Partitioning of Unstructured Problems for Parallel Processing

Horst D. Simon[1]
Applied Research Branch
Numerical Aerodynamic Simulation (NAS) Systems Division
NASA Ames Research Center, Mail Stop T045-1
Moffett Field, CA 94035

March 23, 1994

Abstract

Many large scale computational problems are based on unstructured computational domains. Primary examples are unstructured grid calculations based on finite volume methods in computational fluid dynamics, or structural analysis problems based on finite element approximations. Here we will address the question of how to distribute such unstructured computational domains over a large number of processors in a MIMD machine with distributed memory. A graph theoretical framework for these problems will be established. Based on this framework three decomposition algorithms will be introduced. In particular a new decomposition algorithm will be discussed, which is based on the computation of an eigenvector of the Laplacian matrix associated with the graph. Numerical comparisons on large scale two and three dimensional problems demonstrate the superiority of the new spectral bisection algorithm.

Keywords: graph partitioning, domain decomposition, MIMD machines, substructuring.
AMS Subject Classification 68R10, 05C50, 65F15
CR Subject Classification G.2.2, G.1.3

[1]The author is an employee of Computer Sciences Corporation. This work is supported through NASA Contract NAS 2-12961

1 Introduction

Many large scale computational problems are based on unstructured computational domains. Primary examples are unstructured grid calculations based on finite volume methods in computational fluid dynamics, or structural analysis problems based on finite element approximations. One of the key problems when implementing such large scale unstructured problems on a distributed memory machine is the question of how to partition the underlying computational domain efficiently.

In this paper we investigate three algorithms for the partitioning problem for unstructured domains. All three algorithms considered here are recursive, i.e., the computational domain is subdivided by some strategy into two subdomains, and then the same strategy is applied to the subdomains recursively. In this way a partition into $p = 2^k$ subdomains is obtained after carrying out k of these recursive partitioning steps. The three algorithms considered here thus only differ by the partition strategy of a single domain into two subdomains. The three algorithms are:

- recursive coordinate bisection (RCB)
- recursive graph bisection (RGB)
- recursive spectral bisection (RSB)

The first two algorithms have been used by a number of researchers, in particular RCB is a very intuitive approach, which comes immediately to mind, when considering the partitioning problem. Recursive spectral bisection (RSB) is a very recent development and is based on the graph partition algorithm proposed by Pothen, Simon, and Liou [19]. Our main result is that RSB is a significant improvement over the other two algorithms. Williams [23], who also investigated RSB has arrived at a similar conclusion.

In section 2, we will formulate a general framework for the partitioning problem, based on some graph theoretical notation. In section 3 the three partitioning algorithms will be introduced and other related approaches will be discussed. Their qualitative behavior will be investigated on a sample problem. Section 4 will provide a quantitative comparison of the three algorithms on some large structural analysis and CFD problems. Section 5 will offer some conclusions and possible future directions.

The three algorithms considered are only a small subset of the proposed techniques. Techniques of decomposing a large domain into small subdomains for an easier solution have been investigated for quite some time. In the structural analysis community substructuring is a well known technique, and domain decomposition has a long history in the field of numerical solution of PDE's. A complete survey of these developments is beyond the scope of this paper.

Because of the recent interest in using multiprocessors a large number of different algorithms for the partitioning problem have been investigated. There are a number of other approaches to the partitioning problem, which have been motivated by parallel processors and should be mentioned here. Simulated annealing has been used by Williams [23] and Nour-Omid, Raefsky and Lyzenga [16]. Other algorithms motivated by physical considerations arising from structural analysis are the "peeling" algorithm, and a bisection algorithm based on the centroid of a structure and its principal directions. These have been proposed by Nour-Omid [15]. The Kernighan-Lin algorithm [13], a very popular algorithm for the graph bisection problem, has been applied to structures by Vaughn [21]. Farhat [7] has used a variation of the greedy algorithm to obtain partitions that are suitable for implicit solvers, and an inertia type of algorithm for partitions that are suitable for implicit/explicit solvers where a local operator is factored but the interface problem is treated explicitly. The related problem of mapping an unstructured application to the CM-2 has also been investigated by Farhat [8], and by Hammond and Schreiber [12]. Finally, it should be mentioned that the graph partitioning problem has been studied at length in the context of VLSI layout [4], and that there are some interesting connections to multidimensional scaling in statistics (see e.g. [5]). The applicability of some of these techniques to the partitioning of structures and CFD problems requires further investigation.

2 The Partitioning Problem

What constitutes an "efficient" partition is both problem and machine dependent. Given p, the number of processors, one generally would like to partition the given problem into p subproblems of about equal size (load balancing), and at the same time minimize the amount of communication

between the processors. Minimizing the communication is a function of the both the length of the boundary of the subdomains, as well as of the number of neighboring subdomains. For an explicit algorithm load balancing is probably more important than minimizing communication costs, whereas for an implicit algorithm with higher communication requirements the situation might be reverse.

In this work the target machine is a MIMD machine with a moderate number of parallel processors. Specifically we are considering the Intel iPSC/860 with 128 processors [1]. The target application is an explicit two dimensional Euler solver for unstructured meshes, developed by Barth and Jespersen [2, 3]. With this application/machine combination in mind the partitioning problem can be defined more precisely.

The partitioning problem can be considered as a generalization of the graph bisection problem, which is defined as follows: Given an undirected graph G, with the set of vertices V and the set of edges E, $G = (V, E)$, partition $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, such that

$$|E_c| = \{e | e \in E; e = (v_1, v_2); v_1 \in V_1; v_2 \in V_2\} \tag{1}$$

is minimized, subject to some constraint on the partition. Here we chose $|V_1| = |V_2|$, if $n = |V|$ is even, and $|V_1| = |V_2| - 1$, if n is odd.

The iPSC/860 is based on a hypercube interconnect, and permits the allocation of subcubes with a number of processors $p = 2^k, k = 0, 1, ...7$. By applying suitable algorithms for the graph bisection problem recursively, general algorithms for the partitioning problem for $p = 2^k$ can be obtained.

The assumption that the underlying problem can be expressed as an undirected graph is in no way restrictive. For example, for our target application, the upwind finite-volume flow solver for the Euler equations, proposed and implemented on the Cray-2 by Barth and Jespersen[3], one only has to change from the computational mesh to its dual. In a mesh-vertex scheme, solution variables are associated with each vertex of the mesh and flux computation is performed at edges of the non-overlapping control volumes which surround each vertex. Each control volume consists of a number of triangles, surrounding a vertex. Each edge of the mesh joins a pair of vertices and is associated with one edge of the control volume. In the partitioning which we are planning to use, mesh triangles are assigned to processors. Flux computations are identical to the serial computation, but computed by the

individual processors associated with the triangles. Communication is required along edges, which are shared between adjacent triangles residing in different processors. Hence for the purposes of establishing the partitioning of the problem, i.e. the assignment of triangles to different processor, we consider the dual graph. The triangles of the original mesh are the vertices of the dual graph, and two triangles are considered to be adjacent vertices of the dual graph, if and only if they share an edge in the original mesh. A graph partitioning of the dual graph will thus yield an assignment of triangles to processors. In a similar way most general partitioning problems can be transformed to a graph partitioning problem. The approach used here is thus quite generally applicable.

This relationship between the unstructured mesh and its dual is shown in Figures 1 and 2. Figure 1 shows an unstructured grid for a four element airfoil, which has been generated by D. Jespersen, NASA Ames Research Center. The grid has 6019 vertices, 17473 edges, and four bodies. The dual graph of the four element airfoil problem is shown in Figure 2. It has 11451 vertices (triangles in original grid), 16880 edges (interior edges in original grid), and also four bodies. This example will be used in the next section to illustrate the three different partitioning algorithms.

3 Three Partitioning Algorithms

The general idea behind the three partitioning algorithms is to use an optimal strategy to partition a domain into two subdomains, and then to apply the same algorithm recursively for k steps until $p = 2^k$ subdomains have been obtained. All three algorithms thus only differ in the partitioning strategy for a simple domain into two subdomains.

3.1 Recursive Coordinate Bisection (RCB)

This is probably the easiest algorithm conceptually among the three. It is based on the assumption that along with the set of vertices $V = (v_1, v_2, ...v_n)$, there are also two or three-dimensional coordinates available for the vertices. For each $v_i \in V$ we thus have an associated tuple $v_i = (x_i, y_i)$ or triple $v_i = (x_i, y_i, z_i)$, depending on whether we have a two or three dimensional model. A simple bisection strategy for the domain is then to determine

the coordinate direction of longest expansion of the domain. Without loss of generality, assume that this the x-direction. Then all vertices are sorted according to their x-coordinate. Half of the vertices with small x-coordinates are assigned to one domain, the other half with the large x-coordinates are assigned to the second subdomain. RCB is summarized in table 1.

Table 1: Recursive Coordinate Bisection (RCB).

1)	Determine longest expansion of domain (x, y, or z direction)
2)	Sort vertices according to coordinate in selected direction
3)	Assign half of the vertices to each subdomain
4)	Repeat recursively (divide and conquer)

The partition into eight subdomains, which results when applying RCB three times to the dual of the unstructured grid for the four element airfoil is shown in figure 3. Clearly the subdomains in figure 3 are long and skinny as a result of the very fine grid near to the airfoil. The domains are also disconnected and have a few isolated outlying vertices. This very undesirable property of the partition is a result of the very uneven spacing of the grid points. Also connectivity information does not enter the RCB algorithm at all. Hence one should not expect (in general) that the algorithm creates connected subdomains.

3.2 Recursive Graph Bisection

The weakness of the RCB is that the algorithm does not take advantage of the connectivity information given by the graph. After all, the main goal is to minimize the number of graph edges, which are connecting different subdomains. Thus instead of using the Euclidean distance between vertex coordinates, one should rather consider the graph distance between vertices given by $d(v_i, v_j) = |$ *shortest path connecting* v_i *and* $v_j|$. With this change in metric one can define a new partitioning algorithm, which here is called recursive graph bisection.

First two vertices of maximal or near maximal distance in the graph are determined. Then all other vertices are sorted in order of increasing distance from one of the extremal vertices. Finally vertices are assigned to two subdomains according to the graph distance. The only difficulty is the determina-

tion of the diameter (or at least of a pseudo-diameter) of the graph. However, there exist some very good heuristic algorithms for that purpose. These algorithms are also quite well-known in the structures community, since they can also be used for reducing the storage requirements of sparse matrices in envelope or skyline storage format. Here the SPARSPAK implementation of the reverse Cuthill-McKee (RCM) algorithm is used (see [11]).

The SPARSPAK RCM algorithm first finds two pseudo-peripheral vertices in the graph (i.e. vertices which have a very large distance, but which are not necessarily the pair of vertices with maximum distance). Then starting from one of the vertices, the *root* vertex, a so-called *level structure* is constructed. The level structure is a convenient way of organizing the vertices in the graph in sets of increasing distance from the root. Hence the level structure delivered by the SPARSPAK RCM algorithm forms the basis for the recursive graph bisection algorithm. Half of the vertices, the ones which lie closer to the root are assigned to one subdomain, the remaining vertices to the other subdomain. If we start out with a connected graph then by construction it is guaranteed that at least one of the two subdomains (the one including the root) is connected. RCB is summarized in table 2.

Table 2: Recursive Graph Bisection (RGB).

1)	Use the SPARSPAK RCM algorithm to compute a level structure
2)	Sort vertices according to the RCM level structure
3)	Assign half of the vertices to each subdomain
4)	Repeat recursively (divide and conquer)

Figure 4 shows the result of applying RGB to the airfoil problem. The maximum distance in the graph is apparently from a point in the very fine mesh near the front of the large airfoil (red domain) to somewhere in the fine mesh between the third and fourth airfoil (white domain). Compared to RCB the domains are better connected and somewhat more compact. However, the dark blue domain has a very long boundary, winding around the back of the airfoil configuration, as well as a small disconnected component at trailing edge of the first airfoil. Both phenomena are not desirable. From a qualitative point of view RGB appears to be better than RCB, since the domains maintain better connectivities.

3.3 Recursive Spectral Bisection

The third algorithm to be presented here is of quite a different nature and on first glance considerably less intuitive. The recursive spectral bisection algorithm (RSB) is derived from a graph bisection strategy developed by Pothen, Simon, and Liou [19], which is based on the computation of a specific eigenvector of the Laplacian matrix of the graph G. The *Laplacian matrix* $L(G) = (l_{ij}), i, j = 1....n$ is defined by

$$l_{ij} = \begin{cases} +1 & \text{if } (v_i, v_j) \in E \\ \text{-deg(v}_i\text{)} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

It is easily seen that $L(G) = -D + A$, where A is the adjacency matrix of the graph, and D is the diagonal matrix of vertex degrees. Traditionally spectral properties of the adjacency matrix have been investigated in graph theory. Recently, however, Mohar [14] has gathered convincing evidence that the Laplacian matrix is the more natural object for the study of spectral properties of graphs. One of the reasons is that $L(G)$ is closely related to the Laplacian operator. Specifically, if we consider the standard discrete five point Laplacian on a rectangular grid, then the discrete Laplacian and the Laplacian matrix coincide, if Neumann boundary conditions are imposed. Thus one can consider the Laplacian matrix as a generalization of the discrete Laplacian operator for general graphs. This relationship is explored in more detail in [19].

The Laplacian matrix has a number of intriguing properties, which are just listed here. For details and proof see [14]. First note that the bilinear form associated with the Laplacian matrix can be written as follows:

$$x^t L x = - \sum_{(v,w) \in E} (x_v - x_w)^2. \tag{3}$$

From this it follows that $L(G)$ is negative semidefinite. From the definition of L it also follows that the largest eigenvalue λ_1 is zero, and that the associated eigenvector is $\vec{e}$, the vector of all ones. This is simply a consequence of the particular choice of diagonal elements in $L(G)$. If G is connected then λ_2, the second largest eigenvalue, is negative. The magnitude of λ_2 is a measure of connectivity of the graph or its expansion (see [14] and the references therein).

What is of interest here is the eigenvector $\vec{x}_2$ associated with λ_2. It turns out that this eigenvector gives some directional information on the graph. If the components of $\vec{x}_2$ are associated with the corresponding vertices of the graph, they yield a weighting for the vertices. Differences in this weight give a distance information about the vertices of the graph. Sorting the vertices according to this weight provides then another way of partitioning the graph.

Eigenvectors of the adjacency matrix have been used previously to find graph partitionings, e.g., [20] (for a more detailed survey see [19]). The special properties of $\vec{x}_2$ have been investigated by Fiedler [9, 10]. His work gives most of the theoretical justification of the uses of the second eigenvector of the Laplacian matrix for the partitioning algorithm. Hence this eigenvector is called *Fiedler vector* for short.

The actual computational challenge of the RSB algorithm is the effective computation of the Fiedler vector. Here the Lanczos algorithm (see [18]) is used, since it does not require any manipulation of the Laplacian matrix $L(G)$. All that is needed are matrix vector multiplications with $L(G)$. These can be implemented at no additional storage cost, since the Laplacian matrix directly reflects the structure of the graph. Since only one eigenvector is required a special version of the Lanczos algorithm is employed here, which makes use of rational function approximation in order to compute approximations to λ_2. This technique is discussed in [17]. The algorithm from [17] is used directly with only one simple modification, which avoids the unnecessary computation of approximations to the trivial eigenvector $\vec{x}_1 = \vec{e}$. In summary the recursive spectral bisection algorithm is given in table 3.

Table 3: Recursive Spectral Bisection (RSB).

1)	Compute Fiedler vector for graph using the Lanczos algorithm
2)	Sort vertices according to size of entries in Fiedler vector
3)	Assign half of the vertices to each subdomain
4)	Repeat recursively (divide and conquer)

When RSB is applied to the airfoil problem, the partition in figure 5 is obtained. Figure 5 shows that the domains obtained from RSB are both connected (even though there is no theoretical guarantee for it), and nicely rounded and compact. Visually the result of RSB appears to be the most pleasing partitioning.

Table 4: **Number of edges cut $|E_c|$ on airfoil problem ($|E|$ = 16,880)**

partitions	RCB	RGB	RSB
2	118	175	91
4	296	436	208
8	529	618	299
16	863	950	462
32	1193	1334	743
64	1653	1878	1154
128	2218	2529	1763

4 Comparison of the three algorithms

When comparing qualitatively the partitionings resulting from the three algorithms, we have made the following observations in the previous section: RCB creates long, narrow, and disconnected domains. RGB creates more compact domains, but their boundaries are "fuzzy", and sometimes they are disconnected. RSB creates well balanced, connected domains, which yield a visually most pleasing partitioning.

All three algorithms have been implemented both on a Silicon Graphics 4D/25 and a Cray Y-MP. In terms of execution time, RSB is currently the most demanding. However, all three algorithms deliver partitions in a few seconds time on the Y-MP. This is small compared to both the actual execution time of a flow solver, and also the time required to generate grids. Hence we consider the actual runtime of the three algorithms as a less important issue.

In order to get a better quantitative comparison, we consider the number of edges of the original graph, which are connecting the different subdomains. For brevity these edges in the "cut"-set of the partition will be called E_c. Table 4 below gives the number of edges in E_c for the three different algorithms for the unstructured grid discussed above. The three algorithms also have been applied to a more refined grid for the same problem. The dual graph of this more refined grid, which is used as the basis of our computation has 30,269 vertices and 44,929 vertices. The results for this larger problem are given in table 5.

Obviously RSB yields uniformly the best partition, when counting the

Table 5: **Number of edges cut $|E_c|$ on refined airfoil problem ($|E| =$ 44,929)**

partitions	RCB	RGB	RSB
2	126	390	126
4	336	938	290
8	859	1289	546
16	1433	1775	832
32	1997	2452	1180
64	2634	3251	1793
128	3552	4405	2735

number of edges cut. RGB has the worst performance, which is a reflection of the observation about the long boundaries created by RGB. The more interesting and relevant quantity, however, would be the reduction in execution time observed on a parallel machine, using the different partitions. This work is currently in progress, and results will be reported elsewhere [22]. Initial results from a parallel implementation of the Euler solver [3] on the 128 processor iPSC/860 at NASA Ames Research Center indicate, that the reduction in execution time is directly proportional to the reduction in $|E_c|$. For example, the Euler solver with the RSB partition run on 128 processors about 20% faster than the same code with the RCB partition.

Let us finally consider some large three dimensional structures problems. The three algorithms are also applied to the graphs based on the connectivity of finite element models. The two structures considered here are a model of the Space Shuttle solid rocket motor aft skirt and a model of a car body. Both have been discussed by Deuermeyer in [6]. The shuttle aft skirt problem has 12,598 vertices and 91,961 edges, the car body problem has 45,087 vertices and 163,734 edges. These problems have a much higher connectivity that the unstructured grid problems. The results for partitioning these two problems with the three algorithms are given in tables 6 and 7.

The results in tables 6 and 7 demonstrate that the advantage of recursive spectral bisection is even more pronounced for large three dimensional problems. RSB is in some cases almost twice as good as either RCB or RGB. Furthermore it is somewhat a surprise that RGB is performing quite well on these problems. In contrast to the two dimensional airfoil problem, RGB has

Table 6: **Number of edges cut $|E_c|$ on shuttle aft skirt**

partitions	RCB	RGB	RSB
2	837	607	429
4	4818	2139	700
8	9927	3893	1793
16	13772	5968	4251
32	19728	10160	7237
64	25104	16804	12704
128	31011	24604	19084

Table 7: **Number of edges cut $|E_c|$ on car body**

partitions	RCB	RGB	RSB
2	953	807	651
4	3647	1906	1293
8	5211	3639	2492
16	7694	6294	3514
32	11542	9880	4978
64	17463	14557	7834
128	24279	21925	12255

here an advantage over the coordinate bisection scheme. Figures 6, 7, and 8 depict the partionings obtained from the three algorithms on the shuttle aft skirt. It is important to notice that the considerable quantitative differences in table 6 are *not* obviously visible from the figures. These pictures indicate that our visual perception may be inadequate, and that an automatic partitioning algorithm is a necessity when considering large three dimensional problems.

5 Conclusions

We have introduced recursive spectral bisection (RSB), a new algorithm for unstructured grid decomposition. Recursive spectral bisection shows the best performance on unstructured grid problems and large scale finite element problems, when compared to recursive coordinate or graph bisection. As a measure of efficiency the number of edges in the cut-set has been used. Some initial results on a parallel application are very promising. RSB has demonstrated its great potential when used in connection with an explicit Euler solver on an unstructured grid.

Acknowledgement. I wish to thank Tim Barth, V. Venkatakrishnan, and Steve Hammond (NASA Ames) for discussions about unstructured grid computations, Dawson Deuermeyer (Cray Research) for his collection of large FEM problems, Jeff Hultquist, Creon Levit, and Sam Uselton (NASA Ames) for graphics help, and Alex Pothen (Pennsylvania State University) for ideas about graph algorithms and the Fiedler partition.

References

[1] D. Bailey, E. Barszcz, R. Fatoohi, H. Simon, and S. Weeratunga. Performance results on the intel touchstone gamma prototype. In David W. Walker and Quentin F. Stout, editors, *Proceedings of the Fifth Distributed Memory Computing Conference*, pages 1236 – 1246, IEEE Computer Society Press, Los Alamitos, California, 1990.

[2] T.J. Barth. On unstructured grids and solvers. In *Computational Fluid Dynamics, Lecture Series 1990-03*, Von Karman Institute, Belgium, March 1990.

[3] T.J. Barth and D.C. Jespersen. The design and application of upwind schemes on unstructured meshes. In *Proceedings, 27th Aerospace Sciences Meeting*, January 1989. Paper AIAA 89-0366.

[4] S. N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *J. of Comp. System Sciences*, 28:300 – 343, 1984.

[5] C. Chatfield and A. Collins. *Introduction to Multivariate Analysis.* Chapman and Hall, London, 1980.

[6] D. Deuermeyer. Large-scale solutions in structural analysis. *CRAY Channels*, 12(1):15 – 17, 1990.

[7] C. Farhat. On the mapping of massively parallel processors onto finite element graphs. *Computers and Structures*, 32(2):347 – 353, 1989.

[8] C. Farhat, N. Sobh, and K. C. Park. Transient finite element computations on 65536 processors: the connection machine. *Int. J. Num. Meth. Eng.*, 30:27 – 55, 1990.

[9] M. Fiedler. Eigenvectors of acyclic matrices. *Czechoslovak Math. J.*, 25(100):607 – 618, 1975.

[10] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Math. J.*, 25(100):619 – 633, 1975.

[11] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice Hall, Englewood Cliffs, 1981.

[12] S. Hammond and R. Schreiber. *Mapping Unstructured Grid Problems to the Connection Machine.* Technical Report 90.22, RIACS, NASA Ames Research Center, Moffett Field, CA 94035, October 1990.

[13] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical J.*, 49:291–307, 1970.

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

[15] B. Nour-Omid. 1990. (private communication).

[16] B. Nour-Omid, A. Raefsky, and G. Lyzenga. Solving finite element equations on concurrent computers. In A. K. Noor, editor, *Parallel Computations and their Impact on Mechanics*, pages 209 – 227, American Soc. of Mech. Eng., New York, 1986.

[17] B. Parlett, H. Simon, and L. Stringer. Estimating the largest eigenvalue with the Lanczos algorithm. *Math. Comp.*, 38:153 – 165, 1982.

[18] B.N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice Hall, Englewood Cliffs, New Jersey, 1980.

[19] A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430 – 452, 1990.

[20] D. Powers. Graph partitioning by eigenvectors. *Lin. Alg. Appl.*, 101:121 – 133, 1988.

[21] C. Vaughn. Structural analysis on massively parallel computers. In *Proceedings of the Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications*, Pergammon Press (to appear), 1991.

[22] V. Venkatakrishnan, H. Simon, and T. Barth. *A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids.* Technical Report RNR-91-xx, NASA Ames Research Center, Moffett Field, CA 94035, 1991. (in preparation).

[23] R. D. Williams. *Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations.* Technical Report C3P913, California Institute of Technology, Pasadena, Califronia, June 1990.