INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2000; **32**: 903–920

The implementation of an aeronautical CFD flow code onto distributed memory parallel systems

C. S. Ierotheou[a,*], C. R. Forsey[b] and M. Leatham[c]

[a] *Parallel Processing Research Group, University of Greenwich, London, SE18 6PF, U.K.*
[b] *Aircraft Research Association Limited, Manton Lane, Bedford, MK41 7PF, U.K.*
[c] *Oxford University Computing Laboratory, Wolfson Buillding, Oxford, OX1 3QD, U.K.*

SUMMARY

The parallelization of an industrially important in-house computational fluid dynamics (CFD) code for calculating the airflow over complex aircraft configurations using the Euler or Navier–Stokes equations is presented. The code discussed is the flow solver module of the SAUNA CFD suite. This suite uses a novel grid system that may include block-structured hexahedral or pyramidal grids, unstructured tetrahedral grids or a hybrid combination of both. To assist in the rapid convergence to a solution, a number of convergence acceleration techniques are employed including implicit residual smoothing and a multigrid full approximation storage scheme (FAS). Key features of the parallelization approach are the use of domain decomposition and encapsulated message passing to enable the execution in parallel using a single programme multiple data (SPMD) paradigm. In the case where a hybrid grid is used, a unified grid partitioning scheme is employed to define the decomposition of the mesh. The parallel code has been tested using both structured and hybrid grids on a number of different distributed memory parallel systems and is now routinely used to perform industrial scale aeronautical simulations. Copyright © 2000 John Wiley & Sons, Ltd.

KEY WORDS: computational fluid dynamics; hybrid structured and unstructured grids; parallel computing

1. INTRODUCTION

General purpose parallel processing for solving day-to-day industrial problems has been slow to develop. This is due to the lack of suitable hardware from well-established mainstream computer manufacturers, the lack of suitably parallelized applications software and also the lack of standard message passing inter-process communication software to enable portability of the software. The introduction of large-scale parallel computers from most of the major

* Correspondence to: Centre for Numerical Modelling and Process Analysis, University of Greenwich, Woolwich Campus, Wellington Street, London SE16 6PF, U.K.

Received August 1998
Revised February 1999

computer manufacturers has increased the attractiveness of parallel processing for industrial users and work is underway at many sites to parallelize important industrial codes.

The EC funded EUROPORT activity has provided some impetus to this effort, enabling a number of codes to be parallelized in a machine-independent way in a relatively short time scale. The EUROPORT effort was carried out over a 2 year period and was part of the European Commission's ESPRIT III programme. It included the parallelization of nearly 40 industrially relevant codes that covered a broad spectrum of application areas in which high performance computing would be beneficial. This included areas such as fluid mechanics, structural mechanics, crash simulation, forging, animation, chemistry pharmaceuticals, oil, radiotherapy and electromagnetics [1].

This paper discusses one of the aeronautical computational fluid dynamics (CFD) codes included in the EUROPORT programme. The flow solution code, known as ESAUNA, is part of the SAUNA CFD suite developed at the Aircraft Research Association Limited (ARA) under funding from the U.K. Ministry of Defence. The suite allows the novel use of hybrid structured and unstructured meshes in addition to the more common purely block-structured or purely unstructured approach, adopted by most CFD suites. The hybrid approach allows very complex geometries to be accurately defined and solved for, including complete aircraft [2]. The paper also discusses the main issues regarding the porting of the ESAUNA code on distributed memory systems, including the partitioning strategy, the portability issues and the future maintenance and development of the code. Results are shown for test cases that reflect the mesh sizes and computational expense required to perform industrial scale simulations and their performance on a range of parallel platforms.

2. THE SAUNA SUITE

SAUNA is an acronym for structured and unstructured numerical analysis. It is a large CFD suite aimed at computing the steady flow around very complex aircraft configurations, including complete aircraft. The aim of SAUNA is to provide a framework for the complete design and optimization process of complex aircraft configurations. This includes the generation of a mesh to represent the design configuration; the computation of the flow around the geometry; the visualization and interpretation of the results; and the refinement or optimization of the configuration to improve the quality of the design.

In addition to aircraft configurations, the suite has also been used to simulate the flows around other complex configurations, such as ships, submarines, missiles, engine intakes and cars. Depending on the flow regime, in some cases the SAUNA flow solver is used for the simulations, while in others only the grid generation modules are employed with the flow being calculated by other available flow solvers.

The suite has been under continuous development at the ARA for over 10 years under contract to the U.K. Defence Evaluation and Research Agency (DERA).

2.1. The governing Euler equations for aerodynamic flows

The flow solution module ESAUNA (Section 3) solves the three-dimensional Euler or Navier–Stokes equations. The Euler equations describing the conservation of momentum, mass and energy for a compressible inviscid flow can be written using Cartesian tensor form as

$$\frac{\mathrm{d}}{\mathrm{d}t}\iiint_V \mathbf{w}\,\mathrm{d}V + \iint_S \mathbf{H}\cdot\mathbf{n}\,\mathrm{d}S = 0 \tag{1}$$

where $\mathbf{w} = \{\rho u, \rho v, \rho w, \rho E\}^T$ is the vector of conserved variables stored at the vertices of the mesh, V is a volume of surface S, and $\mathbf{n}$ is the unit outward normal to the volume. The flux tensor $\mathbf{H}$ is given by

$$\mathbf{H} = \begin{bmatrix} \rho u & \rho v & \rho w \\ \rho u^2 + p & \rho uv & \rho uw \\ \rho uv & \rho v^2 + p & \rho vw \\ \rho uw & \rho vw & \rho w^2 + p \\ \rho uH & \rho vH & \rho wH \end{bmatrix} \tag{2}$$

The pressure is denoted by p, the density by ρ; the total energy per unit mass is E, the total enthalpy per unit mass is H, and the velocity components are u, v and w. The Navier–Stokes equations describing compressible viscous flow take the same general form, with additional terms to model the viscous stresses in the fluid. The solution to both the Euler and Navier–Stokes equations is obtained by discretizing the domain in some suitable fashion and solving for $\mathbf{w}$.

2.2. Cell-centre and cell-vertex discretization

The solution of the discretized equations is based on a Jameson-type explicit time marching finite volume scheme. Finite volume schemes can be developed using either cell-vertex or cell-centre storage of flow variables. The ESAUNA code uses a cell-vertex storage scheme. In such a scheme, the spatial discretization is achieved by balancing fluxes through the faces of a control volume surrounding each node. There is some evidence that a cell-vertex scheme produces greater solution accuracy than a cell-centred approach [3].

Using a cell-vertex scheme, the computation of the flux balance at all the vertices is obtained by looping over all cell faces of the grid and accumulating flux contributions at the appropriate vertices. Connectivity matrices are used to define the common faces between different connecting polyhedra and thus to correctly perform the flux balancing for each control volume. Non-physical oscillations in the solution are avoided by the introduction of an artificial dissipation term [4,5], sometimes known as artificial viscosity.

Using a cell-centred scheme would involve looping over all faces of the grid with an accumulation of flux contributions at the appropriate cell-centres to complete the flux balancing procedure [3]. A cell-centred scheme is somewhat easier to parallelize due to the less complex connectivity.

3. OVERVIEW OF THE SOLUTION PROCEDURE WITHIN ESAUNA

ESAUNA is the flow solver module of the SAUNA suite and solves Equation (1) above. It is designed to solve either the Euler or Navier–Stokes equations, the latter in conjunction with various turbulence models. Second- and fourth-order dissipation is explicitly added to stabilize the discrete equations (Section 2.2).

The resulting discretized flow equations are a system of first-order ordinary differential equations (ODE), which are integrated in time using a five-stage, fourth-order Runge–Kutta scheme with local time stepping. Convergence to steady state is enhanced using several techniques, including implicit residual smoothing, enthalpy damping and a full approximation storage (FAS) multigrid algorithm (Section 3.2) in which the multigrid smoother is the multistage Runge–Kutta scheme.

Operations in ESAUNA are either cell face based (flux balancing) or cell edge based (dissipation and smoothing). Although there are more edge-based operations, they play a lesser role in the computationally intensive parts of the code since flux balancing is carried out for every stage of the Runge–Kutta time integration scheme, while dissipation and smoothing are carried out only at selected Runge–Kutta stages (Figure 2).

Figure 2 shows that both Euler and Navier–Stokes simulations are performed using the same general procedure. For the Euler equations, residual smoothing is only necessary at stages 1, 3 and 5 of the Runge–Kutta scheme, while for the Navier–Stokes equations residual smoothing is used at every stage. In addition, for the solution of the Navier–Stokes equations, it is necessary to calculate viscous terms as these are needed in the update of the flow variables in the Runge–Kutta scheme.

3.1. Hybrid grid functionality within the SAUNA suite

One standard approach to generating a curvilinear grid to represent the shape of an aerodynamic object is to perform a transformation from a single-block Cartesian grid. Although such an approach has been widely used to represent simple objects, such as a fuselage or a wing, it is somewhat limited when used to model a complex geometry, such as a complete aircraft. There are three commonly used techniques to represent complex geometries.

The first technique is based on the concept of chimera or overlapping grids, where a number of different grid blocks are used to represent different portions of the complete geometry and the blocks are allowed to overlap each other [6]. The complexity in the implementation of this approach lies in the definition of the overlapped interfaces between blocks.

The second technique is the use of a single unstructured mesh to represent the complete geometry [7]. This has a greater degree of flexibility than using a number of structured grids, but suffers from the relatively expensive data storage requirement.

The third approach is based on the connection of structured blocks of grids that do not overlap, however, the interfaces between blocks are defined in an unstructured fashion [2]. This has the disadvantage that the orientation of connecting blocks has to be consistent, thus more effort is needed to ensure this when connecting blocks.

A novel feature of the SAUNA suite is that it is designed to use either block-structured hexahedra and pyramidal grids, unstructured tetrahedral grids or a hybrid combination of

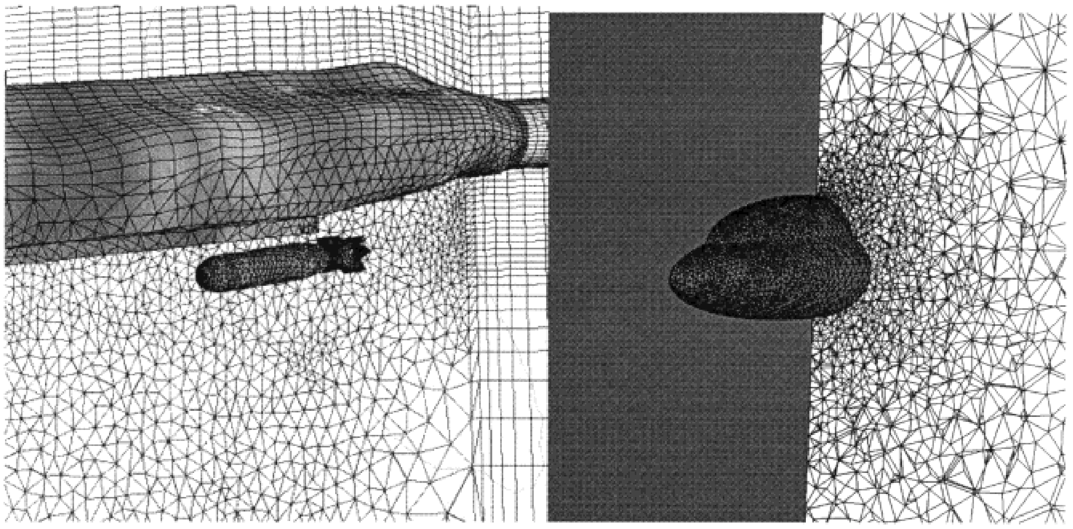
Figure 1. Examples of structured, unstructured and hybrid grids for use with ESAUNA.

both grid types (Figure 1), thus leveraging the best features of all grid types. For the block-structured grids, the approach used is based on the concept of non-overlapping grids, so there is a clean interface between connecting blocks. As mentioned earlier, the connectivity between the blocks is defined using unstructured mesh techniques. This has the added advantage that an unstructured region can be connected to a structured block, thus providing the advantages of both.

3.2. Improving convergence with a multigrid acceleration scheme

A multigrid convergence acceleration technique is used to reduce the computation time for large mesh simulations as well as for viscous flow simulations. The multigrid strategy has been

```
For each multigrid level
      Calculate time step
      Calculate viscous terms (Navier-Stokes simulation only)
      For each Runge-Kutta stage
            If stage = 1,3,5 calculate dissipation
            Calculate flux balance/residual at nodes
            If stage = 1,3,5 smooth residuals (Euler simulation only)
            If stage = 1,2,3,4,5 smooth residuals (Navier-Stokes
            simulation only)
            Update flow variables at nodes
      endfor
endfor
```

Figure 2. Overview of solution procedure within ESAUNA.

implemented for block-structured grids and work is currently underway to extend the use to unstructured meshes [8]. The varying levels of coarseness required for structured blocks are readily obtained by removing alternate grid lines. For example, a fine grid defined as having $nx * ny * nz$ cells has two successive coarser levels defined by $(nx/2) * (ny/2) * (nz/2)$ and $(nx/4) * (ny/4) * (nz/4)$ cells.

The multigrid technique is based on a FAS scheme strategy [9], where the solution on a coarser mesh is prolongated up to the finer mesh and used as a correction to the approximation at the finer mesh. The solution on the fine mesh is restricted to the coarse mesh to provide an initial approximation to the coarse mesh. In practice, the strategy moves from fine mesh to coarse mesh, then from coarse mesh to either an even coarser mesh or back to the finer mesh and so on. Thus, there are a number of cycling strategies that can be employed in moving from fine to coarse mesh and back. The most popular are 'V' and 'W' cycles. In the current implementation of SAUNA, a V cycle is used for the solution of the Euler equations, while a W cycle is used for the solution of the Navier–Stokes equations. The cycles comprise a fine grid and two or three successively coarser grids depending on the complexity of the flow or the size of the total fine grid.

4. DATA STRUCTURES AND STORAGE SCHEME WITHIN ESAUNA

In order to exploit the parallelism present in the ESAUNA flow module, it is necessary to understand the data structures and storage schemes used to represent the flow data. The storage approach used within ESAUNA is based on a one-dimensional array to store the flow variables for the entire fine mesh.

Unstructured grid calculations work directly with this one-dimensional array. For block-structured calculations, a block is extracted from the 'global' one-dimensional array and stored in an equivalent 'local' multi-dimensional array. The local array is then updated and finally, the local array is mapped back to the global one-dimensional array. This process allows the programme to make more efficient use of the available data memory since it is flexible enough to store varying block sizes with no wasted space. However, the disadvantage is that there is a time penalty incurred every time data are copied between the global and local arrays.

Figure 3 shows a very simple representation of two connecting structured blocks. The blocks are assumed to be two-dimensional with the number of cells equal to $nx_1 * ny_1$ and $nx_2 * ny_2$ respectively. The cells within each block are defined in one of two ways, they are either internal (or core) cells or boundary (or outer) cells. The internal cells are referenced in a structured manner, as shown in Figure 4(a), and the block boundary cells are stored in an unstructured manner using an indirect addressing array, as shown in Figure 4(b).

The process of computing the flux balances for each cell will be used to illustrate these storage techniques. The pseudo code in Figure 5 shows that for each block, flow variables are first mapped from their global to their local storage using indirect addressing. After the initialization of local workspace arrays, flux contributions from cell faces are accumulated at each node by looping over all faces of all cells. Finally, the local node contributions for this block are added back into the global array using global references where contributions from other neighbouring blocks are also added.

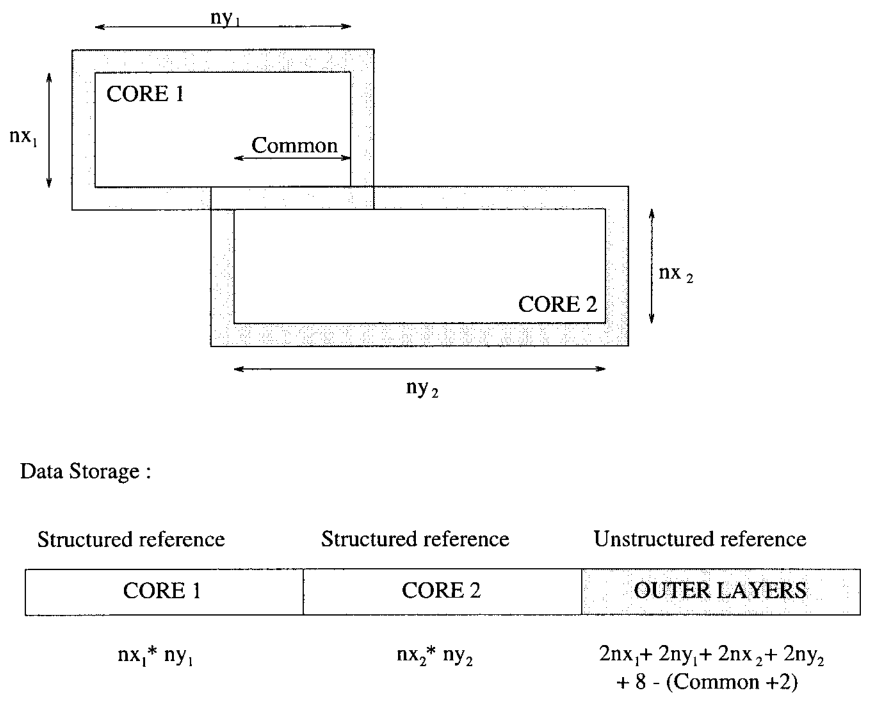

Figure 3. Simple data storage scheme of structured blocks.

For structured blocks, the flux balancing is complete for all internal nodes of a block when calculations in that block are complete, while flux balancing for boundary nodes is complete when looping over all blocks has been carried out. For vector processor architectures, simple vector loops can be used to exploit the accumulation of the flux contributions. For the unstructured grid and interface regions, the flux contributions to the flux balance at each grid node are accumulated directly in the main global data structure using indirect addressing. For vector processor architectures, full vectorization of this process can be ensured by pre-processing the nodal structures using a colour-coded scheme.

```
DO I=2,NX-1
  DO J=2,NY-1
    U_VEL(I,J)=...
  ENDDO
ENDDO
```
(a)

```
DO NODE=1,NBOUNDARY
  UVEL(BNDRY_PTR(NODE)=...
ENDDO
```
(b)

Figure 4. (a) Typical data referencing of internal cells of a block and (b) typical data referencing of block boundary cells.

```
For each block
     Copy flow variables from global array to work arrays
     Initialise all work arrays for flux calculation
     For each face of each cell in the current block
          Calculate flux contribution to face
          Add contribution to nodes in work arrays
     Add accumulated flux contributions to global array
```

Figure 5. Pseudo code for flux balancing computation within ESAUNA.

5. PARALLELIZATION APPROACH FOR DISTRIBUTED MEMORY SYSTEMS

It is not surprising that by far the most computer-intensive part of any flow simulation is the actual flow solution. This is performed by the ESAUNA code in the case of the SAUNA suite. For typical industrial cases, problem sizes range from 350 000 to 1 000 000 grid nodes and require between 250 Mb and 1 Gb of memory to run on a large sequential machine.

Typically, the multigrid scheme with three multigrid levels requires between 100 and 200 fine grid V cycles (Euler) or W cycles (Navier–Stokes) to converge to acceptable accuracy. Traditionally, the time for a typical simulation of flow around even a simple aircraft configuration is in the region of 1–4 h on a Cray YMP-8i vector computer depending on the grid size and flow model chosen.

5.1. Data partitioning—unified grid partitioning scheme

One of the most widely used strategies for executing this kind of simulation on a distributed memory parallel architecture is referred to as the single programme multiple data (SPMD) paradigm. In this approach, each processor executes its own copy of the programme but operates on a subset of the domain data. Therefore, a key consideration for the efficient running of the application in parallel is the partitioning of the original data among the processors, while trying to ensure that each processor is kept equally busy with local computation.

There are a number of partitioning strategies that can be used to distribute the data, but here only three of them will be discussed. These are an intra-block partitioning, where data are partitioned within a structured block; an inter-block partitioning, where data are partitioned between structured blocks; and an unstructured mesh partitioning, where the unstructured mesh region is fragmented into equally weighted partitions. Table I shows a summary of the characteristics and the effects of using the three partitioning approaches to parallelize ESAUNA. Grid partitioning can be applied both to the block-structured grid modules and to the unstructured grid modules. However, the mechanism employed for partitioning is different in both cases. For unstructured grids, a graph partitioning tool such as JOSTLE [10] can be used.

In the original parallel implementation, the structured mesh blocks and the unstructured regions were partitioned independently. The current implementation uses a unified grid partitioning scheme that optimizes an original partition of the mesh through simulated

Table I. Summary of the effects of using the three partitioning approaches to parallelize ESAUNA.

	Inter	Intra	Unstructured
Granularity of data partitioning	At the block level	Within a block	Within the unstructured mesh region
Data partitioning strategy	Assign one or more blocks per processor	Similar to that used for a single block parallelization	Assign cells, faces, and/or vertices to processors
Load balancing	Poor if block sizes are unequal. An approach to remedy this is to create blocks of approximately equal sizes [Leatham M, *ARA Report* # M276/5, 1998]	Relatively good, better than Inter block approach	Achieves a high level of load balancing because of data partitioning strategy
Changes made to the code	Relatively low number of changes, so time to parallelize ESAUNA less than 6 months	Relatively high number of changes, so time to parallelize ESAUNA up to 1 year	Relatively low number of changes (including renumbering the mesh), so time to parallelize ESAUNA less than 6 months
Parallelization of solver	This is straightforward	If solver contains complex serializing data dependencies, the parallelization performance may be poor and solver may need to be replaced	If solver contains complex serializing data dependencies, the parallelization performance may be poor and solver may need to be replaced
Parallelization of FAS multigrid scheme	This is straightforward since different grid levels are defined within a block	This is straightforward if data partitioning is done at coarsest grid level	This is a time-consuming process [8]
Data communication	Few data dependencies between blocks, so relatively low number of data communications	Large number of data dependencies within a block, so relatively higher number of data communications compared with inter-partitioning	Large number of data dependencies within an unstructured mesh, so relatively higher number of data communications compared with inter-partitioning
Scalability of code	This is limited by the number of blocks. Larger blocks may be decomposed to smaller ones, to increase the number of blocks, but this may cause convergence problems	Relatively higher than inter-partitioning for low number of blocks that are large in size	Relatively high as it is based on the size of the unstructured mesh

annealing [Leatham M, Optimal domain decomposition of hybrid grids for a target parallel architecture by simulated annealing methods. *ARA Report* # M276/5, 1998]. The scheme uses a graph-type schematic representation of the mesh. The nodes of the graph represent either blocks of grid or individual unstructured nodes and the edges of the graph are either unstructured edges in the grid, or the faces of blocks. Graph nodes are weighted to account for the number of nodes that they represent and similarly the weight of the edges indicates their associated communication cost.

The initial decomposition is accomplished by using recursive geometric bisection (RGB) to rapidly give a decomposition of the mesh onto the required number of processors. The optimization process (optimizer) then attempts to minimize the cost associated with the partition. The cost of the partition is the sum of communication cost (latency + transfer time) and compute cost. The compute cost includes a term to reflect the associated cost if a processor's local memory size is exceeded.

The key feature of the optimizer is that it attempts to optimize both the partition and the partition to processor mapping. Consequently, the optimized solution is the optimal decomposition onto a given computing resource, rather than the optimal decomposition for a given number of processors. A description of the target parallel platform forms a part of the input to the optimizer, this includes the number of available processors, their relative speed and available memory and the network performance. From an original decomposition onto all of the available processors, the optimizer may reduce the number of nodes on some machines to zero in an attempt to reach an optimal decomposition. For a typical network of workstations, the optimizer will halve the total execution time relative to RGB for the same number of processors.

5.2. Additional data lists required for parallel execution

The examination of the data structures used in the parallel implementation reveals, not surprisingly, that the data structures stored on each processor are the same as those used for a serial execution. As a result of the partitioning, a list defining just the data local to a processor is generated. This list is used in a number of ways to ensure that the execution in parallel is the same as the one in serial. One use is through IF statement execution control masks to determine which processor executes a given statement, another use is through the generation of communication lists to enable the transfer of information between neighbouring processors in the form of messages.

In the original parallel implementation, additional lists were created where the information relating to the complete mesh was stored on each processor, however, most lists become redundant when each processor only stores the information it requires instead of the complete problem. The remaining lists include the data that is updated by the processor and also any 'halo' data used by the processor that is assigned by a neighbouring processor. This minimizes the amount of the local memory required by each processor and hence allows large problems to be computed that could otherwise not be handled by the limited memory available on a single processor. Therefore, in the final implementation of the parallel ESAUNA code the following lists were used:

(i) a list to define the nodes and edges that are common to other neighbouring processors;

(ii) a list of the nodes needed by a processor that it did not already own (Section 5.3);
(iii) a list of the nodes owned by a processor that are needed by another processor (Section 5.3).

5.3. Data communication between processors

In the sequential code, the flux balancing process is computed by adding flux contributions from each cell face to appropriate nodes. In the parallel implementation, within each processor, flux contributions are calculated in the same way (Figure 6(a)). Due to the cell-vertex formulation, the calculation of the cell face fluxes does not require data from any other processor and therefore a halo scheme is not required. When all cell face fluxes within a processor have been calculated and sent to the appropriate nodes, the flux values at all nodes not on the boundary of the grid partition will have been fully updated. However, for nodes on the boundary the flux balance is only partially completed at this stage and the partially completed values must be exchanged via data communication with other processors which share common boundary nodes (Figure 6(b)). The flux balances for these shared nodes can then be completed by adding the locally accumulated balances with the summed values that

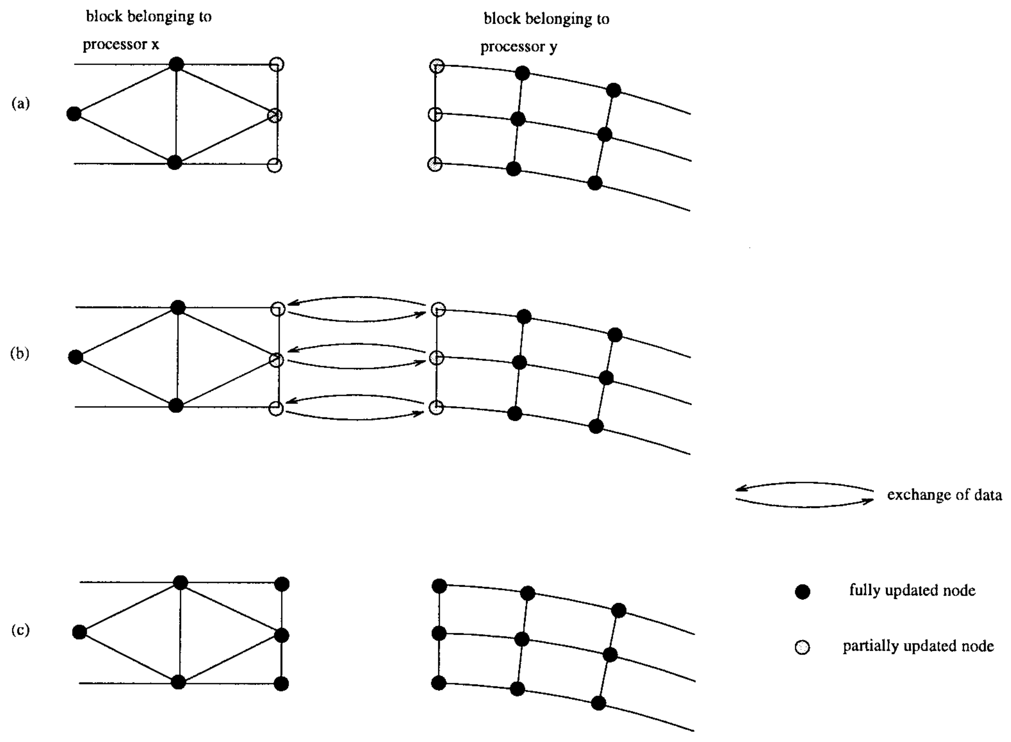

Figure 6. Flux update for nodes that are shared between blocks. (a) Compute partial node updates, (b) exchange partial node data, (c) fully update block nodes with exchanged data.

have just been communicated from a neighbouring processor (Figure 6(c)). A similar procedure is followed for the edge-based dissipation and smoothing operations, which involve the differencing of nodal values. It is worth noting that, unlike the flux balance calculations, data from nodes in adjacent processors may be required in some cases to compute the differences at boundary nodes. A halo scheme is employed to handle these situations.

In the communications approach adopted, all outgoing data from a processor are tagged and communicated using non-blocking sends. Following all the sends, any incoming data are received in arriving order using blocking receives, and the tags are used to identify the data. This leads to only one point of synchronization at each Runge–Kutta stage for each set of transfers.

It may be that the underlying hardware enables the simultaneous operations of computation and data communication between processors. For some parallel architectures, the benefit in using asynchronous communications is significant [11]. In the present study, the use of the Parsytec GC/PP benefited from the use of asynchronous communications. The Parsytec GC/PP employed the Power PC601 processor to perform the computation and the Inmos T805 to perform the data communication. However, in the experiments, the execution of ESAUNA on the Cray T3D and T3E, IBM SP2 and SGi Origin 2000 did not yield any improvements in using asynchronous communications. It is unclear whether this is due to the implementation of the message passing library, the hardware to handle the simultaneous processing of computation and communication, or both.

In order to identify the data to be exchanged between processors and the global to local (and local to global) mappings, extracts from the original code are used that define the key data structures within the code. These extracts from the code are loops that usually relate to the mesh topology, i.e. the relationship between cells, faces and vertices. These loops are used to generate lists that identify all data items to be communicated. These loops have been described in the literature as inspector loops [12].

In addition, executor loops [12] are used to gather the actual data items referenced by the corresponding inspector loops, and this is followed by the communication itself. It is common to use the same inspector loop for many different executors, e.g. the communication of velocity components, pressure and enthalpy all at the same locations defined by the inspector loop.

6. TEST CASES

Three test cases were used to test the efficiency of the parallel code on a number of machines. The first two test cases were run on a Cray T3D, an IBM SP2 and a Parsytec GC/PP. The first test case was the simulation of flow around the M165 wing–body combination (standard AGARD test case [13], Figure 7) using a block-structured grid of 350 000 nodes and with the number of blocks equal to the number of processors. The computation was based on a Euler simulation. Owing to the size of the problem being solved, the convergence rate is accelerated towards the solution using a three-level multigrid procedure.

It was not possible to run the problem on less than four processors due to the limited local processor memory, where each processor had access to 64 MB of memory only. The machines used were based within Europe, the Cray T3D at the FSC (Farnborough, U.K.), the IBM SP2

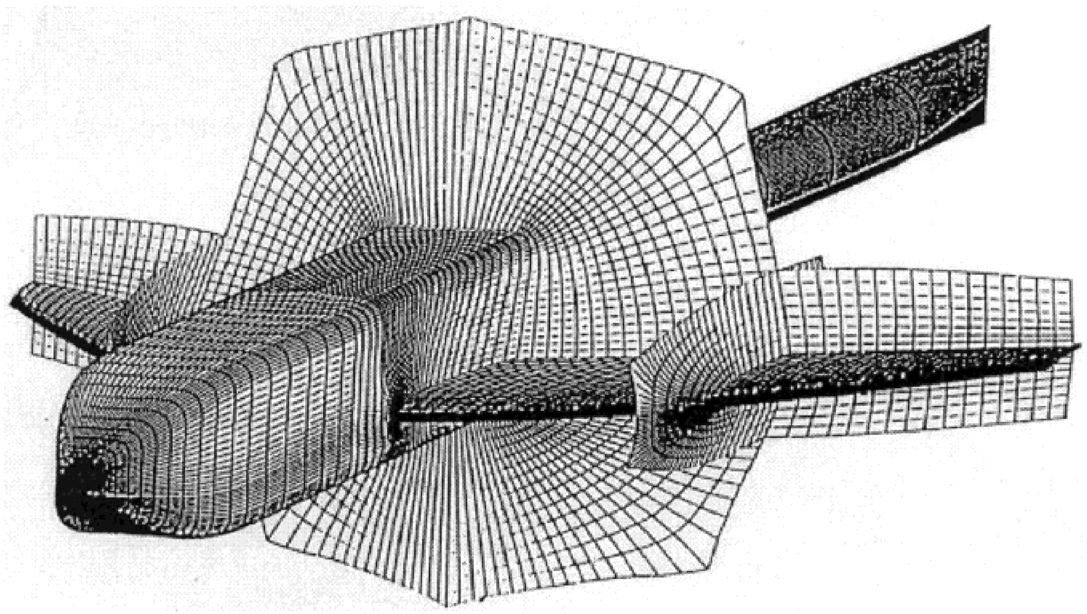

Figure 7. AGARD test case of M165 wing–body geometry.

at CSIRO (Sardinia, Italy) and the Parsytec GC/PP at the University of Paderborn (Paderborn, Germany).

The parallel code is made fully portable using a generic communications library [14] so that results can be generated for several different platforms without code changes, i.e. the same ESAUNA code that was used to produce all the results reported in this paper. In addition, no specific machine-dependent optimizations have been performed in running these test cases but, for each machine, some experimentation to find the best compiler switches was carried out and some generic optimizations used. Although the original sequential code is fully vectorized no attempt has been made so far to optimize the parallel code for RISC processors so there is some bias in favour of the sequential performance in the results.

Figure 8 shows the speed-up factors achieved on the respective machines, using a single grid level (i.e. no multigrid). Figure 9 shows the effect of running ESAUNA with a three-level multigrid procedure on the same case. Clearly, the performance of parallel ESAUNA degrades when multigrid is used, since this reduces the ratio of computation to communication in the parallel execution.

At the coarsest multigrid level, the parallel performance is only between 20 and 50 per cent, while at the finest level it is between 65 and 85 per cent. This effect is more noticeable for the Parsytec GC/PP machine where the communications are performed by the Inmos T805 processor and the computation is performed by the Power PC601 processor. The speed-up factors for the GC/PP look impressive given the relative cost of the GC/PP machine.

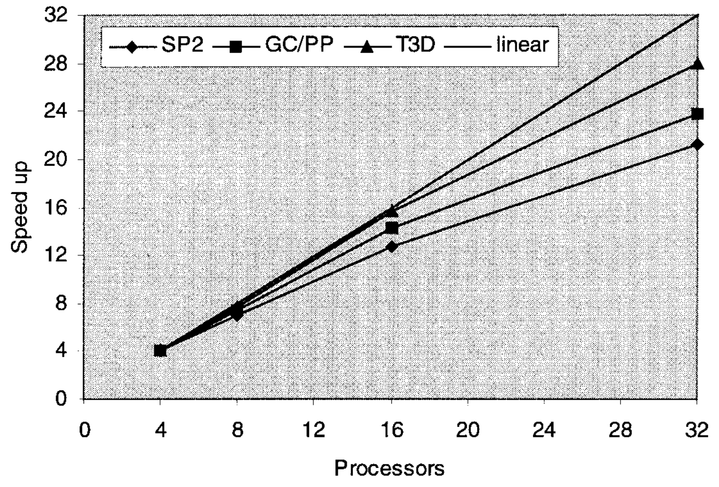

Figure 8. Results for case 1 (no multigrid).

Other contributory factors to the performance degradation include the load balance between processors, as there is up to a 30 per cent difference in the number of nodes assigned to different processors. This is due to the use of a manual grid decomposition for this problem as no automatic procedure existed at the time it was computed. It has already been mentioned that an automatic procedure now exists that would have improved the quality of the partitions and could have generated them in a fraction of the time.

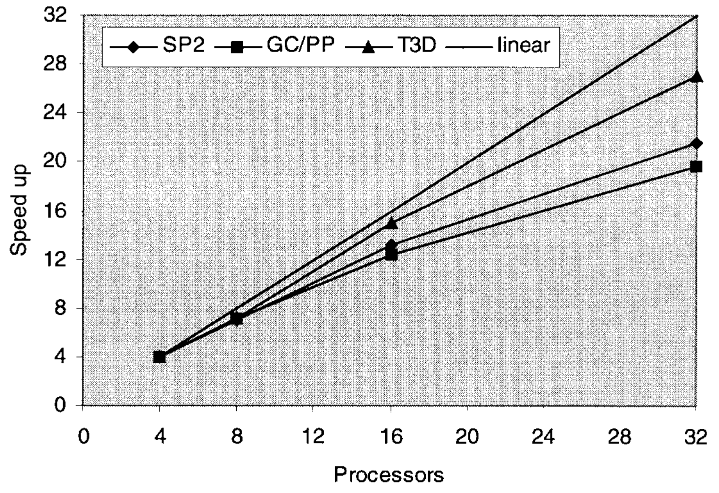

Figure 9. Results for case 1 with multigrid.

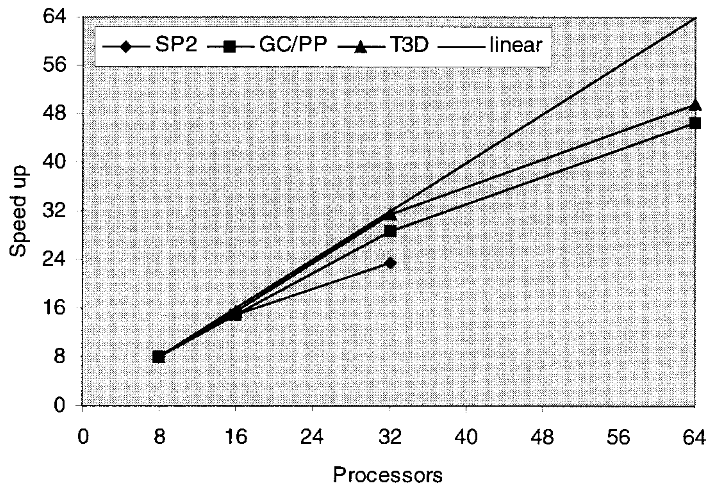

Figure 10. Results for case 2 with three-level multigrid.

The second test case considers the flow over the same M165 wing–body geometry using the Navier–Stokes equations. The solution is sought for a grid of over 750 000 nodes. Clearly, a Navier–Stokes simulation will require more communication between blocks owing to the need to communicate viscous fluxes in addition to the other fluxes, but this is more than compensated by the processor time required to compute these same fluxes.

The results in Figure 10 show the speed-up factors for the same three machines when using a three-level multigrid procedure. It clearly shows that the additional computation (for the flux calculation and for the larger number of nodes) far outweighs the increase in communication for a Navier–Stokes simulation. Hence, the improved speed up factors compared with the Euler simulation. This effect is more apparent with the GC/PP machine, which now performs at a rate comparable with the Cray T3D.

One interesting result to note is that this simulation takes in excess of 3 h on a Cray YMP-8i, a machine regularly used by ARA in the past for production runs. The same simulation can now be performed in less than 0.75 h on a 64 processor Cray T3D.

The third test case is that of a generic civil aircraft configuration. This simulation uses a hybrid block-structured grid for the fuselage and the wings, while using an unstructured mesh for the pylons attaching the engines to the wings (Figure 11). In total there are 585 200 grid points of which the structured components comprises 517 400 hexahedra and 6000 pyramids and the unstructured component comprises 244 300 tetrahedra. Also, there are 985 blocks used to define the complete structured component. The Euler simulation was performed on two different architectures, the first was based on a network of 4250 MHz SGI workstations connected via a thin ethernet (based at ARA). The second architecture was on a Cray T3D up to 32 nodes. The mesh was partitioned using the unified grid partitioning scheme described in Section 5.1. The results for these two architectures are shown in Tables II and III.

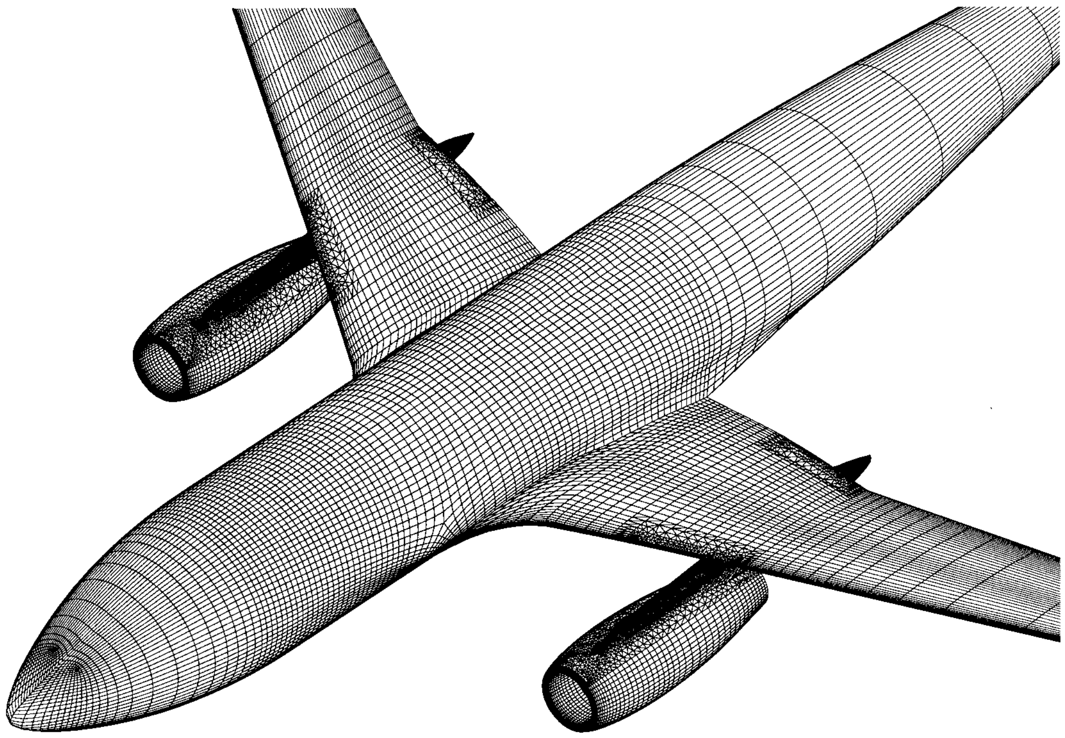

Figure 11. Generic civil aircraft simulated using a hybrid grid of structured blocks and unstructured mesh.

The workstation utilization represents typical use of the code within ARA. This case is at the very limit of memory usage of a single workstation and is used in order to demonstrate comparisons with a single processor execution. Owing to the limitation of memory per node on the T3D, it was only possible to run the simulation on a minimum of eight processors, therefore, the performance improvements shown in Table III are relative to the execution on a T3D using eight processors. Nevertheless, the performance suggested by increasing the number of processors is still very favourable.

Table II. Performance results for test case 3 on a network of SGIs.

Processors	Time (min)	Speed-up
1	326	—
2	183	1.78
4	142	2.29

Table III. Performance results for test case 3 on a Cray T3D.

Processors	Time (min)	Performance improvement[a]
8	181	—
16	92	1.96
32	47	3.88

[a] Performance improvement factor is relative to the execution on 8 processors of the Cray T3D.

7. CONCLUSIONS

The results obtained show that it is possible to obtain high efficiencies using the parallel version of ESAUNA to solve large, complex, industrially relevant problems using hybrid grids on a number of different parallel platforms. There are many implications of the quicker turn-around times: shorter design cycles resulting in reduced time to market, the ability to run a larger number of simulations for a given budget, and the use of parallel ESAUNA in a automated design optimization process for new aircraft design. The coupling of ESAUNA with other simulation codes for multidisciplinary computations, such as flow simulations around an aircraft, coupled with flutter prediction for transient flight simulations now also becomes computationally practical. Likewise, the development of an unsteady flow version of ESAUNA suitable for time-dependent unsteady flows can be considered.

This paper has described the issues involved in parallelizing a large complex CFD code. The time taken to parallelize the code could be estimated at 18 months. This manual task was error-prone and laborious but was the only practical option available at the time. A more viable approach, which is now reaching a level of maturity, is the use of interactive transformation tools. These take sequential Fortran77 code and, through a series of user-guided processes, generate Fortran77 source code (similar in appearance to the original sequential code) with message passing calls to enable execution on a parallel platform. One example of such an interactive tool is the computer aided parallelization tools (CAPTools) [15] that can currently handle single block structured mesh codes [16] as well as unstructured mesh-based codes [17].

REFERENCES

1. Stuben K, Mierendorff H, Thole C-A, Thomas O. *Parallel industrial fluid dynamics and structural mechanics codes, 90–98*, 1996, Proceedings HPCN '96, Springer, Berlin.
2. Childs PN, Shaw JA, Peace AJ, Georgala JM. *SAUNA—a system for grid generation and flow simulation using hybrid structured/unstructured grids*, 1992, ECCOMAS, Brussels.
3. Peace A, Shaw JA. The modelling of aerodynamic flows by solution of the Euler equations on mixed polyhedral grids. *International Journal of Numerical Methods in Engineering* 1992; **35**: 2003–2029.
4. Jameson A, Baker TJ, Weatherill NP. Calculation of inviscid transonic flow over a complete aircraft. *Journal of Aircrafts* 1985; **22**: 855–860.
5. Jameson A, Schmidt W, Turkel E. Numerical solutions of the Euler equations by finite volume methods using Runge–Kutta time stepping schemes. *Journal of AIAA* 1981; **81**: 1259.
6. Benek JA, Steger JL, Dougherty FC. A flexible grid embedding technique with application to the Euler equations. *Journal of AIAA* 1983; **83**: 1944.

7. Morgan K, Peraire J, Peiro J. *Unstructured mesh methods for compressible flows, special course on unstructured grid methods for advection dominated flows*, 1992; 5.1–5.39, AGARD Report No. 787.
8. Leatham M. *Method development for hybrid grids*, 1998, DPhil Dissertation, Oxford.
9. Brandt A, Dinar N. Multigrid solution to elliptic flow problems. In *Numerical Methods in Partial Differential Equations*, Parter S (ed.). Academic Press: New York, 1977; 53–147.
10. Walshaw C, Cross M, Everett MG, Johnson SP. *JOSTLE: partitioning of unstructured meshes for massively parallel machines*, 1994; 273–279, Proceedings of Parallel CFD94.
11. Evans EW, Johnson SP, Leggett PF, Cross M. Automatic code generation of overlapped communications in a parallelisation tool. *Parallel Computing* 1997; **23**: 1493–1523.
12. Saltz J, Mirchandaney R, Crowley K. Run-time parallelisation and scheduling of loops. *IEEE Transactions on Computers* 1991; **40**(4): 603–612.
13. Stanniland DR. *A selection of experimental test cases for the validation of CFD codes*, 1994, AGARD-AR-303.
14. Leggett PF. *CAPLib*, *A portable communications library for automatically generated parallel code*, 1998, University of Greenwich Technical Report, PPRG-98-013.
15. Ierotheou CS, Johnson SP, Cross M, Leggett PF. Computer aided parallelisation tools (CAPTools)—conceptual overview and performance on the parallelisation of structured mesh codes. *Parallel Computing* 1996; **22**: 197–226.
16. Ierotheou CS, Cross M, Johnson SP, Leggett PF. *CAPTools—an interactive toolkit for mapping CFD codes onto parallel architectures*, 1993; 251–259, Proceedings of Parallel CFD 93, North Holland, Amsterdam.
17. Johnson SP, Ierotheou CS, Cross M. *Computer aided parallelisation of unstructured mesh codes*, 1997; 344–353, Proceedings of PDPTA, vol. 1.