

A Parallel Generator of Partitioned Unstructured Meshes Integrated with a Parallel Adaptive FE Solver

D.C. Hodgson¹ and P.K. Jimack¹

Abstract

We describe an algorithm for the generation of large unstructured meshes for use in finite element analysis on parallel distributed memory computers (such as the Intel iPSC/860 or the Cray T3D for example). This generation procedure is then coupled with an efficient parallel finite element solution algorithm and a *a posteriori* error estimate, to provide a mechanism for the reliable adaptive and parallel solution of a wide class of elliptic boundary value problems.

1. Introduction

This paper considers the efficient and reliable solution of elliptic partial differential equations using the finite element method on unstructured grids and parallel computer architectures. In this context therefore, we not only wish to make use of adaptivity and a *a posteriori* error estimation to ensure quantifiable accuracy at low computational cost; we also wish to ensure that a high degree of parallelism is built in to all of our algorithms so as to guarantee parallel efficiency and reliability too.

In recent years much research has been done in developing high quality p.d.e. software for serial computation. Typically, such software includes features such as efficient mesh generation and refinement as well as effective preconditioning for the iterative solvers that are used with unstructured grids (see the public domain package PLTMG [2] for example). Unfortunately, many of the advances incorporated into software such as this are yet to be carried over to non-serial computers, such as distributed memory parallel architectures. The majority of recent research into the solution of unstructured finite element problems on distributed memory machines seems mainly to have concentrated on the problem of data partitioning [4,7,9,10,16,19]. For this paper we follow [14] and [15] and take the view that in order to produce efficient and reliable parallel algorithms it is necessary to consider the data partitioning problem *in conjunction with* the issues of parallel mesh generation and adaptivity.

In section 2 below we outline a new algorithm for the parallel generation of unstructured meshes which are suitably partitioned for the immediate application of a parallel finite element solver. Such a solver,

¹School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK.

along with an efficient parallel preconditioner and a *posteriori* error estimator, is then discussed in section 3. The following section then shows how this error estimator may be combined with the mesh generation procedure to produce an appropriately adapted mesh in parallel. The paper then finishes with a simple illustrative example and a brief discussion of this and future work. Throughout the paper we concentrate on the case of problems in two dimensions although most of the ideas carry over to three dimensions with little conceptual difficulty.

2. Parallel Mesh Generation

The requirements of a good parallel unstructured mesh generation procedure for a distributed memory computer can be listed as follows:

1. meshes should be generated in parallel, and be of "high quality",
2. upon completion, each processor should have generated about the same number of nodes and the mesh should be distributed such that the number of nodes lying on the boundary between processors should be as low as possible.

The first of these requirements is self-explanatory, the second relates to being able to obtain an efficient finite element solution on the generated mesh. In order that each processor will have the same computational load we require that the subproblem that each processor works with should be of about the same size. In addition we also wish to minimize the amount of data transfer that will be required between processors during the solution phase of the algorithm: we model this by attempting to minimize the number of nodes on the partition boundary because the local nature of the finite element method implies that it is only data relating to these nodes that will ever need to be communicated.

Our algorithm achieves the above requirements by making use of a coarse background grid which covers the problem domain (or an approximation to it where the boundary region is curved). At each knot point of the background grid we associate a value, known here as a "point distribution value", which defines the desired spacing of the mesh to be generated in the immediate neighbourhood of this point. Using these values it is then possible to estimate how many nodes we expect will be generated along each edge of the background grid and inside each element. Following this a weighted dual graph is produced such that each vertex of this dual graph represents a coarse element and has a weight equal to the number of predicted nodes for that element, and each edge of the graph represents an edge of the background grid and has a weight equal to the number of nodes that will be generated along it. The vertices of this weighted graph are then partitioned between the processors so as to ensure that the sum of the vertex weights on each processor is

about the same, whilst the sum of the weights of those edges joining vertices on different processors is as low as possible. Hence each processor is assigned a number of elements in the coarse grid which it will work with. The finite element mesh itself is now generated in parallel, with each processor using an implementation of Weatherill's algorithm ([17]) for generating a Delaunay mesh on the interior of each coarse element allocated to it during the partitioning stage.

So long as the *a priori* estimates of the number of nodes that are generated in each subdomain (coarse element), and along each coarse edge, are sufficiently accurate, then by partitioning the weighted dual graph of the background grid in the manner outlined in the previous paragraph, we are able to ensure that requirement 2 above is satisfied. In addition, the first part of requirement 1 is also satisfied by this algorithm as each processor will generate an approximately equal contribution to the final mesh in parallel. The use of a Delaunay mesh generator within each subdomain is designed to ensure that if the coarse background grid is of a good quality then so too will be the final mesh. As an extra precaution local mesh smoothing through the use of a Laplacian filter (as in [18]) may also be applied within each subdomain.

Full details of this parallel mesh generation algorithm may be found in [11] or [12], where technical details on the estimation of vertex and edge weights, the partitioning of the weighted dual graph, and the parallel implementation of the procedure are all discussed. We end this section by quoting one of the experimental results from [11], where it is demonstrated that it is possible to generate a mesh of over 800000 elements on an L-shaped domain in under two minutes using eight processors of an Intel iPSC/860 (in fact in [12] efficient results are given for up to 32 processors). Moreover, it is observed that these elements are partitioned in such a way that each processor has an almost equal number of vertices stored upon it, and that the total number of vertices lying on a boundary between processors is generally as low (or lower than) the number obtained using conventional partitioning algorithms.

3. Parallel Solution and Error Estimation

Having generated an unstructured mesh which covers an arbitrary 2-d problem domain, Ω say, we now concentrate on deriving an efficient and reliable parallel finite element algorithm for solving general second order boundary value problems. We therefore consider the following Dirichlet problem whose weak solution, $u \in H_E^1(\Omega)$, satisfies a relationship of the form

$$a(u, v) = (f, v) \quad \forall v \in H_0^1(\Omega), \quad (3.1)$$

where the bilinear form $a(\cdot, \cdot)$ is symmetric and elliptic, (\cdot, \cdot) represents the usual L^2 -inner product and we assume that the essential boundary

condition $u = u_E$ is satisfied on the entire boundary $\partial\Omega$. For the purposes of this paper we only consider using piecewise linear finite elements to solve (3.1) on our unstructured mesh, although the ideas presented here do extend to higher order elements, through the use of hierarchical basis functions for example (see [1] and references therein). This leads to the necessity of solving one or more sparse linear systems of the form

$$K \underline{x} = \underline{b}, \quad (3.2)$$

where K is a symmetric positive definite matrix, \underline{b} is a known vector and \underline{x} is a vector of unknowns which correspond to values at the nodes of the finite element mesh (see [13] for example).

Using piecewise linear elements, if we order the nodes of the finite element mesh starting with all of the nodes generated inside the first coarse element, followed by all of the nodes inside the second coarse element, up to all of the nodes inside the last coarse element, followed by all of the nodes on the boundary between coarse elements (this excludes those on the Dirichlet boundary), then the resulting finite element system, (3.2), takes the block-arrowhead form

$$\begin{bmatrix} A_{II(1)} & & & & A_{IB(1)} \\ & A_{II(2)} & & & A_{IB(2)} \\ & & \ddots & & \vdots \\ & & & A_{II(N)} & A_{IB(N)} \\ A_{IB(1)}^T & A_{IB(2)}^T & \cdots & A_{IB(N)}^T & A_{BB} \end{bmatrix} \begin{bmatrix} \underline{x}_{I(1)} \\ \underline{x}_{I(2)} \\ \vdots \\ \underline{x}_{I(N)} \\ \underline{x}_B \end{bmatrix} = \begin{bmatrix} \underline{b}_{I(1)} \\ \underline{b}_{I(2)} \\ \vdots \\ \underline{b}_{I(N)} \\ \underline{b}_B \end{bmatrix}. \quad (3.3)$$

Here it is assumed that there are N subdomains (coarse elements) and that $\underline{x}_{I(i)}$ represents those unknowns inside subdomain i and \underline{x}_B represents those unknowns on the boundary between subdomains (i.e. the boundary between coarse elements). Moreover, it is possible to compute and assemble each of the blocks $A_{II(i)}$, $A_{IB(i)}$ and $\underline{b}_{I(i)}$ independently on the processor storing subdomain i . In addition, each processor can compute and assemble its own contributions to the remaining block of K , A_{BB} , independently - storing them in the blocks $A_{BB(i)}$ say (hence $A_{BB} = A_{BB(1)} + \dots + A_{BB(N)}$). For the remaining block of \underline{b} , \underline{b}_B , each processor can again compute its own contributions but some communication is required to enable the full \underline{b}_B to be known on each processor.

This completes the first step of the finite element solution procedure: computing and partially assembling the finite element matrices in parallel. The next step is to solve (3.3) in parallel, respecting the fact that the system is stored in a distributed fashion. We do this using the Schur Complement Method (SCM) with a parallel domain decomposition (DD) preconditioner, as explained in [12]. To derive the Schur Complement Method it is noted that, from (3.3),

$$A_{II(i)} \underline{x}_{I(i)} + A_{IB(i)} \underline{x}_B = \underline{b}_{I(i)} \quad \text{for } i = 1, \dots, N \quad (3.4)$$

4. Adaptivity Through Parallel Remeshing

The final stage in our adaptive remeshing algorithm is to make use of the error estimate, derived from our initial finite element solution, to generate a new mesh which will allow a more accurate answer to be obtained. Recall from section 2 that the parallel mesh generation procedure that we use requires a point distribution value to be defined at each knot of the coarse background grid. In the absence of any other information we would usually choose to set each of these values to be the same in the first instance so as to generate a mesh of uniform density everywhere. Now that we do have some information however (in the form of the error estimate η), we would like to use this to influence the point distribution values at each coarse knot point so as to generate a new mesh which is better suited to the solution we are attempting to find.

One of the simplest ways of achieving this is to let the error at each knot point, i say, of the coarse grid be defined to be

$$E_i = \max_{e=1, N(i)} \{ \max_{\Omega_t \subseteq \Omega'_e} \{ \|\eta\|_{\Omega_t, \infty} \} \}, \quad (4.1)$$

where $e = 1, \dots, N(i)$ is an ordering of each of the coarse grid elements, Ω'_e , with knot i as a vertex, and Ω_t is a finite element contained within the subdomain Ω'_e . We can now use an approach similar to that in [14] and define the point distribution value at knot i to be

$$\min \left\{ \text{previous value}, \frac{\alpha}{E_i} \right\}, \quad (4.2)$$

for some suitable choice of scaling factor α . Note that the form of (4.2) ensures that the new mesh, when it is generated, will never be less dense anywhere than the mesh at the previous level, even when the estimated error is extremely small.

Again, this stage of the algorithm can also be efficiently implemented in parallel, with only a small amount of local data transfer required (to determine maximum error values at those coarse knots lying on the boundary between processors). Having obtained these modified point distribution values we may now re-use the procedure described in section 2 to repartition the coarse background grid and generate a new, refined, finite element mesh which is better suited for approximating the true solution. A summary of the complete algorithm is shown below.

1. Generate a coarse background grid and assign a moderate point distribution value to each knot.
2. Partition the background grid across the processors.

3. In parallel: generate a uniform density, unstructured mesh (which will be load-balanced, with a low interpartition boundary length).
4. In parallel: assemble and solve the finite element equations (using the Schur Complement Method with a parallel Domain Decomposition preconditioner).
5. In parallel: estimate the error on each triangle and use this to assign new point distribution values to each coarse knot point.
6. Repartition the background grid across the processors.
7. In parallel: generate a new, refined, unstructured mesh (again load-balanced and with a low interpartition boundary length).
8. In parallel: assemble and solve the finite element equations (as in step 4).
9. In parallel: estimate the new, reduced, error on each triangle.

Note that the only parts of this algorithm that cannot efficiently be implemented in parallel are steps 1, 2 and 6, which relate to the coarse background grid. Since this is always likely to be much smaller than the final mesh upon which we solve the problem, and since we aim to keep the size of this grid independent of the number of processors being used, we can reasonably expect that the algorithm will scale well as the problem size and the number of processors is increased.

In theory it is possible to extend the above algorithm further, turning it into an iterative process. This would mean taking the error estimate computed in step 9 and using it to further modify the point distribution values on the coarse grid. Then, the algorithm would return to step 6 unless some termination criterion was satisfied. We have yet to implement such an iterative approach however since it is clear that it will not work using the current maximum norm and point distribution value formulae given by (4.1) and (4.2). This is because, as the error gets smaller at each iteration so too will the coarse knot error estimates, E_i , and so, by (4.2), the mesh would not be refined further.

5. Discussion

In this section we illustrate the algorithm that has been described by considering a straightforward test problem with a known solution. We then discuss the merits of the solution that we obtain and consider the strengths weaknesses of our approach.

We consider the problem

$$\Delta u = f \quad \text{on the domain } \Omega = (0, 7) \times (0, 2), \quad (5.1)$$

subject to the Dirichlet boundary condition $u|_{\partial\Omega} \equiv 0$, where the choice of f in (5.1) is made so as permit the unique solution

$$u = \tanh(100r - 40) - \tanh(100r - 60), \quad (5.2)$$

with

$$r = \sqrt{(x - 3.5)^2 + (y - 1.0)^2}.$$

Clearly, as can be seen from the contour plot in figure 1, this solution is flat over most of the domain, Ω , but changes very rapidly in some restricted regions. It is in these regions where we would expect the mesh refinement to be greatest in order to ensure that the computational error is small.

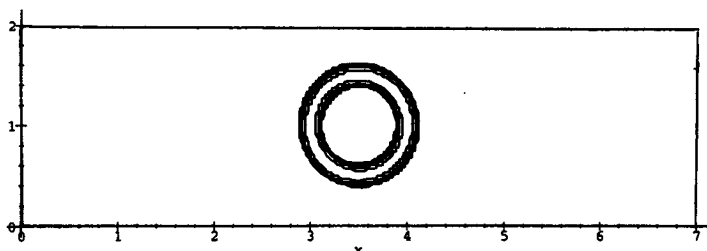


Figure 1: Contour plot of the solution to equation (5.1).

In table 1 below, the parallel solution times are shown for this problem using 1, 2, 4 and 8 processors of an Intel iPSC/860, on a background grid of 440 triangular elements (which is shown in figure 2).

Task	Number of processors			
	1	2	4	8
Gen. & part. uniform mesh	3.036	2.393	2.548	3.320
Sol. + err. est.	11.29	5.777	3.300	2.173
Gen. & part. refined mesh	15.45	9.239	6.565	6.088
Sol. + err. est.	67.94	32.83	17.62	10.25
TOTAL	97.72	50.24	30.03	21.83

Table 1: Parallel solution times (in seconds) on the Intel iPSC/860.

First of all a mesh of uniform density was generated, which contained 4180 elements (see figure 3). Next, the finite element problem was solved on this mesh and an estimate of the error found. This estimate was used to generate a refined mesh of 23886 elements which is shown in figure 4.

As can be seen from this table, the solution and error estimation steps of the algorithm scale well, even when the problem size is fixed, as

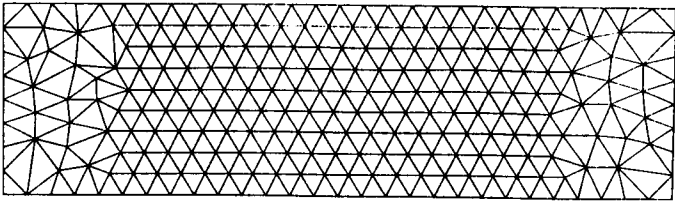


Figure 2: Background mesh used in the solution of equation (5.1).

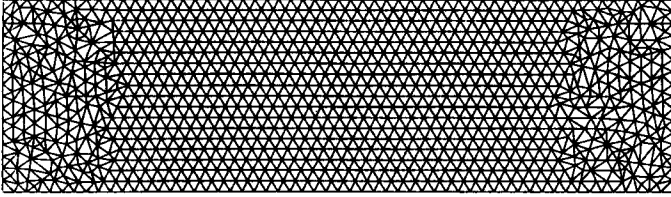


Figure 3: Initial mesh used in the solution of equation (5.1).

in this example. The parts that do not scale so well are those steps which are dominated by the sequential partitioning of the coarse background grid. It should be noted however that when the problem size is allowed to grow with the number of processors this difficulty is significantly reduced (since the coarse grid does not grow with the problem size). Inspection of the meshes shown in figures 2 to 4 show that heavy refinement has indeed taken place in those coarse elements for which the solution u , in (5.2), has large gradients or curvatures. This suggests that for the self-adjoint elliptic problems considered here, we have had some success in achieving our aims of providing reliable and efficient software: both in the sense of achieving automatic mesh refinement and in producing a robust parallel implementation.

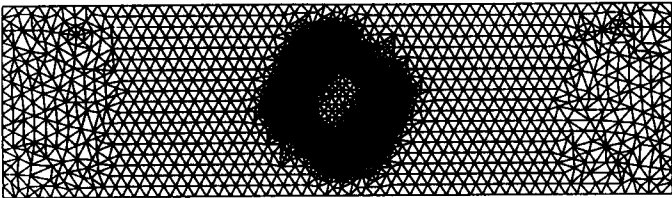


Figure 4: Refined mesh used in the solution of equation (5.1).

There is still a considerable amount of further work to be performed however, especially for non-self-adjoint equations. Here we may have to work further on our error estimation and it will certainly be necessary to make use of alternative iterative solvers and preconditioners. In addition, as already mentioned in section 4, it may well be possible to improve the manner in which we use the error estimate to generate a refined mesh, thus allowing a fully iterative version of the algorithm discussed here to be implemented.

Acknowledgements

Our use of the Intel iPSC/860 parallel computing facility at the EPSRC Daresbury Laboratory throughout this work has been through SERC grant GR/J27066. The first author also acknowledges financial support from the EPSRC in the form of a research studentship.

References

- [1] AINSWORTH, M., (1994), "A preconditioner Based on Domain Decomposition for $h - p$ Finite Element Approximation on Quasi-Uniform Meshes". Mathematics and Computer Science Tech. Report 93/16 (Rev. 4/1994), University of Leicester.
- [2] BANK, R.E., (1990), "PLTMG: A Software Package for Elliptic Partial Differential Equations". SIAM.
- [3] BANK, R.E., WEISER, A., (1985), "Some A Posteriori Error Estimators for Elliptic Partial Differential Equations". *Math. Comp.*, 44, 283-301.
- [4] 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.
- [5] BRAMBLE, J.H., PASCIAK, J.E., SCHATZ, A.H., (1986) "The Construction of Preconditioners for Elliptic Problems by Substructuring, I". *Math. Comp.*, 47, 103-134.
- [6] DRYJA, M., WIDLUND, O.B., (1990), "Some Domain Decomposition Algorithms for Elliptic Problems". In *Iterative Methods for Large Linear Systems*, Academic Press.
- [7] FARHAT, C., LESSOINNE, M., (1993), "Automatic Partitioning of Unstructured Meshes for the Parallel Solution of Problems in Computational Mechanics". *Int. J. Num. Meth. in Eng.*, 36, 745-764.

- [8] GOLUB, G.H., VAN LOAN, C.F., (1989), "Matrix Computations". Johns Hopkins University Press, 2nd edition.
- [9] HENDRICKSON, B., LELAND, R., (1993), "A Multi-level Algorithm for Partitioning Graphs". Tech. Report SAND 93-1301, Sandia National Laboratories.
- [10] 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.
- [11] HODGSON, D.C., JIMACK, P.K., (1994), "Parallel Generation of Partitioned, Unstructured Meshes". School of Computer Studies Research Report 94.19, University of Leeds. Submitted to Computer Systems in Engineering.
- [12] HODGSON, D.C., JIMACK, P.K., (1995), "A Domain Decomposition Preconditioner for a Parallel Finite Element Solver on Distributed Unstructured Grids". School of Computer Studies Research Report 95.1, University of Leeds. Submitted to SIAM J. Sci. Comp.
- [13] JOHNSON, C., (1987), "Numerical Solutions of Partial Differential Equations by the Finite Element Method". CUP.
- [14] KHAN, A.I., TOPPING, B.H.V., (1991), "Parallel Adaptive Mesh Generation". Computer Systems in Engineering, 2, 75-101.
- [15] LÖHNER, R., CAMBEROS, R., MERRIAM, M., (1992), "Parallel Unstructured Grid Generation". Comp. Meth. in Apl. Mech. Eng., 95, 343-357.
- [16] SIMON, H.D., (1991), "Partitioning of Unstructured Problems for Parallel Processing". Computing Systems in Engineering, 2, 135-148.
- [17] WEATHERILL, N.P., HASSAN, O., (1992), "Efficient 3D Grid Generation using the Delaunay Triangulation". CFD '92, 2, 961-968.
- [18] WEATHERILL, N.P., (1988), "A Method for Generating Irregular Computational Grids in Multiply Connected Planar Domains". Int. J. Num. Methods in Fluids, 8, 181-197.
- [19] WILLIAMS, R.D., (1991), "Performance of Dynamic Load Balancing for Unstructured Mesh Calculations". Concurrency: Practice and Experience, 3, 457-481.