CONCURRENCY: PRACTICE AND EXPERIENCE, VOL. 7(1), 17–28 (FEBRUARY 1995)

Dynamic load-balancing for PDE solvers on adaptive unstructured meshes

CHRIS WALSHAW* AND MARTIN BERZINS

School of Computer Studies
University of Leeds
Leeds, UK

SUMMARY

Modern PDE solvers written for time-dependent problems increasingly employ adaptive unstructured meshes (Flaherty *et al.*, 1989) in order to both increase efficiency and control the numerical error. If a distributed memory parallel computer is to be used, there arises the significant problem of dividing the domain equally amongst the processors whilst minimising the inter-subdomain dependencies. A number of graph-based algorithms have recently been proposed for steady-state calculations. The paper considers an extension to such methods which renders them more suitable for time-dependent problems in which the mesh may be changed frequently.

1. INTRODUCTION

Modern PDE solvers for time-dependent applications are currently being written to solve real-life problems on complex spatial domains to an accuracy specified by the user[1]. The desire to compute a solution to a certain accuracy over a complex domain has led to many codes using unstructured meshes with mesh adaptivity controlled by the spatial error estimate.

This desire to control the spatial error means that the position and density of the spatial mesh points may vary dramatically over the course of an integration. This refinement and coarsening is undertaken automatically[1,2]. As an example, taken from the wedge shock problem discussed in Section 5.3, Figure 3 shows the solution domain after the first and final remeshes.

Parallel versions of such codes face the problem of distributing the mesh. For optimal performance the load should be evenly balanced and the communication cost reduced as much as possible by minimising inter-processor dependencies. It is well known that this mapping problem is NP-complete[3], and so heuristics must be employed to obtain a usable algorithm. In addition, for time-dependent problems, the unstructured mesh may be modified every few time-steps and so the load-balancing must have a low cost relative to that of the solution algorithm in between remeshing.

*Present address: School of Maths, Stats and Computing, The University of Greenwich, 6 Wellington Street, London, SE18 6PF.

CCC 1040–3108/95/010017–12

Received 17 February 1993
Accepted 14 March 1994

A number of good load-balancing algorithms (see, for example, References 4 and 5) are based on partitioning a graph that corresponds to the communication requirements of the unstructured mesh. Until now, such algorithms have not addressed the incremental update partitioning problem posed when a mesh with an existing partition is being refined and/or coarsened. The aim of this paper is to propose a new method for this update problem which may be used *in conjunction with* existing graph-based partitioning techniques.

Of the existing partitioning techniques recursive spectral bisection is generally highly regarded[4,5], and an improved version allowing for quadrisection and even octasection has been devised[3]. The spectral algorithm forms a natural starting point for the work presented here and is described in full in Section 2. The limitations of the algorithm for time-dependent adaptive mesh codes are considered in Section 3. In Section 4 a pre-processing step for the algorithm is introduced which addresses these limitations and appears to provide faster, more efficient dynamic load-balancing. In Section 5 a comparison is made between the algorithms and illustrated by some results from two PDE problems. Finally a few future directions for research are offered.

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

2. RECURSIVE SPECTRAL BISECTION

The recursive spectral bisection algorithm (henceforth RSB) is one of a family of recursive bisection methods used for partitioning a graph. Common to these methods is the idea that bisecting a domain is a much easier task than subdividing into p subdomains. The bisection is obtained by a given strategy and then the same strategy is applied to the subdomains recursively. In this manner a partition into $p=2^q$ subdomains can be obtained in q recursive steps. Two other examples are recursive co-ordinate bisection (RCB; see Section 5.2) and recursive graph bisection (RGB). In his paper[4], Horst Simon describes all three algorithms and demonstrates the superiority of RSB over the other two.

2.1. Recursive spectral bisection—motivation

The spectral bisection algorithm is one of a number of methods which partition a graph derived from the mesh. The fundamental idea is to associate the nodes of an undirected graph with variables in the solution vector. The spatial discretisation defines dependencies between solution variables which can then be represented by edges, hence giving rise to a *dual communication graph*. Thus for triangular cell-centred 2D finite volume calculations, for example, each node of the graph represents a triangle and has three edges connecting it to the nodes of the three adjacent triangles.

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

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

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

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

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

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

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

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

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

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

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

Note that the Laplacian has some interesting properties which are detailed more fully in Reference 6. It is easily verified that zero is an eigenvalue, λ_1 say, of $\mathbf{L}$ with associated eigenvector $\mathbf{e}$, where $e_i=1$ for $i=1,\ldots,n$. Thus $\mathbf{L(G)}$ is positive semi-definite and hence $\lambda_1=0$ is the smallest eigenvalue. If $\mathbf{G}$ is connected, then λ_2, the second smallest eigenvalue, is strictly positive (again see Reference 6).

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

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

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

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

Figure 1. The recursive spectral bisection algorithm

This vector $\mathbf{x}_2$, now renders a weighting, x_v, for each vertex of the graph v. Because of the continuous approximation, in general the weightings will not be the $+1$ or -1 initially required. However, this heuristic is not restrictive. Eigenvectors of the adjacency matrix have been previously studied for the information they give on the graph and used for partitioning purposes[7,8]. The special properties of $\mathbf{x}_2$ have been investigated by Fiedler[9], who gives a theoretical justification for bisecting the graph based on the elements of this vector. Hence $\mathbf{x}_2$, often referred to as the Fiedler vector, is used to bisect the graph by sorting the vertices v according to the weighting given in x_v.

2.2. The method of spectral bisection

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

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

Following Simon, Pothen and Liou[4,7], the Lanczos algorithm is used to calculate the Fiedler vector. This is a well known Krylov subspace iterative technique, ideal for finding extremal eigenvalues of symmetric matrices (see, for example, Reference 10). Unfortunately the algorithm requires many steps to avoid misconvergence to non-extremal eigenvalues, but the cost of estimating the current smallest can be reduced by bounding it inside an interval of decreasing size[11]. The bulk of the work per iterative step is one matrix–vector multiplication, together with some vector operations. It can be implemented for the most part with the level 1 BLAS plus a tailor-made matrix–vector multiplication subroutine. Because of the load-balancing constraint, each Lanczos vector is explicitly orthogonalised against $\mathbf{e}$ as described in Section 2.1.

Finally, the Fiedler vector is sorted and the graph bisected. Note that it is not necessary to employ a full sorting procedure, just a partitioning into two equal-sized subsets. Unfortunately there is no theoretical guarantee that the partitions will be connected[8], but experience shows that it is rare that they are not.

3. APPLICATIONS TO TIME-DEPENDENT PROBLEMS

Whilst the RSB algorithm usually gives good results for a static problem, there are a number of areas in which improvements may be made for dynamic partitioning of adaptive meshes.

Most notably the full method is expensive as the cost of the Lanczos method, for a problem size n, is $O(n\sqrt{n})$[7]. While this may not be a problem on a static mesh where the cost can be hidden as a start-up overhead, it may be significant when a time-stepping code is remeshing frequently.

The RSB method may also be sensitive to small perturbations in the mesh. For instance, Williams[5, p. 477] states that 'a small change in mesh refinement may lead to a large change in the second eigenvector.' Combined with the fact that the RSB algorithm has no mechanism for using existing information about the previous partition, heavy node migration may result.

In the following Section a technique is presented that enables a graph-based algorithm to use existing information about the partition of a previous mesh. Again it is described with particular reference to the recursive spectral bisection algorithm but the concept could be applied to any graph-based method.

4. A DYNAMIC PARTITIONING APPROACH

If a partitioned mesh is modified by the addition of new elements or the removal of existing ones, an immediate load imbalance (and hence a new partitioning problem) is created. Provided that the new mesh is based on coarsening or refining of the existing one, as in Reference 1, it is possible to interpolate the existing partition onto the new mesh and to use this partition as a starting point in a *repartitioning* algorithm.

4.1. The concept

The repartitioning algorithm used here assumes that, 'small' changes in the mesh will in all probability lead to 'small' changes in the partitions and, if not, then the method described here will eventually revert to the standard RSB algorithm. Ideally most mesh elements will remain in the same subdomain whilst the boundaries are 'juggled'. Of course it is not clear that such 'juggling' will produce optimal communication costs (see, however, Section 5), but certainly if mesh elements 'close to' the interprocessor boundaries (or bisection boundaries for a recursive bisection algorithm) are the only ones involved, then the issues discussed in Section 3 are all addressed. The information from the preceding partition is utilised and, as a result, both the cost and the amount of node migration should certainly be reduced (the factors being largely dependent on the granularity).

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

4.2. Implementation for recursive spectral bisection

4.2.1. Clustering and iteration

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

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

Note that in determining the Laplacian of the reduced graph the number of vertices in each cluster has no effect. However, each vertex cluster is likely to have more edges than an ordinary graph vertex and hence the corresponding rows and columns of the Laplacian will be less sparse. In addition it is possible for an ordinary graph vertex to have more than one edge in common with a vertex cluster. These multiple edges can be represented in the Laplacian by redefining $l_{i,j} = -|\{e \epsilon E : e = (v_i, v_j)\}|$. Note that it is still only necessary to store row and/or column indices of the Laplacian (see Section 2.2).

4.2.2. Bisection

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

4.2.3. Recursion

The recursive part of the method proceeds much as before. The only slight problem occurs in data migration. This is most easily demonstrated with an example. Suppose, then, there is an existing partition of four subdomains labelled 'A0', 'A1', 'B0' and 'B1' and the initial bisection is between the A and B domains. Thus two clusters are formed, one containing elements lying in either 'A0' or 'A1' and away from the previous bisection boundary, the other similarly of elements in 'B0' and 'B1'. The Fiedler vector is found and the domain rebisected.

At this stage the data lying in the 'wrong' partition are moved and it is here that care must be taken. For example, it may be found that some elements in both 'A0' and 'A1' have

to migrate to the other side of the bisection. Clearly the easiest method is for 'A0' elements to be mapped to 'B0' and 'A1' elements to 'B1'. However, this does not guarantee that the new 'B0' and 'B1' subdomains are connected. This can cause problems in defining the clusters at the next recursive level, and code must be included to ensure either that migrating elements go to the right place, or that disconnected groups of vertices are not used to form a cluster. This is most easily accomplished by always using data which migrated at the previous recursive level in the reduced graph, although this is not the most efficient method in terms of either work or data migration.

4.3. The dynamic recursive spectral bisection algorithm

The iterative preprocessing technique to be added to the recursive spectral bisection (or other) algorithm can now be presented in full, and is summarised in Figure 2. The `spectral bisect` subroutine is just the three operations inside the loop in Figure 1. Henceforth this combined algorithm will be referred to as DRSB.

```
repeat recursively {
    /* create initial reduced-size graph */
    until (reduced-size graph large enough) {
       expand reduced-size graph (by next level set)
       cluster (remaining vertices)
    }
    while (reduced-size graph < full graph) {
       /* bisection stage */
       spectral bisect (input reduced-size graph, output partition)
       if (successful)
           escape to next subdomain
       else if (too many iterations)
           reduced-size graph = full graph
       else
           expand reduced-size graph (by next level set)
           cluster (remaining vertices)
    }
}
```

Figure 2. The Dynamic RSB Algorithm

Initially enough level sets are taken to:

(a) balance the domain (i.e. if one cluster contains more than half of the vertices bisection is not possible)
(b) have a meaningful graph to use spectral bisection (i.e. just using one level set does not give enough information).

The spectral bisection algorithm is now applied and the Fiedler vector calculated. If the bisection fails because of a cluster close to the median of the vector the reduced graph is

expanded by one more level set and spectral bisection reapplied. This may be repeated until the level sets have recovered the full graph (when the bisection cannot fail), but this will have increased the costs to well above that of using full spectral bisection in the first place. Repeated failures suggest that the existing partition is not close to a new optimal partition and so the iterations are terminated early. However, based on the testing carried out so far, this does not appear to happen very often (see Section 5).

There are two important heuristics inherent in this algorithm, namely the number of level sets to make the initial reduced-size graph large enough and the maximum number of iterations. While neither heuristic is crucial to the eventual success of the algorithm, both have important efficiency implications. Early results suggest that three level sets make a good reduced graph to start from (although this may depend on the density of edges in the graph). Subsequently, if the method fails after two or three iterations it may be better to then revert to the full graph. For coarse-grained problems (including the initial recursive levels) the potential saving is considerably greater and a few more iterations may be worthwhile.

5. NUMERICAL TESTING

In order to evaluate the performance of the new algorithm fully, comparisons are made between DRSB and RSB on adaptive mesh solutions of time-dependent PDEs. The aim of the experiments is to assess the dynamic technique by measuring both the possible cost savings and the quality of the resulting separator sets. In addition, results are given for the RCB algorithm (see below, Section 5.2), a computationally inexpensive algorithm, although shown by Simon[4], to give inferior partitions to RSB on steady-state problems.

The test meshes are derived from the integration of two time-dependent PDEs described below (Sections 5.3 and 5.4). In both cases a PDE solver generates an unstructured mesh and integrates the solution in parallel with a variable-step explicit time-marching algorithm. The integration continues while the error estimate in each triangle remains below a predetermined tolerance. Once the tolerance is exceeded sequential code on the host automatically refines or coarsens the mesh in order to yield an approximately uniform numerical error estimate across the domain. At this stage load-balancing becomes necessary and this is also carried out sequentially on the host.

The code currently runs in parallel on a Meiko CS-1 equipped with 32 T800 transputers and a SPARC 2 host.

5.1. Metrics

Three metrics are used to compare the algorithms:

$|E_i|$, **the number of interprocessor edges**. E_i is defined to be the subset of edges which cross interprocessor boundaries after repartitioning. The size of this set gives an indication of the sizes of separator sets that might be expected for linear algebra using domain decomposition or substructuring. Since this is related to the parallel overhead for such techniques it is a meaningful figure. It also gives an indication of the volume of communication traffic required for flushing the halos of the subdomains around the memory and the amount of work replicated on their boundaries.

T_b, **load-balancing time**. The time, in SPARC-2 CPU seconds, to carry out the load-balancing sequentially.

T_s, **solution time**. The time, in T800 transputer CPU seconds, to carry out the time-integration in parallel. This figure gives a measure of how much the size of $|E_i|$ affects the overall solution time. It is, of course, very problem-dependent, but the sum of $T_s + T_b$ is the most important figure to minimise and, as can be seen from the results, simply minimising E_i is not necessarily the most successful strategy.

5.2. Recursive co-ordinate bisection

To give a comparison with a cheap non-graph-based technique, results for the RCB algorithm are also given. Described in Reference 4, this method sorts the mesh elements (in this case triangles) by either x or y co-ordinates and bisects on this basis. In this implementation the sorting was executed alternately on x co-ordinates at one recursive level and then on y co-ordinates at the next. It is a simple, intuitive and, above all, cheap technique, but one which provides somewhat poor separator sets as a result of excluding any graphical information. This is particularly noticeable on very irregular meshes where the subdomains are likely to be disconnected.

5.3. Euler wedge shock problem

This problem is driven by the Euler equations in two space dimensions (see Reference 2 for a full description) which simulate air at Mach 2.5 as it hits a 10 degree wedge and forms a shock front[12]. In its original form this is a steady-state problem, but in the time-dependent form used here the shock starts off along the wedge and rises to its steady state position as time proceeds. The unstructured mesh becomes heavily refined around the front as it forms but remains coarse away from the wedge and shock. Figure 3 shows the solution domains after the first and final remeshes.

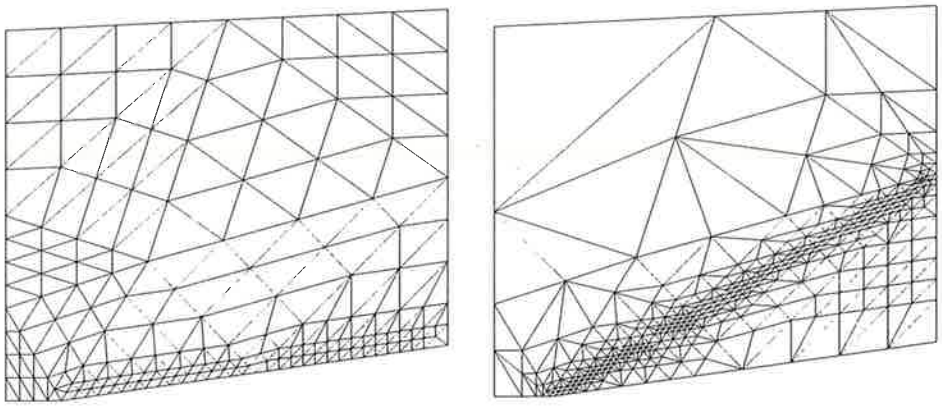

Figure 3. The wedge shock solution domain after the first and final remeshes

Some sample results and the totals for each load-balancing algorithm using 16 processors are given in Table 1. For these experiments there were 4618 residual evaluations and 31 remeshes for each run.

The first result to notice is $|E_i|$, the number of interprocessor edges cut. The figures for DRSB are of the same order as RSB throughout the experiment and the total is actually

Table 1. Sample load-balancing results on 16 processors for the wedge shock problem

	DRSB			RSB			RCB		
n	E_i	T_b	T_s	E_i	T_b	T_s	E_i	T_b	T_s
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1400	147	9.0	127.4	143	26.5	124.3	273	1.0	146.1
1320	129	8.0	13.8	136	24.5	13.8	268	1.0	16.0
1415	134	8.2	152.3	156	26.5	152.3	273	1.1	176.6
1469	144	21.2	208.9	156	28.2	212.1	285	1.0	248.3
860	119	10.4	9.6	137	12.5	9.8	212	0.6	11.6
1589	156	26.9	123.8	163	29.4	121.4	319	1.2	139.0
1572	152	6.4	261.7	160	28.8	257.8	318	1.1	302.8
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
28727	3599	227.4	2074.4	3748	455.8	2074.9	6512	19.9	2415.5

slightly lower. The execution time for DRSB is half that of RSB in total and sometimes as little as a quarter (e.g. n=1572). The high T_b figures for DRSB arise when the reduced graph is expanded back to its original size as described in Section 4.2.2.

When compared with RCB, it is clear that both DRSB and RSB are much more successful in terms of $|E_i|$ but consequently far more time-consuming, and for this reason the T_s figures have been included. Now if the sum $T_b + T_s$ is considered the most efficient technique, for this experiment at least, is DRSB (2302 s) followed by RCB (2435s) and finally RSB (2531s). Of course it is not completely fair to directly add T_b and T_s as the SPARC CPU is about four times faster than that of the transputer. Ideally, however, the load-balancing should run in parallel, as scalability problems arise if this is not the case. The cost of both the DRSB and RSB algorithms is dominated by the Lanczos calculations (implemented with level 1 BLAS and 1 sparse matrix–vector multiply per step) which can be parallelised with good efficiency[13]. Then, even if a parallel efficiency of just 50% is assumed for each algorithm, DRSB will still have the fastest overall solution time with RSB coming in second.

Table 2 gives the totals for the same experiment on varying numbers of processors. If the same thumbnail calculation as before is carried out (i.e. $T_s + 4 \times 0.5 \times T_b/P$) DRSB comes out as the fastest algorithm in each case, although with a decreasing lead as P increases and hence the granularity falls.

5.4. Burger's equation example

This is a well known non-linear convection dominated time-dependent PDE and is described more fully in Reference 1. The solution consists of a pair of L-shaped waves which gradually steepen as they move from bottom left to top right of a square domain and with the rearmost wave eventually overtaking the front wave.

For this example the temporal and local spatial relative error tolerances are varied to give a number of solutions of different accuracy. The smaller of the tolerances result in frequent remeshing of a moderately large number of triangles. The solver commences its run with a uniformly coarse mesh. As this mesh is unsuitable for the solution, large spatial errors occur and the initial time-step fails. The mesh is then refined based on the error estimate to give a new initial mesh. This step–refine sequence is iterated until the mesh reflects the initial conditions of the solution and subsequently the time-steps begin to succeed although with frequent remeshing as the waves propagate.

Table 2. Load-balancing totals for the wedge shock problem

	DRSB			RSB			RCB		
P	E_i	T_b	T_s	E_i	T_b	T_s	E_i	T_b	T_s
4	944	102	7331	919	298	7328	2161	9	7641
8	1941	159	3824	1934	384	3814	2988	14	4051
16	3599	227	2074	3748	456	2075	6512	20	2415
32	5971	308	1200	6212	520	1210	8151	29	1340

Table 3. Load-balancing totals for Burger's equation on 16 processors

	DRSB			RSB			RCB		
TOL	E_i	T_b	T_s	E_i	T_b	T_s	E_i	T_b	T_s
0.05	34038	1928	31796	33851	5931	32100	69675	279	45178
0.01	45833	3603	4123	47195	10714	4107	73904	414	4417
0.005	27748	4313	2579	28842	12587	2549	50518	372	2711

Unfortunately the large mesh sizes needed for the higher accuracy solutions are too big for even the combined memory of the processors and, for tolerances less than 0.05, the solver crashes at meshes of over about 4000 triangles. However, in Table 3 the totals of $|E_i|$, T_b and T_s are given for each algorithm and for three values of TOL.

The first set of figures for TOL=0.05 show much the same as for the wedge shock problem. The DRSB algorithm achieves the same order of $|E_i|$ as RSB but much more quickly. RCB is an order of magnitude faster again but pays for it in the increased work and communications that the solver must use.

For TOL=0.01 and 0.005, however, the story is very different. The remeshing is much more frequent initially on much larger meshes and the solver does not run to completion. As a result the proportion of time spent on the load-balancing is much larger than on a complete run, and the sophisticated methods lose out to RCB.

6. CONCLUSIONS AND FUTURE DIRECTIONS

The results show that the dynamic technique may be used to improve the performance of RSB, both by substantially cutting the calculation time and by decreasing the number of cut edges (though in not all cases). However, RSB is sometimes very inefficient (when the amount of time spent in the solver is relatively low) and in these cases it may not make sense to use DRSB either. It should be remarked, however, that the figures obtained here come from an explicit time-integration solver. If the solver uses implicit integration then linear algebra must be employed and the quality of the separator set is much more important. For example, in a Schur complement method (e.g. see Reference 14), where n ($\approx |E_i|$) is the size of the separator, then the cost of the dense solve for the Schur complement is $O(n^3/P)$.

The method looks very promising and further testing on a range of time-dependent PDEs is under way. The extension of the method to 3-dimensional meshes is algorithmically straightforward, but further work is necessary for an efficient parallel implementation.

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

- other applications. As the DRSB algorithm is graph-based it is not restricted to unstructured mesh problems and could be applied to, for example, matrix partitioning

- other load-balancing methods. The technique of clustering could be applied to other algorithms to improve efficiency. Simulated annealing (see Reference 5) is an obvious example
- accelerated recursive spectral bisection. For non-adaptive meshes it would be nice to have a fast effective partitioning technique. Possibly this could be achieved by generating a coarse partition (using recursive co-ordinate bisection, for example; see Reference 4) and then refining it with the DRSB algorithm.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support of Shell Research Limited. Peter Jimack and David Hodgson are also thanked for their helpful discussions and Justin Ware for providing the example meshes.

REFERENCES

[1] M. Berzins, J. Lawson and J. Ware, 'Spatial and temporal error control in the adaptive solution of systems of conversation laws, in R. Vichnevetsky, D. Knight and G. Richter (Eds.), *Advances in Computer Methods for Partial Differential Equations VII*, IMACS, 1992, pp. 60–66.

[2] J. Ware and M. Berzins, 'Finite volume techniques for time-dependent fluid-flow problems', in R. Vichnevetsky, D. Knight and G. Richter (Eds.), *Advances in Computer Methods for Partial Differential Equations VII*, IMACS, 1992, pp. 794–798.

[3] B. Hendrickson and R. Leland, 'Multidimensional spectral load balancing', Tech. Rep. SAND 93-0074, Sandia National Labs, Albuquerque, NM, 1992.

[4] H. D. Simon, 'Partitioning of unstructured problems for parallel processing', *Comput. Syst. Eng.*, **2**, 135–148 (1991).

[5] R. D. Williams, 'Performance of dynamic load balancing algorithms for unstructured mesh calculations', *Concurrency*, **3**, 457–481 (1991).

[6] B. Mohar, The Laplacian spectrum of graphs, Tech. Rep., Department of Mathematics, University of Ljubljana, Yugoslavia, 1988.

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

[8] D. Powers, 'Graph partitioning by eigenvectors', *Lin. Alg. Appl.*, **101**, 121–133 (1988).

[9] M. Fiedler, 'A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory', *Czech. Math. J.*, **25**, 619–633 (1975).

[10] G. H. Golub and C. F. van Loan. *Matrix Computations, (2 edn.)*, Johns Hopkins, Baltimore, 1989.

[11] B. N. Parlett, H. Simon and L. M. Stringer, 'On estimating the largest eigenvalue with the Lanczos algorithm', *Math. Comput.*, **38**, 153–165 (1982).

[12] L. Demkowicz, J. T. Oden, W. Rachowicz and O. Hardy, 'An *h-p* Taylor-Galerkin finite element method for compressible Euler equations', *Comput. Methods. Appl. Mech. Eng.*, **88**, 363–396 (1991).

[13] Z. Johan, K. K. Mathur, S. Lennart Johnsson and T. J. R. Hughes, 'An efficient communication strategy for finite element methods on the connection machine CM-5 system', Tech. Rep. No. 256, Thinking Machines Corp., Cambridge, MA, 1993 (submitted for publication).

[14] James M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[15] J. E. Flaherty, P. J. Paslow, M. S. Shepherd and J. D. Vasilakis, 'Adaptive methods for partial differential equations', in *Proc. of Workshop on Adaptive Computational Methods for Partial Differential Equations, Rensseleer Poly. Inst., 1988*, SIAM, Philadelphia, 1989.