ICAS-92-4.5.2

PARALLEL IMPLEMENTATION OF THE FEATURE ASSOCIATED MESH EMBEDDING METHOD FOR THE 2D-EULER EQUATIONS (FAME2D)

M W Bowers and R K Cooper

Department of Aeronautical Engineering,
Queen's University of Belfast, U.K.

Abstract

FAME is a new approach to the problem of a flexible mesh system for complex geometries, which are now more frequently encountered in CFD. Three relatively simple parallel versions of FAME2D, characterised by their increasing complexity of domain decomposition, have been developed to run on a chain of transputers. For a single element aerofoil the simplest version produces a speedup of 5.7 using 8 transputers, while the most complex produces a speedup of 8.6 using 13 transputers. All versions can be easily extended to cater for 2 or 3 element aerofoils and they provide a good basis for the parallelisation of FAME3D.

Introduction

The continued development of Computational Fluid Dynamics(CFD) has reached the stage at which solutions to flows over complex geometries i.e complete aircraft, are being actively pursued and obtained.[1,2,3] However, such a capability can only be truly exploited, if it can be incorporated into the design of an aircraft at the earliest possible moment. Presently, this is being hindered by the lack of a flexible mesh system and the need for immense computing power. Constructing a grid for a new aircraft requires large amounts of human interaction and a high degree of expertise, while computers will be soon reaching their ultimate speed limits.

A single grid-generation scheme is required that is able to rapidly generate grids about an aircraft, enabling accurate resolution of geometric features as well as flow features, as and when they develop. The FAME code is an initial attempt to unify the treatment of the above two features.[4] In this approach, more than one mesh is allowed to cover certain parts of the physical domain . A hierarchy of meshes is established and those higher up will take precedence in regions of overlap. Flexibilty is achieved through the use of separate feature-associated meshes together with mesh-embedding.

The requirement for increased computer power can be fulfilled by the use of parallel processors. Investigation of computer architectures best suited to CFD is underway, as well as parallel communications software, and the conversion of present serial codes to the parallel environment.[5,6,7,8,9] The present work describes the modificaton of FAME2D to work on a simple network of processors.

Description of FAME2D

FAME2D is a basic application code employing a two-dimensional grid system generated by FAME and a novel second-order upwind transonic Euler flow solver. A full description of the philosophy and generation strategy of FAME can be found in reference 4. The objective of this section is to give an adequate overview of the code, so that the parallel implementation can be understood. For convenience, we will use a single NACA0012 aerofoil as the reference test wing, although the code is capable of handling a three element wing. The original code has been optimised to run on a CRAY2.

Flow Solver

The Euler equations are solved using the Split-Coefficient Matrix(SCM) method, which utilises information on signal propagation provided by the theory of characteristics.[10] In the form used, it is a non-conservative first-order accurate upwind algorithm with explicit forward time-stepping.[11] The CFL condition is applied locally to improve convergence.

In two space dimensions, x and z, and time, t, the Euler equations in conservative form, can be rewritten in the familiar quasi-linear form,

$$\frac{\partial Q}{\partial t} + [A]\frac{\partial Q}{\partial x} + [B]\frac{\partial Q}{\partial z} = 0 \quad (1)$$

where Q is the solution vector. The SCM scheme consisgs of integrating a decoupled form of the characteristic equations of equation 1, which can be written in the form,

$$\frac{\partial Q}{\partial t} + [A]^{-}\frac{\partial Q^{-}}{\partial x} + [A]^{+}\frac{\partial Q^{+}}{\partial x} + [B]^{-}\frac{\partial Q^{-}}{\partial z} + [B]^{+}\frac{\partial Q^{+}}{\partial z} = 0 \qquad (2)$$

Notice, the coefficient matrices [A] and [B] have been split according to the sign of their eigenvalues, i.e

$$\begin{aligned}[A] &= [A]^{+} + [A]^{-} \\ [B] &= [B]^{+} + [B]^{-}\end{aligned} \qquad (3)$$

The terms such as ,

$$[A]^{+}\frac{\partial Q^{+}}{\partial x}$$

denote the contributions made by the various characteristics. In this particular case, the contribution made by the positive characteristic with the positive slope in the x-direction and a backward difference is used on the derivatives of Q. The first-order upwind scheme described above uses a symmetric five-point computational molecule and requres no added dissipative terms to enforce stability or to capture shocks cleanly. However, for engineering purposes, a second-order accurate scheme is necessary. In particular, a method is required which does not increase the size of the computational molecule or which requires modification near captured shocks to avoid 'wiggles'. These two properties considerably ease the implementation of the algorithm on the complex grid structure to be described later in this section.

Second-order accuracy can be obtained using a type of deferred-correction procedure, in which a forcing term is generated that is an approximaton to the leading dissipative truncation error associated with first-order upwinding.[12] This is subtracted from the right-hand side of the discretised equations. The forcing term is calculated using Equation 2 with the time derivatives set to zero and the spatial derivatives discretised using simple second-order central differencing. The computational molecule remains unchanged and the stability characteristics are those of the first-order scheme. No observations of 'wiggles' during numerical experimentation have occurred.

Grid Structure

FAME classifies geometric features (types 0,1,2,3,4) according to the number of directional constraints to which they are subject, and generates the appropriate mesh. In this particular case, a type 1 and a type 0 have been produced as shown in Figure 1.

The type 0 mesh is the main mesh which underlies all other meshes and covers the whole flow field. Although it is not associated with any particular geometric feature, it must allow the accurate resolution of high-flow gradients. Therefore, mesh density must be greater in the vicinity of the aerofoil than at the boundaries of the flow field. This is achieved by the use of a series of embedded Cartesian meshes (termed 'levels') of increasing mesh spacing (a factor of 2 is used) as one moves towards the far flow field boundary. Each level is comprised of a number of blocks containing 16 mesh points. These blocks play a crucial role in the embedding procedure. Figure 2 shows the distribution of blocks across the embedded levels. The finest levels, level 17 being the most embedded, are located around the nose and trailing edge of the aerofoil. Level 11, which we have called 'ovle', is the first level that fully encloses the whole aerofoil. Level 1 is the coarsest.

The type 1 mesh is associated with the surface of the aerofoil. It is an algebraic C-mesh extending 3 mesh intervals downstream of the trailing-edge and 1 mesh interval into the flow field, perpendicular to the surface.

A hierarchy of meshes can now be established. The type 1 or overlying mesh is used to calculate the flow on the aerofoil surface. Boundary conditions are obtained by linear interpolation from mesh points on the most embedded main mesh level which occupies the region where the overlying mesh boundary point is located. At level ovle, all the overlying mesh boundary conditions will have been collected.

In regions where the main mesh levels overlap, the more embedded level takes precedence over the other level, since it is more appropriate, i.e it has been constructed to resolve higher flow gradients. This precedence is accomplished by replacing flow data on the coarser level by data from the finer level, at those points which coincide in space with points on the coarser level. Boundary conditions for each level are obtained by interpolation from the next least embedded level.

The main mesh has not been constructed to cope with the solid surface, so some points in each level will reside inside it. The Euler algorithm uses a five-point computational molecule and several points on each level may reference inside the surface. At these points, flow data is replaced by data obtained from further up the hierachy using linear interpolation from the four vertices of the appropiate cell of the type 1 mesh. These points on levels less than ovle, are corrected using the method described in the previous paragraph. Thus, levels less than ovle have no interaction with the overlying mesh.

In summary, FAME2D is composed of an Euler routine for the embedded levels; an Euler routine for the overlying mesh and a series of corrections which transfer flow data from more appropiately constructed meshes. The FAME2D algorithm is shown in Figure 3 and describes the strict order in which the corrections are applied. Note, the convergence criterion is applied to the overlying mesh.

Some Comments on FAME2D

Six important observations, with respect to parallelisation, can be made of the FAME2D algorithm:

(a)Boundary conditions for the overlying mesh are collected on embedded levels greater or equal to ovle. The most embedded collect those around the nose and trailing edge while while the least embedded collect those near the centre of the aerofoil.

(b) For a single aerofoil the embedded levels greater than ovle are split into two independent segments, one located around the nose and one around the trailing edge.

(c) Once all the boundary conditions for the overlying mesh have been collected, i.e. at level ovle, then the Euler algorithm can be applied to the overlying mesh as well as the calculation of the second-order corrections followed by the test for convergence. Levels less than ovle have no interaction with the overlying mesh.

(d) The Euler algorithm can be applied to each embedded level of the main mesh independently without the need to communicate with any other levels. Remember, access to flow data on other levels is only necessary to enact corrections.

(e) Corrections proceed in a strict order on any level:
(i) correct those points that reference inside surface
(ii) replace those points that are coincident with points on previous embedded level
(iii) correct boundary conditions on level

Corrections 1 and 2 must be completed on previous level before correction 2 is enacted on present level. We will reference the corrections by their order of precedence.

(f) To obtain an accurate flow solution on an embedded level, the level must have access to flow data on the next embedded level, the next least embedded level and possibly the overlying mesh. This dependency of information is illustrated in Figure 5, showing the block structure of level 13 located around the nose of the aerofoil. In this example, approximately 40% of the solution is updated by the Euler algorithm, although generally, it is just over 50% The rest of the solution is composed of the corrections or are points within the surface. Not only is the Euler algorithm the most computationally expensive but is responsible for generating 50% of the solution on any given level.

Parallel Implementation

Hardware and Communication

Transputers provide a cheap and flexible research tool with which to investigate the potential of applying current CFD algorithms into the parallel envoironment. Our present system consists of a Tandon P.C acting as a host to 18 T800 transputers, with a total of 72 MBytes of RAM.

This combination is a typical MIMD(Multiple Instruction stream, Multiple Data stream) setup, consisting of the Master processor and several Worker processors, each with a separate memory. The Master processor is responsible for the initial distribution of data and then the collecting, collating and finally storing of the results from the Workers. The Workers perform the calulations required to solve the problem. These processors are connected together in some desired network depending on the number of input/output connections on each processor(4 per transputer). The network is needed to transfer data that is required from one processor to another to enable the calculations to proceed correctly.

Messages can be sent/received via this network using communication software. As the majority of CFD codes have been in FORTRAN, it is desirable to use software compatible with this. One such message-passing harness for FORTRAN applications is FORNET(3L)v1.0.[7] This provides a flexible method by which any processor can send messages to any other, whatever the configuration of the processors. Since it isin the developement stage, FORNET(3L) is presentaly limited to linear topolgies such as chains or rings of transputers, but it can be extended to more complex configurations.

Figure 4 shows a typical chain for nine processors as well as some of the notation used in the code logic. Each transputer is identified by its number inode. Thus, the first processor has the value of inode=1 and the last has a the corresponding value inode=9. A processor identifies its nearest neighbours using the terms 'iprev' and 'inext', where 'iprev'=inode-1 and 'inext'=inode+1. The first, second and last nodes in the chain are given the additional indentifiers of 'mast', 'fnode' and 'lnode' respectively. One of the other nodes is given the identifier , 'ovend', which will be explained later.

Domain Decomposition

In general most previous work, as in reference 8, has envolved problems using a single grid. Thus, the physical domain has been split into several subdomains, each containing roughly the same number of mesh points, which ensures good load-balancing. i.e one processor is not being asked to do too much or to little computation with respect to the other processors. This would be an inefficient use of aviable resources.

Unfortunately, it is obvious that this approach would be difficult to apply to the FAME2D grid system. It is possible but would require large amoutnts of setup information (distribution data) and led to complicated communication. A much simpler approach is desirable.

Instead of partitioning in physical space, the problem can be split according to levels. Three versions have been implemented with increasing complexity of setup information as more processors are used.

Version A

In this method the complete levels are distributed across the processors except for the 'mast'. On this processor is placed the overlying mesh; its Euler routine,its second-order correction routine and the convergence test. The processor where level 'ovle' resides is given the additional identifier 'ovend'.

From observations (a) and (f) it is possible to obtain the overlying mesh boundary conditions and apply the Euler algorithm to the embedded levels on each processor. Once this is completed on 'fnode' the new overlying boundary conditions are inserted into the overlying mesh and the whole mesh is sent to inode=3. Inode=3 receives the overlying mesh and inserts the overlying mesh boundary conditions collected from the levels on that processor, and then sends the whole mesh on to inode=4. This updating of the boundary conditions continues until inode='ovend'. Instead of passing the overlying mesh to 'inext'='ovend'+1, it is passed back to inode='mast', where the Euler algorithm is applied and convergence checked. This is shown in Figure 6b.

For those processors with 'mast'<inode≤'ovend', levels which contain points that reference inside the aerofoil surface can be corrected whenever the overlying mesh has been passed on to 'inext'or 'mast'.

After correction 1 is applied to all the levels on 'fnode', correction 2 is applied. At this stage correction 3 cannot be applied to all the levels since the least embedded level on 'fnode' requires access to the most embedded level on inode=3. Thus, the least embedded level on 'fnode' is sent to inode=3, which allows its most embedded level on inode=2 to have its correction 2 made. Immediately, the most embedded level on inode=3 is sent to 'fnode' and correction 3 can be applied on all the levels residing there. 'Fnode' is then ready to receive from 'mast' the overlying mesh with the new updated surface flow values and the old boundary conditions. Simultaneously, correction 2 is applied to all the levels on inode=3 and then it must communicate with inode=4 as described above. As this tide of corrections is continued until 'lnode' is reached, a new time iteration is beginning in the previous processors.

When first-order convergence has be detected on 'mast', this information is gradually passed to the other processors. This results in a delay of n iterations where n=inode before the second-order corrections begin to be calculated on inode. The complete parallel algorithm is illustrated in Figures 6a,b,c,d.

Version B

This is the same as version A except that the overlying mesh is distributed across the processors that contain the embedded levels greater than ovle-1. No computation is now carried out on the Master processor. The distribution can be accomplished using observation (a), i.e. each level has a number of overlying mesh points associated with it.

Consider Figure 7; if the overlying boundary point E is collected by level l, then we place the corresponding surface point B on the same processor as 'l', e.g inode. Similiary, if we assume points D and A are on inode-1 and points, F and C are inode+1. In order to apply the overlying Euler routine to point B, l requires information about the points A and C. Also, to enact correction 1 on points within the cell BCFE inode requires information about point F. Thus, the communication in Figure 6b is replaced with a new overlying mesh communication routine. For example, for all overlying mesh points on inode, inode receives points D and A from inode-1; then sends E and B to inode+1, then inode-1. Finally, inode receives points F and C from inode+1.The overlying mesh Euler routine and a modified convergence check are added to Figures 6c,d at the end of each time iteration.

Version C

Version B still suffers the same restriction as version A in that only complete levels are given to each transputer. From Figure 2, it is easy to predict load-balancing problems as more transputers are used. Is it possible to get a more even distribution of blocks without making significant changes to version B? This can be achieved by making use of observation (b).

For a single aerofoil the data structure of the FAME grid system can be likened to a tuning-fork, to separate limbs joining at a junction into one. In reality, to separate and independent columns of ever increasing thickness (i.e embedded level area). One column immanating from the nose and the other from the trailing edge, both eventually joining at level ovle. Since ovle is simply rectangular in shape, this can be easily divided into two and allocated to the corresponding column. The juction level changes to ovle-1.

Again consider Figure 4; for example, we can distribute the nose column across inode=2('fnnose') to inode=4('lnnose'), and the tail column across inode=5('fntail') to inode=7('lntail'). Levels less than ovle are then distributed across inode=8 to inode=lnode. It is now possible to use the same code with a few addional commuication modifications.

'Lnnose' exchanges levels with 'lntail'+1 rather than 'inext' while 'lntail'+1 exchanges levels with 'lnnose' as well as 'iprev'('lntail'). Also, at the end of each time iteration 'lnnose' and 'lntail' transfer additional boundary condition due to ovle being split.

Results and Discussion

Simple timing tests were carried out to evaluate speedup, S_m and efficiency E, defined as,

$$S_m = \frac{time\ for\ a\ single\ processor}{time\ for\ m\ processors} \tag{4}$$

$$E = \frac{S_m}{m} \tag{5}$$

For version A we include the Master transputer in m while for versions B and C the Master is not included since it is not actively contributing to the computation of the problem. The test conditions are M_∞=0.8 and α=0° while the execution time is the time for 100 iterations to be performed.

It is obvious from the previous description that for good results using version A and version B, a distribution of complete levels across the processors must be found that gives good load balancing. Slightly different strategies were implemented as the number of processors increased.

(i) For a small number of processors the levels are distributed to give as even a distribution of the blocks as possible.

(ii) As (i) becomes increasingly difficult (levels cannot be split) preference is given to distributing levels greater than ovle as extra processors are used.

(iii) Whenever all the levels greater than ovle are distributed i.e one level per processor , preference is given to distributing the coarser levels across the extra processors.

The results for speedup and efficiency are shown in Figures 8a,b. They relate to the best distribution for a given number of processors. For convenience each distribution will be referenced by the number of processors version A uses. Figure 8c shows the distribution of levels, in terms of the number of blocks on each processor, for distributions four to nine.

For version A, speedup initially increases at a fairly uniform rate until distribution 6 where the marginal increase i.e. the speedup gained by using one extra transputer, is significantly reduced. Marginal speedup is reduced to zero using distribution 7. However, distribution 8 causes a dramatic increase of 1.9 but the use of further transputers has no effect and a plateau is quickly reached. A speedup of 1.1 is achieved for the special case of distribution 2 where the whole of the main mesh is placed on Worker 1. Efficiencies between 55% and 71% have been achieved for distributions less than 8, thereafter, they rapidily decrease.

The three distinct sections described above can easily be associated with one of the distribution strategies already explained. Distributions 6 and 7 correspond to strategy two, where preference is given to distributing levels 11 and above. It is these two distributions that are responsible for the kink in the speedup results. This is caused by bad load-balancing. For distribution 6 there is a consistent rise in the number of blocks on Worker 2 to Worker 4. It has been found that Workers 2 and 3 spend 11% of each time iteration waiting to communicate the overlying mesh to inext. This in turn increases the waiting time on fnode for the overlying mesh from the Master. Distribution 7 has a different source of processor idleness. There is significant difference in the number of blocks on Worker 6 and Worker 5. Almost 35% of a time iteration on Worker 5 is waiting in order to communicate its coarsest level with Worker 6. It is slow to return to its Euler routine which then slows down the progression of the corrections and no extra speedup is gained by using distribution 7. Not suprisingly a large gain in speedup is gained by distribution 8 when the difference in the number of blocks on Worker 5 and Worker 6 is reduced.

Ultimately, the maximum speedup achieved depends on how quickly corrections proceed so allowing a new time iteration to begin. The use of extra transputers does quicken the calculation of Euler routine but this is balanced by the need for the corrections to be passed to extra processors, which is the cause for the plateau.

The version B results show same characteristics as version A, although for given number of transputers speedup and efficiency are considerable higher. One would expect this since the overlying mesh is not allocated its own transputer. By moving version B results one unit to the right, they are almost indistinquishable from version A. Two differences emerge; the kink is more pronounced in version A and its plateau value is 0.32 greater than version B's plateau. The distribution of the levels are the same so these differences can only be explained by the difference in the treatment of the overlying mesh.

Whenever the levels are allocated roughly one per transputer the distribution of the overlying mesh becomes significant. This is particularly for levels 12 and 11 which contain the most blocks as well as most of the overlying mesh points. The additional burden of calculating the Euler results for these points, although relatively small, is enough to delay the start of the main mesh Euler routine (the most computational expensive part of the code) on their respective transputers. This has a knock on effect and further hinders the flow of corrections, so producing a reduced plateau value. The more pronounced kink in version A is the result of the extra communication by having to send the overlying mesh back to the MASTER processor. This slows the start of a new time iteration on the first WORKER, especially if there is very bad load-balancing on transputers near 'ovend'.

To evaluate version C, four distinct cases were tested. Case 2N3T envolved allocating one transputer per column and one for the levels less than ovle. Case 3N5T allocated two transputers per column; case 4N7T allocated three transputers per column and 5N9T allocated four. Again, one transputer was allocated for the rest of the levels. To each case extra tranputers were added. These were used to redistribute levels less than ovle. Speedup, efficiency and block distribution are shown in Figures 9a,b,c repectively.

Each case exhibits the same characteristics as version B, i.e. there is a rapid increase in speedup until a plateau is reached. Plateau speedups of 3.3, 6.0, 8.2 and 8.6 were acheived for cases 2N3T,3N5T, 4N7T and 5N9T respectively. Each case has an optimum distribution where a maximum efficiency is achieved. This corresponds to the first point on each plateau. Initially, placing all the levels less than ovle on one transputer creates a bad load-balance and this effects the smooth progression of the correction procedure. The use of more transputers reduces this effect leading to the increase in speedup. Eventually, the best distribution is achieved and no further increase can be obtained by using more transputers.

Figures 7a,b compare the optimum distributions of version C and the corresponding results for version B. Using more than 9 transputers version C is significantly better. Using 13 transputers, version C's speedup is 50% greater than that of version B, while efficiency is 0.66 compared with 0.43. It is interesting to note that the optimum results also seem to approach a plateau. Thus, greater speedups can only be acheived with more transputers if the domain composition is futher refined. For example, split both segments of the most embedded levels and use the extra processors there.

Figure 11 shows the fully converged solution to the test problem. This was obtained using version A and 7 transputers. It illustrates that parallelistasion has no effect on the solution.

Conclusions

Three parallel versions of FAME2D have been implemented, which for a relatively small number of processors, i.e less than 20, arranged in a simple chain gives acceptable speedups. It has been found that better results can be obtained by biasing the use of avaible parallel resources towards the most embedded levels.

The algorithms have several positive attributes:

(a) The coarse chain of processors reflect the data structure of FAME.

(b) Almost all communication occurs between nearest neighbours.

(c) Retaining complete levels or independent sections of levels on the same processor allows large chunks of vectorised code to be retained (important if vector processors can be used).

(d) Domain decomposition is achieved by distributing levels rather than dividing actual physical space, resulting in a flexible parallel code that can be used for future development; for example, shock-fitting; implicit or multigrid acceleration.

(e) Decomposition information is extremely small, easy to obtain and thus allows quick redistribution, if required.

(f) The algorithm can be easily extended to cater for 2 and 3 element aerofoils. Similar speedups can be expected.

Acknowledgments

This work was funded by the Science and Engineering Research Council(SERC) of the United Kingdom.

References

1. Olling, C.R., and Mani, K.K., "Navier-Stokes and

Euler Computations of the Flow Field around a Complete Aircraft," Advanced Aerospace Aerodynamics, Society of Automotive Engineers SP-757, Oct. 1988, pp. 231-242.

2. Raj, P., Olling, C.R., Sikora, J.S, Keen, J.M., Sing, S.W., and Brennan, J.E., "Three-Dimensional Euler/Navier-Stokes Aerodynamic Method (TEAM), Vol. I: Computational Method and Verification," AFWAL-TR-87-3074, June 1989.

3. Thompson, J.F., "A Composite Grid Generation for General Three-dimensional Regions - the EAGLE Code," AIAA Journal, Vol. 26, No. 3, 1988, pp. 271-272.

4. Albone, C.M., and Joyce, G., "An Overlying/Mesh-Generation and Euler Algorithm for Flows Past Multi-Element Aerofoils", RAE Technical Memorandom(Aero 2131), 1989.

5. Gropp, W.D., and Smith, E.B., "Computational Fluid Dynamics on Parallel Processors", Computers & Fluids, vol. 18, no. 3, 1990, pp289-304.

6. Long, L.N., Khan, M.M.S., and Sharp, H.T., "Massively Parallel Three-Dimensional Euler/Navier-Stokes Method," AIAA Journal, Vol. 29, No. 5, May 1991, pp. 657-666.

7. Cooper, R.K., and Allen, R.J., "Fortnet(3L)v1.0: Implementation and Extensions of a Message Passing Harness for Transputers Using 3L Parallel Fortran," Computer Physics Communications, 1992. (to be published)

8. Yadlin, Y., and Caughey, D.A., "Block Multigrid Implicit Solution of the Euler Equations of Compressible Fluid Flow", AIAA paper 90-0106, 1990.

9. David, T., and Blyth, G., "Parallel Algorithms for Panel Methods," International Journal for Numerical Methods in Fluids, Vol. 14, 1992, pp. 95-108.

10. Chakravarthy, S.R., Anderson, D.A., and Salas, M.D., "The Split-Coefficient Matrix Method for Hyperbolic Systems of Gas Dynamic Equations," AIAA paper 80-0268, 1980.

11. Albone, C.M., "An Upwind, Multigrid, Shock-fitting Scheme for the Euler Equations," RAE Technical Report 85004, 1985.

12. Albone, C.M., "A Second-order Accurate Scheme for the Euler Equations by Deferred Correction of a First-order Upwind Algorithm," RAE Technical Report 88061, 1988.

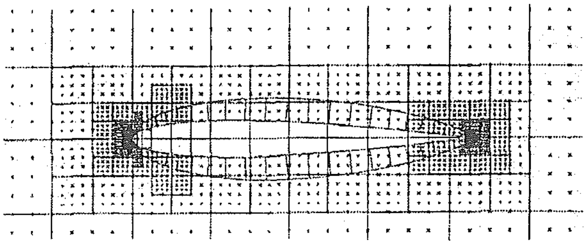

Fig. 1 Type 0 and type 1 meshes for a single element aerofoil.

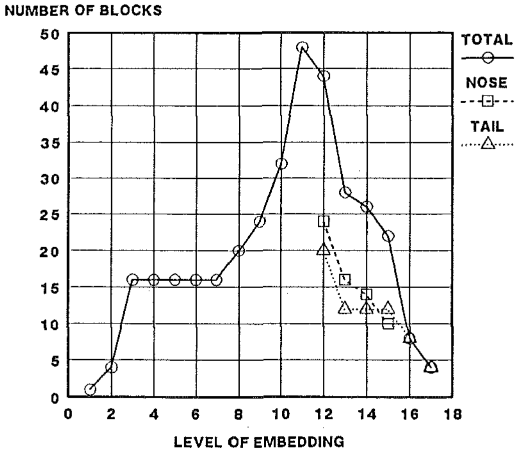

Fig. 2 Distribution of blocks across the embedded levels of the type 0 mesh for a NACA0012 aerofoil.

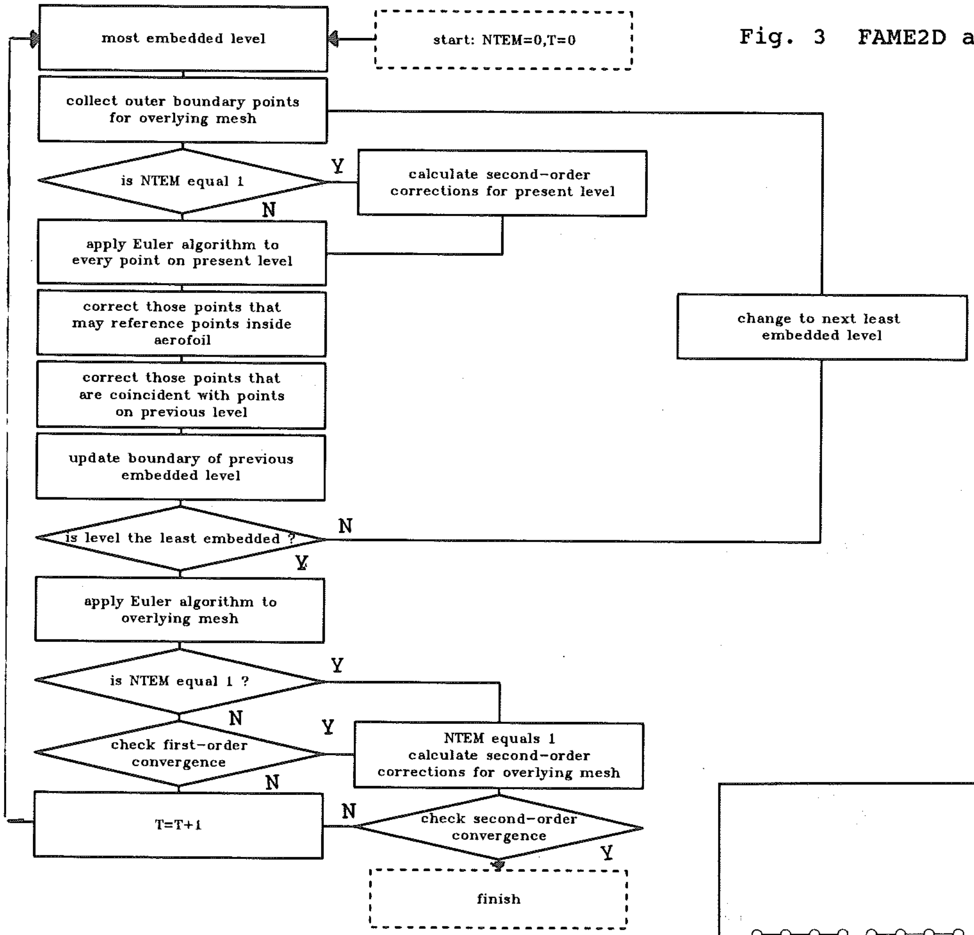
most embedded level
start: NTEM=0,T=0
collect outer boundary points for overlying mesh
is NTEM equal 1
Y
calculate second-order corrections for present level
N
apply Euler algorithm to every point on present level
correct those points that may reference points inside aerofoil
change to next least embedded level
correct those points that are coincident with points on previous level
update boundary of previous embedded level
is level the least embedded ?
N
Y
apply Euler algorithm to overlying mesh
is NTEM equal 1 ?
Y
N
check first-order convergence
Y
NTEM equals 1 calculate second-order corrections for overlying mesh
N
T=T+1
N
check second-order convergence
Y
finish

Fig. 3 FAME2D algorithm.

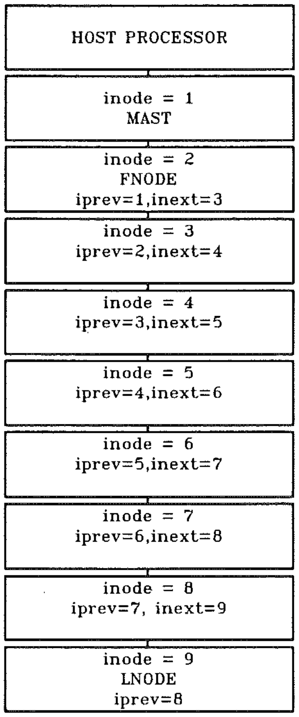

Fig. 4 Parallel network using nine transputers.

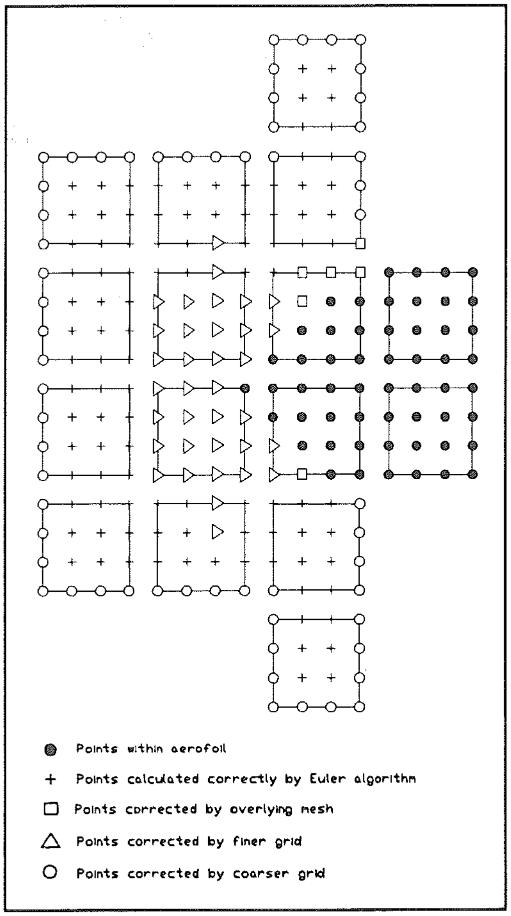
● Points within aerofoil
+ Points calculated correctly by Euler algorithm
□ Points corrected by overlying mesh
△ Points corrected by finer grid
○ Points corrected by coarser grid

Fig. 5 Dependency of flow data on level 13 around the nose of the aerofoil.

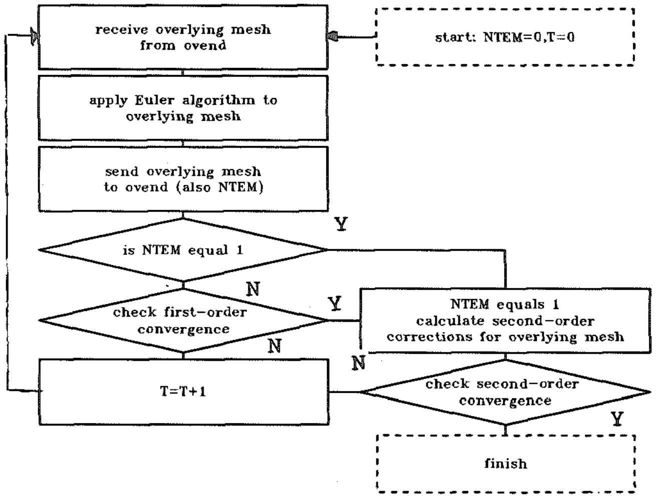
receive overlying mesh from ovend
start: NTEM=0,T=0
apply Euler algorithm to overlying mesh
send overlying mesh to ovend (also NTEM)
Y
is NTEM equal 1
N
check first-order convergence
Y
NTEM equals 1 calculate second-order corrections for overlying mesh
N
N
T=T+1
check second-order convergence
Y
finish

Fig. 6a Parallel code on MASTER transputer.

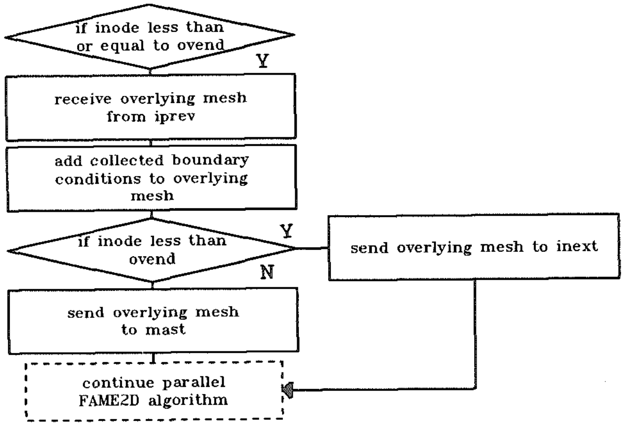
if inode less than or equal to ovend
Y
receive overlying mesh from iprev
add collected boundary conditions to overlying mesh
if inode less than ovend
Y
send overlying mesh to inext
N
send overlying mesh to mast
continue parallel FAME2D algorithm

Fig. 6b Communication routine for the overlying mesh for WORKERS, 'fnode'<inode<'lnode'.

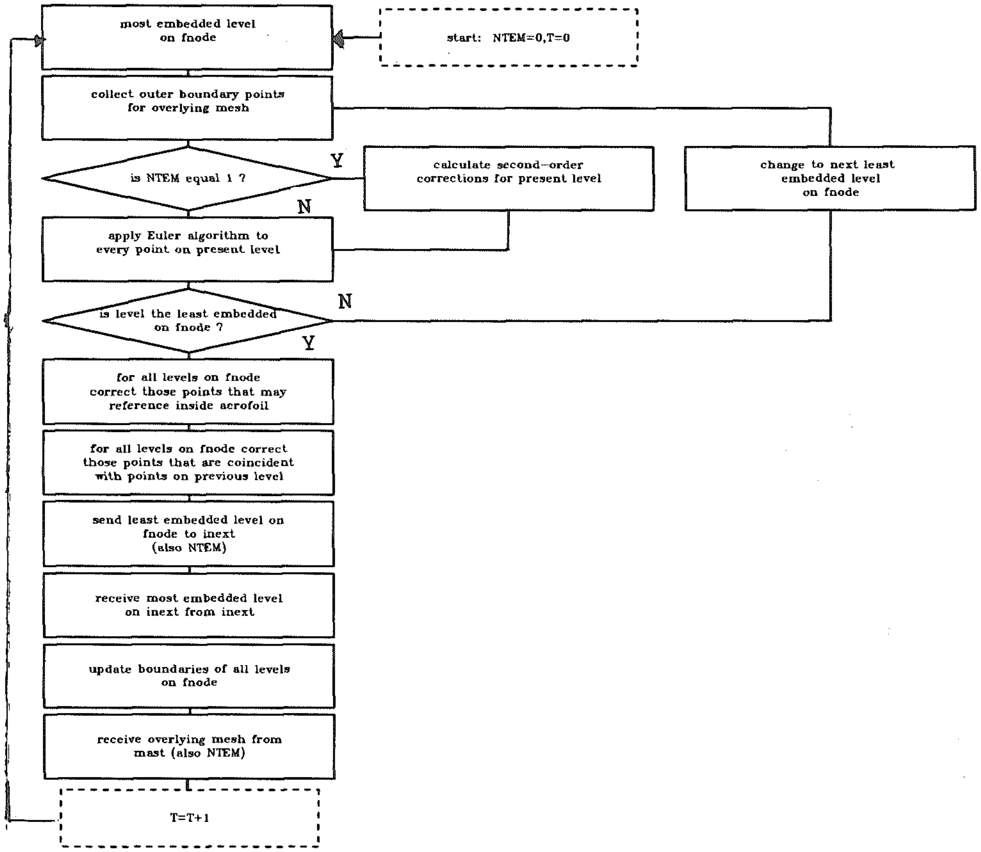
most embedded level on fnode
start: NTEM=0,T=0
collect outer boundary points for overlying mesh
is NTEM equal 1 ?
Y
calculate second-order corrections for present level
change to next least embedded level on fnode
N
apply Euler algorithm to every point on present level
is level the least embedded on fnode ?
N
Y
for all levels on fnode correct those points that may reference inside aerofoil
for all levels on fnode correct those points that are coincident with points on previous level
send least embedded level on fnode to inext (also NTEM)
receive most embedded level on inext from inext
update boundaries of all levels on fnode
receive overlying mesh from mast (also NTEM)
T=T+1

Fig. 6c Parallel code on WORKER 'fnode'.

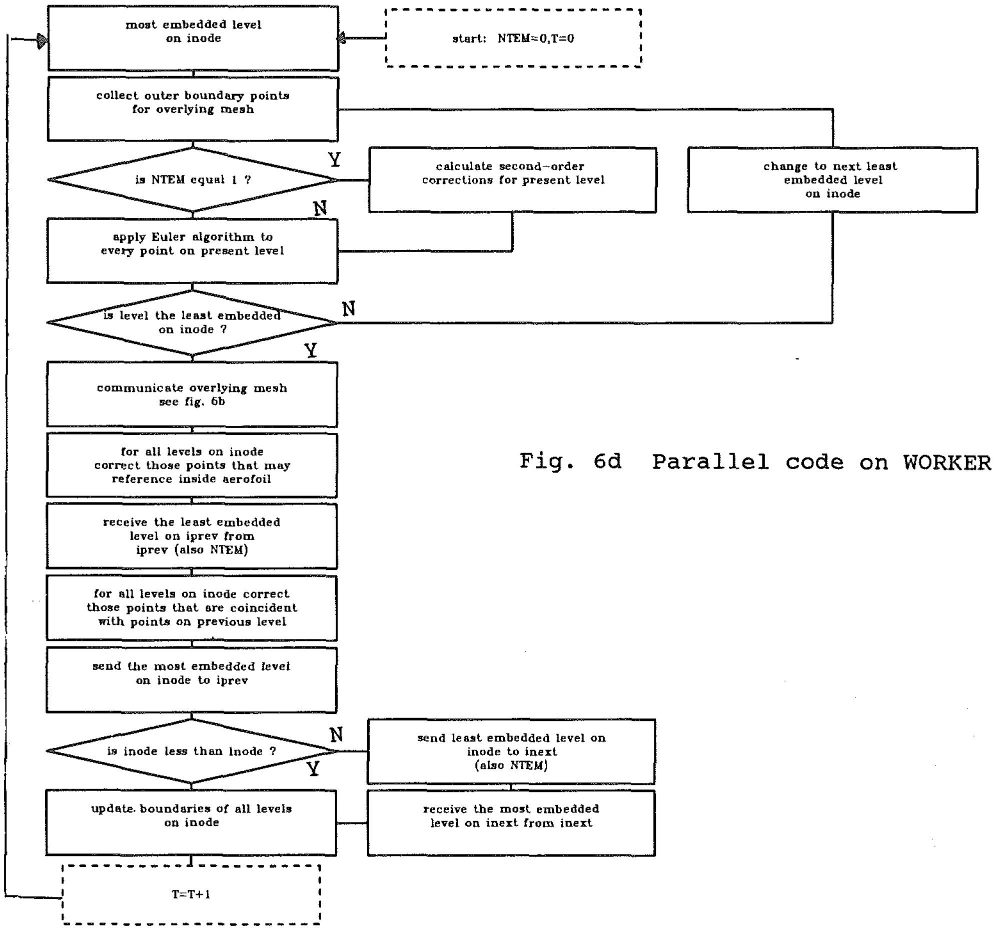
most embedded level
on inode
start: NTEM=0,T=0
collect outer boundary points
for overlying mesh
is NTEM equal 1 ?
Y
N
calculate second-order
corrections for present level
change to next least
embedded level
on inode
apply Euler algorithm to
every point on present level
is level the least embedded
on inode ?
N
Y
communicate overlying mesh
see fig. 6b
for all levels on inode
correct those points that may
reference inside aerofoil
receive the least embedded
level on iprev from
iprev (also NTEM)
for all levels on inode correct
those points that are coincident
with points on previous level
send the most embedded level
on inode to iprev
is inode less than lnode ?
N
Y
send least embedded level on
inode to inext
(also NTEM)
update boundaries of all levels
on inode
receive the most embedded
level on inext from inext
T=T+1

Fig. 6d Parallel code on WORKER inode>'fnode'.

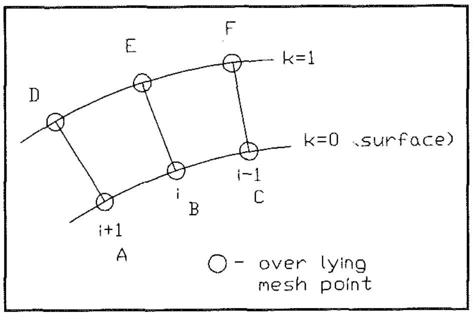

Fig. 7 Type 1 mesh section.

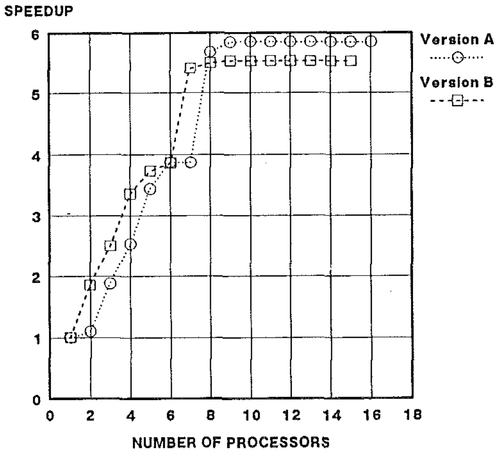

Fig. 8a Version A and B speedups.

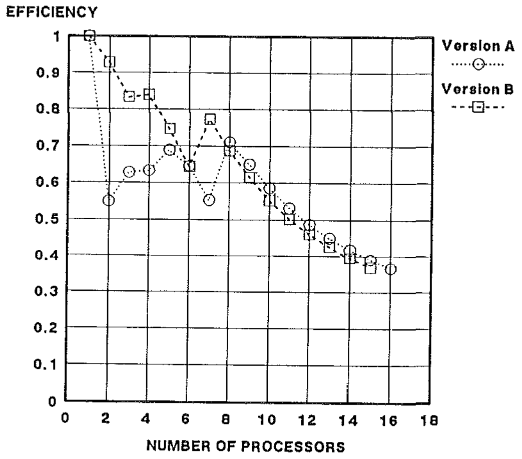

Fig. 8b Version A and B efficiencies.

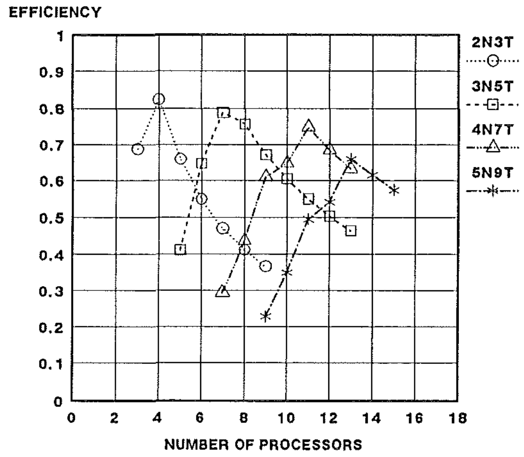

Fig. 9b Version C efficiencies.

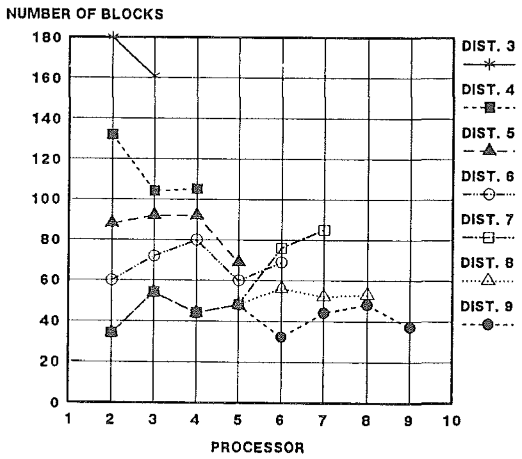

Fig. 8c Distribution of blocks on each transputer for versions A and B.

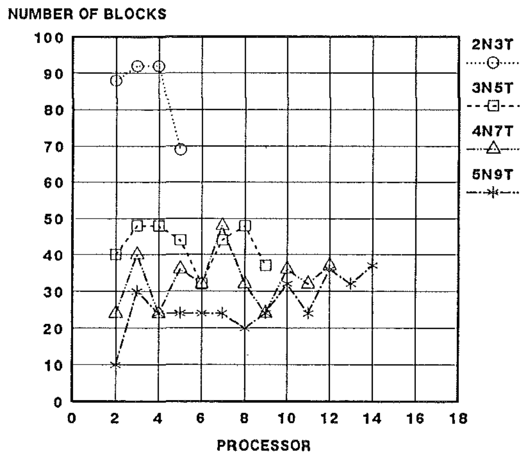

Fig. 9c Distribution of blocks on each transputer for version C.

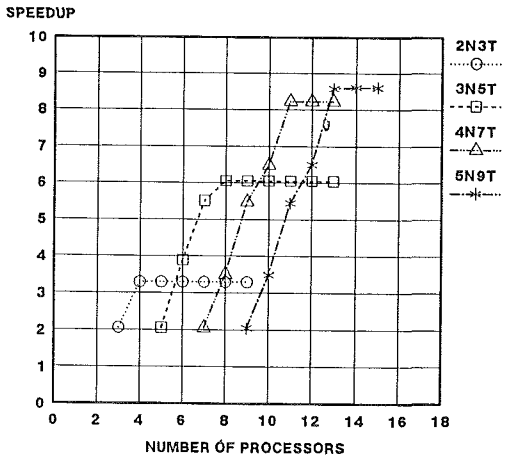

Fig. 9a Version C speedups.

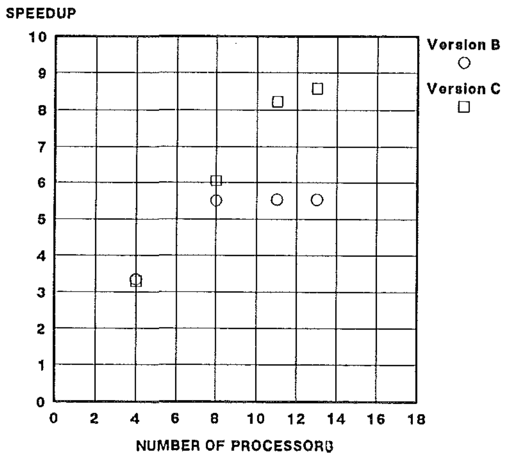

Fig. 10a Comparison of speedups for versions B and C.

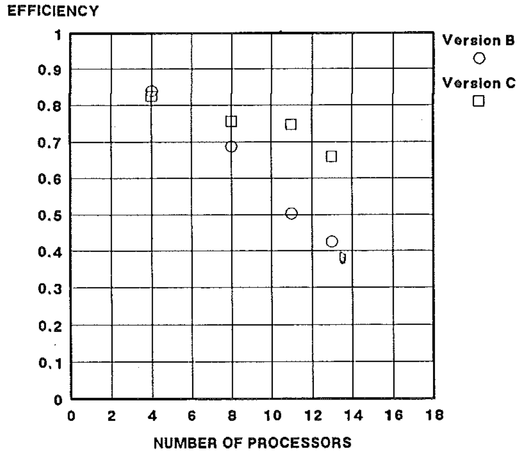

Fig. 10b Comparison of efficiencies for versions B and C.

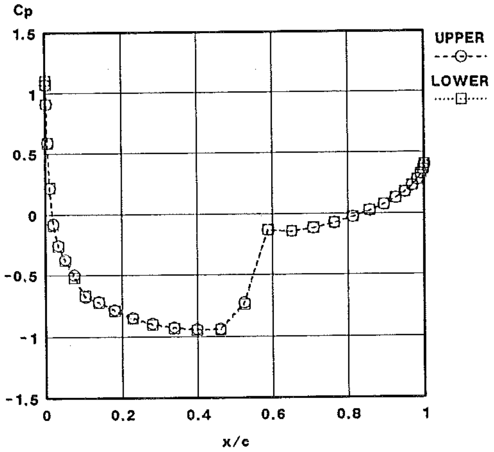

Fig. 11 C_p distribution for NACA0012 aerofoil at M_∞=0.8 and α=0° .