}{d}}}, \quad (2)$$

where $C = \pi^{\frac{1}{d}} (2d)^{\frac{d-1}{d}} P$, and d (= 2 or 3) is the dimension of the mesh. We refer to this cost function as Γ_{template} or Γ_t because of the way it tries to match shapes to chosen templates.

In fact, it will turn out (see, e.g., Section 3.2) that even this function may be too complex for certain optimization needs, and we can define a simpler one by assuming that all subdomains have approximately the same volume, $\Omega S_p \approx \Omega M / P$, where ΩM is the total volume of the mesh. This assumption may not necessarily be true, but it is likely to be true locally (see Section 4.4). We can then approximate (2) by

$$\Gamma_{\text{template}} \approx \frac{1}{C'} \sum_p \partial S_p, \quad (3)$$

where $C' = (\pi P)^{\frac{1}{d}} (2d\Omega M)^{\frac{d-1}{d}}$. This can be simplified still further by noting that the surface of each subdomain S_p consists of two components, the *exterior* surface, $\partial^e S_p$, where the surface of the subdomain coincides with the surface of the mesh ∂M, and the *interior* surface, $\partial^i S_p$, where S_p is adjacent to other subdomains and the surface cuts through the mesh. Thus, we can break the $\sum_p \partial S_p$ term in (3) into two parts, $\sum_p \partial^i S_p$ and $\sum_p \partial^e S_p$, and simplify (3) further by noting that $\sum_p \partial^e S_p$ is just ∂M, the exterior surface of the mesh M. This then gives us a second cost function to optimize:

$$\Gamma_{\text{surface}} = \frac{1}{K_1} \sum_p \partial^i S_p + K_2, \quad (4)$$

where $K_1 = (\pi P)^{\frac{1}{d}} (2d\Omega M)^{\frac{d-1}{d}}$ and $K_2 = \partial M / K_1$. We refer to this cost function as Γ_{surface} or Γ_s because it is just concerned with optimizing surfaces.

2.3 MODIFYING THE GRAPH

To use these cost functions in a graph-partitioning context, we must add some additional qualities to the graph. Figure 1 shows a very simple mesh (1a) and its dual graph (1b). Each element of the mesh corresponds to a vertex in the graph. The vertices of the graph can be weighted as is usual (see Section 2.1), but in addition, vertices store the volume and total surface of their corresponding element (e.g., $\Omega v_1 = \Omega e_1$ and $\partial v_1 = \partial e_1$). We also weight the edges of the graph with the size of the surface they correspond to. Thus, in Figure 1, if $D(b, c)$ refers to the distance between points b and c, then the weight of edge (v_1, v_2) is set to $D(b, c)$. In this way, for vertices v_i corresponding to elements that have no exterior surface, the sum of their edge weights is equivalent to their surface ($\partial v_i = \sum_E |(v_i, v_j)|$). Thus, for vertex v_2, $\partial v_2 = \partial e_2 = D(b,c) + D(c,e) + D(e,b) = |(v_2, v_1)| + |(v_2, v_5)| + |(v_2, v_3)|$.

When it comes to combining elements together, either into subdomains or for the multilevel matching (Section 3) these properties, volume and surface can be easily combined. Thus, in Figure 1c, where $E_1 = e_1 + e_4$, $E_2 = e_3 + e_5$, and $E_3 = e_3$, we see that volumes can be directly summed—for example, $\Omega V_1 = \Omega E_1 = \Omega e_1 + \Omega e_4 = \Omega v_1 + \Omega v_4$, as can edge weights, for example, $|(V_1, V_2)| = D(b,c) + D(c,d) = |(v_1, v_2)| + |(v_4, v_5)|$. The surface of a combined object S is the sum of the surfaces of its constituent parts less twice the interior surface—for example, $\partial V_1 = \partial E_1 = \partial e_1 + \partial e_4 - 2 \times D(a,c) = \partial v_1 + \partial v_4 - 2|(v_1, v_4)|$. These properties are very similar to properties in conventional graph algorithms, where the volume combines in the same way as weight and surfaces combine as the sum of edge weights (although including an additional term that expresses the exterior surface ∂^e). The edge weights function identically.

Note that with these modifications to the graph, it can be seen that if we optimize using the Γ_s cost function (4), the AR mesh-partitioning problem is identical to the cut-edge weight mesh-partitioning problem with a special edge weighting. However, the inclusion of non-integer edge weights does have an effect on the some of the techniques that can be used (e.g., see Section 4.5).

2.4 TESTING THE ALGORITHMS

Throughout this paper, we compare the effectiveness of different approaches using a set of test meshes. The algorithms have been implemented within the framework of JOSTLE, a mesh-partitioning software tool developed at the University of Greenwich and freely available for academic and research purposes under a licensing agreement (available from http://www.gre.ac.uk/~c.walshaw/jostle). The experiments were carried out on a DEC Alpha with a 466 MHz CPU and 1 Gbyte of memory. Due to space considerations, we only include eight test meshes, but they have been chosen to be a representative sample of medium- to large-scale real-life problems and include both

2-D and 3-D examples. Table 1 gives a list of the meshes and their sizes in terms of the number of vertices and edges. The table also shows the aspect ratio of each entire mesh and the mesh grading, which here we define as the maximum surface of any element over the minimum surface, and these two figures give a guide as to how difficult the optimization may be. For example, "uk" is simply a triangulation of the British mainland and hence has a very intricate boundary and therefore a high aspect ratio. The "wing" mesh, on the other hand, is a cube containing a hollowed-out section in the shape of an airplane wing; the AR is therefore reasonably close to 1, but the grading is very high as the mesh goes from very small elements close to the wing to very large ones in the far field.

Table 2 shows the results of the most successful combination of algorithms—surface matching (SM) (see Section 3.2) and local template gain/template cost optimization (LTGTC) (see Section 4.3)—which were chosen as a benchmark for the other combinations. For the four different values of P (the number of subdomains), the table shows the average aspect ratio as given by Γ_t, the edge cut $|E_c|$ (i.e., the number of cut edges, not the weight of cut edges weighted by surface size), and the time in seconds, t_s, to partition the mesh. Notice that with the exception of the "uk" mesh and P = 16, all partitions have average aspect ratios ≤ 1.53, which is within the target range of [1.0, 1.57] suggested in Diekmann, Meyer, and Monien (1998) and Diekmann, Schlimbach, and Walshaw (1998).[1] Indeed, for the "uk" mesh, it is no surprise that the results for P = 16 are not optimal because the subdomains inherit some of the poor AR from the original mesh (which has an AR of 3.82), and it is only when the mesh is split into small enough pieces—P = 32, 64, or 128—that the optimization succeeds in ameliorating this effect. Intuitively, this also gives a hint as to why DD methods are a very successful technique as a solver.

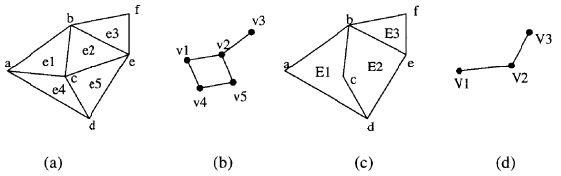

Fig. 1 A simple mesh (a), its dual (b), the same mesh with combined elements (c), and its dual (d)

Table 1
Test Meshes

Mesh	Number of Vertices	Number of Edges	Type	Aspect Ratio	Mesh Grading
uk	4824	6837	2-D triangles	3.82	7.98e+02
4elt-dual	30,269	44,929	2-D triangles	1.08	2.13e+04
t60k	60,005	89,440	2-D triangles	1.80	2.00e+00
dime20	224,843	336,024	2-D triangles	2.11	3.70e+03
cs4	22,499	43,858	3-D tetrahedra	1.32	9.64e+01
wing	62,032	121,544	3-D tetrahedra	1.27	1.08e+06
mesh100	103,081	200,976	3-D tetrahedra	2.02	2.45e+02
cyl3	232,362	457,853	3-D tetrahedra	1.59	8.42e+00

Table 2
Final Results Using Surface Matching and Local Template Gain/Template Cost Optimization

	$P = 16$			$P = 32$			$P = 64$			$P = 128$		
Mesh	Γ_t	$\lvert E_c \rvert$	t_s	Γ_t	$\lvert E_c \rvert$	t_s	Γ_t	$\lvert E_c \rvert$	t_s	Γ_t	$\lvert E_c \rvert$	t_s
uk	1.62	197	0.27	1.46	332	0.40	1.40	559	0.48	1.40	937	1.08
4elt-dual	1.24	898	0.88	1.28	1358	1.18	1.27	1985	1.40	1.29	2737	1.82
t60k	1.34	1031	1.37	1.28	1607	1.62	1.30	2524	2.03	1.31	3806	2.80
dime20	1.43	1889	4.92	1.34	2886	5.32	1.30	4651	6.15	1.26	6732	7.17
cs4	1.47	2625	2.00	1.47	3660	2.52	1.47	5000	3.23	1.48	6629	3.57
wing	1.37	9346	4.67	1.40	13,640	6.48	1.41	15,706	7.57	1.43	17,027	10.27
mesh100	1.53	6020	4.23	1.49	8413	7.23	1.49	11,577	6.38	1.50	15,995	8.00
cyl3	1.47	10,929	8.68	1.52	16,382	10.05	1.52	22,355	12.03	1.51	29,926	15.97

The partitioning times ranged from under 1 second to a maximum of 16 seconds (for the largest 3-D mesh). Experience suggests that this is not an unreasonable overhead for a domain decomposition-based method.

3 The Multilevel Paradigm

In recent years, it has been recognized that an effective way of both speeding up partition refinement and, perhaps more important, giving it a global perspective is to use multilevel techniques. The idea is to match pairs of vertices to form clusters, use the clusters to define a new graph, and recursively iterate this procedure until the graph size falls below some threshold. The coarsest graph is then partitioned, and the partition is successively optimized on all the graphs starting with the coarsest and ending with the original. This sequence of contraction followed by repeated expansion/optimization loops is known as the multilevel paradigm and has been successfully developed as a strategy for enhancing many partitioning approaches. The multilevel idea was first proposed by Barnard and Simon (1994) as a method of speeding up spectral bisection. It was subsequently generalized by Hendrickson and Leland (1993), who employed it to give global partition quality to local refinement algorithms such as that of Kernighan and Lin (1970) and by Vanderstraeten et al. (1996), who used it to speed up stochastic optimization techniques such as simulated annealing (Kirkpatrick, Gelatt, and Vecchi, 1983). Several algorithms for carrying out the matching have been devised by Karypis and Kumar (1995a), while Walshaw

and Cross (1998) describe a method for using imbalance in the coarsest graphs to enhance the final partition quality.

3.1 IMPLEMENTATION

Graph Contraction. To create a coarser graph $G_{l+1}(V_{l+1}, E_{l+1})$ from $G_l(V_l, E_l)$, we use a variant of the edge contraction algorithm proposed by Hendrickson and Leland (1993). The idea is to find a maximal independent subset of graph edges, or a *matching* of vertices, and then collapse them. The set is independent because no two edges in the set are incident on the same vertex (so no two edges in the set are adjacent) and maximal because no more edges can be added to the set without breaking the independence criterion. Having found such a set, each selected edge is collapsed and the vertices, $u_1, u_2 \in V_l$, say, at either end of it are merged to form a new vertex, $v \in V_{l+1}$ with weight $|v| = |u_1| + |u_2|$.

The Initial Partition. Having constructed the series of graphs until the number of vertices in the coarsest graph is smaller than some threshold, the normal practice of the multilevel strategy is to carry out an initial partition. Here, following the idea of Gupta (1996), we contract until the number of vertices in the coarsest graph is the same as the number of subdomains, *P*, and then simply assign vertex *i* to subdomain S_i. Unlike Gupta, however, we do not carry out repeated expansion/contraction cycles of the coarsest graphs to find a well-balanced initial partition but instead, since our optimization algorithm incorporates balancing (of the vertex weights) (see Section 2.1), we commence on the expansion/optimization sequence immediately.

Partition Expansion. Having optimized the partition on a graph G_l, the partition must be interpolated onto its parent G_{l-1}. The interpolation itself is a trivial matter; if a vertex $v \in V_l$ is in subdomain S_p, then the matched pair of vertices that it represents, $v_1, v_2 \in V_{l-1}$, will be in S_p.

3.2 INCORPORATING THE ASPECT RATIO

The matching part of the multilevel strategy can be easily modified in several ways to take AR into account, and in each case the vertices are visited (at most once) using a randomly ordered linked list. Each vertex is then matched with an unmatched neighbor using the chosen matching algorithm, and it and its match are removed from the list. Vertices with no unmatched neighbors remain unmatched and are also removed. In addition to random matching (RM) (Hendrickson and Leland, 1995), where vertices are matched with random neighbors, we propose and have tested three matching algorithms.

"The matching part of the multilevel strategy can be easily modified in several ways to take AR into account, and in each case the vertices are visited (at most once) using a randomly ordered linked list."

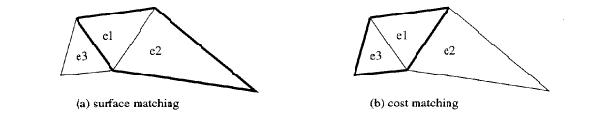

Fig. 2 Surface (a) and cost (b) matching

Surface Matching (SM). As we have seen in Section 2.3, the AR partitioning problem can be approximated by the cut-edge weight problem using (4), the Γ_s cost function, and so the simplest matching is to use the heavy edge approach of Karypis and Kumar (1995a), where the vertex matches across the heaviest edge to any of its unmatched neighbors. This is the same as matching across the largest surface (since here edge weights represent surfaces), and we refer to this as *surface matching.*

Template Cost Matching (TCM). A second approach follows the ideas of Bouhmala (1998) and matches vertices with the neighbor that minimizes the given cost function. In this case, the chosen vertex matches with the unmatched neighbor which gives the resulting cluster the best aspect ratio. Using the Γ_t cost function, we refer to this as *template cost matching.*

Surface Cost Matching (SCM). This is the same idea as TCM only using the Γ_s cost function, (4), which is faster to calculate and matches a vertex with the neighbor that minimizes the surface of the resulting cluster.

Figure 2 motivates the difference between surface matching (SM) and cost matching (SCM and TCM). For surface matching, the graph vertex corresponding to e_1 matches across the largest surface area, in this case with e_2. For cost matching, the graph vertex corresponding to e_1 matches to minimize the aspect ratio (TCM) or surface area (SCM) of the resulting cluster, in this case with e_3.

3.3 RESULTS FOR DIFFERENT MATCHING FUNCTIONS

In Tables 3, 4, and 5, we compare the results in Table 2, where SM was used, with RM, SCM, and TCM, respectively. In all cases, the LTGTC optimization algorithm (see Section 4.3) was used. For each value of *P*, the first

Table 3
Random Matching Results Compared with Surface Matching

Mesh	*P* = 16		*P* = 32		*P* = 64		*P* = 128	
	Γ_t	$\frac{\Gamma(\mathrm{RM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{RM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{RM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{RM})-1}{\Gamma(\mathrm{SM})-1}$
uk	1.65	1.05	1.49	1.06	1.40	1.01	1.39	0.98
4elt-dual	1.29	1.20	1.30	1.09	1.29	1.05	1.29	1.02
t60k	1.36	1.08	1.36	1.26	1.36	1.19	1.37	1.20
dime20	1.45	1.05	1.39	1.16	1.39	1.29	1.35	1.33
cs4	1.58	1.24	1.52	1.12	1.55	1.17	1.53	1.10
wing	1.44	1.17	1.44	1.12	1.44	1.08	1.46	1.07
mesh100	1.59	1.10	1.52	1.05	1.53	1.08	1.57	1.13
cyl3	1.53	1.11	1.52	1.00	1.59	1.14	1.56	1.10
Average		1.12		1.11		1.13		1.12

Table 4
Surface Cost Matching Results Compared with Surface Matching

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Γ_t	$\frac{\Gamma(\mathrm{SCM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{SCM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{SCM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{SCM})-1}{\Gamma(\mathrm{SM})-1}$
uk	1.61	0.98	1.48	1.05	1.39	0.97	1.39	0.99
4elt-dual	1.26	1.09	1.25	0.90	1.26	0.97	1.28	0.98
t60k	1.30	0.90	1.25	0.88	1.31	1.03	1.31	1.01
dime20	1.38	0.88	1.34	0.99	1.30	1.01	1.28	1.07
cs4	1.50	1.06	1.52	1.12	1.51	1.07	1.51	1.07
wing	1.41	1.10	1.42	1.07	1.42	1.02	1.42	0.99
mesh100	1.55	1.02	1.55	1.11	1.52	1.06	1.52	1.04
cyl3	1.51	1.08	1.50	0.95	1.54	1.03	1.55	1.08
Average		1.01		1.01		1.02		1.03

Table 5
Template Cost Matching Results Compared with Surface Matching

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Γ_t	$\frac{\Gamma(\mathrm{TCM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{TCM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{TCM})-1}{\Gamma(\mathrm{SM})-1}$	Γ_t	$\frac{\Gamma(\mathrm{TCM})-1}{\Gamma(\mathrm{SM})-1}$
uk	1.65	1.04	1.48	1.05	1.41	1.03	1.39	0.99
4elt-dual	1.26	1.08	1.28	0.99	1.28	1.03	1.27	0.94
t60k	1.29	0.87	1.31	1.09	1.29	0.96	1.31	1.02
dime20	1.39	0.90	1.34	1.00	1.28	0.92	1.28	1.07
cs4	1.49	1.04	1.47	1.01	1.50	1.05	1.50	1.04
wing	1.41	1.09	1.41	1.04	1.41	1.00	1.42	0.98
mesh100	1.48	0.89	1.47	0.96	1.51	1.03	1.51	1.03
cyl3	1.49	1.04	1.50	0.95	1.53	1.03	1.52	1.03
Average		0.99		1.01		1.01		1.01

column shows the average AR, Γ_t, of the partitioning. The second column for each value of P then compares results with those in Table 2 using the metric $\frac{\Gamma(\mathrm{RM})-1}{\Gamma(\mathrm{SM})-1}$ for RM and so forth. Thus, a figure > 1 means that RM has produced worse results than SM. These comparisons are then averaged and so it can be seen, for example, for P = 16 that RM produces results 12% (1.12) worse on average than SM. Indeed, RM is better than SM in only one case ("uk," P = 128) and up to 33% worse ("dime20," P = 128), with the overall average quality 12% worse than SM. This is not altogether surprising since the AR of elements in the coarsest graph can be very poor if the matching takes no account of it, and hence the optimization has to work with

Fig. 3 Final "element" shapes for random (a) and surface (b) matching

badly shaped elements. This limitation is graphically demonstrated in Figure 3, which shows an example of the shapes of the final 16 clusters in the coarsest graph of an example 2-D mesh. While the shapes for SM (3b) are very good (although the borders are somewhat irregular), the shapes for RM (3a) are extremely poor, and as a result, the partition optimization on the coarser graphs is limited in the improvements that can be made.

When it comes to comparing SM with SCM and TCM (Tables 4 and 5), there is actually very little difference; SCM is about 1.9% worse on average and TCM only about 0.6% worse. This suggests that the multilevel strategy is relatively robust to the matching algorithm, provided the AR is taken into account in some way.

With regard to partitioning time, RM was on average about 32.9% slower than SM; as explained above, this is because the optimization is inhibited by the poor quality of the coarser graph and thus took considerably longer. SCM and TCM were about 14.3% and 8.5% slower than SM, respectively; this is due to the slightly slower matching process. However, the multilevel partitioning is generally very fast, and any of the intelligent matching algorithms (as opposed to random matching) do not add significantly to the optimization time.

Overall, this suggests that SM is the algorithm of choice, although there is little benefit over TCM.

4 The Kernighan-Lin Optimization Algorithm

In this section, we discuss the key features of an optimization algorithm, fully described in Walshaw and Cross (1998) and then in Section 4.3 describe how it can be modified to optimize for AR. It is a Kernighan-Lin (KL) type algorithm incorporating a hill-climbing mechanism to enable it to escape from local minima. The algorithm uses bucket sorting (Section 4.5), the linear time complexity improvement of Fiduccia and Mattheyses (1982), and is a partition optimization formulation. In other words, it optimizes a partition of P subdomains rather than a bisection.

4.1 THE GAIN FUNCTION

A key concept in the method is the idea of gain. The gain $g(v, q)$ of a vertex v in subdomain S_p can be calculated for every other subdomain, S_q, $q \neq p$, and expresses how much the cost of a given partition would be improved were v to migrate to S_q. Thus, if π denotes the current partition and π' the partition if v migrates to S_q, then for a cost function Γ, the gain $g(v, q) = \Gamma(\pi) - \Gamma(\pi')$. Assuming the migra-

tion of v only affects the cost of S_p and S_q (as is true for Γ_t and Γ_s), then we get

$$g(v,q) = \mathrm{AR}(S_q) - \mathrm{AR}(S_q + v) + \mathrm{AR}(S_p) - \mathrm{AR}(S_p - v). \quad (5)$$

For Γ_t, this gives an expression

$$g_{\text{template}}(v,q) = \frac{1}{C}\left[\frac{\partial S_q}{(\Omega S_q)^{\frac{d-1}{d}}} - \frac{\partial\{S_q + v\}}{(\Omega\{S_q + v\})^{\frac{d-1}{s}}} + \frac{\partial S_p}{(\Omega S_p)^{\frac{d-1}{d}}} - \frac{\partial\{S_p - v\}}{(\Omega\{S_p - v\})^{\frac{d-1}{d}}}\right], \quad (6)$$

which cannot be further simplified. However, for Γ_s, since

$$\begin{aligned}\mathrm{AR}(S_q) - \mathrm{AR}(S_q + v) &= \frac{1}{K_1}\{\partial^i S_q - \partial^i (S_q + v)\} \\ &= \frac{1}{K_1}\{\partial^i S_q - (\partial^i S_q + \partial^i v - 2|(S_{q,}\, v)|)\} \\ &= \frac{1}{K_1}\{2|(S_q, v)| - \partial^i v\},\end{aligned}$$

where $|(S_q, v)|$ denotes the sum of edge weights between S_q and v, we get

$$g_{\text{surface}}(v,q) = \frac{2}{K_1}\{|(S_q, v)| - |(S_p, v)|\}. \quad (7)$$

Notice in particular that g_{surface} is the same as the cut-edge weight gain function and that it is entirely localized (i.e., the gain of a vertex only depends on the length of its boundaries with a subdomain and not on any intrinsic qualities of the subdomain that could be changed by non-local migration).

4.2 THE ITERATIVE OPTIMIZATION ALGORITHM

The iterative optimization algorithm has been specifically constructed to exploit the flexibility inherent in the multi-level paradigm and uses imbalance in the coarser graphs to enhance the final partition quality. More specifically, by allowing a large imbalance in the coarsest graphs, a better partition may be found than if balance is rigidly enforced, and by removing this imbalance gradually throughout the multilevel procedure, this quality is not degraded. To this end, the optimization defines a balancing schedule—that is, an increasing series of target subdo-

"The iterative optimization algorithm has been specifically constructed to exploit the flexibility inherent in the multilevel paradigm and uses imbalance in the coarser graphs to enhance the final partition quality."

main weights, T_l, one for each graph G_l. If every subdomain, S_p, is not heavier than this target (i.e., $\max|S_p| \le T_l$), then we say that the graph is sufficiently balanced, and the optimization can concentrate on refinement alone (as long as the balance is not destroyed). However, if $\max|S_p| > T_l$, then the optimization must concentrate on balancing (with some regard to refinement), and this is achieved by determining a balancing flow—that is, a schedule of weight to be transferred, F_{pq}, between every pair of adjacent subdomains, S_p and S_q, which will balance the subdomain weights. Various balancing schedules, together with an algorithm due to Hu, Blake, and Emerson (1998) for determining a balancing flow, are fully described in Walshaw and Cross (1998). Here we use the most successful balancing schedule from that paper and set $T_l = \theta_l \bar{S}$, where $\bar{S} = \lceil |V|/P \rceil$ is just the optimal subdomain weight (see Section 2.1), and

$$\theta_l = \left\lceil 1 + 2\left(\frac{P}{N_{l-1}}\right)^{\frac{1}{2}} \right\rceil,$$

where N_{l-1} is the number of vertices in G_{l-1}, the parent graph of G_l. In other words, a graph G_l is considered balanced if the imbalance is less than $\theta_l = 1 + 2\left(\frac{P}{N_{l-1}}\right)^{\frac{1}{2}}$ for $l > 0$.

The iterative optimization algorithm, as is typical for KL-type algorithms, has inner and outer iterative loops, with the outer loop terminating when no migration takes place during an inner loop. The optimization uses two bucket-sorting structures or bucket trees (see Section 4.5) and is initialized by calculating the gain for all border vertices and inserting them into one of the bucket trees. These vertices will subsequently be referred to as *candidate* vertices and the tree containing them as the *candidate tree*.

The inner loop proceeds by examining candidate vertices, highest gain first (by always picking vertices from the highest ranked bucket), testing whether the vertex is acceptable for migration (see below), and then transferring it to the other bucket tree (the tree of examined vertices). This inner loop terminates when the candidate tree is empty, although it may terminate early if the partition cost (i.e., the average aspect ratio) rises too far above the cost of the best partition found so far. Once the inner loop has terminated, any vertices remaining in the candidate tree are transferred to the examined tree, and finally pointers to the two trees are swapped ready for the next pass through the inner loop.

Migration Acceptance. Let T refer to the target weight for the graph and W represent the weight of the largest subdomain, $W = \max_p |S_p|$. If the required flow from subdomain S_p to subdomain S_q is F_{pq}, a candidate vertex v with weight $|v|$ (> 0) is acceptable for migration from S_p to S_q (with weights $|S_p|$ and $|S_q|$) if

$$\begin{aligned} &\text{(a) } W > T \text{ and } 2F_{pq} > |v| \\ &\text{or} \\ &\text{(b) } W \le T \text{ and } |S_q| + |v| \le T. \end{aligned} \tag{8}$$

These criteria reflect the aim of trying to balance the graph down to the target weight, T, and then keeping it there. If the graph is not yet within the imbalance tolerance (i.e., $W > T$), then (8a) only allows migration, which reduces the required flow. Condition (8b) guarantees that once balance is achieved, the graph cannot become unbalanced again.

Migration Confirmation. The algorithm also uses a KL-type hill-climbing strategy. In other words, vertex migration from subdomain to subdomain can be accepted even if it degrades the partition quality and later, based on the subsequent evolution of the partition, either rejected or confirmed. During each pass through the inner loop, a record of the best partition achieved by migration within that loop is maintained together with a list of vertices that have migrated since that value was attained. If subsequent migration finds a "better" partition, then the migration is confirmed, and the list is reset. Once the inner loop is terminated, any vertices remaining in the list (vertices whose migration has not been confirmed) are migrated back to the subdomains they came from when the optimal cost was attained.

To define a "better" partition, let $\bar{\pi}$ represent the best partition reached so far and π^i the subsequent partition after some migration (i.e., after some iterations of the inner loop). Each partition has a cost associated with it, $C(\pi)$, and an imbalance that depends on $W(\pi)$, the weight of the largest subdomain in that partition. Again, let T represent the target weight for the graph. Denoting $C(\pi^i)$ and $W(\pi^i)$ by C^i and W^i (and similarly for $\bar{\pi}$), then π^i is confirmed as a new optimal partition if

$$\begin{aligned} &\text{(a) } C^i < \bar{C} \\ &\text{or} \\ &\text{(b) } C^i = \bar{C} \text{ and } W^i < \bar{W} \\ &\text{or} \\ &\text{(c) } T \le W^i < \bar{W}. \end{aligned} \tag{9}$$

Condition (9c) simply states that while the graph is unbalanced (i.e., $W^i > T$), any partition that improves the balance is confirmed. Conditions (9a) and (9b) are more typical of KL-type algorithms and confirm any partition that either improves on the optimal cost (9a) or improves on the optimal balance without raising the cost (9b).

4.3 INCORPORATING THE ASPECT RATIO: LOCALIZATION

One of the advantages of using cut-edge weight as a cost function is its localized nature. When a graph vertex migrates from one subdomain to another, only the gains of adjacent vertices are affected. In contrast, when using the graph to optimize AR, if a vertex *v* migrates from S_p to S_q, the volume and surface of both subdomains will change. This, in turn, means that when using the template cost function (2), the gain of all border vertices both within and abutting subdomains S_p and S_q will change. Strictly speaking, all these gains should be adjusted with the huge disadvantage that this may involve thousands of floating-point operations and hence be prohibitively expensive. We have tested (see Table 8) a version that includes full updating but, as alternatives, we propose three localized variants.

Surface Gain/Surface Cost (SGSC). The simplest way to localize the updating of the gains is to make the assumption in Section 2.2 that the subdomains all have approximately equal volume and to use the surface cost function Γ_s from (4). As mentioned in Section 2.3, the problem immediately reduces to the cut-edge weight problem, albeit with non-integer edge weights, and from (7) only the gains of the vertices adjacent to the migrating vertex will need updating. However, if this assumption is not true, it is not clear how well Γ_s will optimize the AR, and below we provide some experimental results.

Surface Gain/Template Cost (SGTC). The second method we propose for localizing the updates of gain relies on the observation that the gain is simply used as a method of rating the vertices so that the algorithm always visits those with highest gain first (using the bucket sort). It is not clear how crucial this rating is to the success of the algorithm, and indeed Karypis and Kumar (1995b) demonstrated that (at least when optimizing for cut-edge weight) almost as good results can be achieved by simply visiting the vertices in random order. We therefore propose approximating the gain with the surface cost function Γ_s from (4) to rate the vertices and store them in the bucket tree structure but using the template cost function Γ_t from (2) to assess the change in cost when actually migrating a vertex. This localizes the gain function.

Local Template Gain/Template Cost (LTGTC). A third possibility we propose is to actually use the template cost function, Γ_t, for adjusting the gain but only adjusting the gain of those vertices adjacent to the migrating vertex. The motivation is that the neighbors of the migrating vertex are likely to have large changes in gain, whereas the gains of other vertices are likely to only change marginally (since they are only affected by the change in volume and surface of subdomains). The disadvantage is that the gains will become progressively more and more inaccurate as the optimization progresses; however, they are still likely to be as accurate as using the surface cost.

Finally note that the implementation, which, when a vertex migrates from subdomain S_p to S_q, involves full updating of the gains of all vertices in and adjacent to the borders of S_p and S_q, is referred to as template gain/template cost (TGTC).

4.4 RESULTS FOR DIFFERENT OPTIMIZATION FUNCTIONS

Tables 6 and 7 compare SGSC and SGTC optimization against the LTGTC results from Table 2. Both sets of results use surface matching (SM). The tables are in the same form as those in Section 3.3 and show that on average the surface gain function provides results that are 12.2% (SGSC) and 14.1% (SGTC) worse than LTGTC.

Note that in earlier results (Walshaw et al., 1998), we concluded that SGTC was the algorithm of choice, and the reason for this discrepancy is explained in the test meshes used. In Walshaw et al. (1998), we did not use the "4elt-dual" and "wing" meshes, which contain the highest mesh grading (the ratio of the largest surface of an element to the smallest), respectively, 2.13e+4 and 1.08e+6. Looking at the results in more detail, then, "4elt-dual" gives average aspect ratios between 31% and 79% worse than LTGTC, while "wing" ranges between 46% and 74% worse. These heavily influence the average results, and the reason we believe this to happen is that the approximation (3) made in Section 2.2, that every subdomain has approximately equal volume, completely breaks down for meshes with very high gradings. For all the other meshes, the SGSC and SGTC optimizations give average ARs from 20% better to 17% worse than LTGTC. In fact, if we exclude the "4elt-dual" and "wing" meshes from the results, on average, SGSC is 1.40% better than LTGTC, and SGTC is 0.36% worse. This leads us to suggest that as a very rough "ballpark" figure, if the mesh grading is of the order 10^3 or less, the surface gain function provides perfectly good results, but if greater than this, a more accu-

Table 6
Surface Gain/Surface Cost Optimization Compared with Local Template Gain/Template Cost

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$
uk	1.67	1.07	1.47	1.02	1.38	0.97	1.41	1.03
4elt-dual	1.35	1.47	1.42	1.52	1.36	1.33	1.38	1.32
t60k	1.27	0.79	1.25	0.87	1.30	1.01	1.27	0.88
dime20	1.36	0.84	1.34	1.00	1.28	0.92	1.26	1.01
cs4	1.48	1.03	1.51	1.10	1.50	1.06	1.50	1.04
wing	1.75	2.01	1.63	1.59	1.65	1.59	1.61	1.42
mesh100	1.48	0.90	1.48	0.97	1.51	1.04	1.52	1.04
cyl3	1.49	1.04	1.52	0.99	1.52	1.00	1.53	1.04
Average		1.14		1.13		1.12		1.10

Table 7
Surface Gain/Template Cost Optimization Compared with Local Template Gain/Template Cost

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{LTGTC})-1}$
uk	1.62	1.00	1.50	1.08	1.41	1.03	1.40	1.00
4elt-dual	1.43	1.79	1.42	1.50	1.37	1.36	1.38	1.31
t60k	1.30	0.90	1.25	0.90	1.29	0.95	1.27	0.87
dime20	1.35	0.82	1.33	0.98	1.30	0.98	1.30	1.17
cs4	1.53	1.15	1.49	1.05	1.51	1.08	1.50	1.04
wing	1.65	1.74	1.65	1.65	1.66	1.62	1.63	1.46
mesh100	1.48	0.91	1.48	0.97	1.52	1.05	1.50	1.01
cyl3	1.52	1.10	1.52	0.99	1.53	1.03	1.52	1.02
Average		1.18		1.14		1.14		1.11

rate estimate of gain is necessary and LTGTC is to be preferred.

Table 8 compares TGTC optimization, the version that uses full updating of gains, with LTGTC and shows that on average LTGTC and TGTC give results that are almost equivalent in quality (TGTC is in fact 0.50% worse than LTGTC) and hence that LTGTC provides a very good approximation to TGTC.

Again, we are not primarily concerned with partitioning times, but it was interesting to note that SGSC and SGTC were on average 28.41% and 24.38% faster than LTGTC. This is because the surface cost function, Γ_s, is much quicker to calculate when assessing or updating the gains (since it does not involve calculating $S_p^{\frac{d-1}{d}}$). TGTC

Table 8
Template Gain/Template Cost Optimization Compared with Local Template Gain/Template Cost

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Γ_t	$\frac{\Gamma(\text{TGTC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{TGTC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{TGTC})-1}{\Gamma(\text{LTGTC})-1}$	Γ_t	$\frac{\Gamma(\text{TGTC})-1}{\Gamma(\text{LTGTC})-1}$
uk	1.64	1.04	1.49	1.06	1.38	0.95	1.38	0.96
4elt-dual	1.23	0.98	1.27	0.96	1.28	1.01	1.28	0.96
t60k	1.33	1.00	1.28	0.98	1.31	1.03	1.31	1.01
dime20	1.39	0.91	1.34	1.01	1.29	0.95	1.29	1.13
cs4	1.48	1.04	1.49	1.04	1.48	1.01	1.49	1.01
wing	1.38	1.01	1.41	1.04	1.41	1.00	1.44	1.02
mesh100	1.52	0.97	1.50	1.01	1.50	1.01	1.51	1.02
cyl3	1.48	1.02	1.51	0.98	1.52	1.01	1.52	1.02
Average		0.99		1.01		1.00		1.02

was more than 50 times slower on average than LTGTC, and we feel that this justifies the assertion that full updating of gains is too expensive.

4.5 INCORPORATING THE ASPECT RATIO: BUCKET SORTING WITH NON-INTEGER GAINS

The bucket sort is an essential tool for the efficient and rapid sorting and adjustment of vertices by their gain. The concept was first suggested by Fiduccia and Mattheyses (1982), and the idea is that all vertices of a given gain g are placed together in an unsorted "bucket," which is ranked g. Finding a vertex with maximum gain then simply consists of finding the (nonempty) bucket with the highest rank and picking a vertex from it. If the vertex is subsequently migrated from one subdomain to another, then the gains of any affected vertices have to be adjusted and the list of vertices that are candidates for migration (re)sorted by gain. Using a bucket sort for this operation simply requires recalculating the gains of affected vertices and transferring them to the appropriate buckets. If a bucket sort were not used and the vertices were simply stored in a list in gain order, then the entire list would require resorting (or at least merge sorting with the sorted list of adjusted vertices), an essentially $O(N)$ operation for every migration.

The implementation of the bucket sort is fully described in (Walshaw and Cross, 1998). It includes a ranking for prioritizing vertices for migration, which incorporates their weight as well as their gain. The nonempty buckets are stored in a binary tree to save excessive mem-

ory use (since we do not know a priori how many buckets will be needed), and this structure is referred to above as a bucket tree.

The only difficulty in adapting this procedure to AR optimization is that with non-integer edge weights, the gains are also real non-integer numbers. This is not a major problem in itself as we can just give buckets an interval of gains rather than a single integer—that is, the bucket ranked 1 could contain any vertex with gain in the interval [0.5, 1.5). However, the issue of scaling then arises since, if using the surface gain function Γ_s (SGSC and SGTC), for a mesh entirely contained within the unit square/cube, all the vertices are likely to end up in one of two buckets (dependent only on whether they have positive or negative gains). Fortunately, we can easily calculate the maximum possible gain when using Γ_s, which would occur if the vertex with the largest surface, $v \in S_p$, for example, were entirely surrounded by neighbors in S_q. The maximum possible gain is then $2 \max_{v \in V} \partial v$ (strictly speaking, $2 \max_{v \in V} \partial^i v$), and similarly the minimum gain is $-2 \max_{v \in V} \partial v$. This means we can easily choose the number of buckets—*B*, for example—and scale the gain accordingly so that for a gain *g*, we calculate the appropriate bucket by finding the integer part of

$$\frac{gB}{4\max_{v \in V} \partial v}.$$

If using Γ_t as a gain function (LTGTC and TGTC), we can approximate the maximum gain (using Γ_s) to get the same scaling, although then the actual number of buckets used only approximates *B*. For either Γ_s or Γ_t, a problem still arises for meshes with a high grading because many of the elements will have an insignificant surface area compared to the maximum and hence be contained in a small number of buckets centered around 0. However, the experiments carried out here all used a scaling that allowed a maximum of $B = 1000$ buckets, and we have tested the algorithm up to $B = 10{,}000$ buckets without significant penalty in terms either of memory or runtime. We have also tested the algorithm with $B = 100$, although with a 4.8% average deterioration in the results.

5 Discussion and Conclusions

5.1 COMPARISON WITH CUT-EDGE WEIGHT PARTITIONING

In Table 9, we compare AR as produced by the edge-cut version of JOSTLE (EC) described in Walshaw and Cross (1998) with the results from Table 2. The EC partitioner never produces average aspect ratios that are actually better than the AR partitioner and, on average, gives results that are 19.8% worse than those of the AR partitioner and can be up to 61% worse. Notice that there is no real consistency in the differences, however (as there is in the differences between SGSC and SGTC compared with LTGTC; see Section 4.4), and we conclude that although an EC partitioner might be expected to produce reasonably good AR results (since a partition with a low value of $|E_c|$ is likely to have compact and therefore well-shaped subdomains), targeting the cost function on AR can provide considerably better results in most cases.

Meanwhile, in Table 10, we compare the edge cut produced by the EC version of JOSTLE with that of the AR version. As might be expected, EC partitioning produces the best results (about 14.4% better than AR). Notice, in particular, the results for the "wing" mesh (the mesh with the highest grading), where the EC partitioner produces partitions with up to 50% fewer cut edges than the AR partitioner, but the AR partitioner produces subdomains with aspect ratios 23% to 61% better. This demonstrates that a good partition for the aspect ratio is not necessarily a good partition for edge cut and vice versa.

In terms of time, the EC partitioner is about two times faster than AR on average. Again, this is no surprise since the AR partitioning involves floating-point operations (assessing cost and combining elements), while EC partitioning only requires integer operations. However, both are extremely fast at producing high-quality partitions.

5.2 GENERIC MULTILEVEL MESH PARTITIONING

In this paper, we have adapted a mesh-partitioning technique originally designed to solve the edge-cut partitioning problem to a different cost function. The question then arises, Is the multilevel strategy an appropriate technique for solving partitioning problems (or indeed other optimization problems) with different cost functions? Clearly, this is an impossible question to answer in general, but a few pertinent remarks can be made:

- For the AR-based cost functions at least, the method seems relatively sensitive to whether the cost is included in the matching. This suggests that, if possible, a generic multilevel partitioner should use the cost function to minimize the cost of the matchings. Note, however, that this may not be possible since a cost function that, say, measured the cost of a mapping onto a particular processor topology would be unable to

Table 9
AR Results for the Edge-Cut Partitioner Compared with the AR Partitioner

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	Γ_t	$\frac{\Gamma(\mathrm{EC})-1}{\Gamma(\mathrm{AR})-1}$	Γ_t	$\frac{\Gamma(\mathrm{EC})-1}{\Gamma(\mathrm{AR})-1}$	Γ_t	$\frac{\Gamma(\mathrm{EC})-1}{\Gamma(\mathrm{AR})-1}$	Γ_t	$\frac{\Gamma(\mathrm{EC})-1}{\Gamma(\mathrm{AR})-1}$
uk	1.68	1.10	1.55	1.19	1.46	1.15	1.41	1.04
4elt-dual	1.32	1.33	1.28	1.01	1.29	1.06	1.29	1.01
t60k	1.38	1.13	1.31	1.09	1.33	1.09	1.32	1.02
dime20	1.50	1.17	1.45	1.33	1.40	1.33	1.38	1.45
cs4	1.52	1.12	1.54	1.15	1.53	1.12	1.51	1.05
wing	1.60	1.61	1.61	1.53	1.61	1.50	1.53	1.23
mesh100	1.55	1.03	1.60	1.22	1.61	1.24	1.61	1.21
cyl3	1.59	1.25	1.63	1.21	1.61	1.18	1.59	1.17
Average		1.22		1.22		1.21		1.15

Table 10
$|E_c|$ Results for the Edge-Cut Partitioner Compared with the AR Partitioner

Mesh	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
	$\lvert E_c\rvert$	$\frac{\lvert E_c\rvert(\mathrm{EC})}{\lvert E_c\rvert(\mathrm{AR})}$	$\lvert E_c\rvert$	$\frac{\lvert E_c\rvert(\mathrm{EC})}{\lvert E_c\rvert(\mathrm{AR})}$	$\lvert E_c\rvert$	$\frac{\lvert E_c\rvert(\mathrm{EC})}{\lvert E_c\rvert(\mathrm{AR})}$	$\lvert E_c\rvert$	$\frac{\lvert E_c\rvert(\mathrm{EC})}{\lvert E_c\rvert(\mathrm{AR})}$
uk	182	0.92	305	0.92	512	0.92	809	0.86
4elt-dual	602	0.67	902	0.66	1515	0.76	2364	0.86
t60k	1016	0.99	1552	0.97	2439	0.97	3624	0.95
dime20	1382	0.73	2368	0.82	3717	0.80	5540	0.82
cs4	2496	0.95	3501	0.96	4666	0.93	6077	0.92
wing	5008	0.54	6866	0.50	9401	0.60	11,877	0.70
mesh100	4782	0.79	7851	0.93	11,100	0.96	15,202	0.95
cyl3	11,377	1.04	16,783	1.02	22,369	1.00	29,432	0.98
Average		0.83		0.85		0.87		0.88

function since at the matching stage no partition, and hence no mapping, exists.

- The optimization relies, for efficiency at least, on having a local gain function so that the migration of a vertex does not involve an $O(N/P)$ or even an $O(N)$ update. Here we were able to localize the updating of gains either by (a) making a simple approximation to localize the cost function or (b) by just ignoring the updating of nonadjacent vertices. However, it is not clear that (a) is always possible or that (b) is always valid. On the other hand, the underlying approach in (a), which essentially decouples the gain from the cost, does look quite promising for more general cost functions. In other

words, we can use a local (and possibly crude) approximation for the gain function and then control the convergence/hill climbing of the KL method (Kernighan and Lin, 1970) with the true cost. In some ways, this could be regarded as a hybrid of the KL method and simulated annealing (SA) (Kirkpatrick, Gelatt, and Vecchi, 1983) because in some ways you could regard SA as KL with a random gain function. This concept of decoupling the gain and cost functions is part of our ongoing research.

- The bucket sort is reasonably simple to convert to non-integer gains, but the process relies on being able to estimate the maximum gain. If this is not possible, it may not be easy to generate a good scaling that separates vertices of different gains into different buckets.

5.3 CONCLUSION AND FUTURE RESEARCH

We have shown that the multilevel strategy can be modified to optimize for the aspect ratio. In Section 2, we gave a definition of aspect ratio and showed how the graph could be modified to take AR into account. In Section 3.2, we described three matching algorithms (modifications of those already in the literature) that can be used to take AR into account and in Section 3.3 concluded that if it is not taken into account (i.e., random matching), the same quality of results cannot be expected. In Section 4.3, we described four ways of incorporating AR into a KL-based optimization algorithm. We then demonstrated in Section 4.4 that we can approximate the cost function to localize the updating of gains reasonably successfully, provided that the mesh grading is not too high. We also showed that we can also localize the updating of gains by just ignoring nonadjacent vertices and concluded that full updating of gains does not provide any significant advantages (and costs a lot more). We also described, in Section 4.5, how to use the bucket sorting of Fiduccia and Mattheyses (1982) for non-integer gains. Finally, in Section 5.1, we showed that partitions with good subdomain aspect ratios can vary greatly from those with a low edge cut.

To fully validate the method, it would be interesting to measure the correlation between the definition of aspect ratio used here and convergence in the solver and verify that it does indeed provide the benefits for DD preconditioners that other researchers, using different definitions of aspect ratio, suggest (e.g., Farhat, Maman, and Brown, 1995; Vanderstraeten et al., 1996). It would also be interesting to extend the ideas to investigate the shaping of subdomains to reflect anisotropic behavior. Finally, although a parallel version of JOSTLE exists (e.g., Walshaw, Cross, and Everett, 1997), it is not clear how well AR optimization, with its more global cost function, will work in parallel, and this is another direction for future research. Some related work on AR optimization already exists in the context of a parallel dynamic adaptive mesh environment (Diekmann, Meyer, and Monien, 1998; Diekmann, Schlimbach, and Walshaw, 1998; Schlimbach, 1998), but none of this work involves multilevel methods, so the question still arises whether parallel multilevel techniques can help in the optimization.

BIOGRAPHIES

Chris Walshaw is a Senior Research Fellow in the School of Computing and Mathematical Sciences at the University of Greenwich. He graduated from Bath University with a BSc in Mathematics and then moved to Edinburgh where he gained an MSc from Edinburgh University and a PhD from Heriot-Watt University, where his doctoral thesis concerned parallel algorithms for systems of differential equations. His postdoctoral work has extended this theme into parallel methods for adaptive unstructured meshes and, in particular, mesh partitioning. Since joining the University of Greenwich in 1993, he has developed the publically-available JOSTLE mesh partitioning software. He is the author of some 45 research papers.

Mark Cross is Professor of Numerical Modelling and Director of the Centre for Numerical Modelling and Process Analysis in the School of Computing and Mathematical Sciences at the University of Greenwich. The centre has about 100 staff and graduate students of which about 10 are associated with the Parallel Processing Group, whose work is focussed on the development of software tools to support the exploitation of such systems by computational modelling software. Professor Cross was educated at the University of Wales, Cardiff and received a PhD in 1972 for work on the modelling of semiconductor lasers. Since then he has worked in industry and academia in both the UK and USA and has been at Greenwich since 1982. His research interests cover computational modelling of metals/materials processes, computational mechanics algorithms, and software tools and the exploitation of HPC systems. The editor of the archival journal, *Applied Mathematical Modelling*, published by Elsevier, he is the author of some 200 research publications.

Ralf Diekmann received his Diploma in computer science and electrical engineering from the University of Paderborn, Germany in 1991. During his diploma, he developed parallel algorithms for combinatorial optimization problems, especially parallel Simulated Annealing algorithms. From 1991 to 1998, he worked as research assistant in the group of Burkhard Monien in Paderborn where his main research areas were algorithms for graph partitioning, distributed dynamic load balancing, and parallel combinatorial optimization. In these areas, he has published more than 30 papers in international conference proceedings and journals. In 1998, he received his Ph.D. in computer science for his work on load balancing strategies for data

parallel applications. Since 1998, he has been a research staff member of the corporate reasearch department of Hilti AG in Liechtenstein. His responsibilities include supercomuting, efficient algorithms and parallel algorithms for dynamic finite element simulations.

Frank Schlimbach is currently studying for a PhD in the School of Computing and Mathematical Sciences at the University of Greenwich. He graduated from the Department of Mathematics and Computer Science at the University of Paderborn with a Diploma in Computer Science. He has been developing the load balancing software for the PadFEM project, an object oriented environment for parallel adaptive finite element analysis. He is the author of around 10 research publications.

NOTE

1. Diekmann, Meyer, and Monien (1998) and Diekmann, Schlimbach, and Walshaw (1998) suggest the value of 1.40 using the square/cube-based definition of AR in Section 2.2—this is equivalent to 1.57 using the circle/sphere-based definition.

REFERENCES

Barnard, S. T., and H. D. Simon. 1994. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice & Experience* 6 (2): 101-17.

Blazy, S., W. Borchers, and U. Dralle. 1995. Parallelization methods for a characteristic's pressure correction scheme. In *Flow simulation with high performance computers II: Notes on numerical fluid mechanics*, edited by E. H. Hirschel. Braunschweig: Vieweg.

Bouhmala, N. 1998. Partitioning of unstructured meshes for parallel processing. Ph.D. dissertation, Inst. d'Informatique, Univ. Neuchatel.

Bramble, J. H., J. E. Pasciac, and A. H. Schatz. 1986. The construction of preconditioners for elliptic problems by substructuring I+II. *Math Comp* 47:103-34; 49:1-16.

Diekmann, R., B. Meyer, and B. Monien. 1998. Parallel decomposition of unstructured FEM-meshes. *Concurrency: Practice & Experience* 10 (1): 53-72.

Diekmann, R., F. Schlimbach, and C. Walshaw. 1998. Quality balancing for parallel adaptive FEM. In *Proceedings of Irregular '98: Solving irregularly structured problems in parallel*, vol. 1457 of LNCS, edited A. Ferreira et al., 170-81. Berlin: Springer.

Farhat, C., N. Maman, and G. Brown. 1995. Mesh partitioning for implicit computations via domain decomposition. *Int J Numer Methods Engng* 38:989-1000.

Farhat, C., J. Mandel, and F. X. Roux. 1994. Optimal convergence properties of the FETI domain decomposition method. *Comput Methods Appl Mech Engng* 115:365-85.

Fiduccia, C. M., and R. M. Mattheyses. 1982. A linear time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, 175-81. Piscataway, NJ: IEEE.

Gupta, A. 1996. Fast and effective algorithms for graph partitioning and sparse matrix reordering. *IBM Journal of Research and Development* 41 (1/2): 171-83.

Hendrickson, B., and R. Leland. 1993. A multilevel algorithm for partitioning graphs. Technical Report SAND 93-1301, Sandia National Labs, Albuquerque, NM.

Hendrickson, B., and R. Leland. 1995. A multilevel algorithm for partitioning graphs. In *Proceedings of Supercomputing '95*. New York: ACM Press.

Hu, Y. F., R. J. Blake, and D. R. Emerson. 1998. An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice & Experience* 10 (6): 467-83.

Karypis, G., and V. Kumar. 1995a. A fast and high quality multilevel scheme for partitioning irregular graphs. Report TR 95-035, Department of Computer Science, University of Minnesota, Minneapolis.

Karypis, G., and V. Kumar. 1995b. Multilevel *k*-way partitioning scheme for irregular graphs. Report TR 95-064, Department of Computer Science, University of Minnesota, Minneapolis.

Kernighan, B. W., and S. Lin. 1970. An efficient heuristic for partitioning graphs. *Bell Systems Tech J* 49:291-308.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220 (4598): 671-80.

Mitchell, S. A., and S. A. Vasavis. 1992. Quality mesh generation in three dimensions. In *Proceedings of ACM Conference on Computer Geometry*, 212-21. New York:ACM Press.

Schlimbach, F. 1998. Load balancing heuristics optimising subdomain aspect ratios for adaptive finite element simulations. Diploma thesis, Department of Mathematics and Computer Science, University of Paderborn.

Vanderstraeten, D., C. Farhat, P. S. Chen, R. Keunings, and O. Zone. 1996. A retrofit based methodology for the fast generation and optimization of large-scale mesh partitions: Beyond the minimum interface size criterion. *Comput Methods Appl Mech Engng* 133:25-45.

Vanderstraeten, D., R. Keunings, and C. Farhat. 1995. Beyond conventional mesh partitioning algorithms and the minimum edge cut criterion: Impact on realistic applications. In *Parallel processing for scientific computing*, edited by D. Bailey et al., 611-14. Philadelphia, PA: SIAM.

Walshaw, C., and M. Cross. 1998. Mesh partitioning: A multilevel balancing and refinement algorithm. Technical Report 98/IM/35, University of Greenwich, London.

Walshaw, C., M. Cross, R. Diekmann, and F. Schlimbach. 1998. Multilevel mesh partitioning for aspect ratio. In *Proceedings of VecPar'98, Porto, Portugal*, 381-94. Porto, Portugal: Universidade do Porto.

Walshaw, C., M. Cross, and M. Everett. 1997. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J Par Dist Comput* 47 (2): 102-8.