Adaptive Time-Dependent CFD on Distributed Unstructured Meshes

Chris Walshaw and Martin Berzins

School of Computer Studies, University of Leeds, Leeds, LS2 9JT, U. K.
e-mails: chris@scs.leeds.ac.uk, martin@scs.leeds.ac.uk

An adaptive-mesh parallel CFD software package for time-dependent problems is described. The package uses unstructured meshes and is intended to be portable across MIMD architectures. A new algorithm for the load-balancing of such meshes and their data structure is discussed and its efficiency is demonstrated numerically.

1. Introduction

Two recent trends in software for CFD problems are the use of unstructured adaptive meshes and the need to implement the software on distributed memory parallel computers. The subject of this paper is the implementation of one particular code of this type for the solution of time-dependent compressible Euler and Navier Stokes problems.

The software described here uses the method of lines with the spatial discretisation performed using a cell-centred finite volume scheme on unstructured triangular meshes, [2], to generate an ODE system which is integrated forward in time. Time-integration may be performed using a variety of methods; in the experiments reported here a Theta method algorithm with functional iteration is employed. The code attempts to control both the spatial and temporal errors and to balance them so that the spatial error dominates, [2]. This spatial error control means that the position and density of the mesh points may vary dramatically over the course of an integration. In order to ensure that the mesh is load-balanced as it evolves it is desirable to use a dynamic load-balancing technique. This approach is in contrast to work on regular grids which lend themselves to parallelisation via compiler directives & Fortran 90/HPF and also differs from recent work on a distributed memory compiler for irregular problems, [3], which uses a static mesh decomposition executed at runtime.

2. Software Outline and Development Strategy

An outline of the structure of the code is given in Figure 1. The four main Modules are mesh generation, evaluation of the residual of the ODE system, time-stepping and mesh refinement & load-balancing. A layered software approach, based on these four modules, has been used to generate the parallel code from the serial versions.

The mesh generation software, Module 1, remains serial code at present as the initial mesh generation may be regarded as a start-up overhead for transient problems. In addition, highly refined meshes can be constructed by repeated applications of the

adaptivity code before any time-stepping takes place and therefore it seems sensible to devote code development resources to parallel remeshing as opposed to parallel mesh generation. However, very complex domains could still pose a sequential bottleneck and so development of a parallel mesh generator is desirable in the future.

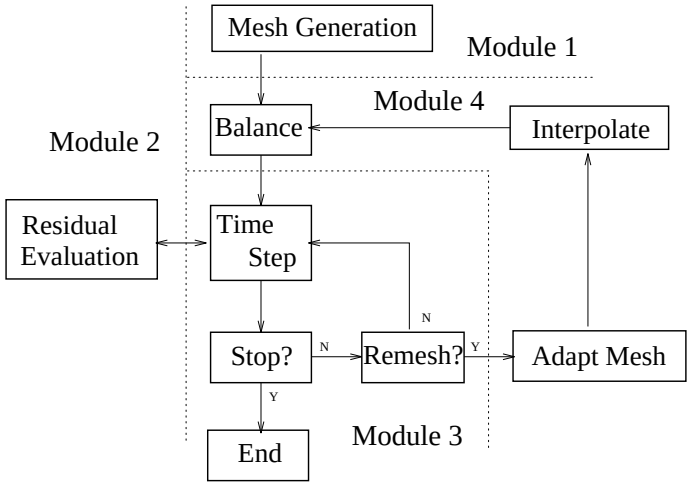

Figure 1. Software Outline

Module 2 contains the bulk of the computational workload (about 80%) in the form of repeated residual evaluations – at least for the explicit functional iteration/time-marching scheme used for hyperbolic problems. Each residual evaluation involves the spatial discretisation of the PDE on the unstructured mesh and so the parallel code for this routine uses a distributed version of the complex pointer-based mesh data structure in C. This is accomplished by the construction of *tiles*.

Conceptually, the tile is a spatially coherent subset of the data defined by a partition of the solution vector (not directly by the mesh). Each of these partitions defines the core of a tile and then the tile as a whole consists of this core, a halo of replicated elements of the solution vector lying in other tiles plus any of the underlying mesh structure required for calculations on core elements. The core of the tile is thus an independent self-contained unit whilst the halo forms a read-only database.

The introduction of some high-level routines to split, move and merge tiles allows the tiles to be moved from processor to processor as required by the partitioning. The routine `flush_halo` copies elements of the solution vector around the system to update the halo database.

Module 3 integrates the temporal ODE system arising from the method of lines approach. The code here involves only simple operations on the solution vector – essentially an extended set of level 1 BLAS. Once the solution vector has been distributed (using the mapping between triangles and solution components) these operations are easily parallelised by the implementation of a `combine` communication primitive for norms and dot products. Routines to communicate updated solution values to the ha-

los have also been written. Parallel linear algebra for implicit time-integration has been developed separately and is currently being integrated into the code.

The adaptivity and load-balancing software, Module 4, is still serial code. Although this has not caused a bottleneck, very large meshes could present a serious obstacle to scalability because the host has some $O(n)$ operations and must broadcast the mesh out to the processors. Hence Module 4 is now due for parallelisation.

2.1. Hardware Platforms and Portability

The software abstraction used is that of a data parallel system with non-local data access provided by explicit message passing calls hidden in a communications library. A single control thread is multiply instantiated for both efficiency and compatibility with sequential and shared memory versions. The machine abstraction is a fully connected network of distributed memory processors connected to a host processor. At present the host has several functions such as I/O, graphics and control of the geometrical data. However as the code is being increasingly parallelised this will be phased out.

The code currently runs in parallel on the development platform of a Meiko CS-1 equipped with 32 T800 transputers and a SPARC 2 host. It has also been ported to the new Meiko CS-2 but the test machine to be used has not yet been fully commissioned.

For portability between different message passing systems the machine specific communications routines are hidden by a minimal set of more general communications primitives, such as `send`, `receive`, `broadcast` and `combine`. These primitives may be written in terms of the current message passing standards (such as MPI or the *de facto* standard PVM). A further library of useful 'macros' are then implemented in terms of these primitives. This library includes the level 1 BLAS, routines for updating halo information, and the linear algebra software. Should porting to different machines be required, all that is necessary is the implementation of the communication primitives.

3. Load-Balancing

The key problem in obtaining optimal performance is ensuring that the load is evenly balanced and that the communication cost is reduced as much as possible by minimising inter-processor dependencies. It is well known that this mapping problem is NP-complete and so heuristics must be employed to obtain a usable algorithm. In addition, the adaptive nature of such codes mean that 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.

A number of good load-balancing algorithms, in particular Recursive Spectral Bisection (RSB) and its extensions, [1], [6] & [9], are based on partitioning a graph which corresponds to the communication requirements of the spatial discretisation of an unstructured mesh. Until now, however, such algorithms have not addressed adaptive mesh codes and the resulting incremental update partitioning problem posed when a mesh with an existing partition is being refined and/or coarsened. The failure to utilise this existing partition suggests that the load-balancing may be unnecessarily computationally expensive. In [7] an iterative preprocessing technique for the update problem was proposed to *enhance* existing graph-based partitioning methods and in Section 3.3 a summary is provided of how this new algorithm can be used in the context of adaptive

meshes.

3.1. Recursive Bisection Methods

The Recursive Bisection approach is based on the principal 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. Simon, [6], considers three strategies – Recursive Coordinate, Graph & Spectral Bisection – and demonstrates the superiority of RSB over the other two.

It should be noted that Recursive Bisection algorithms can create any number of partitions (not necessarily a power of 2). For example, to create 3 partitions, the bisection field (the Fiedler vector in the case of RSB or the x/y coordinates for RCB – see below) can either be divided directly into 3, so called strip-wise partitioning, or bisected with a 1:2 split and then bisection applied again to the larger partition.

3.1.1. Recursive Coordinate Bisection

RCB is a simple and intuitive technique which bisects the mesh by sorting the elements using alternately x and y coordinates. Although the method is computationally inexpensive it can provide poor separator sets due to ignoring communication information. This is particularly noticeable on very irregular meshes where the subdomains are likely to be disconnected, [6].

3.1.2. Recursive Spectral Bisection

Spectral Bisection, [6], associates each mesh element with a node of an undirected graph. The *dual communication graph*, $G(V, E)$ where V is the set of nodes and E the set of edges, is then defined by connecting nodes together (with an edge) when the spatial discretisation gives rise to a dependency between the corresponding mesh elements. This dual graph can be represented by an $n \times n$ symmetric matrix L, known as the Laplacian, where n is the number of graph nodes. The diagonal of L gives the degree of each graph node whilst the off-diagonals are -1 at (i, j) where an edge connects nodes i and j and zero elsewhere. Due to the special properties of this matrix, an approximate solution of the bisection problem can be found by using the eigenvector (or Fiedler vector) associated with the smallest non-trivial eigenvalue of L to weight the graph nodes (see [6] for details).

3.2. Load-Balancing of Adaptive Meshes

Whilst Spectral Bisection usually gives good results for a static problem, it does have a number of drawbacks for the dynamic partitioning of adaptive meshes. In particular the method is computationally expensive; the cost of finding the eigenvector for a problem size n, being at least $O(n \log n)$. This problem has been addressed in the latest version of RSB, [1], where the initial bisection is found on a coarse graph and then projected onto increasingly finer graphs using multigrid type techniques.

A second important consideration for adaptive meshes is that the method tends to be sensitive to small perturbations in the graph. For instance Williams, [9, page 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. This problem has also been recently addressed by Driessche & Roose, [4].

In the next Section an incremental method is presented that enables a graph-based algorithm to use existing information about the partition of a previous mesh. It is analogous to adaptive techniques in the same way that multilevel RSB is analogous to multigrid techniques.

3.3. A Dynamic Partitioning Approach

When 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 this case [2], 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.

The assumption is that, unless the mesh has changed dramatically, the partitions will not need to be changed a great deal. Ideally most mesh elements will remain in the same subdomain whilst just the boundaries are balanced. To effect this, graph nodes away from the inter-processor boundaries are clustered together and represented by a single node in a reduced size graph. In this way the information from the previous partition can be 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). This idea forms the basis of the Dynamic RSB algorithm (DRSB) and a full description together with a discussion of some implementation issues is given in [7].

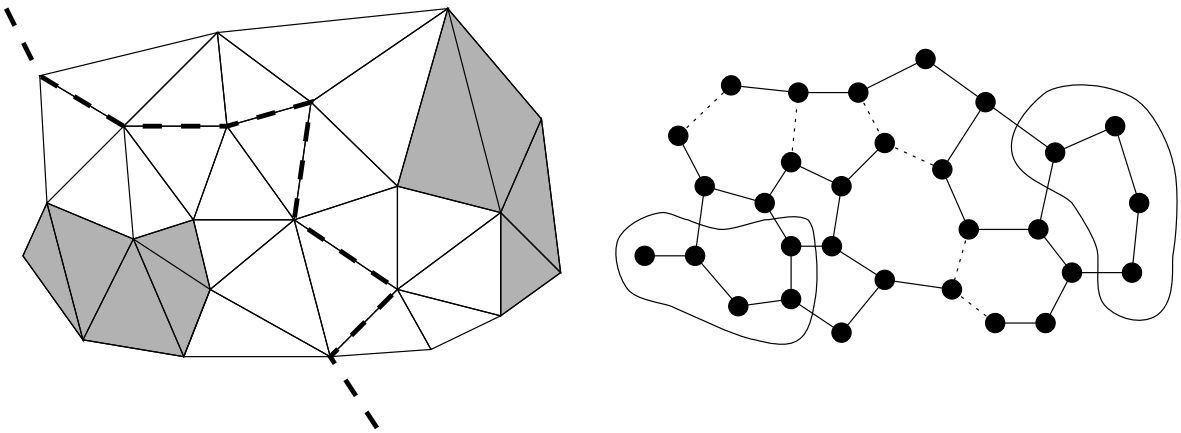

Figure 2. Clustering of mesh elements and the corresponding dual graph

Figure 2 show this clustering technique on a simple mesh. The domain is shown (left) immediately after refinement with the old (and now non optimal) partition interpolated onto the mesh and the mesh elements which have been selected for clustering shaded. On the right the dual graph shows the inter-processor edges as dotted lines and the clustered nodes ringed. These clustered nodes will be represented by a single node in the reduced graph. Spectral bisection (or another graph-based technique) is applied to the

reduced graph and the graph partitioned as before. Of course, it is no longer clear that such balancing will approximate the optimal bisection, but the results ([7] & [8] and below) actually seem to show an improvement over those of the RSB algorithm in most cases.

4. Computational Example – Euler Wedge Shock Problem

The results of the above approach are best illustrated with an example problem. The Euler equations in two space dimensions are integrated with an inflow of air at Mach 2.5 hitting a 10 degree wedge and forming a shock front. In its original form this is a steady state problem, however, 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.

Table 1 shows solution time in seconds (neglecting any remeshing and partitioning carried out on the host) and speed-up results for a typical run. In this case there were 1,239 residual evaluations and 18 remeshes with an average mesh size (per residual evaluation) of 1157 triangles (1967 maximum; 92 minimum). More impressive performance figures can be achieved with larger meshes but, since the on-processor memory is limited, a single processor cannot store more than about 2,000 triangles and so true speed-up figures cannot be produced.

Table 1
Solution time & speed-up for a typical run

P	1	2	4	8	16	32
t (secs)	9374	4747	2410	1247	659	372
speed-up	1.00	1.98	3.89	7.52	14.2	25.3

The most computationally effective load-balancing technique is the one which, when combined with the solution process, produces the lowest overall run-time. Sophisticated methods, such as RSB, usually result in significantly fewer cut-edges than crude techniques, such as RCB. However, they are much more computationally expensive to run and hence can result in a higher execution time overall. Table 2 shows some typical figures for the time-integration of the wedge shock problem on 16 processors. Here n is mesh size, $|E_i|$ is the number of graph edges cut and T_b is the time taken for the load-balancing. The differences in the algorithms can be clearly seen; RSB & DRSB cut significantly fewer edges than RCB but are much slower.

Table 3 compares the solution time (T_s – transputer CPU seconds) with the load-balancing time (T_b – SPARC CPU seconds) and for all but the 32 processor case the total run time, $T_s + T_b$, is lower for DRSB. This is not a fair comparison because the SPARC is a faster than the transputer (by a factor of about 4) but this can be mitigated by the fact that the load-balancing is currently done sequentially. For parallel load-balancing, the cost of the DRSB algorithm is dominated by a Lanczos calculation (implemented with level 1 BLAS and one sparse matrix-vector multiply per step) which can be parallelised with

good efficiency, [5]. Then, even if a parallel efficiency of just 50% is assumed for the whole of the algorithm, DRSB will have a faster overall solution time than RCB for all the given results.

Table 2
Edges cut & load-balancing time on a 16 processor run

	RCB		DRSB		RSB	
n	$\|E_i\|$	T_b	$\|E_i\|$	T_b	$\|E_i\|$	T_b
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1483	253	1.0	109	8.1	119	105.3
1869	283	1.3	131	13.6	117	52.6
1967	290	1.4	125	11.6	112	120.7
1961	284	1.4	118	9.5	114	65.0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
17238	3413	11.9	1685	108.8	1608	660.1

A comparison of the 16 & 32 processor cases begins to show what happens when the work per processor is reduced to the point at which the benefits of the more complex DRSB algorithm are outweighed by the computational effort needed in the extra level of recursive bisection. This situation would be less likely to occur if a parallel version of DRSB were to be used, but is still possible.

Table 3
Solution & load-balancing times for varying numbers of processors

	RCB		DRSB	
P	T_s	T_b	T_s	T_b
4	2,506	6	2,410	48
8	1,304	8	1,247	74
16	762	12	659	109
32	413	17	372	154

5. Conclusions

The development of a prototype parallel adaptive PDE solver has been considered. The key issue of load-balancing unstructured meshes has been discussed and a new algorithm which appears to deal well with adaptive meshes in time-dependent computations has been described. Future work on this package will include the integration of the parallel linear algebra software and the development of parallel load-balancing and parallel adaptivity modules.

Acknowledgements. The authors would like to acknowledge the financial support of Shell Research Limited and to thank Justin Ware for all his help.

REFERENCES

1. S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. In R. F. Sincovec *et al*, editor, *Parallel Processing for Scientific Computing*, pages 711–718. SIAM, 1993.
2. 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, editors, *Advances in Computer Methods for Partial Differential Equations VII*, pages 60–66. IMACS, 1992.
3. R. Das, R. Ponnusamy, J. Saltz, and D. Mavriplis. Distributed Memory Compiler Methods for Irregular Problems – Data Copy Reuse and Runtime Partitioning. ICASE Rep. No. 91-73, Institute for Computer Applications in Science and Engineering, 1991.
4. R. Van Driessche and D. Roose. An Improved Spectral Bisection Algorithm and its Application to Dynamic Load Balancing. Rep. TW 193, Dept. Computer Science, Katholieke Universiteit Leuven, 1993.
5. 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).
6. H. D. Simon. Partitioning of Unstructured Problems for Parallel Processing. *Computing Systems in Engineering*, 2:135–148, 1991.
7. C. H. Walshaw and M. Berzins. Dynamic Load-Balancing For PDE Solvers On Adaptive Unstructured Meshes. Computer Studies Tech. Rep. 92.32, University of Leeds, 1992. (Available by anonymous ftp from agora.leeds.ac.uk in scs/doc/reports/92_32.ps.Z).
8. C. H. Walshaw and M. Berzins. Enhanced Dynamic Load-Balancing Of Adaptive Unstructured Meshes. In R. F. Sincovec *et al*, editor, *Parallel Processing for Scientific Computing*, pages 971–978. SIAM, 1993.
9. R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice & Experience*, 3:457–481, 1991.