Dynamic Load-Balancing For PDE Solvers Using Adaptive Unstructured Meshes

Chris Walshaw and Martin Berzins
School of Computer Studies
University of Leeds

Abstract

Modern PDE solvers written for time-dependent problems increasingly employ adaptive unstructured meshes (see Flaherty et al. [4]) in order to both increase efficiency and control the numerical error. If a distributed memory parallel computer is to be used, there arises the significant problem of dividing up the domain equally amongst the processors whilst minimising the inter-subdomain dependencies. A number of graph based algorithms have recently been proposed for steady state calculations, for example [6] & [10]. This paper considers an extension to such methods which renders them more suitable for time-dependent problems in which the mesh may be changed frequently.

1 Introduction

Modern PDE solvers for time-dependent applications are currently being written so as 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 employs adaptive methods in both space and and time in order to attempt to control the numerical error.

This desire to control the spatial error means that the position and density of the spatial mesh points may vary dramatically over the course of an integration. This refinement and coarsening is undertaken automatically, [1] & [11]. As an example (taken from the wedge shock problem discussed in Section 5.2) Figure 3 shows the initial mesh and Figure 4 the final one after 9 refinement phases.

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 inter-processor dependencies. It is well

known that this mapping problem is NP hard, [6], 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 [10] and [12]) are based on partitioning a graph that corresponds to the communication requirements of the unstructured mesh. Until now, such algorithms have not addressed the incremental update partitioning problem posed when a mesh with an existing partition is being refined and/or coarsened. The aim of this paper is to propose a new method for this update problem which may be used *in conjunction with* existing graph based partitioning techniques.

Of the existing partitioning techniques Recursive Spectral Bisection is generally highly regarded, [10] & [12], and an improved version allowing for quadrisection and even octasection has recently been devised, [6]. The spectral algorithm forms a natural starting point for the work presented here and is described in full in Section 2. The limitations of the algorithm for time-dependent adaptive mesh codes are considered in following Section. In Section 4 a pre-processing step for the algorithm is introduced which addresses these limitations and appears to provide faster, more efficient dynamic load-balancing. In Section 5 a comparison is made between the algorithms and illustrated by some results from a PDE problem. Finally a few future directions for research are offered.

Note that the adaptive code used to motivate the work here employs h-refinement, [1], which in the context of time-dependent problems means that both the number and the distribution of the mesh points change as time progresses. Other adaptive mesh codes may use r-refinement in which a fixed number of mesh points move around the solution domain. However, provided that the points do not overtake each other this does not affect the connectivity of the communication graph and hence the optimal partitioning of the mesh.

2 Recursive Spectral Bisection

The Recursive Spectral Bisection algorithm (henceforth RSB) is one of a family of recursive bisection methods used for partitioning a graph. The common theme is the idea 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. Two other examples are Recursive Coordinate Bisection (RCB) and Recursive Graph Bisection (RGB). In

his paper, [10], Horst Simon describes all three algorithms and demonstrates the superiority of RSB over the other two.

The algorithm is now described in more detail.

2.1 Recursive Spectral Bisection – Motivation

Although this paper is concerned with unstructured mesh problems, spectral bisection is one of a number of methods which actually partition a graph derived from the mesh. The fundamental idea is to associate the nodes of an undirected graph with variables in the solution vector. The **dual communication graph** is then defined by connecting nodes together when the spatial discretisation gives rise to a dependency between the corresponding variables. The connections are made by specifying (undirected) edges between the nodes. Thus for cell-centred 2D finite volume calculations, for example, a node of the graph might be 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.

Consider, then, an undirected graph $G = G_n(V, E)$, where V is the set of n nodes or vertices, E the set of edges and G represents the connectivity of elements in the discretisation of the domain. A partition P of the domain is given by separating V into p mutually exclusive subsets.

Subsets of V can be defined by labelling each vertex. At each recursive stage a subgraph is to be *bisected* and so a variable x_v will be associated with each vertex $v \epsilon V$ and given the value $+1$ or -1 according to which subset it is in. Thus define $\mathbf{x}$ by

$$x_v \stackrel{def}{=} \begin{cases} +1 & \text{if } v\epsilon \text{ 'left' partition} \\ -1 & \text{if } v\epsilon \text{ 'right' partition} \end{cases}$$

The communication cost or number of edges between these two subsets (a cost that should be minimised) can now be defined with the quadratic form:

$$C(\mathbf{x}) \stackrel{def}{=} \sum_{(v,w)\epsilon E} (x_v - x_w)^2$$

where the sum is over vertices which are connected by an edge of the graph.

The minimisation of C is not an easy problem to solve in its current formulation. Consider, however, the Laplacian of the graph $L(G) = [l_{ij}]$, for $i, j = 1, \ldots, n$, given

by

$$l_{ij} \stackrel{def}{=} \begin{cases} -1 & \text{if } (v_i, v_j) \epsilon E \\ +\deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

or alternatively $L(G) = D - A$, where D is the diagonal matrix of vertex degrees and A is the adjacency matrix of the graph. It can now be seen that

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

and so $L(G)$ is the matrix associated with the quadratic form $C(\mathbf{x})$. This equivalence holds for any vector $\mathbf{x}$ (not just $x_v = +1$ or -1).

Note that the Laplacian has some interesting properties which are detailed more fully in [7]. It is easily verified that zero is an eigenvalue, λ_1 say, of L with associated eigenvector $\mathbf{e}$, where $e_i = 1$ for $i = 1, \ldots, n$. Thus $L(G)$ is positive semi-definite and hence $\lambda_1 = 0$ is the smallest eigenvalue. If G is connected then λ_2, the second smallest eigenvalue, is strictly positive (again see [7]).

If the x_v are now allowed to be *continuous* rather than *discrete* variables then C is minimised by the eigenvector corresponding to the smallest eigenvalue of L. From above, the eigenvector $\mathbf{x}_1 = \mathbf{e}$ (i.e. $x_v = 1$ for $v = 1, \ldots, n$) corresponding to $\lambda_1 = 0$ certainly minimises C (because $C = \mathbf{x}^t L \mathbf{x} = 0$) but locates all the vertices in the same subset (and hence trivially requires no communication). There is, however, the additional restriction of load-balancing, i.e.

$$\sum_{v\epsilon V} x_v = 0.$$

This is equivalent to $(\mathbf{x}, \mathbf{e}) = 0$, where $(.,.)$ refers to the inner product, and hence it is necessary to find the smallest eigenvalue with eigenvector orthogonal to $\mathbf{e}$. Since L is symmetric its eigenvectors form an orthogonal set and hence C is non-trivially minimised by the eigenpair $(\lambda_2, \mathbf{x}_2)$ where λ_2 is the smallest positive eigenvalue.

This vector, $\mathbf{x}_2$, now renders a weighting, x_v, for each vertex of the graph v. Because of the continuous approximation, in general the weightings will not be the $+1$ or -1 initially required. However this heuristic is not restrictive. Eigenvectors of the adjacency matrix have been previously studied for the information they give on the graph and used for partitioning purposes, [9]. The special properties of $\mathbf{x}_2$ have been investigated by Fiedler and his work, [3], gives theoretical justification for bisecting the graph based on its entries. Hence $\mathbf{x}_2$, often referred to as the Fiedler vector, is used as a bisection field, with the vertices v of the graph sorted according to the weighting given in x_v and the graph bisected on this basis.

2.2 The Method of Spectral Bisection

The three steps of the algorithm are summarised in figure 1 and described below.

```
repeat recursively {
    create Laplacian (input graph, output Laplacian)
    find 2nd eigenvector (input Laplacian, output Fiedler vector)
    sort and bisect (input Fiedler vector, output partition)
}
```

Figure 1: The Recursive Spectral Bisection Algorithm

Working either from the dual communication graph or directly from the mesh the Laplacian is created. The entries of this matrix L do not need to be stored at all, only the row (or column) indices of off-diagonal non-zeroes (all of value -1) are required. These can be stored as a vector of packed sparse vectors together with an indexing vector indicating the start of each new row (column). The diagonal entries are then given by the number of entries in each row (column). The fact that L is symmetric may also be used to further reduce the number of entries, but at the risk of significantly complicating the matrix-vector multiplication routine.

Following Simon, Pothen and Liou, [9] & [10], the Lanczos algorithm is used to calculate the Fiedler vector. This is a well-known Krylov subspace iterative technique, ideal for finding extremal eigenvalues of symmetric matrices (see for example [5]). Unfortunately the algorithm requires many steps to avoid misconvergence to non-extremal eigenvalues but the cost of estimating the current smallest can be reduced by bounding it inside an interval of decreasing size, [8]. The bulk of the work per iterative step is one matrix-vector multiplication, together with some vector operations. It can be implemented for the most part with the level 1 BLAS plus a tailor made matrix$\times$vector subroutine. Because of the load-balancing constraint, each Lanczos vector is explicitly orthogonalised against $\mathbf{e}$ as described in Section 2.1.

Finally the Fiedler vector is sorted and the graph bisected. Note that it is not necessary to employ a full sorting procedure, just a partitioning into two equal sized subsets.

3 Applications to Time-Dependent Problems

Whilst the RSB algorithm usually gives good results for a static problem, there are a number of areas in which improvements may be made for dynamic partitioning of adaptive meshes. Most notably the full method is expensive as the cost of the Lanczos method for a problem size n, falls somewhere between $O(n)$ and $O(n^2)$ with a large coefficient. It is difficult to be more precise as the number of iterations required for convergence to $\mathbf{x}_2$ increases with n. While this may not be a problem on 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.

The method may also be sensitive to small perturbations in the mesh. For instance Williams, [12] 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 a technique is presented that enables a graph-based algorithm to use existing information about the partition of a previous mesh. Again it is described with particular reference to the Recursive Spectral Bisection algorithm but the concept could be applied to any graph based method.

4 A Dynamic Partitioning Approach

If 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.

4.1 The Concept

The repartitioning algorithm used here assumes that, unless the mesh has changed dramatically, it is probable that the partitions will not need to be changed a great deal. Ideally most mesh elements will remain in the same subdomain whilst the boundaries are 'juggled'. Of course it is not clear that such 'juggling' will produce optimal communication costs (see however Section 5), but certainly if mesh elements 'close to' the inter-processor boundaries (or bisection boundaries for a Recursive Bisection algorithm) are the only ones involved then the issues discussed in Section 3 are all addressed. 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).

To effect this idea mesh elements which are far enough away from a inter-processor (or bisection) boundary to be ignored for the processes of repartitioning are chosen (by some heuristic – see Section 4.2). If the subdomains are compact enough this should result in clusters of mesh elements separated by a strip of elements alongside the boundaries. Moving to the dual graph these clusters are now treated as a *single* vertex. The partitioning algorithm then proceeds as before on this reduced size graph (a considerable cost saving) and for the redistribution it is expected that the clustered nodes will remain in the same partition (a node migration saving). In this way the adaptive techniques used to coarsen the mesh are mirrored to derive an adaptive technique to decrease the size of graph used in partitioning.

4.2 Implementation for Recursive Spectral Bisection

4.2.1 Clustering and Iteration

A method of selecting those graph vertices which are 'close to' the previous bisection boundary and those that are 'far enough away' from it must be found. At present this part of the code has been implemented as follows. Firstly vertices with an edge crossing the boundary are chosen. This defines the first level set of working vertices, L_1. Next all vertices sharing an edge with a vertex in L_1 are sought, giving a second level set, L_2. A third level set, L_3, is defined by selecting vertices with edges into L_2 and so on. Assuming the graph is connected, this gives an iterative technique which converges to (or more properly terminates with) the full graph. Note that at each stage a level set L_q is determined by the previous level set L_{q-1} alone, eliminating the need for full graph searches.

After each successive level set is chosen the clusters can be defined by *connected* groups of vertices which do not lie in one of the existing level sets. To create the reduced size graph, edges between cluster vertices are collapsed until each cluster is represented by a single vertex. Edges from cluster vertices into the last level set remain. This working graph is the one which is input into the Spectral Bisection algorithm.

Note that in determining the Laplacian of the working graph the number of vertices in each cluster has no effect. However each vertex cluster is likely to have more edges than an ordinary graph vertex and hence the corresponding rows and columns of the Laplacian will be less sparse. In addition it is possible for an ordinary graph vertex to have more than edge in common with a vertex cluster. These multiple edges can be represented in the Laplacian by redefining $l_{i,j} = -|\{e \epsilon E : e = (v_i, v_j)\}|$. Note

that it is still possible to store the Laplacian by storing only row and/or column indices (see Section 2.2).

4.2.2 Bisection

Because of clustering some of the entries in the resultant Fiedler vector represent more than one vertex. Thus in order to find the median of the vector an entry associated with a cluster is counted with a multiplicity of the number of vertices in that cluster. In itself this is easy to implement but a problem arises if a cluster lies across the median point. Of course it is not possible to bisect a cluster and so in this case the bisection has failed. Early experience suggests that this does not happen too often, but when it does it is either because the reduced graph is not 'wide enough' or because the mesh has changed significantly from the previous partition. Thus either the working graph is expanded by one more level set or possibly the full graph is reinstated and spectral bisection applied again.

4.2.3 Recursion

The recursive part of the method proceeds much as before. The only slight problem occurs in data migration. This is most easily demonstrated with an example. Suppose, then, there is an existing partition of 4 subdomains labelled '00', '01', '10' and '11' and the initial bisection is executed based on the first digit. Thus two clusters are formed, one containing elements lying in either '00' or '01' and away from the previous bisection boundary, the other similarly of elements in '10' and '11'. The Fiedler vector is found and the domain re-bisected.

At this stage the data lying in the 'wrong' partition moved and it is here that care must be taken. For example, it may be found that some elements in both '00' and '01' have to migrate to other side of the bisection. Clearly the easiest method is for '00' elements to be mapped to '10' and '01' elements to '11'. However this does not guarantee that the new '10' and '11' subdomains are connected. This can cause problems in defining the clusters at the next recursive level and code should be included to ensure either that migrating elements go to the right place, or that disconnected groups of vertices are not used to form a cluster. This is most easily accomplished by always using data which has recently migrated in the working part of the graph, although this is not the most efficient method in terms of either work or data migration.

4.3 The Dynamic Recursive Spectral Bisection Algorithm

The iterative pre-processing technique to be added to the Recursive Spectral Bisection (or other) algorithm can now be presented in full and is summarised in figure 2. The `spectral bisect` subroutine is just the three operations inside the loop in figure 1. Henceforth this combined algorithm will be referred to as DRSB.

```
repeat recursively {
    until (working graph large enough) {
        expand working graph (by next level set)
        cluster (remaining vertices)
    }
    while (working graph < full graph) {
        spectral bisect (input working graph, output partition)
        if (successful)
            escape to next subdomain
        else if (too many iterations)
            working graph = full graph
        else
            expand working graph (by next level set)
            cluster (remaining vertices)
    }
}
```

Figure 2: The Dynamic RSB Algorithm

Initially enough level sets are taken to

(a) balance the domain (i.e. if one cluster contains more than half of the vertices bisection is not possible);

(b) have a meaningful graph to use spectral bisection (i.e. just using one level set does not give enough information).

The Spectral Bisection algorithm is now applied and the Fiedler vector calculated. If the bisection fails because of a cluster close to the median of the vector the working graph is expand by one more level set and spectral bisection re-applied. This may be repeated until the level sets have recovered the full graph (when the bisection cannot fail), but this will have increased the costs to well above that of using full spectral bisection in the first place. As consistent failure suggests that the partition should be in a different place it is better terminate the iterations early. Fortunately this does not appear to happen very often (see Section 5).

There are two important heuristics inherent in this piece of pseudo-code, namely the number of level sets to make the initial `working graph large enough` and the maximum number of iterations. Of course neither are crucial to the eventual success of the algorithm but they both have important implications for the efficiency of the technique. Early results suggest that 3 level sets make a good working graph to start from (although this may depend on the density of edges in the graph). Subsequently, if the method fails after 2–3 iterations it may be better to then revert to the full graph. For coarse grained problems (including the initial recursive levels) the potential saving is considerably greater and a few more iterations may be worthwhile. Thus it is desirable parameterise this number with the granularity.

5 Results

The dynamic technique was initially tested on a number of different meshes by using RSB to get an initial partition and then DRSB to modify it. Of course in these cases the load was already balanced but it is interesting that the new algorithm would more often than not provide a partition with slightly fewer inter-processor edges. This was a surprise as it might be suspected that DRSB would never work as well, however there was never sufficient difference to draw any real conclusions.

Having established that the algorithm worked it was decided to compare DRSB results with those obtained from the full RSB algorithm. Although the two algorithms are not in direct competition, the aim was to assess the effectiveness of the dynamic technique by measuring both the possible cost savings and the quality of the resulting separator sets.

The test meshes were derived from the integration of a time-dependent PDE described below (Sections 5.2. The PDE solver generates an unstructured mesh and proceeds to integrate the solution with a variable step explicit time marching algorithm. The integration continues while the error estimate in each triangle remained below a pre-determined tolerance. Once the tolerance was exceeded adaptive code automatically refines or coarsens the mesh in order to yield an approximately uniform numerical error estimate across the domain. At this stage load balancing becomes necessary; the existing partition was interpolated onto the new mesh and then both DRSB (using the interpolated partition) and RSB (ignoring it) were run to partition the domain. For this example the meshes were all fairly small (700–900 triangles), and so the domain was only partitioned into 8 subdomains.

5.1 Metrics

Three metrics were used to compare the algorithms:

Mflops

The bulk of the cost of the spectral algorithm lies in the Lanczos iterations to find 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. Of course the DRSB algorithm has a lot more administrative work to do and a proper timing would have given a better assessment, however much of the level set search code has not been implemented very efficiently yet.

$|G_a|$ - Average Problem Size

Whilst the Mflops count gives a good indication of the cost of the algorithms, the inclusion of number of Lanczos iterations do not allow it to be very illuminating about the sizes of the sub-graphs, G_s, used for partitioning. Accordingly the average problem size for each mesh, $|G_a|$, defined as the sum of sizes of all the bisection problems divided by the number of bisection problems, is given. For RSB this figure is just a linear function of n. Thus if l is the number of recursive levels (so that $p = 2^l$) then the number of bisection problems is given by

$$\sum_{j=1}^{l} 2^{l-1} = 2^l - 1$$

and so for RSB

$$|G_a| = \frac{n + 2.n/2 + 4.n/4 + \ldots}{2^l - 1} = \frac{n.l}{2^l - 1}.$$

For DRSB however, each problem size is dependent on the granularity, the changes in the mesh and the previous bisection boundary as well n and p. In addition if the bisection fails because of the clustering both the number of problems will fail and, if iterations repeatedly fail and the code resorts to the full graph, the problem size can dramatically increase. It is interesting to note, however, that for both examples the DRSB figure is of the same order as $|E_i|$. Of course, this relationship cannot hold as p varies (since $|G_a|$ decreases and $|E_i|$ increases as p increases for fixed n) but as we might expect $|G_a^t|$ is largely dependent on $|E_i^{t-1}|$ where t denotes the time-level.

$|E_i|$ – the inter-processor edges

If E_i is defined to be the subset of edges which cross inter-processor boundaries after repartitioning then another metric is simply the size of this set. This measure was used because it gives an indication of the sizes of separator sets that might be expected for linear algebra using domain decomposition or sub-structuring. Since

this is related to the parallel overhead for such techniques it is a meaningful figure. It also gives an indication of the volume of communication traffic required for flushing the halos of the subdomains around the memory.

5.2 Euler Wedge Shock Problem

This problem is driven by the Euler equations in two space dimensions (see [11] for a full description) which simulate air at Mach 2.5 as it hits a 10 degree wedge and forms a shock front, [2]. 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.

The results are as follows for $p = 8$ where n = number of mesh elements.

	DRSB			RSB		
n	Mflops	$\|G_a\|$	$\|E_i\|$	Mflops	$\|G_a\|$	$\|E_i\|$
860	0.5	75	71	5.1	368	75
833	0.5	71	70	5.0	357	70
813	0.9	92	72	4.8	348	74
799	0.5	71	67	4.5	342	75
768	0.5	73	65	4.5	329	74
725	0.5	69	66	4.1	310	71
899	1.4	133	91	5.4	385	79
908	0.9	90	87	5.5	389	84
878	0.8	90	80	5.3	376	79
7483	6.5	764	669	44.2	3204	681

Table 1: Wedge Shock Comparison Results

From the Mflops count, for these problems DRSB is just under seven times times faster than RSB on average and up to ten times faster. The DRSB Mflops counts which are below average arose from the first iteration failing and the code having to expand the working graph. This is particularly noticeable for the 7th mesh (Mflops count of 1.4) where the first iteration failed in two of the bisection problems, one of these being at the second recursive level. In mitigation however, the 7th mesh had changed fairly dramatically from the 6th (725 $\rightarrow$ 899 triangles) with much refinement around the shock and de-refinement away from it. At only one stage did the code revert to the full sub-graph (for the 3rd mesh) and since this was at the third recursive level it did not add greatly to the cost.

The number of inter-processor edges are on average smaller for DRSB. Again, as with the early results, this is somewhat counter-intuitive as it might be expected that the method would get trapped in local minima in the minimisation problem. Possibly this effect might show up more on a problem with coarser granularity.

Rough timings were taken for the codes running sequentially. The solution and remeshing took about 1480 seconds, RSB took 63 seconds and DRSB 15 seconds. As the time-stepping is explicit there is no requirement to solve a system of non-linear equations and so a sophisticated partitioning algorithm such as RSB might be considered unnecessary in this case. However, if we assume about 90% efficiency then on 8 processors the integration code might take about 3.5 minutes and the difference in costs between RSB and DRSB starts to look significant. Parallelisation of the partitioning code (potentially very efficient for the Lanczos algorithm) may redress this balance.

6 Conclusions and Future Directions

The method looks very promising but further testing on a wide range of time-dependent PDEs is required. The DRSB algorithm provides effective load-balancing but it is possible that the cost may still be too high relative to the solution cost. In particular evaluation in conjunction with linear algebra which actually makes use of the separator sets would appear to be desirable. It is conceivable that repartitioning is not necessary after a relatively small amount of adaptivity and some criteria are needed for making this decision – which will also depend upon the cost of the load-balancing algorithm.

The code also needs some tuning work – both algorithmic for robustness and also implementation improvements for efficiency. Once accomplished some proper timings could be taken. The volume of node migration is another valuable metric to use and some more coding is necessary to optimise this. The changes needed for 3 dimensional meshes should be fairly minor but a parallel version will take a lot more work.

A few further possibilities for the technique can be summarised as follows:-

- Other Applications. As the DRSB algorithm is graph based it is not restricted to unstructured mesh problems and could be applied to, for example, matrix partitioning amongst other things.
- Other Load Balancing Methods. The technique of clustering could be applied to other algorithms to improve efficiency. Simulated Annealing (see [12]) is an obvious example.

- Accelerated Recursive Spectral Bisection. For non-adaptive meshes it would be nice to have a fast effective partitioning technique. Possibly this could be achieved by generating a coarse partition (using Recursive Coordinate Bisection for example – see [10]) and then refining it with the DRSB algorithm.

Acknowledgements. The authors would like to acknowledge the financial support of Shell Research Limited. Pete Jimack and Dave 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*, 1992.

[2] L. Demkowicz, J. T. Oden, W. Rachowicz, and O. Hardy. An *h-p* Taylor-Galerkin finite element method for compressible Euler equations. *Comp. Meth. Appl. Mech. Engrg.*, 88:363–396, 1991.

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

[4] J. E. Flaherty, P. J. Paslow, M. S. Shepherd, and J. D. Vasilakis. Adaptive Methods for Partial Differential Equations. In *Proc. of Workshop on Adaptive Computational Methods for Partial Differential Equations, Rensseleer Poly. Inst., 1988*, SIAM, Philadelphia, 1989.

[5] G. H. Golub and C. F. van Loan. *Matrix Computations (2nd ed.)*. John Hopkins, Baltimore, 1989.

[6] 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.

[7] B. Mohar. The Laplacian Spectrum of Graphs. Technical Report, Dept. of Mathematics, Univ. of Ljubljana, 61111 Ljubljana, Yugoslavia, 1988.

[8] B. N. Parlett, H. Simon, and L. M. Stringer. On Estimating the Largest Eigenvalue With the Lanczos Algorithm. *Math. Comp.*, 38:153–165, 1982.

[9] A. Pothen, H. D. Simon, and Kang-Pu L. Partitioning Sparse Matrices with Eigenvectors of Graphs. *SIAM J. Matrix Anal. Appl.*, 11:430–452, 1990.

[10] H. D. Simon. Partitioning of Unstructured Problems for Parallel Processing. Tech. Rep. RNR-91-08, NASA Ames Research Center (to appear in Computing Systems in Engineering), 1991.

[11] 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*, 1992.

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

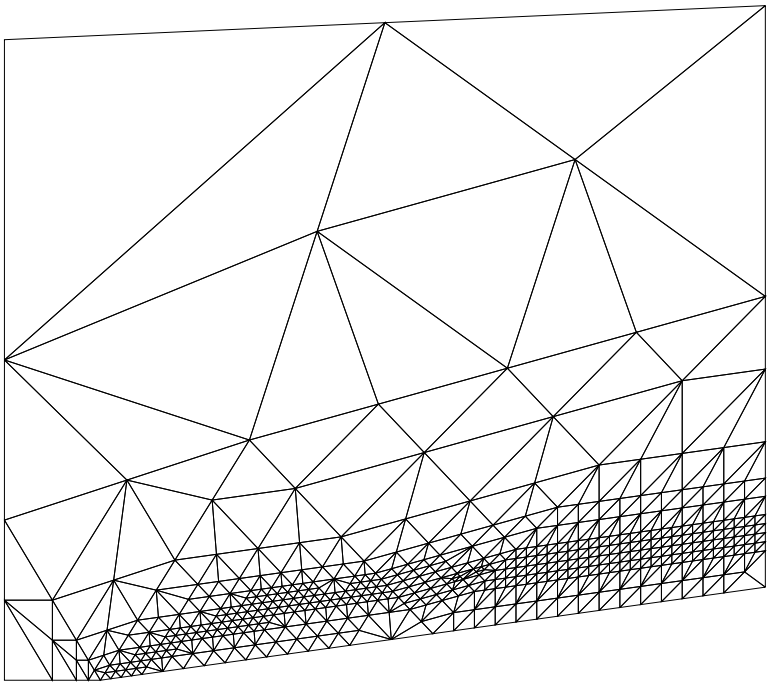

Figure 3: The initial mesh from the wedge shock problem

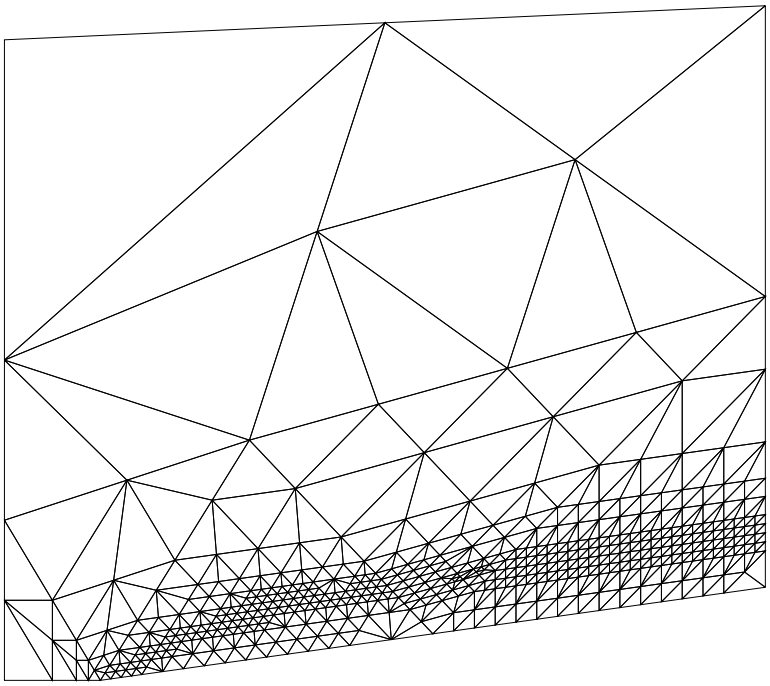

Figure 4: The final mesh from the wedge shock problem