Enhanced Dynamic Load-Balancing Of Adaptive Unstructured Meshes

Chris Walshaw* Martin Berzins*

Abstract

Modern PDE solvers increasingly use adaptive unstructured meshes in order to discretise complex geometries and control numerical error. The problem of dividing up the domain equally for a distributed memory parallel computer whilst minimising the inter-subdomain dependencies can be tackled with graph-based methods such as Recursive Spectral Bisection. This paper describes an extension to such methods which renders them more suitable for time-dependent problems where frequent remeshing may occur, possibly with only relatively small changes to the mesh. Numerical testing shows that this new approach gives a good speed-up for mesh partitioning when used to enhance Recursive Spectral Bisection. It is also shown that the overall computational time can be less than that needed when cheaper but more naive load-balancing algorithms are used.

1 Introduction

PDE solvers for time-dependent applications are currently being written to obtain accurate solutions to real life problems with the solution process as automatic as possible. The use of an unstructured mesh allows the code to cater for completely general geometries and hence a wide range of problems in both two and three space dimensions. In addition, such software may employ adaptive methods in space and/or time in order to to control the numerical error. The desire to control the spatial error in time-varying solutions means that the position and density of the spatial mesh points may vary dramatically over the course of an integration.

Parallel versions of such codes face the problem of distributing the mesh. For optimal performance the load should be evenly balanced and the communication cost reduced as much as possible by minimising interprocessor dependencies. It is well known that this mapping problem is NP hard, [3], and so heuristics must be employed to obtain a usable algorithm. In addition, for time-dependent problems, the unstructured mesh may be modified every few time-steps and so the load-balancing must have a low cost relative to that of the solution algorithm in between remeshing.

A number of good load-balancing algorithms (see for example [6] and [9]) are based on partitioning a graph that corresponds to the communication requirements of the spatial discretisation routine on an unstructured mesh. Of these techniques Recursive Spectral Bisection (RSB) is generally highly regarded and an improved version allowing for quadrisection and even octasection has recently been devised, [3]. The spectral algorithm

*School of Computer Studies, University of Leeds, Leeds, England. E-mails: chris@scs.leeds.ac.uk and martin@scs.leeds.ac.uk

forms a natural starting point for the work presented here and in Section 2.2 we give a brief overview of the method.

Until now, however, such algorithms have not addressed adaptive mesh codes and the resulting incremental update partitioning problem posed when a mesh with an existing partition is being refined and/or coarsened. The failure to utilise this existing partition suggests that the load-balancing may be unnecessarily computationally expensive. In [7] an iterative preprocessing technique for the update problem was proposed *to enhance* existing graph-based partitioning methods and in Section 3 we give a summary of this new algorithm.

This repartitioning algorithm, henceforth referred to as Dynamic Recursive Spectral Bisection (DRSB), has now been integrated into an adaptive parallel PDE solver and observations on the success of this exercise and some further results are given in Sections 4 & 5 respectively. As a comparison with other partitioning techniques we also include results of partitions supplied by both RSB and by Recursive Coordinate Bisection (RCB).

2 Recursive Bisection Methods

The Recursive Bisection approach is based on the principal that bisecting a domain is a much easier task than subdividing into p subdomains. The bisection is obtained by a given strategy and then the same strategy is applied to the subdomains recursively. In this manner a partition into $p = 2^q$ subdomains can be obtained in q recursive steps. Horst Simon, [6], describes three different bisection strategies – Recursive Coordinate Bisection (RCB), Graph Bisection (RGB) & Spectral Bisection (RSB) – and demonstrates the superiority of RSB over the other two.

It should be noted here that Recursive Bisection algorithms can create any number of partitions (not necessarily a power of 2). For example, to create 3 partitions, the bisection field (the Fiedler vector in the case of RSB or the x/y coordinates for RCB – see below) can either be divided directly into 3, so called strip-wise partitioning, or bisected with a 1:2 split and then bisection applied again to the larger partition.

2.1 Recursive Coordinate Bisection

RCB is a simple and intuitive technique which bisects the mesh by sorting the elements using alternately x and y coordinates. Some results are provided as a computationally inexpensive alternative to DRSB. The method however provides poor separator sets due to excluding communication information and this is particularly noticeable on very irregular meshes where the subdomains are likely to be disconnected, [6].

2.2 Recursive Spectral Bisection – An Overview

Spectral partitioning is one of a number of methods, see [6], which partition a graph derived from the mesh. The fundamental idea is to associate each mesh element with a nodes of an undirected graph. The **dual communication graph**, $G(V, E)$ where V is the set of nodes and E the set of edges, is then defined by connecting nodes together (with an edge) when the spatial discretisation gives rise to a dependency between the corresponding mesh elements. In the example employed here, a cell-centred 2D finite volume scheme, a node of the graph is used to represent a triangle and then each node will have three edges connecting it to the triangles it is adjacent to and may in addition have edges to those triangles it shares a corner with. Figure 1 (page 974) shows a simple example of a mesh and its dual communication graph.

The dual graph can now be represented by an $n \times n$ symmetric matrix L, known as

the Laplacian, where n is the number of graph nodes. The diagonal of L gives the degree of each graph node whilst the off-diagonals are -1 at (i,j) where an edge connects nodes i and j and zero elsewhere. This positive semi-definite matrix has a number of interesting properties (see [6]). In particular, if $\mathbf{x}$ is a vector which describes the bisection of a given graph, i.e. $x_i = 1$ if node i is in subset A or -1 if in B, then the communication cost or number of interprocessor edges of this bisection is given by

$$C(\mathbf{x}) \stackrel{def}{=} \frac{1}{4} \sum_{(v,w)\epsilon E} (x_v - x_w)^2 \equiv \frac{1}{4}\mathbf{x}^t L \mathbf{x}.$$

The bisection problem can now by formulated as:–

$$\text{Mininise } \mathbf{x}^t L \mathbf{x} \text{ subject to } \sum_1^n \mathbf{x} = 0.$$

Although this discrete problem is NP hard, it may be approximated by a continuous eigenvector problem if the constraint of discrete x values is relaxed to allow continuous values. The Laplacian as defined always has a zero eigenvalue λ_1, [5], which corresponds to the eigenvector $\mathbf{e}$ (where $e_i = 1 \; \forall \; i$) and places all the nodes in one subset. Choosing an eigenvector $\mathbf{x}$ orthogonal to $\mathbf{e}$ guarantees that the load-balancing constraint is satisfied, since $< \mathbf{e}, \mathbf{x} >= \sum_1^n \mathbf{x}$. Thus we select the smallest positive eigenvalue λ_2 and its corresponding eigenvector $\mathbf{x}_2$, known as the Fiedler vector from the original study of its properties, [2]. This eigenvector gives a weighting for each node which, when used to sort them, usually gives a near optimal bisection.

2.3 Applications to Solution-Adaptive Problems

Whilst Spectral Bisection usually gives good results for a static problem, it may not be so suitable for the dynamic partitioning of adaptive meshes. In particular the method is computationally expensive; the cost of finding the eigenvector for a problem size n, being at least $O(n \log n)$. While this may not be a great drawback for a static mesh where the cost can be hidden as a start-up overhead, it may be significant when a time-stepping code is remeshing frequently.

In addition the method tends to be sensitive to small perturbations in the mesh. For instance Williams, [9] page 477, states that 'a small change in mesh refinement may lead to a large change in the second eigenvector.' Combined with the fact that the RSB algorithm has no mechanism for using existing information about the previous partition, heavy node migration may result.

In the next Section an incremental method is presented that enables a graph-based algorithm to use existing information about the partition of a previous mesh. It is described with particular reference to the Recursive Spectral Bisection algorithm but could used to enhance the performance of any graph-based method.

3 A Dynamic Partitioning Approach

When a partitioned mesh is modified by the addition of new elements or the removal of existing ones an immediate load-imbalance (and hence a new partitioning problem) is created. Provided that the new mesh is based on coarsening or refining of the existing one, as in [1], it is possible to interpolate the existing partition onto the new mesh and to use this partition as a starting point in a *repartitioning* algorithm.

This technique makes the assumption that, unless the mesh has changed dramatically, the partitions will not need to be changed a great deal. Ideally most mesh elements will remain in the same subdomain whilst just the boundaries are balanced. In particular if mesh elements 'close to' the interprocessor boundaries are the only ones partitioned then the information from the previous partition is utilised and, as a result, both the cost and the amount of node migration should certainly be reduced (the factors being largely dependent on the granularity). Of course it is not clear that such balancing will produce optimal communication costs but the results (Section 5 & [7]) actually seem to show an improvement over those of the RSB algorithm in most cases.

A full description of the Dynamic RSB algorithm (DRSB) together with a discussion of some implementation issues is given in [7]; here we provide a brief summary. In order to chose the subsets of nodes which are to be repartitioned, **level sets** of the graph are defined. Each level set, L_q, is specified by selecting nodes within q edges of the old interprocessor boundary (which has been interpolated onto the new mesh). A suitable q (for example $q = 2$ or 3) is chosen to give a meaningful subgraph and then all connected groups of nodes outside L_q are formed into clusters. A reduced graph is now given by representing each of these clusters by a single node, weighted according to the number of nodes that it represents.

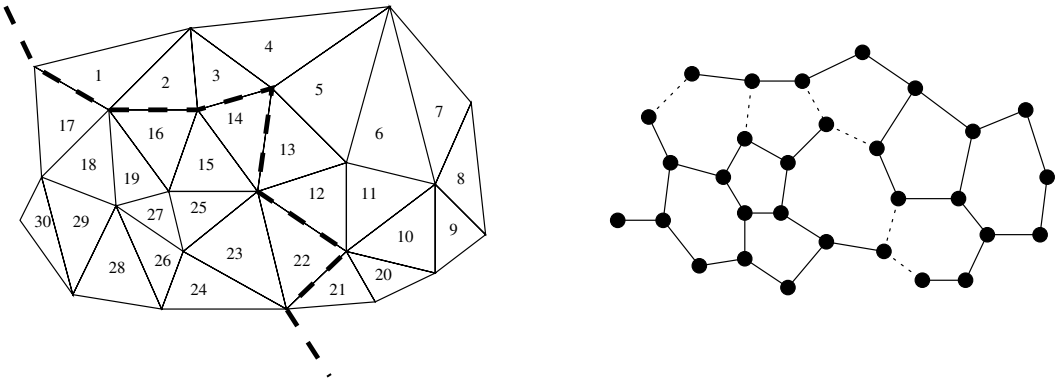

FIG. 1. *A simple mesh and the corresponding dual communication graph*

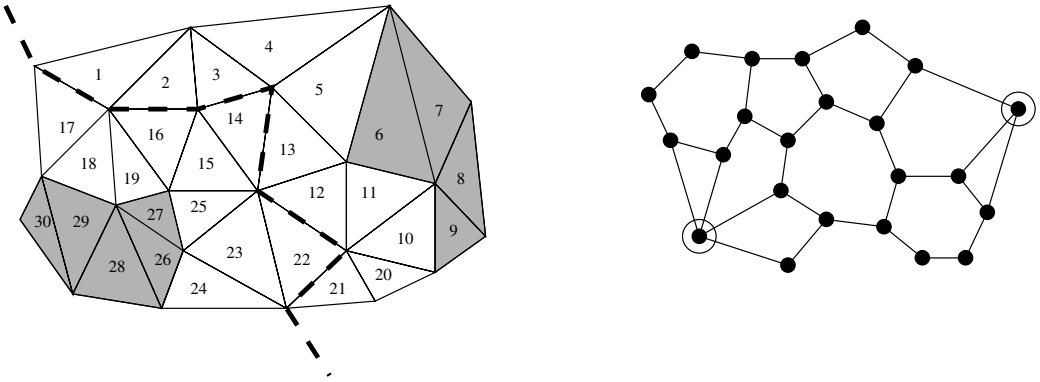

FIG. 2. *Clustering of mesh elements and the corresponding reduced graph*

Figures 1 & 2 show this clustering technique with $q = 2$ on a simple mesh. Figure 1

shows the mesh (left) immediately after refinement with the old (and now non optimal) partition interpolated onto the mesh – the heavy dashed line. The dual graph (right) shows the interprocessor edges as dotted lines. Figure 2 shows the mesh elements which have been selected for clustering (shaded) and on the right the reduced dual graph with each cluster represented by a single node (ringed). On this small mesh the cost savings will not be large, however for partitions of much greater granularity the reduced graph will constitute a considerably smaller proportion of the full graph.

Spectral bisection (or another graph-based technique) is now applied to the reduced graph and the graph partitioned as before – the only difference is that, when sorting the Fiedler vector, clusters are counted with the multiplicity of the number of nodes contained in them. As a worst case this sorting procedure may fail to produce an evenly balanced load (if the bisection falls in the middle of a cluster) and in this event the reduced graph is expanded by one level set and bisection applied again. Assuming the graph is connected, this gives an iterative technique which converges to (or more properly terminates with) the full graph. Efficiency, however, may dictate that the iterations terminate early.

The size reduction in the graph can afford considerable cost savings and, for the redistribution, it is expected that the clustered nodes will remain in the same partition (a node migration saving). Initial tests of the dynamic technique in isolation have shown that up to an order of magnitude speed-up may be found when the algorithm is used to enhance RSB, [7].

4 Integration into a Parallel Adaptive PDE Solver

The new mesh partitioning algorithm has been applied to PDE examples taken from compressible flow problems with moving shock type features. As is usual for such problems explicit method-of-lines time-integration is employed with spatial error control, [1] & [8], and this involves only ODE function evaluations and vector operations. For a partitioned data structure these operations may be implemented easily in parallel using distributed versions of standard BLAS routines. The bulk of the work (80–90%) therefore lies in the ODE residual routine which has thus been the initial target for parallelisation.

The discretisation used derives from a cell-centred finite volume scheme on a mesh of unstructed triangles. The residuals are calculated by evaluating the flux across each of the triangle edges and summing these figures for each element of the solution vector based at that triangle's centroid. As a result flux calculations are duplicated on edges lying between triangles on different processors. An alternative might have been to partition the edges as well as the triangles but in a fully parallel solver this would have called for an extra communication phase to flush the residuals around the system. This part of the code involves the distribution of a rich C data structure which must be partitioned into 'tiles' – conceptually spatially coherent subsets of the mesh and solution data which each reside on a processor. Each tile has a core (data assigned to that processor) selected by the partitioning algorithm plus a halo (copies of data assigned to other processors) defined by the discretisation stencil.

A key question in the design of a mesh partitioning algorithm is the trade-off between the cost of the algorithm and the improvement in the performance of the solver it is designed for. The experiments reported here concentrate on the overhead of different partitions and the effect on the number of replicated edge calculations (with communication costs ignored). In order to investigate this issue a prototype code was constructed in which the remeshing and partitioning is carried out by the host and the tiles transmitted to their

home processor. For each residual evaluation the solution vector is distributed by the host – each tile receiving its core variables plus any required halo data. The residual is calculated in parallel (with the amount of time spent here recorded) and transmitted back to the host for time-integration.

5 Numerical Experiments

In order to discover whether a less than optimal partition would significantly change the efficiency of residual evaluation a wedge shock example was tested with the results reported in Table 1. The computation was performed on a Meiko computing surface consisting of a SPARC 2 front end and 16 T800 transputers. For this particular example the remeshing was fairly infrequent with 2927 residual evaluations (563 successful time-steps) and only 19 remeshes. The meshes were also fairly small – averaging 867 elements over all the residual evaluations. Both these factors tend to favour the sophisticated partitioning algorithms and indeed any test of this kind is very problem dependent. However in mitigation no communications were included and there was no parallel linear algebra both of which would show poor performance with non optimal partitions.

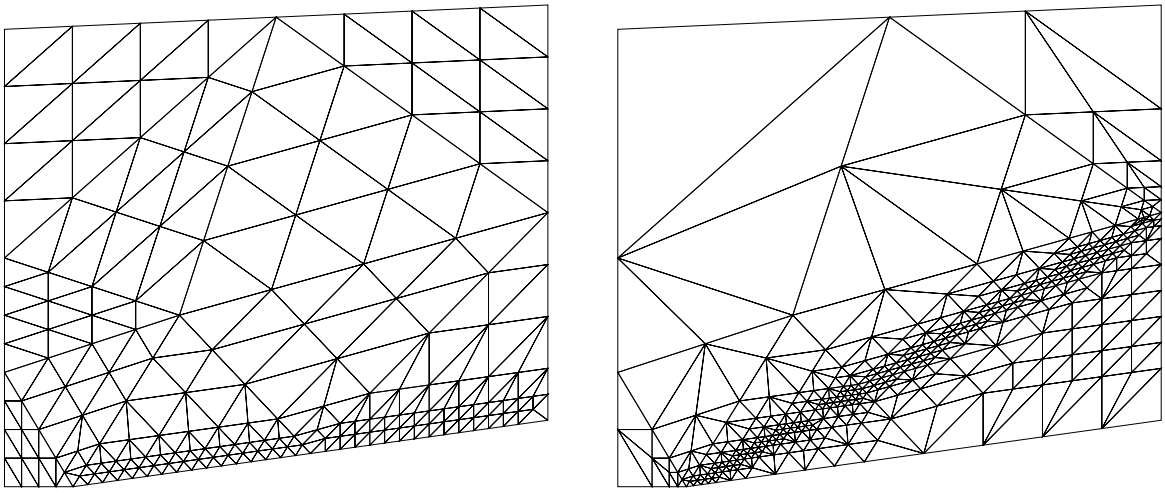

FIG. 3. *The wedge shock solution domain after the first and final remeshes*

5.1 The Wedge Shock Example

This problem integrates the Euler equations in two space dimensions (see [8] for a full description) with an inflow of air at Mach 2.5 hitting a 10 degree wedge and forming a shock front. In its original form this is a steady state problem, however, in the time-dependent form used here the shock starts off along the wedge and rises to its steady-state position as time proceeds. The unstructured mesh becomes heavily refined around the front as it forms but remains coarse away from the wedge and shock. Figure 3 shows the solution domains after the first and final remeshes.

5.2 Comparison Results

The following metrics are used in Table 1 to compare the load-balancing algorithms:

Mflops: The chief expense of the spectral algorithm lies in the eigenvector calculations of the Fiedler vector and the number of floating point operations in this part of the code were totalled to give a measure of the savings afforded by using the dynamic technique.

TABLE 1
Load Balancing Comparison Results

	DRSB				RSB				RCB		
n	Mflops	$\|E_i\|$	T_b	T_r	Mflops	$\|E_i\|$	T_b	T_r	$\|E_i\|$	T_b	T_r
324	0.9	69	2.6	8.0	1.6	73	3.5	7.7	107	0.3	9.9
299	1.4	68	3.6	15.4	1.5	67	3.1	15.7	113	0.2	17.8
556	1.0	84	3.2	27.9	3.8	82	7.0	28.3	157	0.4	31.8
743	2.5	89	7.1	45.3	5.9	88	10.8	44.8	187	0.6	48.5
764	0.9	86	3.5	126.6	6.1	90	11.3	125.6	192	0.6	146.0
492	0.6	88	2.3	5.5	3.1	86	5.9	5.8	148	0.4	6.5
674	2.7	105	7.1	13.4	4.8	107	9.0	12.9	172	0.5	15.0
682	1.4	104	4.4	31.3	4.9	109	9.3	30.7	176	0.5	34.2
865	4.4	118	10.5	68.9	6.8	130	12.4	72.2	208	0.7	77.3
831	2.0	115	5.8	41.3	6.3	132	11.5	44.2	203	0.7	48.9
1149	5.8	128	13.9	75.1	10.2	144	18.5	75.8	264	0.9	88.1
1064	1.8	125	5.9	27.4	9.0	135	16.3	27.8	270	0.8	32.9
1102	1.1	136	5.0	91.0	9.7	140	17.6	92.8	272	0.9	104.7
1150	1.0	140	4.7	49.7	10.1	151	18.2	51.8	276	0.9	57.0
811	2.6	127	7.9	8.2	5.9	134	11.2	8.0	202	0.6	9.2
851	5.3	124	11.1	38.5	6.3	133	11.5	39.1	220	0.7	44.3
1024	3.6	134	9.2	86.7	8.1	139	14.7	88.6	234	0.8	94.9
1107	1.2	146	4.8	139.1	9.0	151	16.2	146.1	249	0.9	159.8
963	2.4	136	7.1	9.6	7.3	140	13.4	9.8	241	0.8	10.6
15451	42.6	2122	119.9	908.9	120.4	2231	221.4	927.7	3891	12.1	1037.4

$|E_i|$ – **the interprocessor edges**: If E_i is defined to be the subset of edges which cross interprocessor boundaries after repartitioning then another metric is simply the size of this set. This measure gives an indication of the number of duplicated flux calculations and the interprocessor communication.

T_b – **load-balancing time**: This figure represents the rough cost in seconds of finding the mesh partitions in serial on a SPARC 2. It serves only to demonstrate the relative costs of the different load-balancing algorithms and it is felt that the Mflops count gives a better asymptotic measure of cost.

T_r – **residual evaluation time**: These figures give the time in seconds on 16 T800 transputers for all of the parallel residual evaluations taken at this mesh resolution. No communication overheads are added in but for each residual the figure chosen was that of the slowest processor. These figures are not directly comparable to T_b as the SPARC is about 4 times faster than a T800.

The table shows that the new mesh partitioning algorithm can involve up to an order of magnitude fewer Mflops than RSB but is still expensive compared to RCB. In looking at the costs of residual evaluation, however, it is clear that the new algorithm results in a more efficient calculation than RCB due to less replication of edge calculations (and significantly less communication).

Benchmarking computations on a serial machine show that the load-balancing accounts for less than 2% of the cost of the complete integration when the new algorithm is used. This percentage drops to 0.2% when RCB is used. However the computational saving

when DRSB is used to partition the mesh shows a 14% saving over RCB for the parallel residual evaluations. These results suggest that in a fully parallel system, assuming that load-balancing remains at the same proportion of the overall cost (note that RSB can be parallelised with good efficiency, [4]), the DRSB code will be more efficient than the RCB code. Furthermore the communications costs for RCB are almost double those of DRSB. In addition, if parallel linear algebra were employed, the poor quality of the separator sets might seriously degrade the efficiency of a naive partition.

These investigations thus suggest that the new algorithm has promise for providing an efficient load-balancing approach for the adaptive solution of time-dependent PDEs. Further work is in progress to validate this claim.

Acknowledgements. The authors would like to acknowledge the financial support of Shell Research Limited. Peter Jimack and David Hodgson are also thanked for their helpful discussions and Justin Ware for providing the example meshes.

References

[1] M. Berzins, J. Lawson, and J. Ware, *Spatial and Temporal Error Control in the Adaptive Solution of Systems of Conversation Laws.* To Appear In: *Proc. of 7th IMACS Conf. on Computer Methods for PDEs, Rutgers Univ.*, 1992.

[2] M. Fiedler, *A Property Of Eigenvectors of Nonnegative Symmetric Matrices and its Applications to Graph Theory*, Czech. Math. J., 25 (1975), pp. 619–633.

[3] B. Hendrickson and R. Leland, *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations.* Tech. Rep. SAND 92-1460, Sandia National Labs, Albuquerque, NM., 1992.

[4] Z. Johan, *Data Parallel Finite Element Techniques for Large-scale Computational Fluid Dynamics.* PhD. Thesis, Stanford University, 1992.

[5] A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning Sparse Matrices with Eigenvectors of Graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.

[6] H. D. Simon, *Partitioning of Unstructured Problems for Parallel Processing*, Computing Systems in Engineering, 2 (1991), pp. 135–148.

[7] C. H. Walshaw and M. Berzins, *Dynamic Load-Balancing For PDE Solvers On Adaptive Unstructured Meshes.* School of Computer Studies Rep. No. 92.32 (submitted for publication), 1992.

[8] J. Ware and M. Berzins, *Finite Volume Techniques for Time-Dependent Fluid-Flow Problems.* To Appear In: *Proc. of 7th IMACS Conf. on Computer Methods for PDEs, Rutgers Univ.*, 1992.

[9] R. D. Williams, *Performance of dynamic load balancing algorithms for unstructed mesh calculations*, Concurrency, 3 (1991), pp. 457–481.