INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2002; **40**:171–187 (DOI: 10.1002/fld.362)

Aerospace simulations on parallel computers using unstructured grids

N. P. Weatherill*, O. Hassan, K. Morgan, J. W. Jones,
B. G. Larwood and K. Sorenson

Department of Civil Engineering, University of Wales Swansea, Singleton Park, Swansea SA2 8PP, U.K.

SUMMARY

If complex simulations on realistic configurations are to be performed, it is critical that the large volume of data that will be produced can be handled efficiently. In our work, we have chosen to parallelize all the steps in the computational cycle; unstructured mesh generation, solvers, adaptation and visualization. In this way, data is distributed at the early stage of mesh generation and is never brought together, thereby preventing data bottlenecks. Using these parallel modules, large-scale simulations have been performed for both computational fluid dynamics and computational electromagnetics. The paper briefly describes the approaches taken to parallelizing the unstructured grid techniques and examples are given using meshes to a quarter of a billion elements. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: computational fluid dynamics; computational electromagnetics; unstructured grids; parallel processing

1. INTRODUCTION

The advent of the vector supercomputers of the 1970s, of which the CRAY 1S is perhaps the most famous, had a major influence on scientific simulation. The technology acted as a catalyst for new algorithms that were able to address emerging and challenging applications. In the last few years, the next generation of computers has emerged in the form of massively parallel computer hardware. Again there is an opportunity for the simulation community to utilize this new computer power to open new avenues for research and to attempt real-world simulations that in the past were out of the range of all but a handful of extremely expensive computers.

In the areas of computational aerodynamics and electromagnetics (primarily focused on radar cross section), it is clear from our estimates (see Figures 1 and 2) that very large computational grids are required for future simulations. However, with the continuous growth

* Correspondence to: N. Weatherill, Department of Civil Engineering, University of Wales Swansea, Singleton Park, Swansea SAZ 8pp, U.K.

Contract/grant sponsor: European Union; contract/grant number: 25050
Contract/grant sponsor EPSRC (Esprit 25050)

Received May 2001
Revised April 2002

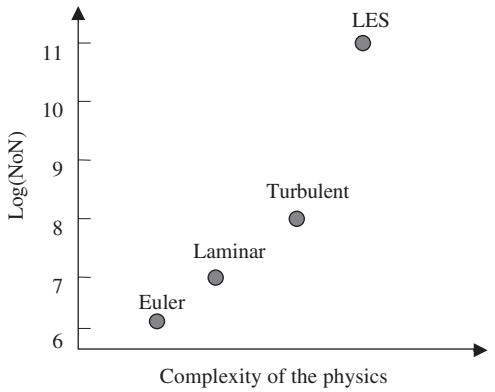


Figure 1. Mesh requirements for computational fluid dynamics for a complete aircraft (NoN—number of nodes).

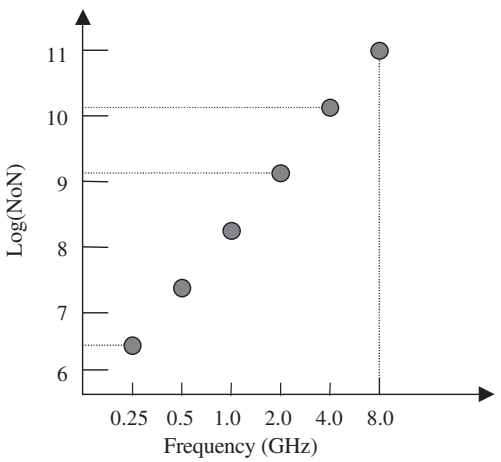


Figure 2. Mesh requirements for the simulation of electromagnetic scatter from an aircraft 20 m in length (NoN—number of nodes).

in speed of modern parallel computer platforms and suitable software, it will be feasible to perform the next generation of simulations (see Figure 3).

Some years ago, therefore, we embarked on a long-term research programme to enhance our software capability in CFD and CEM so as to provide the necessary basis for the next generation of simulations. It was deemed necessary to parallelize all the different steps in the computational cycle, from geometry input, to unstructured mesh generation, to simulation, to visualization and data mining and onto mesh adaptation. To aid in the usability of such software, a parallel simulation user environment (PSUEII), within which all the parallel modules were embedded, was also developed.

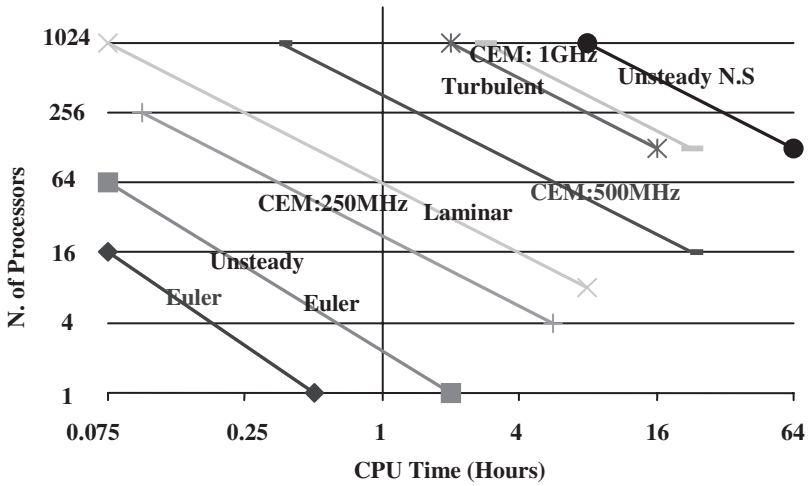


Figure 3. Estimated time for simulations in the next 5 years.

2. BASIC TECHNOLOGY

Details of some of the developments and performance of these modules have been previously reported [1–5]. However, an overview of the basic technology will be given.

2.1. Geometry modelling

A CAD model contains information on the topology and the shape of a geometry. The topology is constructed from nodes, edges, faces and volumes, whilst the geometry is defined in terms of vertices, curves and surfaces. The topology can be extracted from a CAD model. From the geometry data, it is possible to construct a mathematical model of curves and surfaces. A parametric representation is used so that grid generation on a curve and a surface then reduces to grid generation in one and two dimensions, respectively. Typical complex aerospace geometries may use many hundreds of individual surfaces, some of which may be degenerate. The issue of ensuring the geometry is watertight with a consistent topology is a large research topic in its own right. It is, therefore, considered to be outside of the domain of the computational simulation and is, therefore, not addressed within the current implementation of the PSUEII.

2.2. Unstructured grid generation

To handle complex shapes, unstructured grids of tetrahedra offer the maximum flexibility. In our approach to unstructured grid generation, the required mesh point density within a domain is defined in terms of a background grid and a set of point, line and planar sources [6]. Unstructured surface grids are generated using an advancing front algorithm operating in the space of the parametric co-ordinates of the surfaces [7]. A volume mesh is constructed using a Delaunay algorithm with automatic point insertion [6].

The approach adopted for parallel mesh generation is based upon geometrical partitioning of the domain [2]. To generate a grid in parallel, the complete domain is divided into a set of smaller sub-domains, and then a grid generated in every sub-domain independently. An assembly of the sub-domain grids forms the final grid for the total domain. A manager/worker model is employed, in which the initial work is performed by the manager who then distributes the grid generation tasks to the workers. The workers generate the grids using the sequential Delaunay algorithm [6]. The manager can recombine all the sub-domain grids or, if the grid is particularly large, leave the partitioned grid on disk.

The procedure can be described in four stages:

Stage 1: Apply the geometrical partitioning scheme on the computational domain.
Stage 2: Generate meshes on the inter-domain boundaries.
Stage 3: Generate grids in each sub-domain using different processors.
Stage 4: Post-process the grid, including node smoothing of inter-domain boundaries, work load redistribution, and build the inter-domain node-element communication table.

In this approach, following the generation of the surface grid, the domain is sub-divided using planar cuts through the domain. From these cuts a bounding curve is obtained which is defined by existing edges in the surface triangulation. Although these edges are not in general planar, the points can be mapped to the plane of the cut surface. On the cut planar surface, a grid can be generated and then the edge points transformed back to their original positions to provide the final inter-domain grid. High-quality inter-domain grids can be obtained in this way.

The division of the domain, or a subsequent sub-domain, follows two stages. The first step is to find a suitable cutting plane to split the domain into two. The longest axis of the enclosing box of the domain is found and then two options are available:

(i) A simple approach, which does not account for the distribution of nodes within the domain, is to place the cutting plane halfway along this axis.
(ii) An alternative approach, designed to account for the distribution of nodes within the domain, is to select the boundary node with the minimum co-ordinate in the sense of the longest axis. A search is begun, moving through the surface triangulation, marking triangles and advancing along the longest axis and maintaining a front which is normal to the axis. Given that the number of surface triangles within a domain is known, the search continues until half the triangles have been identified. A plane is then defined with a position along the major axis computed as the average of the axis co-ordinate of all the nodes in the advancing triangles.

The proximity of the proposed cutting plane with any 'natural' boundaries, such as the leading and trailing edges of a wing is then checked. This is performed using the following algorithm. If necessary, the position of the cutting plane is modified to ensure that the distance between the plane and any boundary is not less than the minimum background grid spacing at that location. This procedure effectively defines regions where a cutting plane cannot be placed. Figure 4 shows these invalid regions with shaded areas for the Thrust Supersonic car geometry.

Step 1: Determine the normal vector for all triangles in the current domain.
Step 2: Inspect all normal vectors and 'flag' each triangle where the angle between the normal vector of the triangle and the normal to the cutting plane is less than 5°.

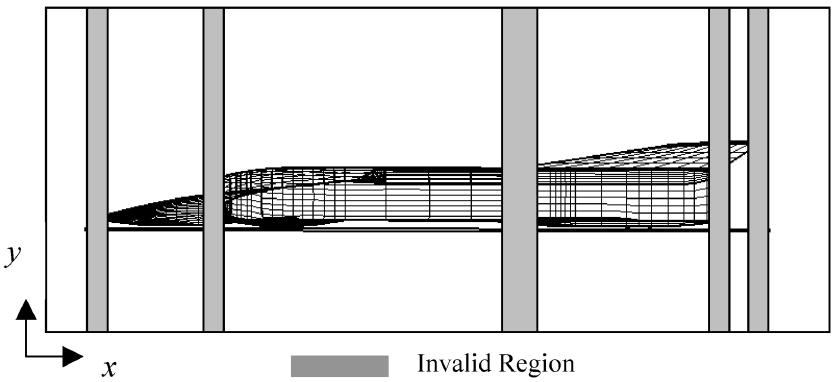


Figure 4. Invalid planar cut regions for thrust SSC geometry.

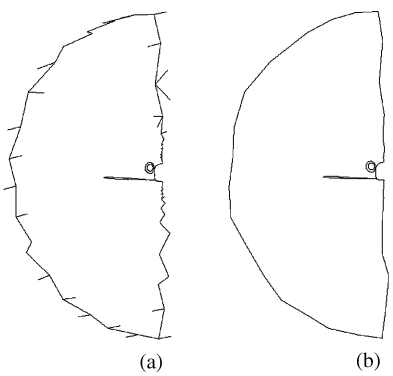


Figure 5. (a) Initial coarse rim, (b) smoothed rim.

Step 3: Build invalid regions around the flagged triangles.
Step 4: Ensure the cutting plane does not lie within an invalid sector. If necessary the cutting plane is moved.

Once a valid position for the cutting plane has been established it is then necessary to derive a closed 'rim' of edges from the surface triangulation of the domain that will form the boundaries for the two-dimensional Delaunay triangulation procedure. Initially, a coarse rim is found by extracting all edges that intersect the cutting plane, as shown in Figure 5(a). A closed loop of edges is then formed by deleting all edges which contain a node that occurs only once in the loop and then by performing a directed graph analysis, Figure 5(b). The directed graph analysis is performed by finding the start and end nodes and assigning a weight to each node along all possible paths. The shortest single route from start to end nodes is found by starting from the end node and moving to the adjacent node which has the lowest weight measure.

Once the closed loop has been defined, further pre-processing is performed prior to the generation of the grid on the cutting plane. Three steps are required. The first two are designed

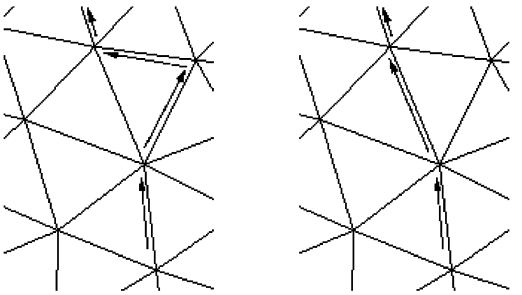


Figure 6. (a) Rough initial edge, (b) edge smoothed.

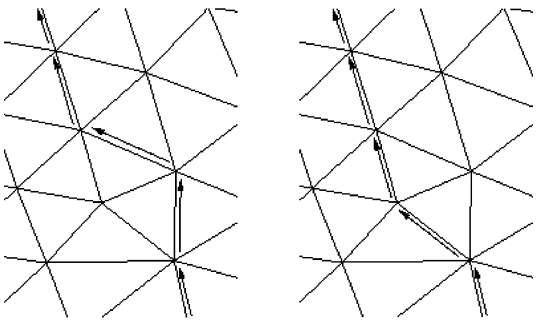


Figure 7. (a) Initial edge path, (b) smooth edge path.

to provide a smooth closed loop, whilst the final step involves the projection of the nodes of the closed loop onto the cutting plane:

(i) Consider the path of the loop as shown in Figure 6(a). It is clear that a path more aligned with a given direction can be derived when the closed loop contains two edges which form two of the three edges of a given surface triangle. The two edges in the loop are replaced by the third edge in the triangle, Figure 6(b). This procedure forms the first pre-processing step.

(ii) Consider two consecutive edges in the closed loop. These form two edges in two triangles that share a common edge, Figure 7(a). Clearly, therefore, there are two paths that connect the two vertices of the two triangles that share the common edge. The angle between the two edges that form part of the closed loop is computed and compared with the angle formed by the edges that form the alternative path. The path that subtends the largest angle is chosen, Figure 7(b).

(iii) Before the inter-domain grid can be generated the final step involves the projection of the nodes on the closed loop onto the cutting plane. Given a cutting plane of constant x, say x_0, a node with co-ordinates (x, y, z) will be mapped to the position (y, z) on the plane $x = x_0$. Given that the ordering of the boundary nodes is known, any boundary edges

crossover, which could result from this simple mapping, can be immediately detected and the co-ordinates of the node modified to untangle the edges. The inter-domain boundary mesh is then generated using a two-dimensional Delaunay triangulation algorithm. The boundary nodes are then mapped back to their original positions on the closed loop. It has been found that this can result in the intersection of triangular faces of the inter-domain grid and the original domain surface grid. This arises when two nodes cannot see each other and hence an intersection arises. However, this problem can be easily solved by recursively adding nodes along the edge of the triangle of the inter-domain surface mesh that intersect with the domain surface mesh.

2.2.1. Inter-processor communication. The structure of the parallel grid generator is a single programme multiple data model. Sub-domain boundary data is passed from the manager to worker processors using message passing interface (MPI) [8]. To avoid latency problems occurring within the network, data, such as the boundary triangles and point co-ordinate arrays, are sent packed into a buffer, rather than being sent with individual calls to the send routine provided by the MPI library. This operation of packing data into a buffer creates a contiguous message from irregularly spaced data structures. The packing of data has another advantage that allows the programme to be run across a heterogeneous network of workstations. Switching between MPI and parallel virtual machine (PVM) [9] is relatively straightforward. The geometrical partitioning approach requires a minimal amount of communication between master and slave, and zero communication between slaves.

2.2.2. Dynamic load balancing. If $\mathbf{N}$ sub-domains are created as a result of the domain decomposition procedure, and $\mathbf{N}$ processors are available, then there can be one task per processor. Since the workload for each sub-domain may be different or, alternatively, a network of heterogeneous workstations is used as the computing environment, then this can produce an inefficient parallel performance. Effectively, the worker with the largest workload, or the slowest worker delays the process. Since, in the proposed strategy, there is no communication between processors, it is practical to sub-divide the domain into $\mathbf{M}$ partitions and if $\mathbf{M} > \mathbf{N}$ it is possible to distribute more than one task to a processor. This procedure will be termed *dynamic load balancing* [2].

For a computer platform with $\mathbf{N}$ processors, it is recommended to generate i times $\mathbf{N}$ tasks, where i is an integer. This avoids any imbalance due to tasks that require significantly more computational effort than others.

An additional form of dynamic load balancing can also be implemented in which a processor sub-divides again an already sub-divided domain. This is a particularly effective procedure when the computational demands of a sub-domain grid exceed the memory capacity of a processor.

It is apparent that the strategy of geometrical grid partitioning, dynamic load balancing and further geometrical partitioning provides a means by which it is possible to generate arbitrarily large meshes on computers with modest computational specifications. This is a very powerful argument for the geometrical partitioning strategy.

2.2.3. Inter-domain communication. Once the grid has been generated, it is necessary to ensure that the relevant communication information between sub-domains is available for any simulation. Once the required number of sub-domains has been reached in the sub-division

process, the sub-domains are assembled and farmed out among the available processors for volume grid generation. During the assembly process the sub-domains are renumbered so that all sub-domains begin at node number one, defined as local numbering, since all new points generated on the inter-domain boundaries are assigned a global number, with the mapping between local and global stored. It is this mapping that enables the communication information to be found. This then enables an arbitrary transformation between local and global numbering to be performed making it possible to construct the necessary communication links for any solver.

2.2.4. Load redistribution. A well-balanced workload among the sub-domains is essential for all parallel simulation algorithms. Since the domain decomposition is carried out on the initial grid, it proves difficult to predict the exact number of elements in each sub-domain. However, a workload redistribution algorithm can be introduced as a post-processing step.

The generalized load-balancing problem is an important area for research in its own right. We have adopted the technique developed by Hu and Blake [10]. The technique minimizes the Euclidean norm of the migrating load. The algorithm is simply based on solving the system $Lx = b$ where L is the Laplacian matrix of the graph

$$(L)_{ij} = \begin{cases} -1 & \text{if } i \neq j, \quad (i,j) \in E \\ \deg(i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$b_i = S_i - W \tag{2}$$

Here S_i is the current load in domain i, W is the average load and E represents the communication between adjacent sub-domains. A conjugate gradient method is used to solve this system. The weight to be transferred between domains i and j is then given by $x_i - x_j$.

2.2.5. Smoothing internal boundaries. The communication table between sub-domains also enables operations to be performed on the mesh. A post-processing step can be implemented in the Delaunay mesh generator to smooth inter-domain boundary nodes using a Laplacian smoother

$$r_0^{p+1} = r_0^p + \frac{w}{N} \sum_{i=1}^{N} (r_0 - r_i) \quad i = 1, \ldots N \tag{3}$$

where r denotes the nodal co-ordinates, N is the total number of neighbour points i, to the node o, ω is the relaxation parameter and p is the iteration level.

An example of the domain decomposition is shown in Figure 8. The figure shows the iterative procedure of domain decomposition, and also the opportunities for parallelization at the decomposition stage. The initial surface triangulation is split into two halves, and the resulting inter-domain boundary is shown below it. This then leads to two separate sub-domains, which can be either sent to the volume mesh generation stage, or further sub-divided until the required number of sub-domains is reached. In this case, further sub-division has taken place and the same process has been repeated. It is at this stage that parallelization can

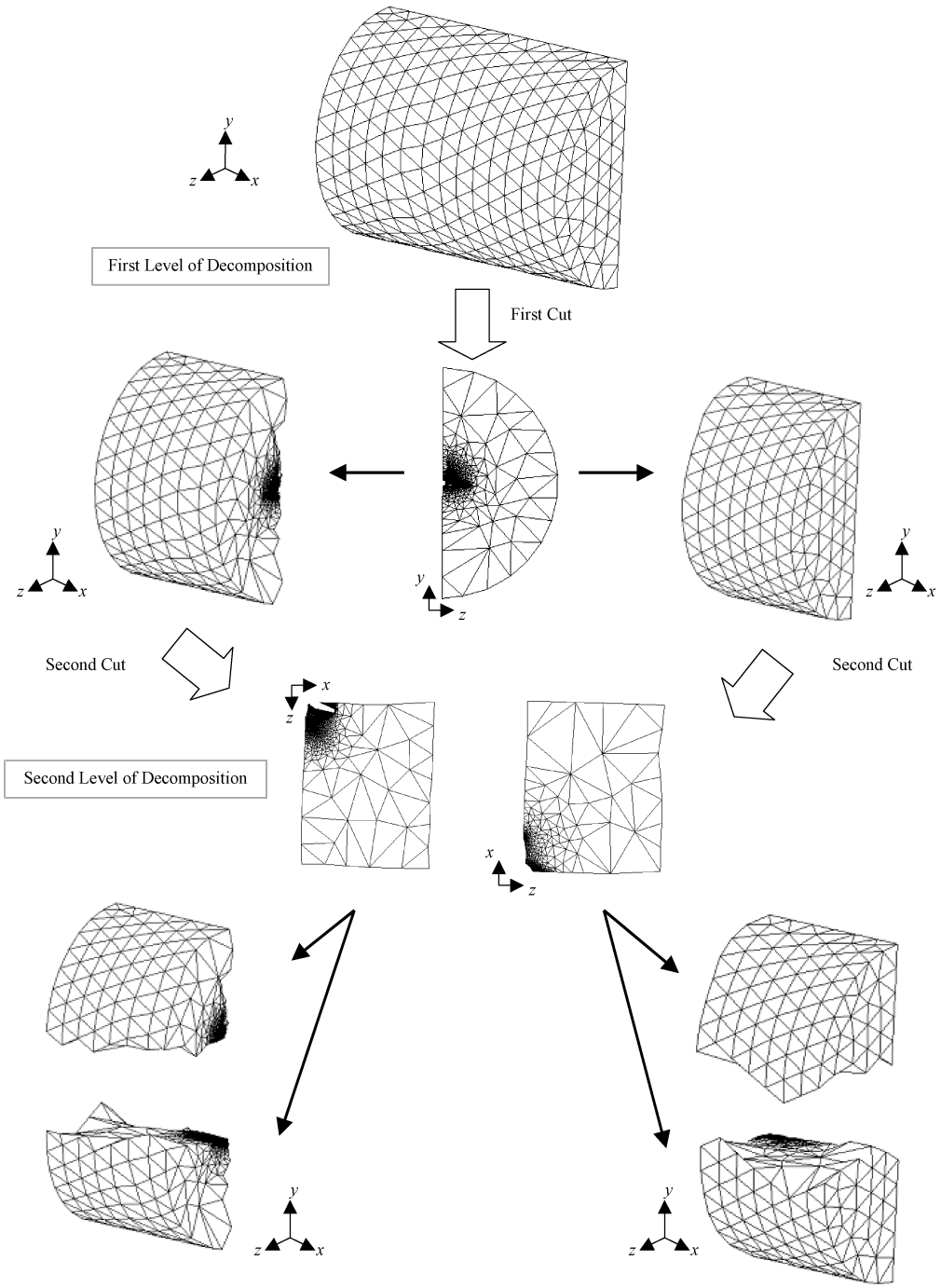


Figure 8. Domain decomposition.

take place, whereby each sub-domain can be farmed out among the available processes for sub-division and inter-domain meshing.

2.3. Solution procedure

The solver technology used for both fluids (Euler or Navier–Stokes equation) and electromagnetics (Maxwell's equations) is finite element based with explicit time integration.

The spatial solution domain is discretised into a general assembly of elements. A piecewise linear variation of the approximation solution space is assumed. A Galerkin approximate variational formulation is used. The mesh is represented in terms of an edge-based data structure.

Stabilization and discontinuity capturing is achieved by replacing, on each edge, the actual (convective) flux function by an approximate numerical flux function. Explicit finite difference procedures are employed to discretize the time dimension.

In the parallel implementation, the mesh is partitioned. Within each partition, nodes and edges are locally numbered. Edges are not duplicated and, where two or more domains meet, the domain with the lower partition number stores the interface edges. Interface nodes are duplicated in sub-domains and, for each node, the domain with the higher partition number performs the updating of the solution values. At the start of a time step, the interface nodes obtain contributions along the interface edges local to each particular sub-domain. These partially updated interface nodal contributions are then broadcast to the corresponding interface nodes in the neighbouring sub-domains. While the data is being sent, each processor performs a loop over the interior edges. This is then followed by receiving the interface node contributions in order to update all interface nodal values. The sending of the updated values back to the interface nodes completes a time step of the procedure [3]. Performing two loops over the edges, the first over the interface edges and the second over the interior edges, allows communication and computation to overlap if allowed by the parallel computer hardware.

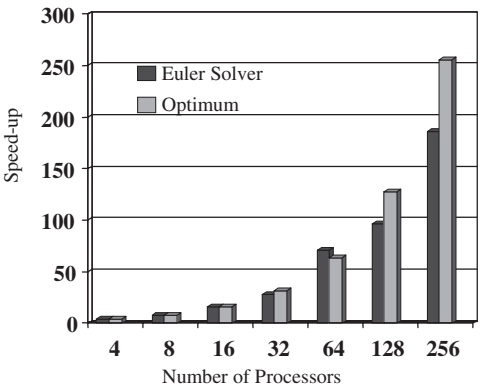
Typical performance data for the parallel solvers, for Euler, Navier–Stokes and the Maxwell equations are given in Figure 9.

2.4. Adaptation

Given a solution on an initial mesh, *h*-refinement can be employed to provide additional resolution where indicated by an error estimator. Mesh refinement is applied in the different sub-domains, with care taken to ensure consistency of the grid should refinement be required along interface boundaries. Nodes added on the configuration geometry are taken back to the original surface which requires details of the geometry to be sent to the different processors.

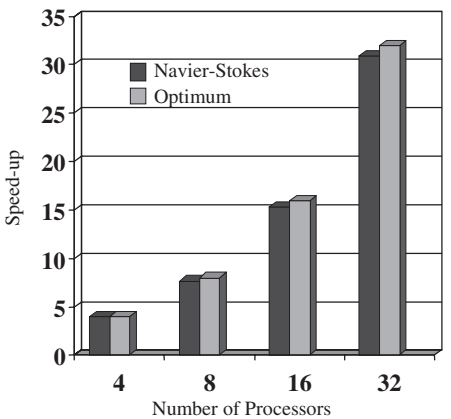
2.5. Visualization

The basis of the parallel visualization toolkit is for all the searching and computationally intensive work to be performed on the processors and only the data required to be rendered sent to the workstation for visual display. The parallel visualization module, ViPar, implements the geometry data transfer method of distributing the visualization process. The parallel architecture is composed of a Master Process running on the graphics workstation, and a



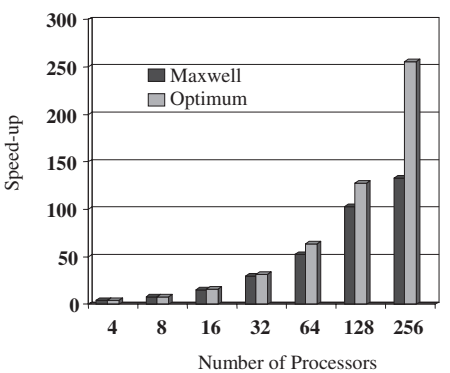
Processors	Speed -up
4	4
8	7.52
16	16.36
32	28.28
64	71.08
128	96.28
256	185. 7

Number of Elements: 1.7 million
Number of edges 1.92 million



Processors	Speed -up
4	4
8	7.68
16	15.4
32	30.9

Number of Elements: 1.97 million
Number of edges 2.32 million



Processors	Speed -up
4	4
8	8
16	15.2
32	29.6
64	52.8
128	102. 8
256	132. 8

Number of Elements: 1.90 million
Number of edges 2.25 million

Figure 9. Performance data for the parallel solvers.

number of Slave Processes running on the parallel platform. The current implementation of ViPar assumes that the architectures on which it is executed conform to:

1. The graphics workstation should support the Open-GL library for rendering. Today, this is not seen as a major restriction since virtually all desktop platforms support Open-GL.

2. The parallel computer is assumed to be of a multiple instruction, multiple data (MIMD) architecture. Again, this is not much of a restriction since most modern parallel computers use the MIMD architecture.
3. The parallel computer is assumed to use distributed memory architecture. This, in fact, is not a restriction since it can be emulated on almost any type of parallel computer including shared memory parallel computers and even clusters of networked workstations.

A set of library routines has been developed to handle the communication. Fast search routines, which operate between sub-domains, and hence are held on different processors, have been developed that utilise octree data structures [11].

Given a geometry, the visualization, meshing, simulation, adaptation and post-processing are all achieved in parallel without a requirement, at any stage in the cycle, to bring together, within one domain, the simulation data. As such, it is our premise that no computational bottlenecks are created. Given this development the new software and hardware technology can be fully exercised.

3. ENGINEERING SIMULATIONS

3.1. Computational electromagnetics

As an example of a large-scale simulation, the propagation of a single wave through an engine duct is considered. Maxwell's equations, written in the time domain, are solved using an explicit time integration procedure. The frequency of the simulation required is 10 GHz. The wavelength, therefore, is

$$\lambda = (\text{Speed of light})/(\text{Frequency}) = 3 \text{ cm}$$

The duct has dimensions (metres) of $6.2 \times 0.45 \times 0.45$. If it is assumed that 10 grid nodes per wavelength are required then, the duct, of electrical length $210 \times 15 \times 15$, requires a mesh of 47.25 million nodes, or approximately 250 million elements.

To generate a uniform mesh with the required size, a background spacing was set at 3 units. The details of the mesh generated are given in Table I.

Figure 10 shows the number of elements generated in each of the 128 partitions. Although there is some variation, after the application of the re-distribution algorithm, all partitions contain almost the same number of elements.

Figure 11 shows the number of partitions generated by each processor. This data highlights the implementation of dynamic load balancing. For example, processors 10 and 21 each generated grids for two domains, whilst processor 7 generated grids in 7 domains.

Figure 12 again, indirectly, shows the effectiveness of dynamic load balancing. All the processors are kept busy, independent of the number of partitions for which they generate grids. As an example, although processor 7 generated grids for 7 domains, it computed for almost the same length of time as processor 21 that generated just two domains.

To demonstrate the flexibility of the geometrical partitioning approach, this same mesh was generated using 32, 16, 8 and 2 processors. Table II shows the times taken.

The time required to simulate one cycle of the wave, using 32 processors, was 8 h. To complete the simulation on a machine with 1024 processors would take approximately 4.5 days.

Table I. Statistics of the unstructured mesh.

Geometry	No. of surfaces	No. of curves
	94	188
Mesh	No. of partitions	No. of processors (R12,000 400 MHz)
	128	32
Surface mesh	No. of nodes	No. of triangles
	1 834 328	3 668 652
	Time for generation	
	1.6 h	
Volume mesh	No. of nodes	No. of tetrahedra
	44 078 548	236 356 076
	Size of volume mesh file	Size of communication data file
	5.1 GB	0.23 GB
	Time for generation	
	18 h	
Communication	Boundary faces	
	9 068 482	

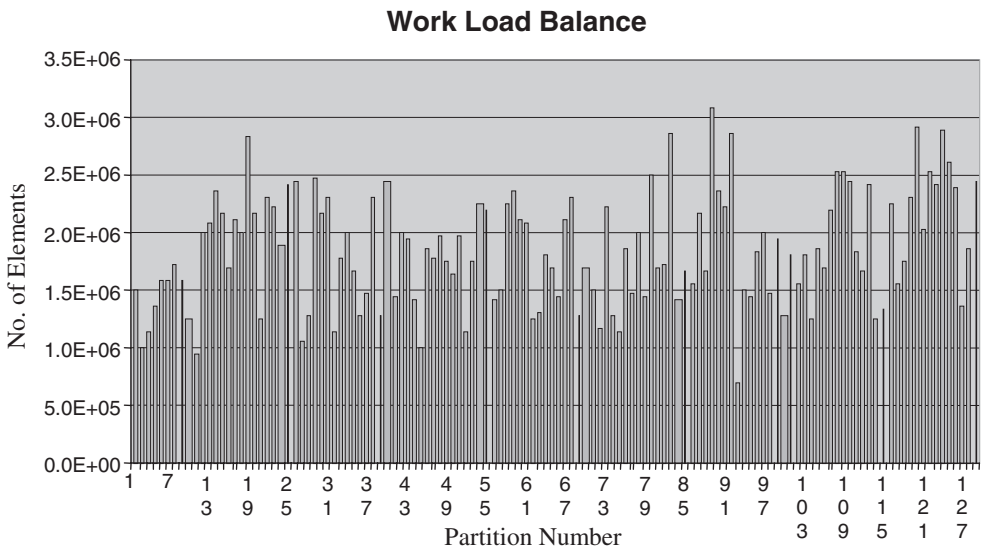


Figure 10. Number of elements generated in each partition.

The mesh and solution data are 7.63 Gbytes in size. This data in 128 partitions across 32 processors can be visualized using the parallel visualization tool ViPar. To load and display the surface data took 11 min 50 s. To produce a cutting plane anywhere in the domain took 2 min 14 s. Figure 13 shows the mesh elements coloured by the partition, in which they reside, and Figure 14 shows the wave propagation after one cycle.

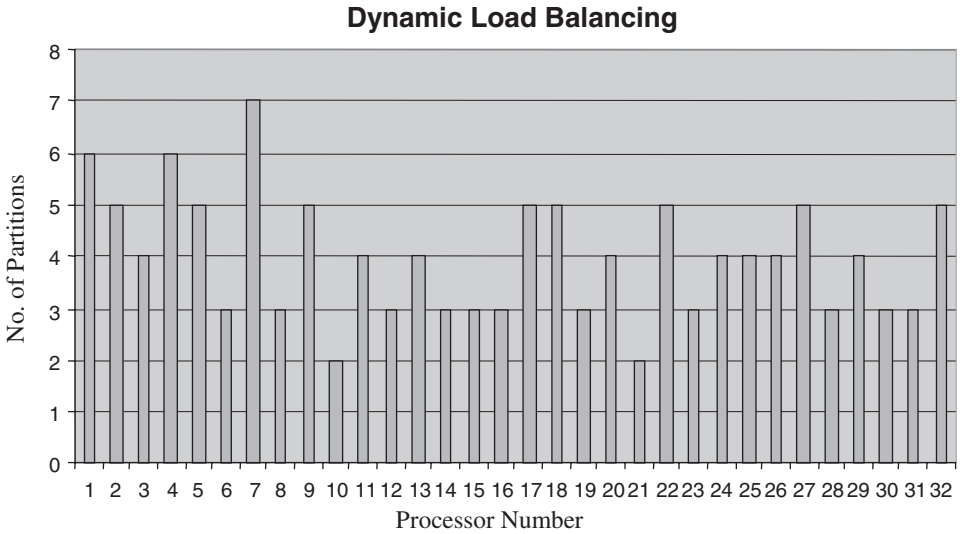


Figure 11. Number of partitions generated per processor.

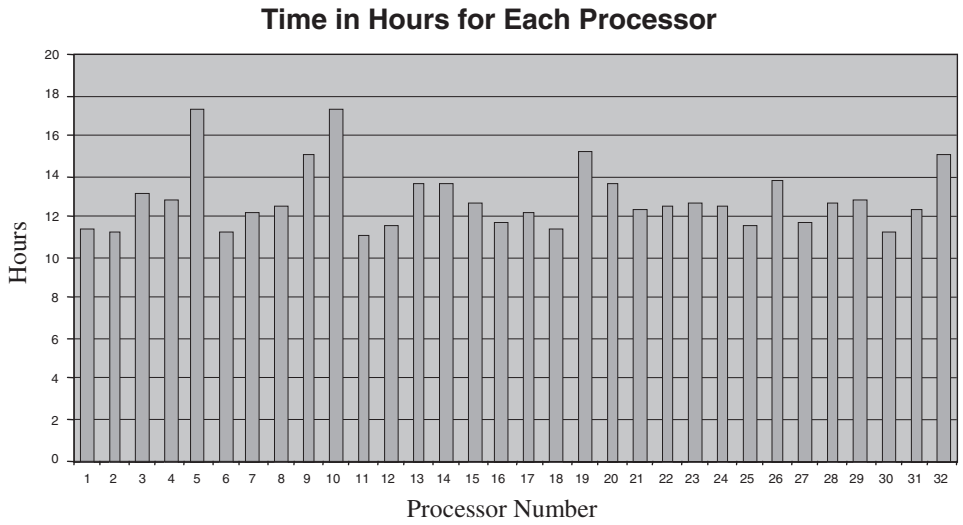


Figure 12. Number of hours computing per processor.

Table II. Time taken to generate the mesh on a different number of processors.

Number of processors	Time (h)
32	18
16	24
8	40
2	256

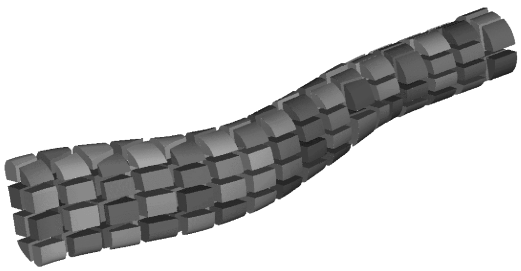


Figure 13. Aerospace duct divided into 128 partitions.

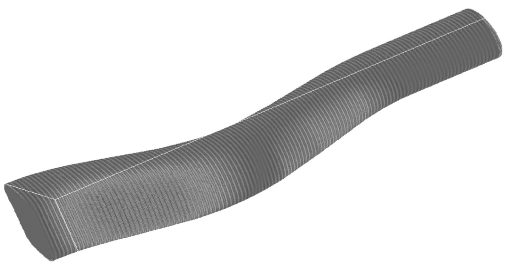


Figure 14. CEM solution after 1 cycle.

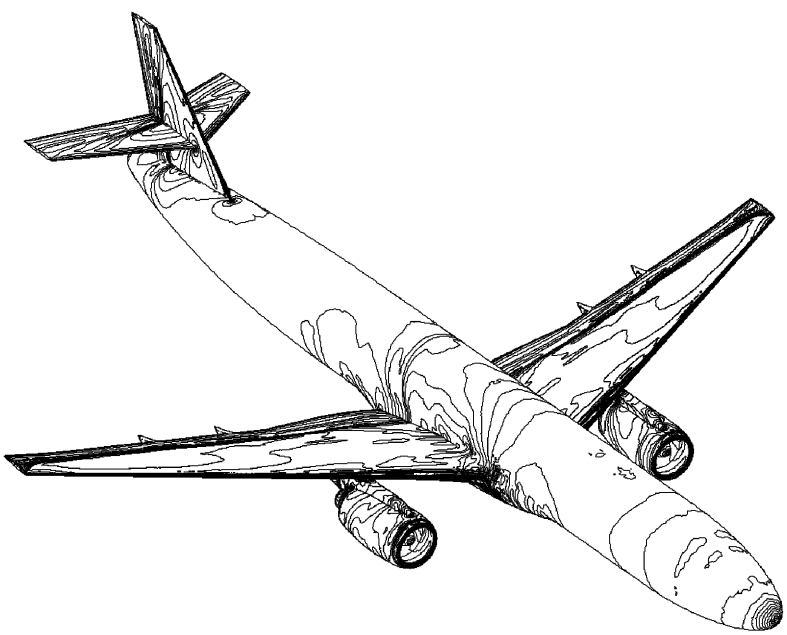


Figure 15. Surface contours of pressure are shown on the aircraft after mesh refinement.

Table III. Statistics of the unstructured mesh.

Geometry	No. of surfaces	No. of curves
	68	161
Mesh	No. of partitions	No. of processors (R12,000 400 MHz)
	8	8
Surface mesh	No. of nodes	No. of triangles
	287 778	575 552
Volume mesh	No. of nodes	No. of tetrahedra
	5 424 167	33 885 651
	Size of volume mesh file	Size of communication data file
	694 MB	1.2 MB
Adapted mesh	No. of nodes	No. of tetrahedra
	19 944 869	119 681 529
	Boundary faces	
	1 448 014	
	Size of volume mesh file	
	1.41GB	

3.2. Computational fluid dynamics

As an illustration of a large-scale computational fluid dynamics computation, the inviscid flow around a complete aircraft is considered with the application of mesh refinement. An unstructured mesh of tetrahedra was generated in parallel. A solution was obtained and then the mesh refined (Figure 15). Table III provides some data on the meshes generated.

This provides an illustration of the size of meshes that can be generated for an aircraft geometry. Work is in-hand to use these techniques to perform a systematic study of adaptive refinement, where the initial mesh is the order of 5 million elements and subsequent multiple levels of refinement increase the mesh to the order of 100 million elements.

4. SUMMARY

Unstructured grid technology for computational fluid dynamics and computational electromagnetics has been enhanced to maximize the efficient use of parallel computer hardware. All steps of the computational cycle have been parallelized including data visualization, mesh generation, solution algorithms and mesh adaptation. These developments now enable large-scale simulations to be performed on 'departmental' computers.

ACKNOWLEDGEMENTS

The authors wish to thank the European Union for funding the JULIUS project (Esprit 25050) under which some of this work was undertaken. B. Larwood would also like to thank EPSRC for financial support. The aerospace duct test case was kindly provided by BAE Systems, Sowerby Research Centre.

REFERENCES

1. Morgan K, Brookes PJ, Hassan O, Weatherill NP. Parallel processing for the simulation of problems involving scattering of electromagnetic waves. *Computer Methods in Applied Mechanics and Engineering* 1998; **152**: 157–174.
2. Said R, Weatherill NP, Morgan K, Verhoeven NA. Distributed parallel Delaunay mesh generation. *Computer Methods in Applied Mechanics and Engineering* 1999; **177**:109–125.
3. Morgan K, Weatherill NP, Hassan O, Brookes PJ, Said R, Jones JW. A parallel framework for multi-disciplinary aerospace engineering simulation using unstructured meshes. *International Journal for Numerical Methods in Fluids* 1999; **31**:159–173.
4. Manzari MT, Hassan O, Morgan K, Weatherill NP. Turbulent flow computations on 3D unstructured grids. *Finite Elements in Analysis and Design* 1998; **30**:353–363.
5. Weatherill NP, Turner-Smith EA, Marchant MJ, Hassan O, Morgan K. An integrated software environment for multi-disciplinary computational engineering. *Engineering Computations* 1999; **16**(8):913–933.
6. Weatherill NP, Hassan O. Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering* 1994; **37**: 2005–2039.
7. Morgan K, Peraire J, Peiro J, Hassan O. The computation of three-dimensional flows using unstructured grids. *Computational Methods in Applied Mechanics and Engineering* 1991; **87**:335–352.
8. Gropp W, Lusk E, Skjellum A. *Using MPI*: *Portable Parallel Programming with the Message Passing Interface*. MIT Press: Cambridge, MA, 1994.
9. Geist A, Beguelin A, Dongara J, Jiang W, Manchek R, Sunderam V. *PVM3 User's Guide and Reference Manual*. Oak Ridge National Laboratory: Oak Ridge, TN, 1994.
10. Hu YF, Blake RJ. An optimal dynamic load balancing algorithm. *PrePrint DL*-P-95-011, Daresbury Laboratory, 1995.
11. Jones JW, Weatherill NP. The visualization of large unstructured grid data sets within a parallel problem solving environment. *Applied Mathematical Modelling*, submitted for publication.

