Multilevel Mesh Partitioning for Aspect Ratio

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

[1] School of Computing and Mathematical Sciences, The University of Greenwich, London, SE18 6PF, UK. {C.Walshaw, M.Cross}@gre.ac.uk

[2] Department of Computer Science, University of Paderborn, Fürstenallee 11, D-33102 Paderborn, Germany. {diek, schlimbo}@uni-paderborn.de

Abstract. Multilevel algorithms are a successful class of optimisation techniques which address the mesh partitioning problem. They usually combine a graph contraction algorithm together with a local optimisation method which refines the partition at each graph level. To date these algorithms have been used almost exclusively to minimise the cut-edge weight, however it has been shown that for certain classes of solution algorithm, the convergence of the solver is strongly influenced by the subdomain aspect ratio. In this paper therefore, we modify the multilevel algorithms in order to optimise a cost function based on aspect ratio. Several variants of the algorithms are tested and shown to provide excellent results.

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 behaviour via variable mesh densities. Meanwhile, the modelling of complex behaviour 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 maximised 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. [12].

A particularly popular and successful class of algorithms which address this mesh partitioning problem are 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 optimisation method which, starting with the coarsest graph, refines the partition at each graph level. These algorithms have been used almost exclusively to minimise the cut-edge weight, a cost which approximates the total communications volume in the underlying solver. This is an important goal in any parallel application, to minimise the communications overhead, however, it has been shown, [18], 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 paper therefore, we modify the multilevel algorithms (the matching and local optimisation) in order to optimise a cost function based on AR. We also abstract the process of modification in order to suggest how the multilevel strategy can be modified into a generic technique which can optimise arbitrary cost functions.

1.1 Domain decomposition preconditioners and aspect ratio

To motivate the need for aspect ratio 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. [2]. 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, [4]. 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 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. Whilst an algorithm such as the multigrid method as the solver on the subdomains is relatively robust against shape, the number of global iterations are heavily influenced by the AR of subdomains, [17]. Essentially, the subdomains can be viewed as elements of the interface problem, [7, 8], and just as with the normal finite element method, where 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.

1.2 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 optimisation 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 which take account of AR. In Section 4 we then describe a Kernighan-Lin (KL) type iterative local optimisation algorithm and describe two possible modifications which aim to optimise 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:

- In §2.2 we describe how the graph can be modified to take AR into account.
- In §3.2 we describe three matching algorithms based on AR.
- In §4.3 we describe two ways of using the cost function to optimise for AR.
- In §4.4 we describe how the bucket sort can be modified to take into account non-integer gains.

2 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 which represent the data dependencies in the mesh. 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 $\bigcup_P S_p = V$. To evenly balance the load, the optimal subdomain weight is given by $\overline{S} := \lceil |V|/P \rceil$ (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).

The definition of the mesh-partitioning problem is to find a partition which evenly balances the load or vertex weight in each subdomain whilst minimising 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 $S_p \leq \overline{S}$ and such that Γ is minimised.

2.1 The aspect ratio and cost function

We seek to modify the methods by optimising the partition on the basis of AR rather than cut-edge weight. In order to do this it is necessary to define a cost function which we seek to minimise 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 optimise (indeed it is for this reason that most mesh partitioning algorithms attempt to minimise the total cut-edge weight rather than the maximum between any two subdomains) and so instead we choose to minimise the average AR

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

There are several definitions of AR, however, and for example, for a given polygon S, a typical definition, [15], is the ratio of the largest circle which can be contained entirely within S (inscribed circle) to the smallest circle which entirely contains S (circumcircle). However these circles are not easy to calculate for arbitrary polygons and in an optimisation code where 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 [16] 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 3d) over that of some ideal shape with identical area (volume in 3d).

Suppose then that in 2d 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 optimisation 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 circle). These definitions of AR are easily extendible to 3d 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 $C = 1/(4\pi)^{1/3}3^{2/3}$ for the sphere. Note that henceforth, in order to talk in general terms for both 2d & 3d, given an object S we shall use the terms ∂S or *surface* for the surface area (3d) or perimeter length (2d) of the object and ΩS or *volume* for the volume (3d) or area (2d).

Of the above definitions of AR we choose to use the square/cube based formulae for two reasons; firstly because we are attempting to partition a mesh into interlocking subdomains (and circles/spheres are not known for their interlocking qualities) and secondly because it gives a convenient formula for the cost function of:

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

where $C = 2dP$ 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 for example §3.2) that even this function may be too complex for certain optimisation 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 §4.5). We can then approximate (2) by

$$\Gamma_{\text{template}} \approx \frac{1}{C'} \sum_p \partial S_p \tag{3}$$

where $C' = 2dP^{\frac{1}{d}}(\Omega M)^{\frac{d-1}{d}}$. 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 optimise:

$$\Gamma_{\text{surface}} = \frac{1}{K_1} \sum_p \partial^i S_p + K_2 \tag{4}$$

where $K_1 = 2dP^{\frac{1}{d}}(\Omega M)^{\frac{d-1}{d}}$ 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 optimising surfaces.

2.2 Modifying the graph

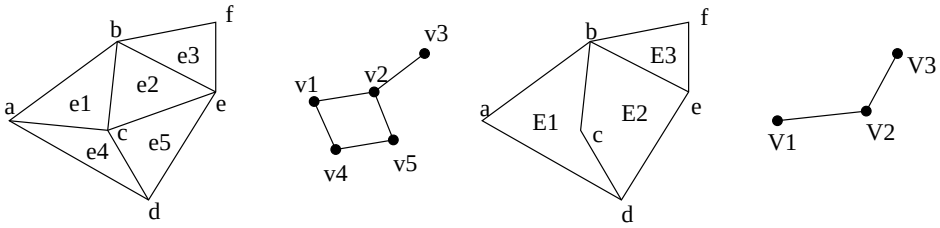

Fig. 1. Left to right: a simple mesh (a), its dual (b), the same mesh with combined elements (c) and its dual (d)

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 (to carry out load-balancing) 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 which 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, $\partial v_2 = \partial e_2 = D(b,c) + D(c,e) + D(e,b) = |(v_2, v_1)| + |(v_2, v_3)| + |(v_2, v_5)|$.

When it comes to combining elements together, either into subdomains, or for the multilevel matching (§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, e.g. $|(V_1, V_2)| = D(b,c) + D(c,d) = |(v_1, v_2)| + |(v_4, v_5)|$. The surface of a combined object S is the sum of the surfaces of its constituent parts less twice the interior surface, e.g. $\partial V_1 = \partial E_1 = \partial e_1 + \partial e_4 - 2 \times D(a,c) = \partial v_1 + \partial v_1 - 2|(v_1, v_4)|$. 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 which 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 optimise 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 §4.4).

2.3 Testing the algorithms

Table 1. Test meshes

mesh	no. vertices	no. edges	type	aspect ratio	mesh grading
uk	4824	6837	2d triangles	3.39	7.98e+02
t60k	60005	89440	2d triangles	1.60	2.00e+00
dime20	224843	336024	2d triangles	1.87	3.70e+03
cs4	22499	43858	3d tetrahedra	1.07	9.64e+01
mesh100	103081	200976	3d tetrahedra	1.63	2.45e+02
cyl3	232362	457853	3d tetrahedra	1.28	8.42e+00

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 6 test meshes but they have been chosen to be a representative sample of medium to large scale real-life problems and include both 2d and 3d 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 optimisation

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. Meanwhile, 'dime20' which has a moderate aspect ratio, has been very heavily refined in parts and thus has a high mesh grading – the largest element has a surface around 3,700 times larger than that of the smallest.

Table 2. Final results using template cost matching and surface gain/template cost optimisation

	$P = 16$			$P = 32$			$P = 64$			$P = 128$		
mesh	Γ_t	$\|E_c\|$	t_s	Γ_t	$\|E_c\|$	t_s	Γ_t	$\|E_c\|$	t_s	Γ_t	$\|E_c\|$	t_s
uk	1.48	206	0.12	1.31	331	0.12	1.23	543	0.22	1.25	917	0.50
t60k	1.16	1003	1.63	1.10	1547	2.07	1.11	2437	2.33	1.11	3647	2.65
dime20	1.22	1623	5.78	1.20	2868	5.17	1.15	4406	5.70	1.12	6620	7.57
cs4	1.22	2727	0.85	1.22	3738	0.90	1.23	5066	1.12	1.23	6747	1.60
mesh100	1.25	5950	3.20	1.24	8752	3.53	1.26	12467	4.13	1.28	17346	5.13
cyl3	1.21	11141	10.05	1.21	15944	10.77	1.23	22378	13.02	1.22	29719	13.18

Table 2 shows the results of the final combination of algorithms – TCM (see §3.2) and SGTC (see §4.3) – which were chosen as a benchmark for the other combinations. For the 4 different values of P (the number of subdomains), the table shows the average aspect ratio as given by Γ_t, the edge cut $|E_c|$ (that is 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, all partitions have average aspect ratios of less than 1.30 which is well within the target range suggested in [6]. Indeed for the 'uk' mesh it is no surprise that the results are not optimal because the subdomains inherit some of the poor AR from the original mesh (which has an AR of 3.39) and it is only when the mesh is split into small enough pieces, $P = 64$ or 128, that the optimisation succeeds in ameliorating this effect. Intuitively this also gives a hint as to why DD methods are a very successful technique as a solver.

3 The multilevel paradigm

In recent years it has been recognised that an effective way of both speeding up partition refinement and, perhaps more importantly 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 optimised on all the graphs starting with the coarsest and ending with the original. This sequence of contraction followed by repeated expansion/optimisation loops is known as the multilevel paradigm and has been successfully developed as a strategy for overcoming the localised nature of the KL (and other) optimisation algorithms. The multilevel idea was first proposed by Barnard & Simon, [1], as a method of speeding up spectral bisection and improved by Hendrickson & Leland, [11], who generalised it to encompass local refinement algorithms. Several algorithms for carrying out the matching have been devised by Karypis & Kumar, [13], while Walshaw & Cross describe a method for utilising imbalance in the coarsest graphs to enhance the final partition quality, [19].

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 & Leland,

[11]. 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, [10], 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 optimisation algorithm incorporates balancing, we commence on the expansion/optimisation sequence immediately.

Partition expansion. Having optimised 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 aspect ratio

The matching part of the multilevel strategy can be easily modified in several ways to take into account AR 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 neighbour using the chosen matching algorithm and it and its match removed from the list. Vertices with no unmatched neighbours remain unmatched and are also removed. In addition to **Random Matching (RM)**, [12], where vertices are matched with random neighbours, we propose and have tested 3 matching algorithms:

Surface Matching (SM). As we have seen in §2.2, 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 & Kumar, [13], where the vertex matches across the heaviest edge to any of its unmatched neighbours. 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, [3], and matches with the neighbour which minimises the cost function. In this case, the chosen vertex matches with the unmatched neighbour which gives the resulting element 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.

3.3 Results for different matching functions

In Tables 3, 4 & 5 we compare the results in Table 2, where TCM was used, with RM, SM & SCM respectively. In all cases the SGTC optimisation algorithm (see §4.3) was used. For each value of P, the first 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 $\frac{\Gamma(\text{RM})-1}{\Gamma(\text{TCM})-1}$ for RM, etc. Thus a figure > 1 means that RM has produced worse results than TCM. These comparisons are then averaged and so it can be seen, e.g. for $P = 16$ that RM produces results 24% (1.24) worse on average than TCM. Indeed the average quality of partitions produced by RM was 30% worse than TCM. This is not altogether surprising since the AR of elements in the coarsest graph could be very poor if the matching takes no account of it, and hence the optimisation has to work with badly shaped elements.

Table 3. Random matching results compared with template cost matching

	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
mesh	Γ_t	$\frac{\Gamma(\text{RM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{RM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{RM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{RM})-1}{\Gamma(\text{TCM})-1}$
uk	1.50	1.04	1.38	1.25	1.25	1.06	1.23	0.91
t60k	1.20	1.28	1.16	1.59	1.17	1.53	1.17	1.54
dime20	1.30	1.37	1.31	1.57	1.27	1.79	1.23	1.89
cs4	1.29	1.31	1.27	1.21	1.30	1.30	1.26	1.15
mesh100	1.31	1.24	1.29	1.24	1.31	1.19	1.32	1.15
cyl3	1.25	1.19	1.25	1.19	1.26	1.15	1.27	1.22
Average		1.24		1.34		1.34		1.31

When it comes to comparing TCM with SM & SCM (Tables 4 & 5) there is actually very little difference; SM is about 3.5% worse and SCM only about 1.5%. This suggests that the multilevel strategy is relatively robust to the matching algorithm *provided* the AR is taken into account in some way.

Table 4. Surface matching results compared with template cost matching

	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
mesh	Γ_t	$\frac{\Gamma(\text{SM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{SM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{SM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{SM})-1}{\Gamma(\text{TCM})-1}$
uk	1.54	1.13	1.34	1.11	1.24	1.01	1.28	1.10
t60k	1.14	0.87	1.11	1.05	1.12	1.10	1.12	1.08
dime20	1.26	1.18	1.24	1.23	1.15	1.00	1.13	1.04
cs4	1.22	0.97	1.24	1.08	1.24	1.04	1.23	1.00
mesh100	1.20	0.78	1.24	1.03	1.27	1.04	1.26	0.94
cyl3	1.19	0.93	1.21	1.02	1.24	1.05	1.24	1.08
Average		0.98		1.08		1.04		1.04

Table 5. Surface cost matching results compared with template cost matching

	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
mesh	Γ_t	$\frac{\Gamma(\text{SCM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{SCM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{SCM})-1}{\Gamma(\text{TCM})-1}$	Γ_t	$\frac{\Gamma(\text{SCM})-1}{\Gamma(\text{TCM})-1}$
uk	1.47	0.99	1.31	1.00	1.27	1.14	1.25	0.98
t60k	1.11	0.69	1.10	0.99	1.14	1.23	1.13	1.14
dime20	1.23	1.06	1.18	0.91	1.14	0.93	1.13	1.02
cs4	1.23	1.04	1.23	1.04	1.24	1.03	1.23	1.00
mesh100	1.23	0.91	1.25	1.07	1.25	0.99	1.27	0.97
cyl3	1.22	1.06	1.23	1.10	1.23	1.02	1.24	1.06
Average		0.96		1.02		1.05		1.03

We are not primarily concerned with partitioning times here, but for the record, RM was about 0.5% slower than TCM (although this is well within the limits of noise). This is because the optimisation stage took considerably longer (although the matching was

much faster than TCM). SM & SCM were 3.3% & 1.8% faster respectively than TCM. Overall this suggests that TCM is the algorithm of choice although there is little benefit over SM & SCM.

4 The Kernighan-Lin optimisation algorithm

In this section we discuss the key features of an optimisation algorithm, fully described in [19] and then in §4.3 describe how it can be modified to optimise 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 (§4.4), the linear time complexity improvement of Fiduccia & Mattheyses, [9], and is a partition optimisation formulation; in other words it optimises 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 migration of v only affects the cost of S_p and S_q (as is true for Γ_t and Γ_s) then we get

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

For Γ_t this gives an expression which cannot be further simplified, however, for Γ_s, since

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

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

$$g_{\text{surface}}(v,q) = \frac{2}{K_1}\left\{|(S_p, v)| - |(S_q, v)|\right\} \tag{6}$$

Notice in particular that g_{surface} is the same as the cut-edge weight gain function and that it is entirely localised, 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 which could be changed by non-local migration.

4.2 The iterative optimisation algorithm

The serial optimisation 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 optimisation uses two bucket sorting structures or bucket trees

(see below, §4.4) and is initialised 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 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 number of cut edges) 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.

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 optimal partition achieved by migration within that loop is maintained together with a list of vertices which 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.

The algorithm, together with conditions for vertex migration acceptance and confirmation is fully described in [19].

4.3 Incorporating aspect ratio: localisation

One of the advantages of using cut-edge weight as a cost function is its localised 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 optimise 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. As an alternative, therefore, we propose two localised variants:

Surface Gain/Surface Cost (SGSC). The simplest way to localise the updating of the gains is to make the assumption in §2.1 that the subdomains all have approximately equal volume and to use the surface cost function Γ_s from (4). As mentioned in §2.2 the problem immediately reduces to the cut-edge weight problem, albeit with non-integer edge weights, and from (6) 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 optimise the AR and below we provide some experimental results.

Surface Gain/Template Cost (SGTC). The second method we propose for localising the updates of gain relies on the observation that the gain is simply used as a method of rating the elements 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 & Kumar demonstrated that (at least when optimising for cut-edge weight) almost as good results can be achieved by simply visiting the vertices in random order, [14]. We therefore propose approximating the gain with the surface cost function Γ_s from (4) to rate the elements 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 an element. This localises the gain function.

4.4 Incorporating 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 & Mattheyses in [9] and the idea is that all vertices of a given gain g are placed together in a 'bucket' which is ranked g. Finding a vertex with maximum gain then simply consists of finding the (non-empty) 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 which are candidates for migration resorted by gain. Using a bucket sort for this operation simply requires recalculating the gains and transferring the affected vertices to the appropriate buckets. If a bucket sort were not used and, say, 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 [19]. It includes a ranking for prioritising vertices for migration which incorporates their weight as well as their gain. The non-empty buckets are stored in a binary-tree to save excessive memory 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 optimisation is that with non-integer edge weight, 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, i.e. the bucket ranked 1 could contain any vertex with gain in the interval $[1.0, 2.0)$. However, if using the surface gain function, the issue of scaling then arises since 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, if using Γ_s as a gain function, as in SGSC and SGTC, we can easily calculate the maximum possible gain. This would occur if the vertex with the largest surface, $v \in S_p$ say, were entirely surrounded by neighbours 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 and scale the gain accordingly. 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. However the experiments carried out here all used a scaling which allowed a maximum of 100 buckets and we have tested the algorithm with up to 10,000 buckets without significant penalty in terms either memory or run-time.

4.5 Results for different optimisation functions

Table 6 compares SGSC against the SGTC results in Table 2. Both set of results use template cost matching (TCM). The table is in the same form as those in §3.3 and shows that there is on average only a tiny difference between the two (SGTC is 0.5% better than SGSC) and again, with the exception of the 'uk' mesh for $P = 16$ & 32, all results have an average AR of less than 1.30. This implication of this table is that the assumption made in §2.1, that all subdomains have approximately the same volume, is reasonably good. However this assumption is not necessarily true, because for example, for $P = 128$, the 'dime20' mesh, with its high grading, has a ratio of $\max \Omega S_p / \min \Omega S_p =$

2723. A possible explanation is that although the assumption is false globally, it is true locally, since the mesh density does not change too gradually (as should be the case with most meshes generated by adaptive refinement) and so the volume of each subdomain is approximately equal to that of its neighbours.

Table 6. Surface gain/surface cost optimisation compared with surface gain/template cost

	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
mesh	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{SGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{SGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{SGTC})-1}$	Γ_t	$\frac{\Gamma(\text{SGSC})-1}{\Gamma(\text{SGTC})-1}$
uk	1.49	1.02	1.32	1.05	1.24	1.02	1.23	0.92
t60k	1.15	0.95	1.10	0.96	1.12	1.07	1.12	1.11
dime20	1.23	1.03	1.17	0.86	1.15	0.98	1.11	0.91
cs4	1.20	0.90	1.23	1.05	1.24	1.03	1.22	0.97
mesh100	1.24	0.95	1.26	1.10	1.27	1.06	1.27	0.97
cyl3	1.23	1.10	1.22	1.08	1.24	1.06	1.22	1.00
Average		0.99		1.01		1.04		0.98

Again we are not not primarily concerned with partitioning times, but it was surprising to see that SGSC was an average 30% slower than SGTC. A possible explanation is that although the cost function Γ_s is a good approximation, Γ_t is a more global function and so the optimisation converges more quickly.

5 Discussion

5.1 Comparison with cut-edge weight partitioning

In Table 7 we compare AR as produced by the edge cut partitioner (EC) described in [19] with the results in Table 2. On average AR partitioning produces results which are 16% better than those of the edge cut partitioner (as could be expected). However, for the mesh 'cs4' EC partitioning is consistently better and this is a subject for further investigation.

Table 7. AR results for the edge cut partitioner compared with the AR partitioner

	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
mesh	Γ_t	$\frac{\Gamma(\text{EC})-1}{\Gamma(\text{AR})-1}$	Γ_t	$\frac{\Gamma(\text{EC})-1}{\Gamma(\text{AR})-1}$	Γ_t	$\frac{\Gamma(\text{EC})-1}{\Gamma(\text{AR})-1}$	Γ_t	$\frac{\Gamma(\text{EC})-1}{\Gamma(\text{AR})-1}$
uk	1.52	1.09	1.33	1.07	1.26	1.09	1.28	1.14
t60k	1.19	1.18	1.18	1.76	1.17	1.47	1.17	1.55
dime20	1.32	1.45	1.26	1.34	1.25	1.65	1.21	1.72
cs4	1.19	0.86	1.21	0.93	1.20	0.87	1.21	0.92
mesh100	1.22	0.89	1.22	0.91	1.26	1.03	1.24	0.86
cyl3	1.22	1.05	1.23	1.09	1.23	1.00	1.23	1.02
Average		1.09		1.18		1.19		1.20

Meanwhile in Table 8 we compare the edge cut produced by the EC partitioner with that of the AR partitioner. Again as expected, EC partitioning produces the best results (about 11% better than AR). In terms of time, the EC partitioner is about 26% 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.

Table 8. $|E_c|$ results for the edge cut partitioner compared with the AR partitioner

	$P = 16$		$P = 32$		$P = 64$		$P = 128$	
mesh	$\|E_c\|$	$\frac{\|E_c\|(\text{RM})}{\|E_c\|(\text{AR})}$	$\|E_c\|$	$\frac{\|E_c\|(\text{RM})}{\|E_c\|(\text{AR})}$	$\|E_c\|$	$\frac{\|E_c\|(\text{RM})}{\|E_c\|(\text{AR})}$	$\|E_c\|$	$\frac{\|E_c\|(\text{RM})}{\|E_c\|(\text{AR})}$
uk	189	0.92	290	0.88	478	0.88	845	0.92
t60k	974	0.97	1588	1.03	2440	1.00	3646	1.00
dime20	1326	0.82	2294	0.80	3637	0.83	5497	0.83
cs4	2343	0.86	3351	0.90	4534	0.89	6101	0.90
mesh100	4577	0.77	7109	0.81	10740	0.86	14313	0.83
cyl3	10458	0.94	14986	0.94	20765	0.93	27869	0.94
Average		0.88		0.89		0.90		0.90

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 optimisation 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 minimise the cost of the matchings. Note, however, that this may not be possible as a cost function which, say, measured the cost of a mapping onto a particular processor topology would be unable to function since at the matching stage no partition, and hence no mapping exists.
- The optimisation relies, for efficiency at least, on having a local gain function in order that the migration of a vertex does not involve an $O(N)$ update. Here we were able to localise the cost function by making a simple approximation to give a local gain function, however, it is not clear that this is always possible.
- The bucket sort is reasonably simple to convert to non-integer gains, however this relies on being able to estimate the maximum gain. If this is not possible it may not be easy to generate a good scaling which 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 optimise for aspect ratio. To fully validate the method, however, we need to demonstrate that the measure of aspect ratio used here does indeed provide the benefits for DD preconditioners that the theoretical results suggest. It is also desirable to measure the correlation between aspect ratio and convergence in the solver.

Also, although parallel implementations of the multilevel strategy do exist, e.g. [20], it is not clear how well AR optimisation, with its more global cost function, will work in parallel and this is another direction for future research. Some related work already exists in the context of a parallel dynamic adaptive mesh environment, [5, 6, 16], but these are not multilevel methods and it was necessary to use a combination of several complex cost functions in order to achieve reasonable results so the question arises whether multilevel techniques can help to overcome this.

References

1. S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. *Concurrency: Practice & Experience*, 6(2):101–117, 1994.
2. S. Blazy, W. Borchers, and U. Dralle. Parallelization methods for a characteristic's pressure correction scheme. In E. H. Hirschel, editor, *Flow Simulation with High Performance Computers II, Notes on Numerical Fluid Mechanics*, 1995.
3. N. Bouhmala. *Partitioning of Unstructured Meshes for Parallel Processing*. PhD thesis, Inst. d'Informatique, Univ. Neuchatel, 1998.
4. J. H. Bramble, J. E. Pasciac, and A. H. Schatz. The Construction of Preconditioners for Elliptic Problems by Substructuring I+II. *Math. Comp.*, 47+49, 1986+87.
5. R. Diekmann, B. Meyer, and B. Monien. Parallel Decomposition of Unstructured FEM-Meshes. *Concurrency: Practice & Experience*, 10(1):53–72, 1998.
6. R. Diekmann, F. Schlimbach, and C. Walshaw. Quality Balancing for Parallel Adaptive FEM. To appear in Proc. Irregular '98.
7. C. Farhat, N. Maman, and G. Brown. Mesh Partitioning for Implicit Computations via Domain Decomposition. *Int. J. Num. Meth. Engng.*, 38:989–1000, 1995.
8. C. Farhat, J. Mandel, and F. X. Roux. Optimal convergence properties of the FETI domain decomposition method. *Comp. Meth. Appl. Mech. Engrg.*, 115:367–388, 1994.
9. C. M. Fiduccia and R. M. Mattheyses. A Linear Time Heuristic for Improving Network Partitions. In *Proc. 19th IEEE Design Automation Conf.*, pages 175–181, IEEE, Piscataway, NJ, 1982.
10. A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix reordering. *IBM Journal of Research and Development*, 41(1/2):171–183, 1996.
11. B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. Tech. Rep. SAND 93-1301, Sandia National Labs, Albuquerque, NM, 1993.
12. B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. In *Proc. Supercomputing '95*, 1995.
13. G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. TR 95-035, Dept. Comp. Sci., Univ. Minnesota, Minneapolis, MN 55455, 1995.
14. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. TR 95-064, Dept. Comp. Sci., Univ. Minnesota, Minneapolis, MN 55455, 1995.
15. S. A. Mitchell and S. A. Vasavis. Quality Mesh Generation in Three Dimensions. In *Proc. ACM Conf. Comp Geometry*, pages 212–221, 1992.
16. F. Schlimbach. *Load Balancing Heuristics Optimising Subdomain Shapes for Adaptive Finite Element Simulations*. Diploma Thesis, Dept. Math. Comp. Sci., Univ. Paderborn, 1998.
17. D. Vanderstraeten, C. Farhat, P. S. Chen, R. Keunings, and O. Zone. A Retrofit Based Methodology for the Fast Generation and Optimization of Large-Scale Mesh Partitions: Beyond the Minimum Interface Size Criterion. *Comp. Meth. Appl. Mech. Engrg.*, 133:25–45, 1996.
18. D. Vanderstraeten, R. Keunings, and C. Farhat. Beyond Conventional Mesh Partitioning Algorithms and the Minimum Edge Cut Criterion: Impact on Realistic Applications. In D. Bailey *et al*, editor, *Parallel Processing for Scientific Computing*, pages 611–614. SIAM, 1995.
19. C. Walshaw and M. Cross. Mesh Partitioning: a Multilevel Balancing and Refinement Algorithm. Tech. Rep. 98/IM/35, Univ. Greenwich, London SE18 6PF, UK, March 1998.
20. C. Walshaw, M. Cross, and M. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *J. Par. Dist. Comput.*, 47(2):102–108, 1997.