Numerical Experiences with Partitioning of Unstructured Meshes

Y. F. Hu and R. J. Blake

Daresbury Laboratory, SERC, Warrington WA4 4AD, UK

E-mail: yfh@uk.ac.dl.cxa, ga@uk.ac.dl.cxa. Fax: 0925 603634

Abstract

The problem of partitioning unstructured meshes for parallel computing is considered. Four existing algorithms as well as a new hybrid algorithm are tested. The numerical results favour the recursive spectral bisection algorithm and the hybrid algorithm. A generalisation of the recursive spectral bisection algorithm to deal with an arbitrary number of processors is discussed

Keywords Partitioning of grids; communication cost; grid partitioning algorithms.

1. Introduction

The numerical solution of partial differential equations usually involves dividing up the physical domain into small elements or volumes. To solve the problem on a distributed memory parallel computer, the mesh should be further decomposed into subdomains, the number of which usually corresponds to the number of processors, and the problem on each subdomain is then assigned to a unique processor.

The problem now is how to decompose the mesh into subdomains so that each processor has about the same number of elements (load balancing) and the communication costs between processors are minimized.

For a 2-D problem the need for communication arises because the solution within a given element requires information from neighboring elements that share edges, or perhaps points. Thus two processors need to communicate with each other if their subdomains share edges or points.

To simplify the problem it is assumed that the time taken to send data from one processor to any of the others depends only on the quantity of the data. The problem is then equivalent to partitioning the communication graph of the mesh into subdomains of roughly equal size such that the partition cuts the least number of edges of the graph. Assuming that only elements sharing edges with each other need to communicate, then the graph to be partitioned is the dual graph, otherwise if elements sharing edges as well as points need to communicate, then the graph is the true communication graph. Figure 1 shows a mesh, its dual graph and its true communication graph.

When there are only two processors, the graph partitioning problem becomes a graph bisection problem, where a graph $G = (V, E)$ is given with vertices V and edges E, it is required to find a partition $V = V_1 \bigcup V_2$ such that $V_1 \bigcap V_2 = \emptyset$, $|V_1| \simeq |V_2|$ and the number of cut edges $|E_c|$:

$$|E_c| = |\{h \mid h \in E; h = (v_1\ v_2);\ v_1 \in V_1,\ v_2 \in V_2\}|$$

is minimized. Here for a set S, $|S|$ is the number of elements in the set.

The graph bisection problem has been studied before by many authors (see, e.g., [4], [5]) in the context of graph theory as well as VLSI layout. The advance in parallel computing renews the interest in the problem, making it now a very active subject of research. The graph bisection problem is found to be an NP hard problem, so there are no known algorithms that can find the exact solution to the problem in polynomial time. Therefore most of the graph bisection methods seek an approximation to the optimal partition.

To partition a graph into more than two subgraphs, most of the graph partitioning algorithms recursively bisect the graph into $2, 4, \ldots, 2^d$ subdomains of approximately equal size. In this paper four recursive bisection algorithms are studied, including: the recursive coordinate bisection (RCB) (see, e.g., [8]), the recursive graph bisection (RGB) (see, e.g., [8]), the recursive spectral bisection (RSB) (see, e.g., [6], [8], [11]) and a heuristic algorithm MINCUT (see, e.g., [7]). A hybrid algorithm which combines the RGB and MINCUT algorithm is proposed and tested. Numerical results show that among the four existing algorithms, the RSB algorithm is the best. The hybrid algorithm is also found to be very competitive.

In Section 2 the four algorithms are briefly introduced. In Section 3 the performance of the algorithms is compared on some mesh examples and the hybrid algorithm is introduced. Section 4 summarises the numerical findings and discuss an extention of the algorithms to arbitrary number of subdomains.

2. Graph Partitioning Algorithms

Recursive coordinate bisection (RCB) algorithm (see, e.g., [8]) is the most intuitive graph partitioning algorithm. Since in practice all elements are within a physical domain and have some coordinates associated with them, the RCB algorithm partitions the communication according to the coordinates. In the 2-D case it divides the graph into two halves according to the x-coordinates of the vertices, then further into four according to the y-coordinates, etc.. This works well when the mesh is evenly spread over a simple domain. Otherwise it can create long boundaries with disconnected subdomains. Figure 2 shows the result of applying the RCB to a mesh with 788 elements. It creates 142 interboundary edges.

One of the reasons for the inefficiency of RCB is that it does not use any connectivity information about the graph. The *recursive graph bisection* (RGB) algorithm (see, e.g., [8]) tries to remedy this. Assume that the graph is connected and define the distance between two vertices as the length of the shortest path connecting the two. The RGB algorithm first finds the two vertices that are furthest apart. Then, starting from one (the root) of the vertices, the half of the vertices that are closer to the root forms one subdomain, the rest forms the other. This process is then recursively executed on the subdomains. Figure 3 shows the result of applying the RGB algorithm to the 788 element mesh, resulting in 142 shared edges.

The *recursive spectral bisection* (RSB) algorithm (see, e.g., [6], [8], [11]) is based on the following consideration. Assume that there are n vertices in the graph numbered $1, 2, \dots, n$. Each vertex of the graph is assigned a value of either 1 or -1, and the value assigned to vertex i is denoted by x_i. This creates a bisection of the graph in which the vertices having value 1 form one subdomain and those with value -1 form the other. Clearly the number of shared or cut edges resulting from the bisection is

$$|E_c| = \frac{1}{4} \sum_{i \leftrightarrow j} (x_i - x_j)^2, \tag{1}$$

where $i \leftrightarrow j$ means that there is an edge connecting the vertices i and j. In order to keep the load balanced each of the two subdomains is required to have the same number of vertices. Thus the sum of all the values associated with the vertices is zero, that is

$$\sum_{i=1}^{n} x_i = 0. \tag{2}$$

The problem of minimising the communication cost for the bisection is equivalent to minimising (1) subject to (2) with x_i taking the value of either 1 or -1. This is an integer programming problem which is difficult to solve when the number of vertices n is large. Ignoring the integer constraints, the quadratic (1) can just be minimised subject to the linear constraint (2). This gives a continuous minimisation problem, but without an extra constraint the solution is simply the zero vector. Remembering that when all the variables takes the value 1 or -1, the sum of the squares should be n, the number of vertices giving the extra constraint

$$\sum_{i=1}^{n} x_i^2 = n. \tag{3}$$

The quadratic (1) can now be minimised subject to (2) and (3).

The quadratic (1) can be written as

$$|E_c| = \frac{1}{4} x^T L x, \tag{4}$$

where $x = (x_i)$ is the vector composed of all the values to be assigned to vertices and L is an $n \times n$ matrix known as the Laplacian matrix of the graph. It has the simple form

$$L_{ij} = \begin{cases} -1, & \text{if } \; i \neq j, \; i \leftrightarrow j, \\ \deg(i), & \text{if } \; i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Here $\deg(i)$ is the degree of the vertex i, defined as the number of edges connected with the vertex i. The matrix L satisfies $Le = 0$ with e the vector of all ones. Applying the necessary condition for the constrained minimisation problem gives

$$Lx = \mu e + \lambda x, \tag{5}$$

with μ and λ two Lagrange multipliers to be decided. Multiplying both sides of (5) with e^T gives $\mu = 0$. Thus x has to be an eigenvector of L and the number of cut edges is $|E_c| = \frac{1}{4} n\lambda$. In order to minimise the number of cut edges, x

needs to be the eigenvector of the smallest eigenvalue of L that satisfies the two constraints (2) and (3).

The matrix L is positive semi-definite with smallest eigenvalue $\lambda_1 = 0$ and corresponding eigenvector e. Clearly e is not a solution to the problem since it does not satisfy the load balancing constraint (2). If the graph is connected, then it can be shown that the next smallest eigenvalue λ_2 is positive (see [5]). Any eigenvector associated with this eigenvalue satisfies (2) because it is orthogonal to the first eigenvector e. It will also satisfy (3) by proper scaling, thus it will be the solution of the constrained minimisation problem. This eigenvector gives a value for each vertex of the graph, and the graph can be bisected by separating those with smaller values from those with greater values. The procedure can then be repeated on the subdomains. This gives the RSB algorithm. In practice this algorithm almost always gives connected subdomains if the original graph is connected, however there can be exceptions (see Section 3 for an explanation). Figure 4 shows the result of applying the RSB algorithm to the 788 element mesh, the partition has 108 shared edges. In this case all the subdomains are connected.

The *MINCUT* algorithm (see, e.g., [7]) is based on a heuristic of Kernighan and Lin ([4]). It is again a recursive bisection algorithm. Starting with a load balanced bisection, it first calculates a table recording the changes in the number of edges cut for each vertex that results from moving that vertex from one half of the graph to the other. Then each iteration it moves the vertex which will reduce most the number of cut edges from the side in surplus to the side in deficit. This vertex is then locked and the table recording the changes of the number of edges cut is updated. The procedure is repeated until the rest of the unlocked vertices are locked. The iterative procedure is restarted with the bisection corresponding to the largest reduction in the number of edges cut. Should the iterative process not result in any reductions then the algorithm is terminated and the resulting subdomains are further bisected in the same way. The initial bisections are generated randomly and the final result is very dependent on the initial choice. Figure 5 shows the best result among 3 runs of the algorithm on the 788 element mesh, it has 133 inter boundary edges.

3. Numerical Results

The above four algorithms have been implemented on a Convex C220 computer in Fortran. Before presenting the results some details of the implementation of the RGB and RSB algorithms are given.

For the recursive graph bisection (RGB) algorithm, it is necessary to find two vertices of a graph with the largest distance between them. This is done approximately by picking any vertex first as the root, and assigning its neighbour vertices as level 1, then the neighbours of neighbours as level 2, etc., until every vertex is assigned a level value. Taking the vertex with the greatest level value (denote the value as l_{max}) as a new root, the procedure is repeated until l_{max} does not increase. The last root is taken as the root for the RGB algorithm and the vertices are bisected according to their level values relative to the root. If the graph is not connected, then when the level sets are assigned, at some stage it will be found that all the neighbours of a vertex have already been assigned level values, but the vertices of the graph have not been exhausted. An artificial edge is created linking an assigned vertex with a vertex that has not been assigned a level value. In this way the graph will finally be connected. Of course a cut through the artificial edges is not counted in the edges cut $|E_c|$.

For the recursive spectral bisection (RSB) algorithm, it is necessary to find the second smallest eigenvalue and eigenvector of the matrix L. This is done using the Lanczos algorithm, which is an iterative process. Each iteration involves multiplication of L with a vector. Because of the sparsity of the matrix L, this multiplication can be done cheaply in about $O(mn)$ operations with m the largest degree of the vertices. Since the smallest eigenvalue of L is known to be 0 with e the associated eigenvector, this information can be used to speed up the algorithm. Following Pothen, Simon and Liou ([6]), the vector e is used to deflate the Lanczos algorithm by orthogonalizing the Lanczos vectors against e. Note that, theoretically, as long as the initial Lanczos vector is orthogonal to e, all the subsequent Lanczos vectors will be orthogonal to e too, and our numerical experience does not show any exception. However to be safe, in all the subsequent numerical results the algorithm is implemented with orthogonalization at each iteration. The maximum iteration number for the Lanczos algorithm is set to be 500, and the stopping criterion for the algorithm is set to be $\|Lx - \lambda x\| < \epsilon$ with $\epsilon = 10^{-6}$ and λ, x the current estimates of the second smallest eigenvalue and the normalized eigenvector. The Lanczos algorithm is coded in double precision.

When the graph is disconnected, the second eigenvector may not be a good separator for the graph for the following reasons. Suppose the graph G has two disconnected components G_a and G_b, then by a suitable change of ordering, the matrix L can be written as a block diagonal matrix with two diagonal elements L_a and L_b the Laplacain matrices of G_a and G_b. Both matrices have 0 as their smallest eigenvalues. Assume λ_a, λ_b and x_a, x_b are their second smallest eigenvalues and eigenvectors respectively, and λ_a is smaller than λ_b, then $x = (x_a, 0)^T$ is the eigenvector associated with the second smallest eigenvalue of L. Clearly x is not a good separator for partitioning the graph because of all the zero elements in it. For practical problems the physical domain considered can indeed be disconnected, and even if the initial graph is connected, the subgraphs resulting from bisections can still be disconnected. Figure 1 (b) is a simple example with 4 vertices, any load balanced bisection of the graph will have to create disconnected subgraphs. In addition, since the Lanczos algorithm is an iterative process, disconnected subgraphs can also be created if the solution is not a good approximation of the actual eigenvector.

Therefore before starting the Lanczos algorithm a routine which ensures the connectivity of the graph is executed. This is done in a similar manner to the RGB algorithm.

The four algorithms have been tested on 4 meshes, two of them (with 788 and 3564 elements) are generated from a package DIME ([12]), the 771 mesh is supplied by C. Walshaw ([10]) and the 5520 mesh is from DYNA3D ([1]). The first three are 2-D meshes and the last is a 3-D mesh. For each mesh, both its dual graph as well as its true communication graph (this is not available for the 771 mesh) are partitioned. The graphs are partitioned into 16 subdomains. Table 1 reports the edges cut $|E_c|$ as well as the CPU time (in seconds) for each algorithm on the 7 problems. Here $|E_c|$ is taken to be the edges cut in the dual graph by the partition, in other words the number of shared edges (shared surfaces in 3-D cases) between subdomains, so as to have a unique index to compare the results of partitioning using dual graphs and using true communication graphs. Of course in doing so we bear in mind that this is more to the advantage of the results using the dual graph.

The MINCUT is sensitive to the randomly generated initial bisections. The algorithm was run for three times and the best edges cut and the total CPU time are reported.

From Table 1 it is clear that the simple recursive coordinate bisection (RCB) algorithm takes little time, but the numbers of edges cut is relatively high compared with other algorithms. The recursive graph bisection (RGB) algorithm takes the least time, it is comparable with or even worse than RCB on 2-D problems, but much better on 3-D problems, which confirms an observation of Simon ([6]). The MINCUT gives very competitive numbers of edges cut, but suffers from very long execution times. The recursive spectral bisection (RSB) algorithm gives very good results on edges cut in reasonable times. Therefore comparatively this is the best algorithm of the four.

The edges cut for the partitions using the dual graph and using the true communication graphs do not show much difference. For most algorithms using the true communication graphs gives much higher CPU time. Therefore it may be preferable to use dual graphs for partitioning. Incidentally for recursive spectral bisection (RSB), dual graphs usually take more iterations for the Lanczos algorithm to converge, which accounts for the reverse order of CPU time for the algorithm on the 788 mesh.

The very competitive results on edges cut for the MINCUT suggest the possibility of trying to modify the algorithm to reduce the CPU time. Randomly generated initial bisections have very high numbers of edges cut, and the MINCUT algorithm takes many iteration to reduce the number of edges cut, which results in long CPU times. It is natural to try to start with a reasonably good initial partition and to improve upon it using MINCUT. On the other hand this initial partition needs to be generated quickly, otherwise the algorithm still can not compete with the RSB algorithm. From Table 1 it is clear that it takes very little time to bisect graphs by finding the root and diameter of a graph, as RGB does. Thus at each recursive step, a sensible strategy would be to find first a bisection of the graph using the RGB algorithm, then improve upon it using the MINCUT algorithm. The resulting algorithm is called MINGRAPH. Figure 6 shows the results of using the algorithm on the 788 mesh, resulting in 116 shared edges. The full results for the hybrid algorithm are reported in the last column of Table 1. The MINGRAPH algorithm is better than the recursive spectral bisection algorithm in terms of the number of edges cut in 6 out of the 7 cases and is faster in 5 out of the 7 cases.

4. Discussions

In this paper four graph partitioning algorithms as well as a hybrid algorithm has been tested, the recursive spectral bisection and the hybrid algorithm are found to be more efficient than the others.

All the algorithms discussed here are recursive bisection algorithms. However two recursive optimal bisections do not necessarily generate an optimal quadrisection. Direct quadrisection even octasection using two or three eigenvectors of the Laplacian matrix are being explored ([2]). For quadrisection, instead of associating each vertex i of a graph with a scalar, it can be associated with a vector $(x_i, y_i)^T$ with x_i and y_i taking the values 1 or -1 and producing four different vectors. For a quadrisection, the communication cost to be minimised is found to be $\frac{1}{4}(x^T Lx + y^T Ly)$, the problem can be made continuous by adding the constraints $e^T x = 0$, $e^T y = 0$ (load balancing); $x^T x = n$, $y^T y = n$ and $x^T y = 0$. By applying the necessary condition of constrained minimisation and some manipulation of linear algebra it is possible to show that the minimum is reached if the $n \times 2$ matrix (x, y) is an orthogonal transformation of the matrix composed of the second and third smallest normalized eigenvectors of the Laplacian. The vectors x and y can then be used as separators for partitioning in a suitable way ([2]).

It has been assumed that the data communication times between processors are the same, yet this is generally not true. Heterogenous data transmission costs can be accommodated by first of all finding a partition, then each subgraph is assigned to an initial processor (a colour) and the colouring is changed in a systematic way to reduce the communication cost (see [7] for some discussion). However on in many cases the difference in communication times are not important compared with the startup costs and the costs for the message passing itself ([9], [3]).

Since in practice the mesh is usually changed adaptively according to some gradients, it is also necessary to look at efficient dynamic graph partitioning algorithms, some possibilities have been suggested ([10]).

Once the partitioning is done, it is necessary to give a scheme for optimising the scheduling of message passing. This problem will be discussed in the accompanying paper [3].

Finally, all the five algorithms discussed in Sections 1 to 3 partition the meshes into numbers of subdomains which are powers of two. In the following a procedure is suggested for the case when the number of processors available is not a power of two. The same process could equally well be used to partition grids into heterogeneous systems where the nodes have different speeds.

In this case a modification to the algorithms is needed, that is, instead of the mesh being partitioned into equal halves each iteration, it should be partitioned into two parts with appropriate numbers of elements in each parts. For example, partitioning a mesh of 788 elements into 15 subdomains could result in the following allocations of numbers of elements

$$\overbrace{53,\ \ldots,\ 53}^{8},\ \overbrace{52,\ \ldots,\ 52}^{7}. \tag{6}$$

In the first iteration the 788 elements are divided into two parts, the first part having 7 portions, i.e. $53 \times 7 = 371$ elements, the second part having the 8 remaining portions, i.e. $53 + 52 \times 7 = 417$ elements. Next time the 371 elements will be further divided into $53 \times 3 = 159$ elements and $53 \times 4 = 212$ elements; the 417 elements be divided into $53 + 52 \times 3 = 209$ elements and $52 \times 4 = 208$ elements, and so on. This procedure is illustrated in Figure 7.

All the five algorithms can be used to partition a mesh into two parts with unequal numbers of elements in each parts. Take the recursive spectral bisection algorithm for example. It was shown in Section 2 that if the mesh is to be partitioned into two equal halves, then the RSB algorithm can be motivated by minimising (4) subject to the constrains (2) and (3). In order to partition n elements into two parts with one having αn $(0 < \alpha < 1)$ elements and the other having $(1-\alpha)n$ elements, each element is assigned with a value of either 1 or -1. The problem of minimising interboundary edges becomes that of minimising (4) subject to constraints (3) and

$$\sum_{i=1}^{n} x_i = \alpha n - (1-\alpha)n = (2\alpha - 1)n. \tag{7}$$

Here x_i is either 1 or -1. If the integer requirement is relaxed and replaced by (3) instead, then the necessary condition for the constrained minimisation problem gives

$$Lx = \mu e + \lambda x. \tag{8}$$

Multiplying both sides of (8) with e^T gives

$$\mu = (1 - 2\alpha)\lambda.$$

Since $Le = 0$, defining $y = x + (1 - 2\alpha)e$, (8) can also be written as

$$Ly = \lambda y.$$

The number of cut edges can equally be written as

$$\begin{aligned} |E_c| &= \frac{1}{4} x^T L x \\ &= \frac{1}{4} y^T L y \\ &= \frac{1}{4} \lambda \|y\|^2 \\ &= \alpha(1 - \alpha) n \lambda, \end{aligned}$$

which for $\alpha = \frac{1}{2}$ reduces to the standard bisection result discussed in Section 2. Thus using similar arguments to those presented in Section 2, the vector $y = x + (1 - 2\alpha)e$ should be the second smallest eigenvector of L. Once the vector y has been calculated, the mesh can be partitioned into two unequal parts according to the vector $x = y - (1 - 2\alpha)e$, or simply according to y since each element of $(1 - 2\alpha)e$ is the same.

In Table 2 the results of using the RSB algorithm to partition the dual graph of the mesh with 788 elements into various numbers of subdomains are listed. In general the number of shared edges increases about linearly to sublinearly with the number of subdomains, as expected.

References

[1] J. O. Hallquist and D. J. Benson, DYNA3D Users's Manual (nonlinear dynamic analysis of structures in three dimensions), Tech. Rep. UCID-19592, Rev. 3, University of California, Lawrence Livermore National Laboratory, 1987.

[2] B. Hendrickson and R. Leland, An improved graph partitioning algorithm for mapping parallel computations, Sandia National Laboratories, Albuquerque, NM 87185, 1992.

[3] Y .F. Hu and R. Blake, Some algorithms for the scheduling of message passing, November 1992 (submitted to Parallel Computing).

[4] B. W. Kernighan and S. Lin, an efficient heuristic procedure for partitioning graphs, Bell Systems Tech. J. 49 (2) (1970) 291-308.

[5] B. Mohar, The Laplacian spectrum of graphs, technical Report, Department of Mathematics, University of Ljubljana, Ljubljana, Yugoslavia, 1988.

[6] A. Pothen, D. H. Simon and K. P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM J. Matrix Anal. Appl. 11 (1990) 430-452.

[7] P. Sadayappan, F.Ercal and J. Ramanujam, Cluster partitioning approach to mapping parallel programs onto a hypercube, Parallel Computing 13 (1990) 1-16.

[8] H. D. Simon, Partitioning of unstructured problems for parallel processing, Computer Systems in Engineering, 2 (1991) 135-148.

[9] V. Venkatakrishnan, H. D. Simon and T. J. Barth, A MIMD implementation of a parallel Euler solver for unstructured Grids, Report RNR-91-024, NAS Systems Division, Applied Research Branch, NASA Ames Research Center, 1991.

[10] C. Walshaw and M. Berzins, Dynamic load-balancing for PDE solvers using adaptive unstructured meshes, School of Computer Studies,

University of Leeds, 1992.

[11] R. D. Williams, Performance of dynamic load balancing algorithms for unstructured mesh calculations, Concurrency: Practice and Experience, 3 (1991) 457-481.

[12] R. D. Williams, DIME: A user's Manual, Caltech Concurrent Computation Report C3P 861, 1990.

Table 1 comparison of five partitioning algorithms*

Graphs	RCB	RGB	RSB	MINCUT	MINGRAPH
Dual−788	142/3	142/1	108/20	133/26	116/4
True−788	142/4	143/3	117/18	97/55	104/10
Dual−771	232/1	201/1	150/14	188/26	148/4
Dual−3564	293/43	334/3	261/154	500/697	233/73
True−3564	293/51	359/13	288/162	263/1096	243/181
Dual−5520	2876/110	1737/8	1000/264	896/1702	822/302
True−5520	2876/126	1117/26	987/489	903/2648	861/270

* The result is in the form $|E_c|$ / CPU time (in seconds).

Table 2 partitioning using RSB: the number of shared edges $|E_c|$ against the number of processors $nprocessor$

$nprocessor$	$\|E_c\|$	$nprocessor$	$\|E_c\|$
2	29	18	129
3	39	19	128
4	43	20	136
5	57	21	144
6	65	22	152
7	60	23	154
8	69	24	165
9	73	25	165
10	78	32	181
11	89	48	236
12	91	64	282
13	96	80	334
14	105	96	358
15	118	112	410
16	108	128	433
17	121		