INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids **31**: 159–173 (1999)

A PARALLEL FRAMEWORK FOR MULTIDISCIPLINARY AEROSPACE ENGINEERING SIMULATIONS USING UNSTRUCTURED MESHES

K. MORGAN*, N.P. WEATHERILL, O. HASSAN, P.J. BROOKES, R. SAID AND J. JONES

Department of Civil Engineering, University of Wales Swansea, Swansea, SA2 8PP, UK

SUMMARY

High performance parallel computers offer the promise of sufficient computational power to enable the routine use of large scale simulations during the process of engineering design. With this in mind, and with particular reference to the aerospace industry, this paper describes developments that have been undertaken to provide parallel implementations of algorithms for simulation, mesh generation and visualization. Designers are also demanding that software should be easy to use and, here, this requirement is addressed by embedding the algorithms within a convenient computer environment. This environment allows for the integration of arbitrary application software, which enables the development of a multidisciplinary engineering analysis capability within a unified computational framework. Copyright © 1999 John Wiley & Sons, Ltd.

KEY WORDS: parallel framework; aerospace engineering; unstructured meshes

1. INTRODUCTION

Computational methods applied within an engineering design environment must be easy to use, enable rapid problem set-up, be fast and efficient and provide consistent and accurate results. These are major challenges, particularly in the aerospace industry, where the simulations which are attempted are generally large computationally and the tasks involved in data preparation and analysis are complicated and time consuming [1].

With the advent of fully automatic mesh generators [2,3], unstructured mesh methods have proved particularly attractive for addressing the problems associated with pre-processing for complex aerospace configurations. However, to meet the simulation requirements in areas such as aerodynamics and electromagnetics, the execution time requirements of unstructured mesh solvers need to be reduced. This is particularly important as the problem sizes increase and the emphasis moves towards computationally demanding multidisciplinary analysis. When the generation of very large grids is attempted with current mesh generators, it is found that major memory demands are placed on the computer system. Ways need to be determined to reduce these memory demands, with the intention of enabling large grids to be generated with a moderate computational capability. A further requirement is for the introduction of new methods that will allow the analyst to quickly view results computed on very large grids.

* Correspondence to: Department of Civil Engineering, University of Wales Swansea, Swansea, SA2 8PP, UK.

CCC 0271–2091/99/170159–15$17.50

In this paper, we illustrate an approach that addresses these problems by utilizing high performance parallel computing. Standard message passing libraries are used to produce parallel implementations of an unstructured mesh generator, of aerodynamic and electromagnetic solvers and of a visualization capability. These parallel codes are developed for use on a variety of parallel platforms, such as distributed workstations, shared memory computers or machines such as the CRAY T3D. To enable the advocated approach to be readily applied within an industrial context, the procedures which have been developed have been included within a parallel simulation user environment (PSUE) [4,5]. The PSUE is an enhanced graphical user capability for complex and multiple problem definition, using unstructured meshes, and provides access to all the tools required for pre- and post-processing. This environment is not application specific and arbitrary simulation software can be readily integrated, thus enabling multidisciplinary analyses. These features ensure that the PSUE has the potential to provide an all encompassing environment for computational engineering.

2. PARALLEL UNSTRUCTURED MESH SOLUTION ALGORITHMS

The solution of real-world aerospace engineering problems frequently requires the use of very large meshes [6]. For example, the computation of steady turbulent viscous flow over a complete aircraft will require a minimum of 10 to 20 million tetrahedral elements, while 100 million elements would currently be a typical minimum for a practically interesting time domain electromagnetic wave scattering simulation about the same configuration. Parallel computing platforms, with their current performance characteristics and their proposed future enhancements, offer the possibility of obtaining the solution to such large problems in realistic elapsed time scales. However, the simulation software has to be suitably parallelized before this power can be accessed.

2.1. Serial unstructured mesh solution algorithms

For computational aerodynamic simulations, the governing equations are taken to be the Favre averaged Navier–Stokes equations for three dimensional compressible flow, with the addition of a k–ω two equation turbulence model [7]. The resulting equations are discretized in space using a Galerkin approximate variational formulation [8], with a piecewise linear representation for the solution over a general tetrahedral mesh. In the computational implementation [9], the unstructured mesh is represented in terms of an edge based data structure, as this minimizes the memory requirements of the algorithm [10]. The steady state solution is obtained by using an explicit multistage procedure to advance the solution in time, with added acceleration devices[11]. Stabilization is achieved by the addition of artificial dissipation, constructed in the JST manner along each edge of the mesh [12].

Problems in the area of computational electromagnetics are governed by Maxwell's equations. Although the solution of these equations is often approached in the frequency domain [13], it is the time domain method of solution that is adopted here [14]. This approach means that the governing equations can be expressed in a conservative form and that it is then possible to apply, with minimum modification, solution algorithms that are very similar to those applied to the simulation of aerodynamic flows. An edge based implementation of a Galerkin approximate variational procedure is applied and the solution is advanced in time by a forward difference explicit method. Stabilization is achieved by the use of a Lax–Wendroff flux function on each edge [15].

2.2. Domain decomposition

The first step in achieving a parallel implementation of these solution algorithms is the decomposition of the mesh which is to be employed into a number of sub-domains. For present purposes, the computational domain is represented in terms of an unstructured mesh of linear tetrahedral elements and such a mesh can be decomposed by recursive spectral bisection (RSB) [16]. The implementation adopted provides a colouring of the nodes in the mesh into an appropriate number of sub-domain groups and generally produces well-balanced sub-domains, with low communication requirements. This should result in good parallel performance of any parallel solution algorithm. However, the implementation of the method is expensive in terms of computer memory demands, e.g. a CRAY YMP-EL with 256 MWords of main memory can be used to decompose meshes consisting of less than 8 million elements. In an attempt to increase the size of mesh that can be handled, an alternative direct partitioning algorithm based upon bandwidth minimization [17] has also been investigated. In this approach, a connection matrix is constructed from the node association information defining the unstructured mesh. This matrix indicates the sparsity distribution of any global matrix that would be formed during an unstructured mesh assembly process. A standard bandwidth minimization is employed on the nodal numbering. The nodes are coloured sequentially, with the sub-division between the colours being determined by the accumulated number of connected edges. This approach requires less memory than RSB and can be used to decompose meshes of up to sixteen million elements on a CRAY YMP-EL with 256 MWords of main memory. When the mesh has been sub-divided, communication data structures are defined which enable inter-domain transfer of information.

2.3. Parallelization philosophy

In the sequential version of the solution algorithms, with the edge data structure, the main computational work is in the form of GATHER and SCATTER operations between nodes and edges. Nodal quantities are accumulated by looping over all the edges within the mesh.

In the parallel implementation of the algorithms, computations within the sub-domains are performed on different processors of the computer platform and data is transferred between processors by using standard message passing libraries, such as MPI or PVM. For these explicit finite element based solution procedures, a high level of data abstraction can be achieved and this is preferable to a low level of parallelization, where it is necessary to treat parallelization at an algorithm level. The approach requires the identification of the different types of data and data structures involved in the formulation. Working with these abstractions, and identifying the parts of the procedure where communication takes place, it is possible to build the necessary data structure for communication and use it for all codes based on similar sequential data structures [18].

For the parallel algorithms, with the sub-divided mesh, edges are regarded as being owned by only one domain and are not duplicated. Interior edges and communication edges are defined for a typical sub-domain, I. An interior edge for sub-domain I is an edge for which both nodes belong to I. As data locality is achieved during the GATHER process from points to edges, interior edges require no communication. Communication edges are edges for which one node belongs to domain I and the other belongs to domain J. This edge is allocated to sub-domain I if $J > I$. Communication edges will contribute to the accumulation of contributions to nodes in J and, in this case, communication between processors takes place. In a similar way, points are owned by one domain, but duplication is required to enforce data locality. This leads to the requirement for a halo of dummy points. Interior points and

communication points are defined for sub-domain *I*. Interior points are those belonging to *I* which are such that all surrounding edges also belong to *I*. As data locality is achieved during the SCATTER process from edges to points, these points require no communication. Communication points are those that do not belong to *I*, but at least one surrounding edge belongs to *I*. For these points, communication is required. Figure 1 shows, in schematic form, the ownership of nodes and edges at a sub-domain interface.

At the start of a time step, the communication nodes obtain contributions from the communication edges. These partially updated interface nodal contributions are then broadcast to the corresponding nodes in the neighbouring sub-domains. Following a loop over the interior edges, the broadcast information is received and all nodal values are subsequently updated. The sending of the updated values back to the communication nodes completes a time step of the procedure. The procedure is implemented in such a way that it allows computation and communication to take place concurrently. Elements and nodes are locally renumbered within the sub-domains, to help minimize memory cache access times.

This philosophy was initially applied in the parallelization of the computer code for the solution of the compressible Navier–Stokes equations [9,19] and the concept is sufficiently general to enable the same procedures to be directly applied to the code which simulates the transient electromagnetic wave scattering by aerodynamic configurations [20]. An important requirement has been that of ensuring that the procedures employed enable portability across different computers, with the result that the codes are not optimized for performance on a particular platform.

3. UNSTRUCTURED MESH GENERATION

During the last decade, there have been major developments in unstructured mesh generation algorithms, both in terms of speed of generation and mesh quality. Today, meshes of 2 to 3 million elements may be routinely generated on workstations and on PCs, using Delaunay based procedures [2,3]. However, the computer memory demands of these algorithms mean that it is necessary to devise new procedures, if meshes consisting of up to, and beyond, 100 million elements are to be routinely generated. One method of accomplishing this is to incorporate the unstructured mesh generation process within a parallel framework.

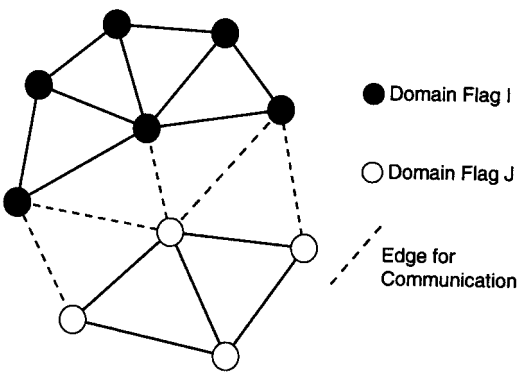


Figure 1. Ownership of nodes and edges at the interface between two sub domains.

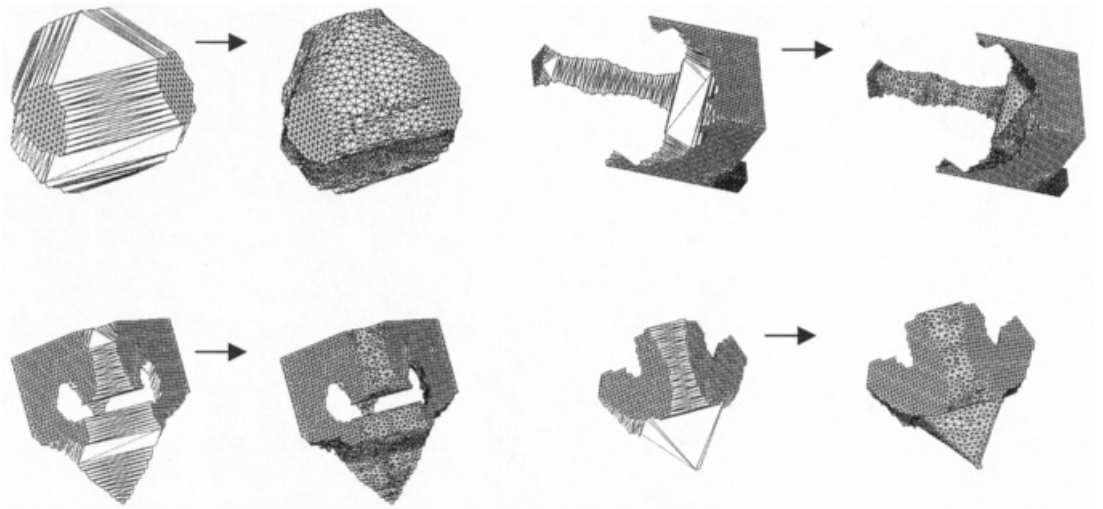


Figure 2. Construction of the surface grids for an example in which four sub-domains are employed.

The approach adopted [21] employs a geometrical decomposition of the domain, followed by the use of different computer processors to generate the meshes within each sub-domain. As a first step, a surface triangulation of the boundary of the computational domain is produced and a volume mesh is created by connecting the corresponding boundary nodes by a Delaunay algorithm. A modified greedy algorithm is employed to produce a decomposition of this initial discretization into n sub-domains, according to an equal volume criterion, where the value of n can either be selected by the user or determined by the resources available on the parallel computer platform. The boundary surfaces between the sub-domains are identified and a smooth triangulation of these surfaces is achieved. This step, which is performed in parallel, is the most challenging within the procedure, for given a surface consisting of planar faces from the initial tetrahedra, a smooth triangulation with mesh point density consistent with the defined mesh control function has to be generated. The approach that is followed [22] allows for point insertion, edge swapping and smoothing to be applied, as illustrated in Figure 2.

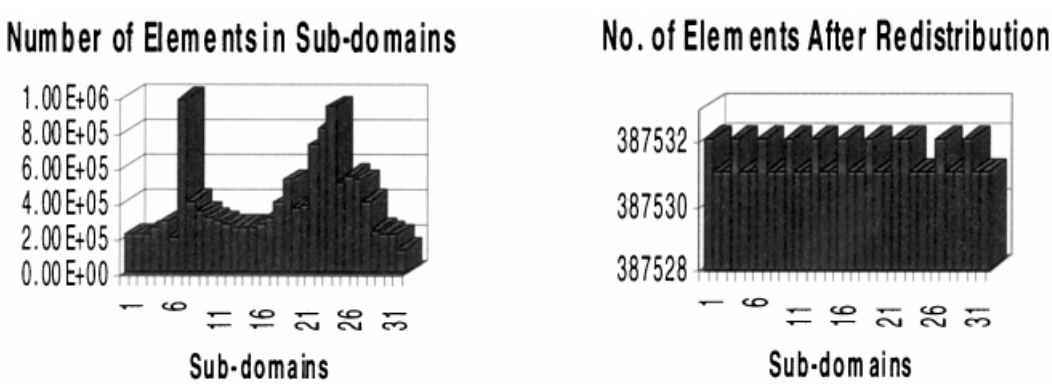
A master processor distributes the n sub-domains to the m worker processors, utilizing message passing in the form of MPI, and meshes are generated on the worker processors using a sequential Delaunay algorithm. When the mesh generation is complete, the sub-domain meshes can be distributed on to different platforms, or collected together on the master. If the number of sub-domains, n, is the same as the number of workers, m, this forms a static load balancing implementation and the efficiency is effectively governed by the length of time it takes to mesh the largest sub-division. However, if $n > m$, this is termed dynamic load balancing and this can be a more efficient process computationally. It should be observed that this approach is applicable to both shared and distributed memory machines, so that meshes of arbitrary size can be generated on computers with very modest memory. Using this method, large meshes have been generated for use in electromagnetic wave scattering simulations. Table I shows the data profile of a mesh of 12 million elements that has been generated in this fashion. Following generation, the elements need to be redistributed to improve the balancing between the sub-domains, before the mesh can be used in practice.

4. VISUALIZATION

Visualization of key data can prove to be a major problem when a simulation is attempted on a very large mesh. To illustrate the difficulties that may be encountered, consider the mesh generation procedure. This can be regarded as being based upon three distinct phases, which may be termed specify, generate and evaluate. In the specify phase, the user interacts with a geometry and specifies all the required input data. In our implementation, this phase utilizes advanced algorithms embedded within an easy-to-use graphics user interface (GUI). When all the required input parameters are specified, the mesh is generated using relatively high compute intensive algorithms with little user interaction. The evaluate phase involves the user in evaluating the suitability and quality of the generated mesh. This is often user interactive intensive, using statistical analysis of mesh quality and visualization of regions in the domain where poorly formed elements may reside. Again, in the evaluate phase, the role of a GUI is important. With current mesh generation algorithms, it is often found that the time required for the generate phase is small compared with the time spent in the specify and evaluate phases.

Typically, using a Delaunay generator, an unstructured mesh of 2 million elements is generated in less than 30 min on a modest workstation. However, the specify and evaluate phases may take several hours, or even several days, depending upon the complexity of the geometry. Experience suggests that the key to effective interaction at the specify and evaluate phases is the response time for the manipulation of the data on the screen of the workstation. For small data sets, it is easy and efficient to specify the data necessary to control the density of the mesh in different regions of the domain and this can normally be accomplished in a few minutes. The problem arises with large data sets, when the interaction time between the user and the computer graphics is excessive. It can be difficult to move objects on the screen and the user and the graphics become disconnected, causing frustration and very inefficient working. A similar situation arises in the evaluate phase. With large data sets, it becomes difficult to visualize aspects of mesh quality and to interact with the generated mesh. This type of problem is also encountered once solution data has been generated, when it is critical to be able to extract key variables and present the data in a meaningful way. To remove such

Table I. The characteristics of a grid of 12 million elements generated in parallel and the effect of redistributing the elements



bottlenecks, special algorithms for visualization on parallel computer platforms need to be developed.

In our work, the basic approach adopted for parallel visualization is based upon the master–slave concept. Communication between the master and the slaves is performed using an in-house communications library that utilizes high-speed sockets [23]. MPI is deemed to be inappropriate here, as it does not have facilities for real-time systems. The master is typically executed on a graphics workstation, while the slaves are executed on either a network of workstations, on a parallel machine or on a combination of both. All rendering on the screen is performed solely by the master, which only stores the primitives that are to be drawn and does not store meshes. To maximize communication and computation, multithreading is used. The main thread deals with the interaction with the user, whilst the slaves are started and stopped as and when data is ready to be received from the master. This allows the user to continue to manipulate data on the screen in parallel with the arrival of new data. The data is spread across the slaves, which perform no rendering. The domain decomposition is achieved using a greedy algorithm, which is found to provide an adequate data decomposition.

5. THE PARALLEL SIMULATION USER ENVIRONMENT (PSUE)

The PSUE is designed as a software environment which incorporates the tools required for pre- and post-processing of engineering simulations and which allows the integration of arbitrary applications software, to capitalize on these tools [4,5]. The environment provides a framework for reducing the time required for problem definition and for post-processing, whilst the unified environment is ideal for multidisciplinary design applications, as the data is handled in one consistent format. The PSUE is designed to hide many aspects of computational engineering which are not of prime interest or relevance to the engineer, e.g. the setting-up and subsequent execution of an application on a parallel platform. A major design philosophy of the PSUE is that modules are either owned by the PSUE or by the user. Modules owned by the PSUE are termed generic modules and are closely coupled, through common data structures, into the environment. Modules owned by the user are integrated into the PSUE through pre-defined format data channels, which take the form of file, pipe or socket transfer. This framework allows users to capitalize on the generic functionality of the PSUE, while enabling them to also use their own modules. A guiding principle throughout the development of the PSUE has been that the efficient performance of the system on large unstructured grids is of paramount importance. Hence, particular attention has been given to memory management, efficient algorithms and the effective use of high performance computing and networking. The ability to conduct parallel processing from inside the environment was deemed to be essential.

5.1. Functionality of the PSUE

As the PSUE is a framework within which arbitrary modules can be integrated, it is inappropriate to be too prescriptive in terms of its functionality. However, in line with the design philosophy, the PSUE contains generic modules that provide a set of working tools. A general schematic of these modules is given in Figure 3.

The primary role of the workstation interface is to provide visual validation of each user action and to guide the user through the system. The interface is based upon X, OSF/Motif

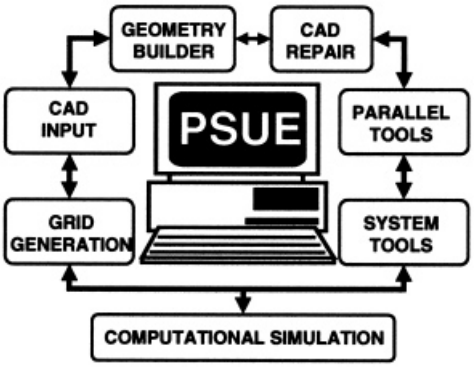


Figure 3. A general schematic of the modules available within the PSUE.

and OpenGL library routines [24–26] for UNIX based platforms. Basic geometry creation and manipulation can be performed and this enables the building of outer boundaries for computational domains and the prescription of the data necessary to control the grid density.

The time required to prepare geometries for mesh generation is presently a major concern within industrial groups. It is frequently the case that the format of geometries inside CAD packages does not conform to the format required for mesh generation. Holes, gaps, overlaps, intersecting surfaces are issues that must be addressed before a grid can be generated on a surface. It should be noted that, in this context, the issue of the validity of a geometry is not a well-defined concept. For example, a detailed geometrical model of the fuselage of an aircraft will include a valid gap around a door, but such detail may not be required for a typical engineering simulation. However, it is a valid geometrical feature, but a feature that would inevitably cause a problem within the grid generation procedure. This case is to be contrasted with that of two surfaces that meet along an edge and leave a hole or a gap. In this case, the geometry is invalid. Both these cases require the use of geometrical tools to prepare the geometry for grid generation. Such tools must be easy to use and capable of dealing with the wide variety of problems which can be encountered. The PSUE has the capability to detect topological and geometrical errors such as edges lying out of faces or gaps between two faces. A list of inconsistencies is provided to the user and, using repairing functionality, the model can be interactively modified in order to repair it prior to grid generation.

The grid generation modules available within the PSUE include the ability to generate 2D planar grids, surface grids of triangles and volume grids of tetrahedra [2,3]. The grids are linked to grid quality analysis modules that provide statistical data, including histograms. Mesh cosmetic operations, such as edge and face swapping, may be applied to improve the quality of a mesh after it has been generated. When a solution has been computed on a given mesh, a mesh adaptation procedure is provided which is capable of selectively performing h-refinement of the mesh, according to some adaptation or error indicator [27].

A key feature of the PSUE is the provision of a set of computing tools. The parallel tool functionality provides a framework for preparing and executing applications on a variety of computer platforms. The basic philosophy is that the PSUE may be running on a workstation of modest computational capability. However, there are tasks, within the pre- and post-processing and in application execution, which require more computer power. To

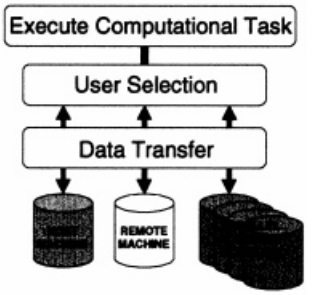


Figure 4. The computer platform options available within the PSUE.

alleviate this potential bottleneck, the user has, at the execution of any process, the option of defining the computer platform appropriate to the required task. Figure 4 shows, in schematic form, these basic options. The remote execution of any task within a generic module of the PSUE is automatic. The data that is required is automatically transmitted to the external computer platform, is executed and appropriate data transmitted back to the host.

The parallel tools module provides users with the flexibility to employ, through a system call, one of the domain decomposition strategies. The environment provides a basis for job preparation and execution and the ability to use XPVM, PVM, MPI has been incorporated. The PSUE offers two options for establishing a parallel platform. These are setup and default setup. Under setup, four options are available: single processor, multi-processor, distributed networks and remote platform. Single processor, multi-processor and distributed networks are constructed using XPVM. Remote platform is used, for example, with an ftp link integrated into the PSUE. Data files and analysis modules are transferred to any remote machine where upon compilation and execution can be performed. Applications can be executed once the user has defined the network setup and the correct message passing environment. Monitoring the performance of a parallel code is a significant task when load balancing and efficiency are the prime targets of the software developer. The PSUE offers two performance monitoring facilities as generic, namely XPVM and PARAGRAPH. However, any performance monitoring tool can be added through the proper PSUE script file. The functionality offered by the parallel computer module is illustrated schematically in Figure 5.

At present, the comprehensive interrogation of analysis data is performed by external packages, such as AVS, Ensight, etc. These external packages are initiated from the PSUE and data transfer is in the form of temporary files. For packages that allow automated startup, such as AVS, this can be utilized so that the user may immediately visualize the data by a minimal number of key inputs. For other packages, temporary files are stored and the user

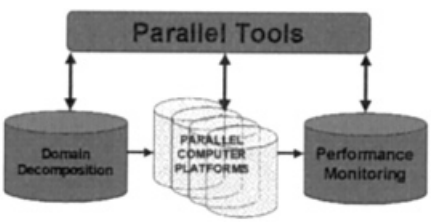


Figure 5. The functionality provided by the parallel tools module.

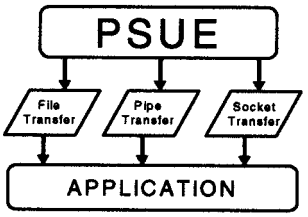


Figure 6. Mechanics of data transfer between the PSUE and an external application.

notified of their type and location. If users require their own analysis package to be coupled into the PSUE, then data may be passed using pipes and sockets as well as the temporary file system. In the case of a coupled package, the user is responsible for any automated startup. The parallel visualization capability has been incorporated into the PSUE so that, for large datasets, users are able to manipulate images using parallel processing and thus ensure effective interaction with data.

5.2. Integration of applications into the PSUE

The PSUE has an architecture that enables arbitrary applications to be coupled into the environment. User software for engineering simulation, CAD repair, grid generation, etc. can be directly coupled into the PSUE as software applications. Applications within the PSUE are customized using an application definition data file. In this file, the user specifies the location of the application module. In the case of a large number of applications being requested, the PSUE automatically sets up a scrolling window containing the required application widgets. The user is responsible for the application window. Data transfer for coupled applications may be achieved by using file, pipe and socket transfer. On requesting a coupled application, a dialog requests a decision from the user as to which type of transfer to make. This is known as the PSUE handshake. When file transfer is selected, the dialog contains a series of file types with varying attributes and structure. On selection of one of these file types, the PSUE produces a file and informs the user of its location. The application itself is then executed and the user is responsible for loading the relevant file. On return to the PSUE, a file saved in the application may be loaded back into the PSUE. When pipe transfer is selected, the application itself is executed and the immediate standard input is the actual data. This requires the application to accept this data as standard input before it can continue. For a socket transfer, the host must be able to set up the socket connection. Changing one system file on the host is all that is required. On selection, the application can receive or send data via the socket at any time. The data types and structures that are used in the file transfer are the same as the types and structures of the data sent by the pipe and socket transfer methods. These options are illustrated in Figure 6.

6. APPLICATIONS

The software for simulating compressible aerodynamic flows and electromagnetic wave propagation has been integrated into the PSUE. The integration was performed with data transfer utilizing a pipe connection. The geometry builder within the PSUE is used to build an outer boundary for the relevant configurations. Inside the PSUE, the mesh is generated, all flow parameters and boundary conditions are defined and the mesh is partitioned. Boundary

Table II. Performance data for the parallel inviscid flow algorithm on the Silicon Graphics Challenge

NoP	RT	RSU
1	6000	1
2	2500	2.4
4	1009	5.9
8	565	10.6

conditions are defined using the generic boundary condition editor. This enables the user to pick curves, surfaces and volume domains and, by using an existing database of boundary conditions or by allowing the user to define new conditions, to set the appropriate conditions. The user specifies the code to be executed in a script file and then a push button for that application automatically appears in the application window within the PSUE. Data, such as the grid and the boundary conditions, is then passed to the application through the data transfer interface.

A number of examples have been simulated to determine the level of computational performance that can be achieved by the resulting procedures.

6.1. Steady inviscid flow over an aircraft configuration

For the analysis of a steady inviscid flow over a wing, body, pylon, nacelle configuration, the grid employed consists of 7335316 tetrahedral elements. Table II shows the performance of the parallel flow solver on an eight processor Silicon Graphics Challenge computer. Here, NoP denotes the number of processors employed, RT denotes the run time in minutes required to perform 1000 time steps and RSU denotes the relative speed-up obtained. It is clear that the strategy adopted to renumber the nodes and elements within the different partitions is effective and super-linear speed-up is achieved, because of an effective use of the cache memory.

Table III indicates the performance achieved when the solution of the same problem is attempted on a CRAY T3D. Here MinEd, MaxEd, MinPo and MaxPo denote, respectively, the minimum and maximum number of edges and points in any domain. For a mesh of this size, the limited memory available on a single node prevents a computation on one processor. Hence, the performance data is compared to a four processor computation and it is seen that, with 256 processors, an effective speed-up of 185 can be claimed.

The number of time steps required to converge the solution on such a grid is around 2000 which means that the solution is computed on the CRAY T3D in around 30 min, using 256 processors.

Table III. Performance data for the parallel inviscid flow algorithm on CRAY T3D

NoP	MinEd	MaxEd	MinPo	MoxPo	RT	RSU
4	469 771	484 465	665 231	70 130	650	1.00
8	230 459	244 827	33 262	37 357	345	1.88
16	114 139	125 040	16 631	19 674	159	4.09
32	55 144	64 926	8316	10 609	92	7.07
64	26 760	33 475	4158	5930	44	14.77
128	12 826	16 980	2079	3239	27	24.07
256	6189	8762	1040	1822	14	46.42

Table IV. Parallel performance statistics for the turbulent flow code

NoP	MinEd	MaxEd	TNIE	RT	RSU
4	498 389	498 409	3933	1545	1.0
8	249 179	249 205	3093	776	2.0
16	124 550	124 607	2578	404	3.84
32	62 307	62 164	2007	200	7.73

6.2. Steady turbulent flow over a wing/body configuration

The generation of meshes that are appropriate for viscous flow simulations over general aerodynamic configurations is a non-trivial task which is currently receiving considerable attention [28]. With our current capability, a mesh of 244 334 nodes and 1 687 850 elements has been generated for a wing/body configuration and a flow simulation is performed at a free stream Mach number of 0.8, a Reynolds number of 40 million and an angle of attack of 0.5 degrees. The grid partitioning statistics, together with the performance of the flow algorithm on the CRAY T3D, are presented in Table IV. Here, TNIE denotes the total number of communication edges in the mesh.

6.3. Electromagnetic scattering by an aircraft configuration

The first example in this area corresponds to the simulation of scattering of a plane electromagnetic wave by a perfectly conducting complete aircraft. The wave frequency is 100 MHz, the wavelength $\lambda = 3$ m and the length of the aircraft is 6λ. The mesh used for the calculation contains 2 242 682 elements and this is decomposed into partitions using the RSB approach and the bandwidth minimization algorithm. Table V shows the performance of the parallel algorithm on the meshes decomposed using RSB, whilst Table VI demonstrate the performance of the parallel algorithm on the meshes decomposed using the bandwidth minimization approach.

It can be observed that the performance figures are not as good as those obtained for the aerodynamic simulations. This is because the electromagnetic algorithm involves less computation per time step and, hence, for a mesh of this size, communication plays a more important role. It is also interesting to note that, on this evidence, using the RSB or the bandwidth minimization algorithm for domain decomposition produces little difference in the performance levels that can be achieved.

Table V. Performance data for the parallel electromagnetic algorithm on CRAY T3D

NoP	MinEd	MaxEd	MinPo	MaxPo	RT	RSU
4	549 429	569 181	77 704	83 761	4516	1.0
8	271 198	294 234	38 852	45 304	2232	2.0
16	132 911	148 268	19 426	23 565	1174	3.8
32	64 814	76 339	9713	12 811	613	7.4
64	32 166	38 441	4857	6724	342	13.2
128	14 971	20 659	2429	3833	176	25.7
256	7142	10 701	1214	2179	136	33.2

The decomposition was performed by the RSB approach.

Table VI. Performance data for the parallel electromagnetic algorithm on CRAY T3D

NoP	MinEd	MaxEd	MinPo	MaxPo	RT	RSU
4	560 666	560 673	82 018	82 567	5100	1.0
8	280 314	280 342	40 959	43 245	2634	1.9
16	140 120	140 176	21 058	22 985	1198	4.3
32	69 974	70 093	10 753	13 812	659	7.7
64	34 819	35 052	5577	9182	371	13.7
128	17 082	17 531	2890	6023	252	20.2
256	7776	8770	1496	4315	195	26.2

The decomposition was performed by the bandwidth minimization approach.

To illustrate the computer time requirements of a more practical case, a higher frequency simulation is performed for the same aircraft configuration. The wave frequency is increased to 300 MHz, corresponding to a wavelength $\lambda = 1$ m and an aircraft length of 18λ. The mesh used contained 15182752 elements, 2553495 nodes and 17872347 edges and the computation of 36 cycles of the incident wave required about 3 h on a CRAY T3D with 256 processors.

7. CONCLUSIONS

The paper has summarized work directed at providing a system, for computational engineering in the aerospace industry, which fully exploits parallel computer platforms. To meet the requirement for rapid turn-around times in application areas such as computational aerodynamics and computational electromagnetics, a method for parallelizing explicit unstructured mesh finite element based solution algorithms has been outlined. The approach that has been followed is based upon a high level data abstraction, which reduces the problem of parallelization to that of handling communication of data between edges connecting individual grid partitions. For simulations involving large meshes, a parallel unstructured mesh generator and a parallel visualization framework have also been introduced. To provide the analyst with a general, flexible and easy to use computer environment from which problem definition, application execution and post-processing can be undertaken, the developments have been incorporated within a parallel simulation user environment. The performance of the complete system has been demonstrated on realistic aerospace engineering problems, employing meshes consisting of up to 16 million tetrahedral elements.

ACKNOWLEDGMENTS

The authors are grateful to the EC for providing financial support for the development of the PSUE under the CAESAR project of ESPRIT. This development also benefited from collaboration with the other key partners in the CAESAR project, in particular IPK (Germany), the Sowerby Research Centre of British Aerospace (UK), DASA (Germany), Odense (Denmark) and BASF (Germany). The development team of the PSUE also included Mr. M. Sotirakos, Mr. E. Turner-Smith and Dr Yao Zheng. The authors wish to thank the UK Engineering and Physical Sciences Research Council for providing access to the CRAY T3D at the Edinburgh Parallel Computer Centre, under research grant GR/K42264, and for supporting the related parallelization activity under research grants GR/J12321, GR/L18860

and GR/J91234. Support for the work on computational aerodynamics was provided in part by British Aerospace Defence Ltd. and British Aerospace (AIRBUS) Ltd.

REFERENCES

1. D.P. Hills, 'Numerical aerodynamics: past successes and future challenges from an industrial point of view', in J.-A. Desideri *et al.* (eds), *Computational Methods in Applied Sciences '96—Invited Lectures and Special Technological Sessions at the ECCOMAS Conferences*, John Wiley & Sons, Chichester, 1996, pp. 166–173.
2. N.P. Weatherill and O. Hassan, 'Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints', *Int. J. Numer. Methods Eng.*, **37**, 2005–2039 (1994).
3. J. Peraire and K. Morgan, 'Unstructured mesh generation including directional refinement for aerodynamic flow simulation', *Finite Elements Anal. Design*, **25**, 343–356 (1997).
4. M.J. Marchant, N.P. Weatherill, E.A. Turner-Smith, Y. Zheng and M. Sotirakos, 'A parallel simulation user environment for computational engineering', in B. Soni, J. Häuser, J.F. Thompson and P. Eiseman (eds), *Proceedings of the 5th International Conference on Numerical Grid Generation in Computational Field Simulation*, Mississippi State University Press, 1996.
5. N.P. Weatherill, M.J. Marchant, E.A. Turner-Smith, Y. Zheng, M. Sotirakos and O. Hassan, 'The design of a graphical user environment for multi-disciplinary computational engineering', in J.-A. Desideri *et al.* (eds), *Numerical Methods in Engineering '96—Proceedings of the 2nd ECCOMAS Conference on Numerical Methods in Engineering*, John Wiley & Sons, Chichester, 1996, pp. 810–818.
6. N.A. Verhoeven, R. Said, N.P. Weatherill and K. Morgan, 'Delaunay mesh generation on distributed parallel platforms', in B.H.V. Topping (ed.), *Proceedings of EURO-PAR97: Parallel and Distributed Computing for Computational Mechanics—Preprocessing and Solution Procedures*, Saxe-Coburg Press, Edinburgh, 1999 (in press).
7. D.C. Wilcox, *Turbulence Modelling for CFD*, DCW Industries Inc, La Canada, CA, 1993.
8. K. Morgan and J. Peraire, 'Unstructured grid finite-element methods for fluid mechanics', *Reports Progr. Phys.*, **61**, 569–638 (1998).
9. M.T. Manzari, O. Hassan, K. Morgan and N.P. Weatherill, 'Turbulent flow computations on 3D unstructured grids', *Finite Elements Anal. Design*, **30**, 353–363 (1998).
10. K. Morgan, J. Peraire, J. Peiró and O. Hassan, 'Unstructured grid methods for high speed compressible flows', in J.R. Whiteman (ed.), *The Mathematics of Finite Elements and Applications—Highlights 1993*, John Wiley & Sons, Chichester, 1994, pp. 215–241.
11. J. Peraire, J. Peiró and K. Morgan, 'Finite element multigrid solution of Euler flows past installed aero-engines', *Comput. Mech.*, **11**, 433–451 (1993).
12. A. Jameson, W, Schmidt and E. Turkel, 'Numerical simulation of the Euler equations by finite volume methods using Runge–Kutta time stepping schemes', *AIAA Paper 81-1259*, 1981.
13. L. Harari, K. Grosch, T.J.R. Hughes, M. Malhotra, P.M. Pinsky, J.R. Stewart and L.L. Thompson, 'Recent developments in finite element methods for structural acoustics', in M. Kleiber and E. Onate (eds), *Archives of Computational Methods in Engineering 3*, 1996, pp. 131–309.
14. A. Taflove, 'Re-inventing electromagnetics: supercomputing solution of Maxwell's equations via direct time integration on space grids', *AIAA Paper 92-0333*, 1992.
15. K. Morgan, O. Hassan and J. Peraire, 'A finite element procedure for electromagnetic scattering simulations', in F. El Dabaghi, K. Morgan, A.K. Parrott and J. Périaux (eds), *Approximations and Numerical Methods for the Solution of Maxwell's Equations*, Clarendon Press, Oxford, 1998, pp. 313–327.
16. H.D. Simon, 'Partitioning of unstructured problems for parallel processing', *Comput. Systems Eng.*, **2**, 135–148 (1991).
17. C. Greenhough and R.F. Fowler, 'Partioning methods for unstructured finite element meshes', *Report RAL-94-092*, Rutherford Appleton Laboratory, Didcot, 1994.
18. J. Cabello, 'Parallel explicit unstructured grid solvers on distributed memory computers', *Adv. Eng. Software*, **26**, 189–200 (1996).
19. K. Morgan, N.P. Weatherill, O. Hassan, M.T. Manzari, L.B. Bayne and P.J. Brookes, 'Parallel processing for large scale aerospace engineering simulations', in D.R. Emerson, A. Ecer, D. Emerson, J. Périaux, N. Satofuka and P. Fox (eds), *Parallel Computational Fluid Dynamics—Recent Developments and Advances Using Parallel Computers*, Elsevier Science, Amsterdam, 1998, pp. 15–22.
20. K. Morgan, P.J. Brookes, O. Hassan and N.P. Weatherill, 'Parallel processing for the simulation of problems involving scattering of electromagnetic waves', *Comput. Methods Appl. Mech. Eng.*, **152**, 157–174 (1998).
21. N. Verhoeven, N.P. Weatherill and K. Morgan, 'Dynamic load balancing in a 2D parallel Delaunay mesh generator', in A. Ecer, J. Périaux, N. Satofuka and S. Taylor (eds), *Parallel Computational Fluid Dynamics—Implementations and Results Using Parallel Computers*, Elsevier Science, Amsterdam, 1995, pp. 641–648.
22. R. Said, N.P. Weatherill and K. Morgan, 'Distributed parallel Delaunay mesh generation', *Comput. Methods Appl. Mech. Eng.* (1999) (in press).
23. J.A. Jones and N.P. Weatherill, 'Parallel visualisation of computational engineering data', in B.H.V. Topping (ed.), *Advances in Computational Mechanics with High Performance Computing*, Civil-Comp Press, Edinburgh, 1998, pp. 1–9.

24. *X Window System Programming Manuals*, *Volumes 1–6*, O'Reilly & Associates, CA, USA.
25. *OSF/Motif Programmer's Reference*, Prentice Hall, New Jersey, USA, 1991.
26. *Open GL Architecture Review Board—Open GL Reference Manual: The Official Reference Document for Open GL*, *Release 1*, Addison-Wesley, 1992.
27. N.P. Weatherill, O. Hassan, M.J. Marchant and D.L. Marcum, 'Grid adaptation using a distribution of sources applied to inviscid compressible flow simulation', *Int. J. Numer. Methods Fluids*, **19**, 739–764 (1994).
28. J. Peraire and K. Morgan, 'Automatic generation of unstructured meshes for Navier Stokes flows', *AIAA Paper 98-3010*, 1998.

