Efficient Parallel Generation of Partitioned, Unstructured Meshes

by

D C Hodgson & P K Jimack

Abstract

In this paper we introduce a method for generating unstructured meshes in parallel which are partitioned in a "good" way. When solving a partial differential equation on a parallel distributed memory machine, the mesh should be decomposed so that the communication requirement of the numerical solver is minimised and also the amount of work to be performed on each processor is approximately equal. Most previous work in this area has concentrated on partitioning a mesh that has already been generated. We introduce a method which actually generates the partitioned mesh in parallel whilst producing a good quality decomposition and compare this method with other approaches.

1 Introduction

In this paper we are concerned with the generation of unstructured meshes for use in the finite element analysis of structural problems on parallel distributed memory computers. At present the use of distributed memory computers, such as transputer arrays or the Intel iPSC/860 for example, is less widespread than that of shared memory multiprocessor machines (such as the Cray family) for solving very large problems. The main reason for this is possibly because of the need to redesign "conventional" algorithms to take full advantage of the features offered by distributed memory machines. However the relative inexpensive of these machines together with their versatility means that their use is beginning to grow.

The use of unstructured meshes has a number of advantages over structured meshes but in particular they allow the straightforward representation of geometrically complicated domains. In this paper we consider the parallel generation of unstructured triangulations in 2-d however all of the concepts carry over to 3-d without difficulty. The usual approach to solving finite element problems in parallel on a distributed memory machine is to decompose the mesh into a number of subdomains and to allocate each of these subdomains to one processor. This decomposition of the elements of the mesh should have two main features. Each subdomain should contain approximately the same number of node points so as to achieve "load-balancing" (note that the bulk of the work in a typical code comes in solving linear systems of the form $K\underline{u} = \underline{f}$ and so the number of nodes in the mesh is more significant than the number of elements). Also, the number of node points which lie on the boundary between different subdomains (we will refer to such points as "interpartition boundary vertices" or IBVs) should be kept to a minimum since the amount of interprocessor communication required to solve the systems $K\underline{u} = \underline{f}$ will depend heavily upon this number.

There has been a considerable amount of research into the problem of partitioning an existing mesh across distributed memory in a manner compatible with the above (see for example [1, 2] and references therein). However this is a far from ideal approach since if we wish to solve very large problems in parallel we do not want the mesh generation itself

to be a serial bottleneck. We therefore study the problem of generating a partitioned mesh in parallel.

In order to create a mesh in parallel, the domain must be split up into subregions which can then be meshed simultaneously. Some parallel mesh generators, such as [3, 4], simply farm out subregions for gridding without distributing these in an intelligent manner for the solution procedure. If kept in a distributed fashion then the generated elements may be unevenly shared amongst the processors and there is no guarantee that any of the subregions allocated to a single processor will in fact be connected to one another. Although such methods may be able to generate grids in parallel reasonably efficiently they have the disadvantage that they cause the generation and partitioning to be performed as separate stages with all the communication overheads associated with such an approach. A more desirable technique is therefore to join these stages by distributing subregions in such a way that the total number of node points generated when triangulating each processor's subregions will be approximately the same, and that the union of the subregions allocated to each processor forms a "good" subdomain. We describe such an approach and contrast it with the techniques of Topping and Khan ([5]) and Löhner *et al* ([6]), which have a similar philosophy.

In section 2 we describe how the parallel mesh generator works. Section 3 explains implementation details and in section 4 we show how effective this technique is for a number of test problems by comparing the results obtained against those when using the traditional approach of partitioning an already generated mesh, applying a measure of partition quality discussed in [1].

2 The Parallel Mesh Generator

Parallel mesh generation is usually performed by splitting the domain up into subregions which can then be meshed simultaneously. There are numerous strategies for creating subregions of the domain, such as simple Cartesian splitting, quadtree splitting or using a background grid ([6]). We use the background grid method as this gives us an initial mesh to solve on and use the code in an adaptive framework. Elements of the background mesh are assigned to each processor which can then either be meshed individually or the subregion defined by the union of the elements can be meshed. In order to load balance the parallel mesh generation, the background grid elements should be distributed in such a way that the amount of meshing that each processor has to perform is approximately the same. By estimating the number of vertices which will be generated within each coarse background element, the background grid can be decomposed so that the generated knot points are evenly distributed amongst the subdomains/processors, so ensuring the load balancing of the numerical solver.

When producing meshes in parallel it is essential to ensure that the generated vertices

coincide along interpartition boundaries, i.e. two processors sharing a common subdomain boundary should have non-conflicting information on the number and positions of any vertices lying on that boundary. This may be achieved easily if the knot points are generated along background element edges *before* the grid is distributed for the parallel meshing. The information concerning the number of vertices along background grid edges proves useful since it may be used to estimate the amount of communication required for a given decomposition of the background mesh. It is therefore possible to distribute this initial mesh so that the parallel p.d.e. solver will be load balanced and its communication costs minimised when solving on the final generated grid.

2.1 The Mesh Generation Method

The mesh generator that our parallel technique is based upon is a two-dimensional version of Weatherill's algorithm ([7]). The method uses Delaunay triangulation to mesh points that may be created in a number of ways. Mesh density is governed by point distribution values associated with the generated vertices which define the desired spacing around each point. Given a discretised boundary of a domain the algorithm can create internal points driven by the distribution of the boundary vertices. The resulting mesh is boundary conforming and has no additional points added along edges of the domain. This is the method that is used to produce the initial, background grid.

In order to create the desired fine mesh, the point spacing is controlled by the background mesh. Each coarse grid vertex is given a point distribution value and the values for generated nodes are obtained from the interpolated spacing from this background mesh. If coarse mesh elements are to be treated as separate subdomains (and therefore meshed individually) then this interpolation is simple. However, if the subregions are defined by the union of the background elements allocated to each processor, the coarse grid triangle in which the newly formed vertex is generated, must be first established (See [8] for an efficient search algorithm).

Given a subdomain boundary and the necessary background grid information, Weatherill's generation procedure can be used to mesh this subdomain, producing a suitable point distribution throughout. Since the generator does not produce new points on the boundary of the domain it meshes, boundary nodes must first be created either along either all subdomain edges or all coarse grid edges (if each coarse grid element is to be meshed) before invoking the mesher. For an edge, i, of the background grid we determine the number of fine mesh edges (N_i) we require along it by using the formula

$$N_i = \left\lfloor \frac{l_i}{\alpha^{\frac{\delta_1+\delta_2}{2}}} \right\rfloor \qquad (1)$$

in which l_i is the length of the edge i, δ_1, δ_2 are the point distribution values at either end of the edge and α is the value used to control global grid point density ([7]). In order to

position the vertices along a coarse grid edge in accordance with the point distributions at its ends, we use a similar technique to Khan & Topping [4] but force the number of created edges to be N_i. The following C code is used to determine the length of each fine mesh edge segment.

```
/* l = length of coarse element edge */
/* pd1 = larger point distribution value */
/* pd2 = smaller point distribution value */
/* num = no. of fine edges to be generated */
/* ALPHA = global point density factor */
pd1 *= ALPHA;
pd2 *= ALPHA;
dx =  pd1;
l2 = l;
i = 0;
while(i++ < num){
   l1 = l2 * dx / (l2 + dx - pd2);
   edge[i] = l1; /* Save length of each edge */
   l2 -= l1;
   dx = (pd1 * l2 + pd2 * (l - l2)) / l;
}
```

The points are generated starting from the end which has the larger point distribution value. Vertices are positioned so that the length of each corresponding edge is made equal to the size of the interpolated point distribution value at that vertex. The edges generated in this way do not extend the full length of the background edge. This problem is rectified by distributing the remaining distance over the created edge segments in proportion to their lengths.

Weatherill's method was chosen to perform the mesh generation for a number of reasons. A primary requisite was for a method which would not create additional points on the subdomain boundary. This fact means that two processors, whose respective subdomains share a common boundary, do not produce two different discretisations of the same boundary. Another requirement this generator satisfies is that it can be controlled to place vertices where most needed in the domain. The mesh generation technique also proves to be very efficient and the run time grows approximately linearly with the number of points generated ([7]). This means that if the coarse mesh can be distributed so that each processor will create a similar number of vertices (see next section), the parallel generation procedure should be well load balanced.

2.2 The Partitioner

The partitioning method should be able to decompose the background grid so that each subdomain will have an equal number of vertices generated in it. This will not only equalise each processor's meshing time, but will also aid load balancing of the parallel p.d.e. solver when solving on the final grid. In addition to this, the partitioner should minimise the amount of communication required in the solution phase.

We may express this partitioning problem in terms of graph theory by looking at a weighted dual graph of the background mesh. The dual graph represents each element by a vertex and has neighbouring elements connected by an edge. Associated with each graph node is a weight stating the number of mesh points that will be generated in the corresponding coarse grid element. Obviously it is impossible to know *a priori* the number of vertices that will be created within a background element and so the approach used here is to set the graph node weights by using an accurate prediction (see section 2.3). The dual graph edges are weighted as well; with the value of the number of fine mesh edges that are to be placed along the corresponding background edge (i.e. N_i from equation (1)). This weight will relate to communication costs when solving the partial differential equation on the decomposed grid. The problem that must therefore be solved is to split the weighted dual graph into subgraphs whose total nodal values are approximately equal, while minimising the sum of edge weights cut (the cut-weight) by the partition.

A common and well-liked approach to solving the unweighted graph partitioning problem is the recursive spectral bisection method (RSB [2]). Although extremely computationally expense (requiring the calculation of an eigenvector of an $n \times n$ matrix, where n is the number of elements in the grid), the technique does perform better than most algorithms in minimising the cut-weight. (In [9] & [10] two techniques are explained for speeding up the method using a multi-grid type approach). This computational expense is not considered to be problematic for our purposes however, since the partitioner will only be required to decompose relatively coarse background meshes. In [11] Hendrickson & Leland suggest an extension of the RSB method to be able to handle the weighted partitioning problem and hence solve the background grid allocation stage for the parallel mesh generator.

Hendrickson & Leland consider the bisection problem as that of a discrete optimisation, which may be written

$$\text{minimise } \frac{1}{4}\sum_{e\leftrightarrow f} w_{E_{ef}}(x_e - x_f)^2$$

$$\text{subject to } \quad \sum_i x_i w_{N_i} = 0 \ ,$$

where $x_i = \pm 1$, corresponding to element i being in one or other subdomain, w_{N_i} are

the nodal weights, $w_{E_{ef}}$ the edge weights and the first summation is over the edges of the dual graph whilst the second is over the V vertices. Defining the weighted Laplacian matrix L of the dual graph as

$$L_{ij} = \begin{cases} -w_{E_{ij}} & \text{if vertices } i \ \& \ j \text{ are connected} \\ \sum_{k=1_{k \neq i}}^{V} w_{E_{ik}} & \text{if } i = j \end{cases}$$

the problem may be written in matrix terms as

$$\text{minimise } \frac{1}{4} \underline{x}^T L \underline{x} \quad \text{subject to} \quad \underline{x}^T \underline{w}_N = 0 \ , \ \ x_i = \pm 1 \ .$$

Due to the complexity of this problem the variables x_i are allowed to be continuous rather than discrete. By introducing $\underline{y}$, $\underline{s}$ and D, where

$$y_i = x_i \sqrt{w_{N_i}} \ \ , \ s_i = \sqrt{w_{N_i}} \ \text{ and } \ D = diag\left(\frac{1}{\sqrt{w_{N_i}}} \right) \ ,$$

it can be rewritten as

$$\text{minimise } \underline{y}^T S \underline{y} \quad \text{subject to} \quad \underline{y}^T \underline{s} = 0 \tag{2}$$

(where $S = D^T L D$), neglecting the factor of one quarter.

We can find a solution to (2) in a similar way to RSB, by finding the eigenvector $\underline{u}_2$ corresponding to the smallest positive eigenvalue λ_2 of S (see [11]). As with RSB this computation is best performed by a version of the Lanczos algorithm ([12]). The partitioning vector $\underline{x}$ we require is therefore given by

$$x_i = \frac{u_{2_i}}{\sqrt{w_{N_i}}} \ .$$

We note that since the problem is solved using continuous variables, the property that $x_i = \pm 1$ is lost and therefore the solution obtained will not be optimal. The partitioning is made by sorting the V vertices of the dual graph according to the size of their entry in $\underline{x}$ and placing elements represented by $x'_i : i = 1 \ldots n$ in one subdomain (with $\underline{x}'$ being the sorted vector) and those by $x'_i : n+1 \ldots V$ in the other, with n chosen so that

$$\sum_{i=1}^{n} w'_{N_i} - \sum_{i=n+1}^{V} w'_{N_i}$$

(where w'_{N_i} is the weight of the node represented by x'_i) is as small as possible. The effect of this is to produce a decomposition of the mesh into two subdomains of approximately equal weight which has a relatively small number of vertices lying on the boundary between the two subdomains.

Rather than use a spectral technique, in [6] Löhner *et al.* employ a simple graph-based, "wavefront" type of scheme to perform the background grid splitting. Although

very computationally cheap, their method appears to have two inherent drawbacks. The algorithm proceeds by choosing suitable elements until a subdomain's total nodal weight exceeds the desired value of $\frac{1}{P}\sum_{i=1}^{V} w_{N_i}$,where P is the required number of subdomains. Clearly a subdomain's weight could be significantly larger than desired if, for example, the last element allocated to it has a very large weight, and the last subdomain to be created is likely to have a much smaller total weight than the others. The spectral method should not have such a drawback since the load balancing constraint is explicitly included in the formulation of the partitioning problem, thus ensuring more evenly weighted subdomains. Löhner's partitioner creates background grid subdomains which have good aspect ratios but fails to take into consideration the number of vertices that will be created along interpartition boundaries and could therefore produce coarse grid decompositions which lead to high communication costs for the solver.

Topping and Khan [5] use a genetic algorithm to tackle the background grid partitioning problem. This method explicitly includes the load balancing constraint and also takes into account interpartition boundary edges in order to minimise the cut-weight but is, however, very computationally intensive. Due to this fact, their method is best used with very coarse background meshes. Since the weighted RSB technique is much faster, we may have much finer background grids which allows us to produce well-balanced partitions even when running on a large number of processors. Allowing finer background meshes means being able to handle domains which have complex boundaries and also gives greater control over mesh density of the final grid. Using a finer initial grid, when creating a mesh of a particular size, means that each background element will contain fewer generated triangles, thus increasing the likelyhood of being able to produce a partition which contains a small number of IBV's.

When scaling up to a massively parallel system the partitioning stage is likely to become a bottleneck due to the fact that a sufficiently large background mesh will be required. In order to avoid this problem the partitioner itself could be parallelised (see [13] for a parallel implementation of standard RSB). It would then be necessary to determine the number of processors required to perform the partitioning most efficiently. An alternative approach is to employ a serial multi-level version of the weighted spectral algorithm (see [10]) which proves to be reasonably inexpensive.

2.3 The Estimator for Vertex Weights

In order for the background grid to be decomposed so that the generated subdomains contain an equal number of vertices, we need to know the number of points that will be created within each background element (i.e. w_{N_i}). Since it is not possible to know this information without actually performing the meshing, we must use an estimation method to predict w_{N_i}. (We choose to load balance knot points rather than elements

since a typical iterative FEM solver ([1]) requires this.)

The estimation method that we use is the 2-D equivalent of the formula used to decide the number of points/edges to be placed along each background element edge (equation (1)). For an element i of area A_i with point distribution values (p.d.v.'s) δ_{i_1}, δ_{i_2} and δ_{i_3} at its vertices, the formula

$$w_{N_i} = \frac{A_i}{\left(\alpha \frac{\delta_{i_1}+\delta_{i_2}+\delta_{i_3}}{3}\right)^2} \tag{3}$$

(where α is the global point density control value) gives an estimate of the number of knot points that will be generated within it. This method is not particularly designed to predict the actual number of points created but instead to produce a value proportional to the actual one, since this is all that is required. Experimentation shows that (3) accurately predicts numbers of generated points when an element's three point distribution values do not differ too greatly. For example, figure 1 shows the performance of the estimator when the point distribution values of all the coarse grid vertices of a test mesh are set to the same constant. Due to the variation in the sizes of coarse grid elements, a wide ranging number of fine mesh vertices are created in the background elements. The (almost) linear plot shows that the predictions are proportional to the actual values. Using the same coarse test grid, figure 2 shows the results when there is a large variance between the point distribution values at the vertices of some of the elements. As can be seen the performance of the estimator is severely impaired.

In order to make more accurate predictions where the point distribution values have a great variance, the element in question can be "virtually" refined by placing an imaginary vertex at its centroid and the value of w_{N_i} is then taken as the sum of the estimates for the three child elements. It can be seen that this approach may help since the point distribution values of each child element will vary less. By employing this technique and "virtually" refining a background element a number of levels, the performance of the estimator is greatly improved. The estimate for a coarse grid element when applying increasing levels of virtual refinement does converge, as shown in figure 3 for a typical example. Figure 4 shows the performance of the estimator when using virtual refinement for the same test problem as shown in figure 2.

In their method Topping *et al.*[5] choose to use a neural network to predict the number of fine mesh triangles which will be created within a particular background element. This technique proves to be very accurate in its predictions but requires the neural net to be trained with a number of test meshes. In [14] they find that the predictor becomes very unreliable when having to predict values greater than those it had been given during the training stage and so it is vital to train the neural net on similar sized problems to those for which it is intended. It is also found that they require two neural networks for the size of problems they are tackling, one to predict small numbers and the other for the prediction of much larger values (up to 1,226 fine grid elements per background

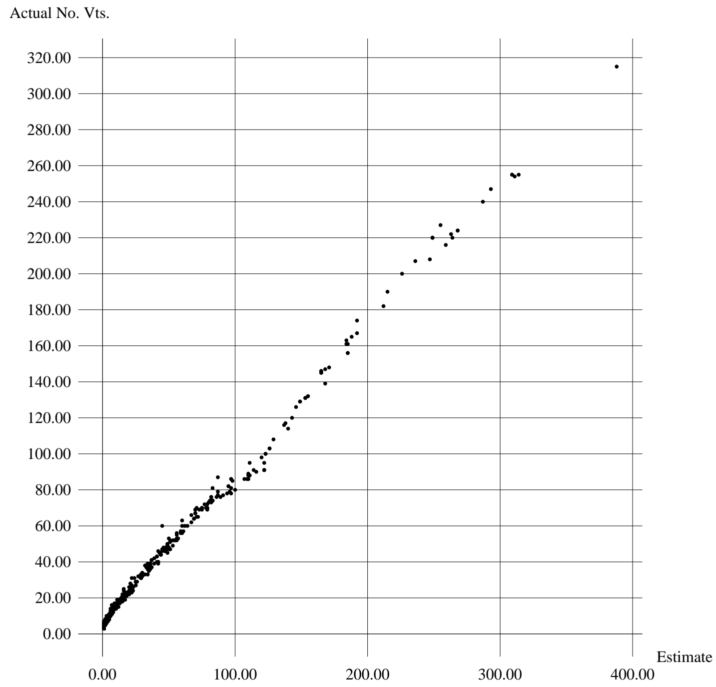


Figure 1: Performance of estimator when p.d.v.'s are constant: Each point in the graph represents a coarse grid element. The graph shows the actual number of vertices generated within each coarse element plotted against the estimated value.

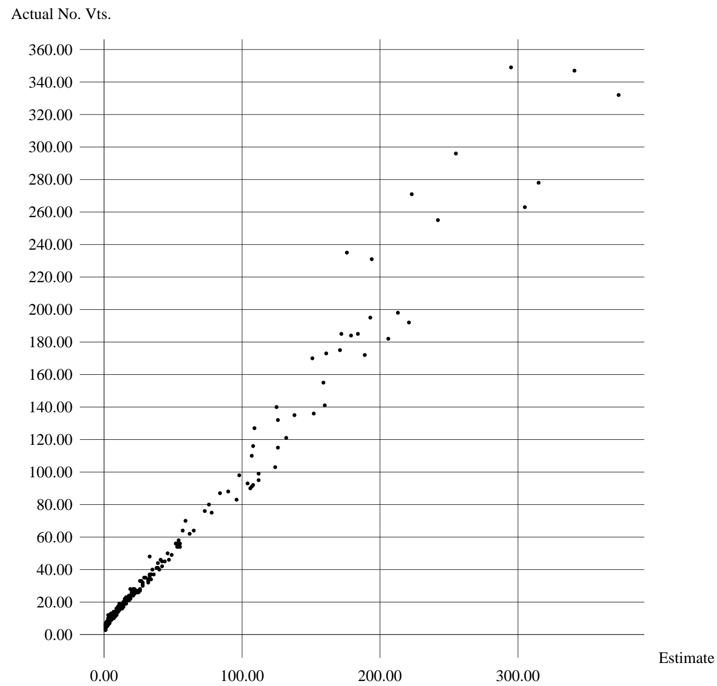


Figure 2: Performance of estimator when p.d.v.'s may differ.

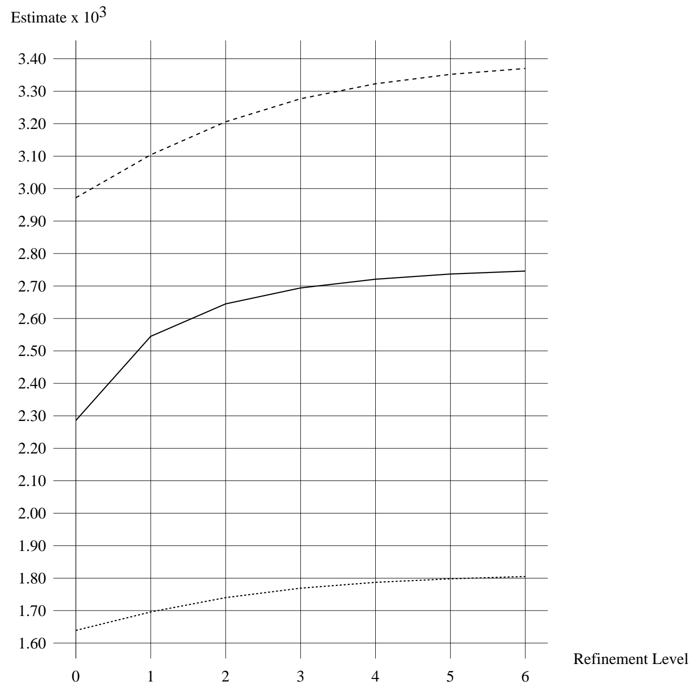


Figure 3: Convergence of estimate for three typical elements.

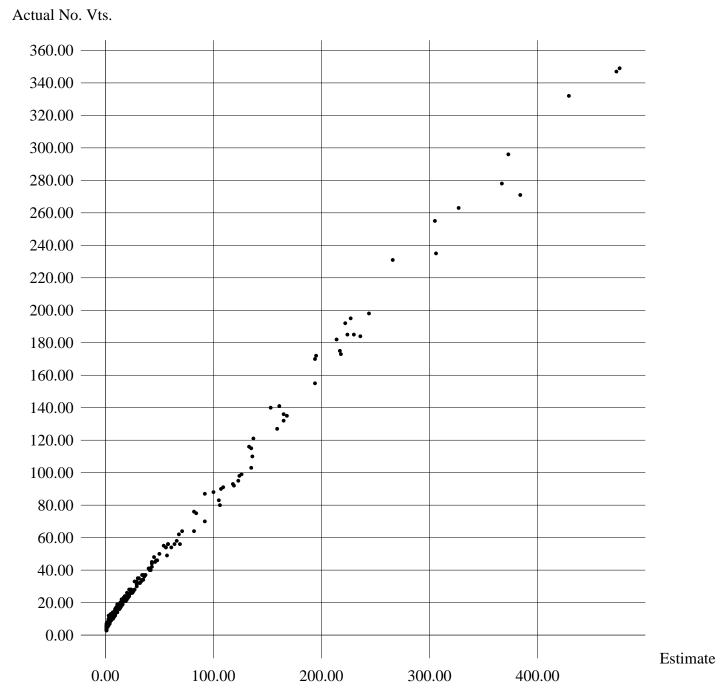


Figure 4: Performance of estimator using virtual refinement.

element). The estimator we use, although not as accurate, appears to be able to handle much larger predictions reasonably well, even when the background grid elements are not well shaped. We emphasise that the most important feature of the predictor is to produce an estimate which is proportional to the actual value (shown by the linear plot of figure 4) rather than the accuracy of the prediction.

3 Implementation Details and Issues

In this section we address the issues concerning the implementation of the mesh generation method on a parallel machine and discuss details of our implementation.

Given the boundary points of a domain and the associated connectivities, an initial coarse, or background, triangulation is obtained using the Delaunay mesher ([7]). This meshing algorithm creates points in such a way that the boundary point distribution is extended into the domain in a smooth manner. The number of elements produced can be controlled by adjusting the global point density value α in order to create a background grid of a suitable size. The background mesh should contain a sufficient number of elements so that the partitioner is able to produce well balanced subdomains. As the number of processors is increased, the number of background elements may also need to be made greater if the partitioning is to remain well balanced.

To control the creation of the fine mesh each vertex of the background grid is assigned a value which dictates the desired spacing about that point. These point distribution values can be derived by various means. Using the generator in an adaptive context, the distribution values can be determined by solving the finite element problem on the coarse mesh and estimating the errors over each element ([4]). In order to generate a suitable mesh for an initial value problem the point distribution values may be set based upon the initial data. The generator can be used to produce meshes in which the point creation is driven by the boundary point distribution by leaving the point distribution values as they are after the background grid generation stage and applying a smaller value of α for the subsequent production of the fine mesh. If the desired distribution of vertices in the final grid is known *a priori*, then the point distribution values can be set by hand or by using a suitable function. We use this latter approach for the purpose of testing the method here.

In our implementation the background mesh creation, the prediction of numbers of generated points and the partitioning stage are all carried out on a single, master processor. Once the background grid decomposition is known, the coarse grid elements are distributed to the processor to which they have been assigned. The discretisation of the subdomain boundaries is performed in parallel with each processor generating the points along its entire subdomain boundary. There is no problem with vertices generated by two different processors not coinciding along the same background edge, since the method

used to position the nodes (see section 2.1) creates a unique discretisation of that edge as defined by the point distribution values at its ends. By generating vertices from the end of the edge which has the larger point distribution value, or the end at which the background node has the greater global node number if the distribution values are equal, it is ensured that exactly the same floating point operations are performed by the processors in question, so producing no discrepancies in the coordinates of the shared vertices, provided an homogeneous parallel system is used.

The fine grid vertices are numbered locally to the processor on which they are created. This does not pose any problems since for a typical parallel finite element solver (see for example [1]) a global numbering scheme of vertices and elements is not required, however, a data structure must be built which gives information concerning data transfer across interpartition boundaries. Using the parallel solver in [1] each processor needs a list of local vertices which are shared with neighbouring subdomains. For the correct updating of values associated with interpartition boundary vertices, two processors with a common subdomain boundary must have the shared vertices listed in the same order. Such a data structure can be built quite simply when using our mesh generation approach: two processors sharing a common interpartition boundary have their IBVs listed with the coarse grid vertices first followed by, for each background interpartition boundary edge, the fine mesh vertices along it. To achieve consistent node orderings the coarse grid vertices and edges on each processor are listed in the same order as the global node and edge numberings respectively (this is performed on the master process) and the fine mesh vertices created along subdomain interface edges are listed in order from the end which has the greater coarse grid global node number.

The fine mesh created does not strictly conform to the Delaunay criterion because of the constraint that the interpartition boundary edges must naturally appear in the grid. However, each subregion (either a coarse grid element or a whole subdomain) is Delaunay conforming. The complete Delaunay structure could be recovered once the parallel meshing has taken place but would require a certain amount of inter-processor communication. Doing this would also then create the need to assign the newly formed elements to processors, thus causing extra computation and communication. Since the original background mesh is Delaunay conforming, the subdomains produced should be of a reasonable shape and therefore we do not bother to Delaunay triangulate across subdomain boundaries or background grid edges.

The final stage of mesh generation is that of mesh smoothing. A Laplacian filter ([15]) is used to perform the smoothing which alters the position of each grid point ($\underline{v}_o$) internal to the whole domain according to

$$\underline{v}_o^{n+1} = \underline{v}_o^n + \frac{\omega}{m} \sum_{i=1}^{m} \left(\underline{v}_i^n - \underline{v}_o^n \right) \ ,$$

where m is the number of neighbouring vertices and ω is a relaxation parameter. In order to perform the smoothing calculation for interpartition boundary vertices, communication is required between processors sharing common subdomain boundaries. The amount of communication is strongly related to the number of IBVs, so a good partitioning of the background mesh will not only aid in minimising the amount of communication in the parallel solver but will also lower the cost of the smoothing operation. An alternative approach is to simply apply the smoothing calculation to those vertices on the interior of each subdomain which has the advantage that no inter processor communication is required. When choosing to mesh individual background grid elements, rather than the subregion defined by the union of a processor's elements, if the coarse grid/fine grid hierarchy is required to be maintained (e.g. for domain decomposition preconditioning purposes) then the smoothing should only be performed on the grid points internal to each background element. Assuming the initial coarse grid is smoothed this should be perfectly adequate.

4 Evaluation

To test our parallel mesh generator, the above algorithms have been impelmented on an Intel iPSC/860. Two sample domains were chosen to test whether the method could perform well on example geometries. The background grids created contain differing numbers of elements depending on the number of processors the parallel mesh generation was to be executed on. Different functions were used to assign the point distribution values to coarse grid vertices in order to simulate regions of stress. Figures 5 and 6 show example background and corresponding fine meshes for both test domains generated on four processors. The example meshes shown here consist of only four subdomains in order to aid clarity, however, an arbitrary number of subregions may be generated. The first test is a straightforward example on a simple domain in which heavy refinement is required in three of the corners. The second test is a more contrived example designed to show that the method can work on difficult domain geometries.

A number of meshes of differing numbers of elements were generated on various numbers of processors in order to test the quality of the produced partitions. The two major factors influencing the quality being the extent to which the partitioning is load balanced and the cost of the communication required when solving on the distributed mesh.

These two issues are examined in this section together with the effect that any load-imbalance has on a parallel finite element solver.

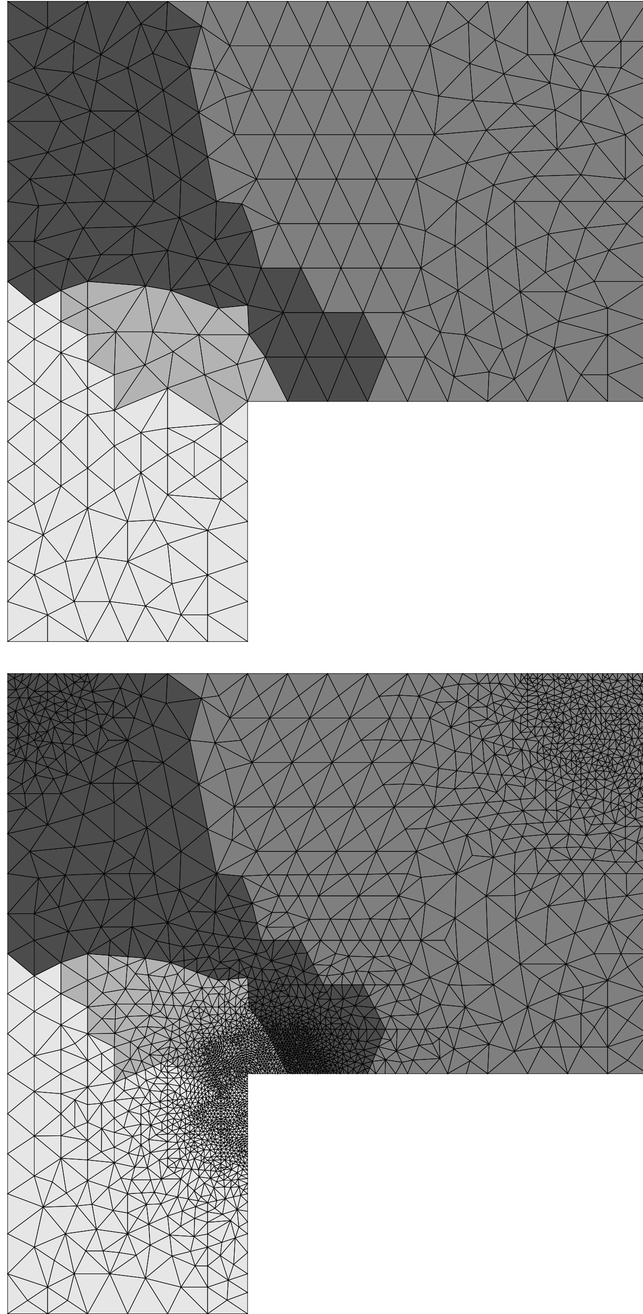
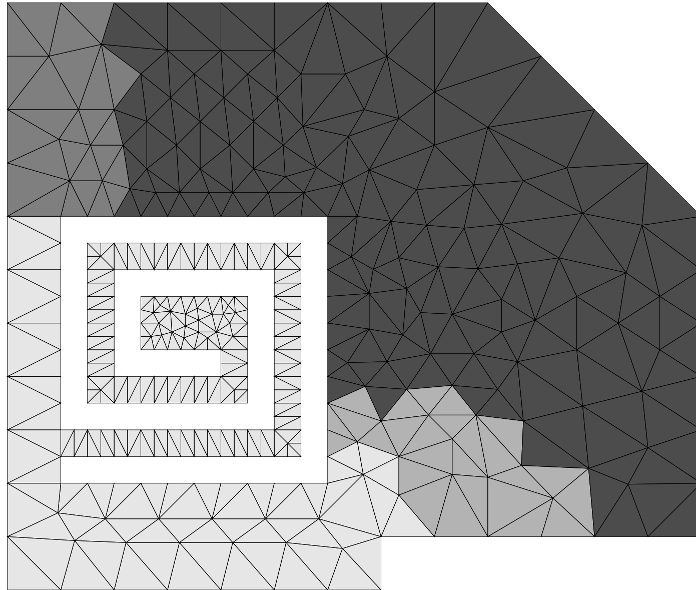


Figure 5: Coarse and refined meshes of Geometry 1 generated on four processors.



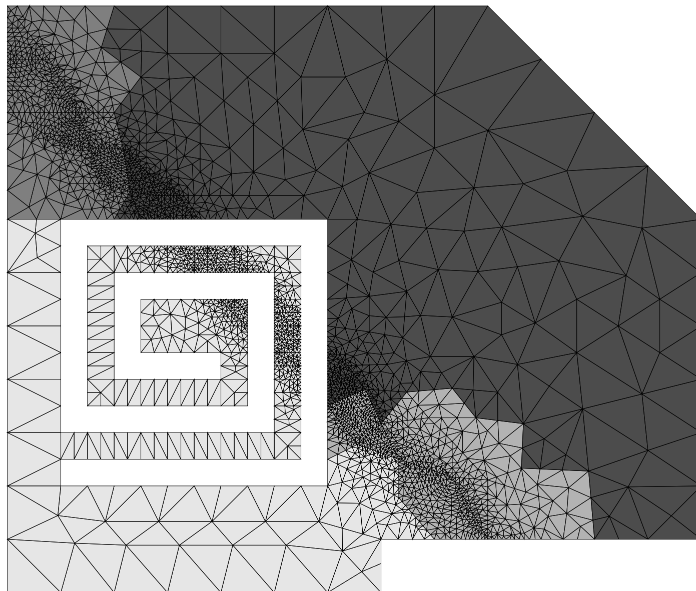


Figure 6: Coarse and refined meshes of Geometry 2 generated on four processors.

4.1 Load Balance

In a typical finite element code the largest computational cost in the solution phase is the linear system solve or solves. Using a conjugate gradient technique, for example, the cost of each iteration is governed by the number of unknowns, i.e. the mesh vertices. When solving on a parallel machine each processor should be given the same number of unknowns thus ensuring that the processors perform the same amount of work. We therefore evaluate the quality of load balancing achieved with our parallel mesher by considering the total number of vertices created on each processor. Tables 1 and 2 show results for a number of large meshes of both test domains. Indicated in the tables are the number of vertices generated by each processor, the time required for the meshing and the percentage by which each subdomain's number of vertices differs from the average.

In the creation of the smaller meshes the difference between the number of points generated in the subdomains is reasonably small and also the corresponding meshing times are well balanced. For the larger grids this is not the case. For example, in test four on domain one there is a difference of 7500 vertices between the processors with the maximum and minimum values. Although this number appears quite large, in terms of the percentage it differs from the mean value over the eight subdomains (+7.8%) it is not so bad. The percentage differences obtained with our method are comparable in size to those quoted by Khan [16]. The great variation in the meshing times observed in the case of the larger meshes can be attributed to our inefficient implementation of the Delaunay grid generator. Since the execution time of our mesher grows nonlinearly with the number of vertices created (instead of the linear run-time suggested by Weatherill [7]) the number of points generated within each background element becomes a factor. Hence the generation times for two subdomains containing an equal number of mesh points differ quite considerably if one of the subdomains generated consists of a large number of coarse grid elements each containing reasonably few fine elements and the second subdomain has fewer, more densely refined background elements. One would expect that with a linear run-time implementation of the mesh generator the meshing time on a processor will be proportional to the number of vertices it creates. Therefore if the mesh is partitioned so that each subdomain generated contains approximately the same number of mesh points, the processor's meshing times should also be well load balanced.

Domain 1 Test 1	Coarse mesh : 1052 elements Final mesh : 25942 elements		
Sub.	No. vts.	%age diff.	Gen. time
1	6776	+1.4	7.4
2	6595	-1.4	7.9

Domain 1 Test 2	Coarse mesh : 1188 elements Final mesh : 79222 elements		
Sub.	No. vts.	%age diff	Gen. time
1	10235	+0.6	11.5
2	10117	-0.5	11.8
3	10262	+0.9	12.2
4	10077	-0.9	13.4

Domain 1 Test 3	Coarse mesh : 1206 elements Final mesh : 168590 elements		
Sub.	No. vts.	%age diff	Gen. time
1	21772	+1.4	24.4
2	21338	-0.6	26.2
3	20917	-2.6	26.8
4	21867	+1.8	31.1

Domain 1 Test 4	Coarse mesh : 1354 elements Final mesh : 836183 elements		
Sub.	No. vts.	%age diff	Gen. time
1	52254	-1.3	81.7
2	50071	-5.4	55.0
3	53507	+1.1	80.2
4	49508	-6.5	101.9
5	57097	+7.8	98.8
6	52764	-0.3	72.2
7	53645	+1.3	112.1
8	54685	+3.3	95.0

Table 1: Load balancing results for domain 1.

Domain 2 Test 1	Coarse mesh : 1099 elements Final mesh : 24933 elements		
Sub.	No. vts.	%age diff	Gen. time
1	6542	+0.4	7.2
2	6484	-0.4	7.4

Domain 2 Test 2	Coarse mesh : 1193 elements Final mesh : 85343 elements		
Sub.	No. vts.	%age diff	Gen. time
1	10895	-1.2	11.5
2	10826	-1.9	14.6
3	10778	-2.3	13.7
4	11629	+5.4	13.5

Domain 2 Test 3	Coarse mesh : 1259 elements Final mesh : 151678 elements		
Sub.	No. vts.	%age diff	Gen. time
1	19710	+1.4	21.5
2	19660	+1.1	28.5
3	19833	+2.0	23.8
4	18552	-4.6	25.6

Domain 2 Test 4	Coarse mesh : 1361 elements Final mesh : 615478 elements		
Sub.	No. vts.	%age diff	Gen. time
1	39757	+1.0	46.2
2	39581	+0.5	52.4
3	39880	+1.3	87.5
4	36569	-7.1	75.6
5	39584	+0.5	70.1
6	40357	+2.5	76.2
7	40950	+4.0	59.6
8	38303	-2.7	61.0

Table 2: Load balancing results for domain 2.

4.2 Communication Costs

The second issue concerning the efficiency of the parallel solver is that of communication cost. In order to maximise the efficiency of any parallel method it is required that the overheads in the computation are kept to a minimum. The overhead is the time spent not performing useful computation and includes communication and synchronisation expense. The cost of sending a message from one processor to another is proportional to its length and therefore we aim to keep the size of any messages communicated down to a minimum. In a typical parallel p.d.e. solver (e.g. see [1, 17]) communication is required for any vertices which lie on the boundary between subdomains. A standard measure of communication cost is thus the cut-weight of the dual graph of the partitioned mesh. However, this metric fails to take into consideration that local communication between adjacent subdomains occurs concurrently and so a more accurate figure for communication expense can be found by calculating the maximum number of IBVs that are held on any processor (Max N_B) as shown in [1].

The cut-weights and values of Max N_B for meshes created by the parallel mesh generator (PMG) are compared to those values obtained by applying standard partitioning techniques (recursive spectral bisection, modified recursive graph bisection and recursive node cluster bisection) to the fine meshes produced. Recursive spectral bisection (RSB) [1, 2] is the un-weighted version of the partitioner which is employed to perform the coarse grid partitioning stage of our parallel mesh generation technique. As noted in section 2.2, this method is very computationally expensive but does produce superior results compared to most other approaches. The modified recursive graph bisection (MRGB) method [1] is a simple and cheap graph-based algorithm which is a modification of the recursive graph bisection method discussed by Simon [2]. Each bisection begins by finding two approximately extremal vertices of the dual graph and then builds a partition up around them by forming two sets. Each set first consists of those vertices which are at most one edge from an "extremal" vertex, then at most two edges, etc., until all of the vertices of the dual graph are claimed. Some exchange of data is then done at the end to ensure that each set is the same size. Recursive node cluster bisection (RNCB) [1] is a hybrid of RSB and MRGB. It uses the concept of node clusters ([18]) to reduce the size of the partitioning problem and so decrease the execution time. Two clusters of nodes are formed by partitioning a fixed percentage of the elements of the grid using the graph algorithm described above. The two groups of elements which have been built up around each "extremal" node of the dual graph are then treated as single, high degree, nodes of a modified graph which may be partitioned spectrally. If necessary extra constraints can be added to the continuous optimisation problem to ensure that the two high degree nodes, which represent clusters of finite elements, are forced into different halves of the partition.

Problem		Cut-weight of final mesh				Max N_B in final mesh			
Domain	Test	PMG	RSB	RNCB	MRGB	PMG	RSB	RNCB	MRGB
1	1	131	102	116	148	130	101	115	147
1	2	588	-	472	652	423	-	337	453
2	1	99	237	298	402	97	235	296	400
2	2	591	-	1383	1068	408	-	829	858

Table 3: Partitioning results.

Table 3 shows the cut-weight and Max N_B values for test cases one and two on both domains. The RSB results for test two are not obtainable due to the high computation time and the amount of memory required to perform the calculation. Because of the vast amount of disk space required to store the larger meshes, the partitioners are not run on these grids. We note that since the RSB, RNCB and MRGB partitioners balance elements rather than vertices as the PMG does, these experiments can only give a good estimate of how the methods compare.

The results show that in terms of minimising communication costs the decompositions produced by the parallel mesher are competitive with partitions generated by the other methods. Results on domain one are characteristic of those that we have found throughout the testing of the method, in that the partitionings are slightly worst than those produced by RSB and RNCB but are better than the MRGB results. In the tests on domain two the PMG method, somewhat surprisingly, out-performs all the other methods by a considerable margin. This may be due to the fact that the imbalance in the allocation of subdomain elements which the PMG produces allows a partitioning with a smaller cut-weight to be made.

As larger meshes are produced, and assuming the initial background mesh is kept the same, there is a possibility that the results obtained using the parallel method may compare not so favourably since in decomposing the coarse grid the partitioner will have much less freedom in determining the best bisections compared to a method working on the final mesh. Hence it may be necessary to increase the density of the background mesh in order to produce good quality partitions.

4.3 Effect of Load Imbalance on Parallel Solver

In section 4.1 it was seen that our parallel mesh generation method often produces partitionings in which the unknowns are allocated unevenly amongst the processors. (We note that although the decomposition is not perfect, the unknowns are roughly equidistributed.) Here we determine what effect this load imbalance has on a typical parallel

finite element solver. We look at a parallel Poisson solver (introduced in [1]) which uses the partitioned matrix method (PMM) to solve the system of linear simultaneous equations. The PMM ([19]) consists of applying an iterative method, in our case preconditioned conjugate gradients, to the linear system in which the unknowns are reordered to produce a block-arrowhead matrix. The solver uses diagonal scaling as the preconditioner and is implemented on the iPSC/860. Current research is looking into more effective parallel preconditioning techniques for use on unstructured grids.

The execution times when solving on the distributed meshes created in test 2 on both domains are compared against those obtained using the RNCB and MRGB partitionings. Both the RNCB and MRGB decompositions will be load balanced (exactly in terms of elements and reasonably well balanced in vertices) but will induce different solution times due to the extent to which the number of IBVs has been minimised. The following table shows the results.

Problem		Soln. Time (seconds)			No. of
Domain	Test	PMG	RNCB	MRGB	Its.
1	2	86.1	84.8	87.6	574
2	2	96.1	96.8	97.5	574

In test two on domain one the RNCB partitioning gives a small reduction in the solution time over that using the PMG decomposition. The PMG partitioning actually proves to be better than that of MRGB since the load imbalance in this case is quite small (+0.9% maximum) and the amount of communication required is smaller. In the second example the PMG decomposition has a much greater imbalance (+5.4% maximum) but manages to out-perform the other partitionings because of its much lower communication requirement. These results show that a reasonable amount of load imbalance is not prohibitive and that the balance in load is not the sole factor here since communication requirements in the solver also make a contribution.

Other experimentation has shown this load imbalance to be more significant as is illustrated below for a mesh of similar size (82376 elements) where the maximum positive deviation is 19% (which is the worst case we have encountered).

Soln. Time		No. of
PMG	MRGB	Its.
159.8	150.6	900

Even though the value of Max N_B in the PMG decomposed mesh is half that of the MRGB figure, there is a large difference in the solution times caused by the imbalance in load. In cases where many more solution iterations are required, for example in non-linear problems, this degradation in performance will become much more apparent and therefore needs to be rectified. We shall discuss this briefly in the next section.

Since with our method the subdomain grids are generated in parallel rather than by generating the whole mesh in serial and then partitioning it, any small increase in the solution time when solving on the PMG decomposition over other partitionings is outweighed by the smaller cost required to generate the distributed mesh. This becomes particularly apparent when larger meshes are created using more processors.

5 Discussion

The method presented in this paper is able to produce large meshes of complex geometries in parallel in which the partitions obtained are suitable for efficient parallel finite element solution. The mesh generation should itself be well load balanced. As we have noted the method may produce partitions in which some subdomains contain many more vertices than the others. Such mesh decompositions may, on occasions, degrade the performance of the parallel solver, as shown in section 4.3, in which case it may be sensible to rebalance the partitions. The same problem of load imbalance occurs in parallel adaptive remeshing codes and a number of repartitioning methods have been proposed to solve this dynamic load balancing problem (e.g., [18, 20, 21, 22]). An additional requirement of such a method is to keep to a minimum the number of elements which need to be moved from one processor to another and hence minimise computation and communication overheads. Since meshes generated with our method are reasonably well partitioned in terms of load balance and communication requirements, any repartitioning will be achievable relatively cheaply. In many circumstances exact load balancing may be accomplished by simple element migration across subdomain boundaries at very little expense.

By meshing individual coarse elements a two level mesh hierarchy is obtained which may be used for preconditioning purposes. This mesh hierarchy allows the problem to be structured in such a way that unstructured domain decomposition preconditioners may be applied in the solution phase. Issues concerning load balancing and rates of convergence of such techniques are at present being addressed.

Overall we conclude that the proposed method performs as well as any decomposition approach that we have considered but has the advantage of allowing the generation of the mesh to be performed in parallel in such a way that the entire mesh never has to be stored on a single processor and without the need for a large amount of interprocessor communication.

Acknowledgements

We thank Chris Walshaw for providing us with his Lanczos code and Peter Dew and Martin Berzins for their valuable suggestions. We also acknowledge the SERC Laboratory at Daresbury for allowing us the continued use of their parallel computing facilities.

The first author acknowledges financial support from SERC in the form of a research studentship.

References

[1] Hodgson, D.C., Jimack, P.K., (1993), "Efficient Mesh Partitioning for Parallel P.D.E. Solvers on Distributed Memory Machines", Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia.

[2] Simon, H.D., (1991), "Partitioning of Unstructured Problems for Parallel Processing", Computing Systems in Engineering, Vol. 2, No. 2/3, 135-148, 1991.

[3] Arthur, T., Bockelie, M.J., (1993), "A Comparison of Using APPL and PVM for a Parallel Implementation of an Unstructured Grid Generation Program", Tech. Report 191425, NASA Computer Sciences Corporation, Hampton, Virginia.

[4] Khan, A.I., Topping, B.H.V., (1991), "Parallel Adaptive Mesh Generation", Computer Systems in Engineering, Vol. 2, No. 1, 75-101.

[5] Topping, B.H.V., Khan, A.I., (1992), "An Optimization-Based Approach to the Domain Decomposition Problem in Parallel Finite Element Analysis", Technical Report, Heriot-Watt University, Edinburgh.

[6] Löhner, R., Camberos, R., Merriam, M., (1992), "Parallel Unstructured Grid Generation", Computer Methods in Apl. Mech. Eng., 95, 343-357.

[7] Weatherill, N.P., Hassan, O., (1992), "Efficient 3D Grid Generation using the Delaunay Triangulation", Comp. Fluid Dynamics '92, 2, 961-968.

[8] Peraire, J., Vahdati, M., Morgan, K., Zeinkiewicz, O.C., (1987), "Adaptive Remeshing for Compressible Flow Computations", Journal of Computational Physics, 72, 449-466.

[9] Barnard, S.T., Simon, H.D., (1992), "A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems", Tech. Report RNR-92-033, NASA Ames Research Center, Moffett Field, CA.

[10] Hendrickson, B., Leland, R., (1993), "A Multi-level Algorithm for Partitioning Graphs", Tech. Report SAND 93-1301, Sandia National Laboratories.

[11] Hendrickson, B., Leland, R., (1992), "An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations", Tech. Report SAND 92-1460, Sandia National Laboratories.

[12] Parlett, B.N., Simon, H.D., Stringer, L., (1982), "Estimating the Largest Eigenvalue with the Lanczos Algorithm", Math. Comp., 38, 153-165.

[13] Leete, C.A., Peyton, B.W., Sincovec, R.F., (1993), "Toward a Parallel Recursive Spectral Bisection Mapping Tool", Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia.

[14] Topping, B.H.V., Khan, A.I., (1993), "Sub-domain Generation Method For Non-Convex Domains", Technical Report, Heriot-Watt University, Edinburgh.

[15] Weatherill, N.P., (1988), "A Method for Generating Irregular Computational Grids in Multiply Connected Planar Domains", Int. J. Numerical Methods in Fluids, 8, 181-197.

[16] Khan, A.I., (1993), "Computational Schemes for Parallel Finite Element Analysis", Ph.D. Thesis, Heriot-Watt University, Edinburgh.

[17] Venkatakrishnan, V., Simon, H.D., Barth, T., (1992), "A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids", The Journal of Supercomputing, Vol. 6, No. 2, 117-127.

[18] Walshaw, C.H., Berzins, M., (1992), "Dynamic Load-Balancing for PDE Solvers on Adaptive Unstructured Meshes", Computer Studies Research 92.32, University of Leeds.

[19] Keyes, D.E., Gropp, W.E., (1987), "A Comparison of Domain Decomposition Techniques for Elliptic Partial Differential Equations and their Parallel Implementation", SIAM J. Sci. Stat. Comput., Vol. 8, No. 2, 167-202.

[20] Van Driessche, R., Roose, D., (1993), "An Improved Spectral Bisection Algorithm and its Application to Dynamic Load Balancing", Technical Report TW 193, Dept. Computer Science, K.U. Leuven, Belgium.

[21] Vidwans, A., Kallinderis, Y., (1993), "A Parallel Dynamic Load Balancing Algorithm for 3-D Adaptive Unstructured Grids", 11th CFD Conference, Orlando.

[22] Özturan, C., deCougny, H.L., Shepard, M.S., Flaherty, J.E., (1993), "Parallel Adaptive Mesh Refinement and Redistribution on Distributed Memory Computers", Technical Report, Rensselaer Polytechnic Institute, Troy, NY.