Massive Parallel Implementation of the Aircraft Euler Method and Performance Tests on Different Computational Platforms

by

Edgar A. Gerteisen
Numerical Fluid Dynamics
Dornier Luftfahrt GmbH

and

Antony Jameson
MAE Department
Princeton University

Parallel CFD '93/GAMNI Conference

Paris

10-12 May 1993

Massive Parallel Implementation of the Aircraft Euler Method and Performance Tests on Different Computational Platforms

by
Edgar A. Gerteisen
Numerical Fluid Dynamics
Dornier Luftfahrt GmbH
and
Antony Jameson
MAE Department
Princeton University

1. SUMMARY

The Aircraft Euler Method, so–called AIRPLANE code, has already demonstrated its capabilities in predicting the flow properties about configurations of highly complex geometry [1,2]. However, the algebraic systems with many degrees of freedom, resulting from a discretization of the physical space, imply very time consuming design loops even on todays supercomputers.

By this means, the massively parallel implementation of the Aircraft Euler Method is motivated. An introduction to the numerical method itself is presented and the guidelines for the parallelization strategy are outlined. The massively parallel realization is carried out with PARMACS [3,4,5] as message passing programming environment, which is available on different massive parallel platforms. Therefore, a broad installation base of this production code, AIRPLANE–PAR, also considering future platforms is already realized.

Different platforms have been considered to obtain performance results of the method. However, they should be regarded as preliminary since no access to the modern massive parallel computers such as CONNECTION CM–5, INTEL Paragon, MEIKO CS–2, NEC Cenju2 or PARSYTEC GC could be arranged so far.

2. INTRODUCTION

Experience in a large spectrum of applied Computational Fluid Dynamics (CFD), especially concerning practical flow simulation problems, shows very clearly the demand for CFD packages which allow to handle configurations of highly complex geometry in a user–friendly and economic way. Such tools are crucial for the development of future more competitive products in different industrial fields, ranging from the design of cooling systems for semi–conductors to aircraft design. However, the required very fine space discretization result in a system of corresponding equations with many degrees of freedom. Even on todays supercomputers, fine grid resolutions imply very time consuming flow simulation if Euler– and Navier–Stokes methods are applied, disabling their routine application in aerodynamic design loops. That situation can certainly be broken down by future MIMD computational platforms with their prospect of TeraFlop performance.

An attractive approach to mesh generation for complex three dimensional shapes is offered by an unstructured space discretization using tetrahedra. The combination of a method for constructing tetrahedral meshes and a finite element technique for solving the Euler equations has resulted in a powerful tool in calculating flows over complete aircraft geometries, the AIRPLANE code [1,2]. Compared to a corresponding structured space discretization, this method has the drawback of higher computational costs but the benefit of structured schemes is outweighed for multiply connected regions such as the domain around a multi–element airfoil or complicated three dimensional surface definitions, where the structure imposed by rectangular cells becomes very restrictive. For dealing with such complex shapes multiblock methods have been proposed which are based on the use of several different mesh blocks, each composed of a structured mesh [6]. The definition of the mesh blocks and the generation of a corresponding block topology description file is supported then by sophistcated interactive tools [7,8,9], but the difficulty and the amount of engineering time for mesh generation about three–dimensional configurations of high complexity can get, however, considerable. Therefore, it is generally accepted that the unstructured approach, because of its potential flexibility (Fig. 1) with respect to the generation of appropriate discretizations, will be the most cost effective line in future aerodynamic applications.

At the Dornier company, this code is currently used in its original version for aerodynamic design studies of regional airliner configurations. Unfortunately, the computational time for numerical flow simulations with many de-

grees of freedom, even for non–viscous modelling, is tremendous and not acceptable for routine application within aerodynamic design optimization loops. If a typical mesh about a complete aircraft configuration appropriate to an Euler simulation consists of about 500,000 mesh points (Fig. 1), the computational time for an unstructured Euler simulation on a todays Super–Mini computer like CONVEX C220 requires about fifty to sixty CPU hours. It can easily be estimated that a corresponding Navier–Stokes simulation in the high Reynolds–number range with an appropriate much finer mesh resolution would be beyond the scope of todays supercomputers.

Different approaches to parallel processing are available on todays hardware. One central question is: should each processing element execute its own program, or should they all receive the same instruction of a central source [10]. These two possibilities are called SIMD (single instruction, multiple data) and MIMD (multiple instruction, multiple data), respectively. The MIMD design is seen as a more general design, capable of performing well on a broader range of applications. Although much scepticism exists about the prospects for general decomposing compilers for distributed memory machines, there may be high–level program development environments that greatly support the user in specific fields [11,12,13]. Recently, the synchronization of message passing computers is realized by language extensions in form of "message passing constructs" to an ordinary FORTRAN 77 code [3,4,5]. Also a message passing standard is coming up in the near future [14] and it can be expected and is already promised [15] that today's existing tools will provide a transformation to the emerging standard.

The massively parallel implementation of the AIRPLANE code has been realized by applying the PARMACS library [3,4,5]. This is a convenient approach since, as an EC–standard parallelization tool, PARMACS is available on different parallel platforms ranging from supercomputers such as CRAY to different loosely coupled workstation clusters, and therefore a broad installation base of the method is realized. The parallel version of the AIRPLANE code, AIRPLANE–PAR, consists of several modules: the mesh generation module, the domain splitting module, the preprocessing module, and the flow solver module. Thereby the flow solver module is the most time consuming, and it is the only module that has been parallelized so far. Also sophisticated graphic facilities are essential, in order to control the generated mesh, visualize the decomposed mesh and analyze the calculated solution. Therefore, different graphic tools and interfaces to professional graphic software are included in the whole package.

3. GOVERNING EQUATIONS

Within the continuum limit and when viscous effects can be neglected, the motion of a compressible fluid is governed by the Euler equations. This set of equations can be achieved from the principles of classical mechanics and thermodynamics in an integral form by applying the conservation law of mass, momentum and energy to a control volume Ω with the control surface $\partial\Omega$ [16]. The system to solve can be written, neglecting the body forces and the stress forces, as:

$$\frac{d}{dt}\int_{\Omega} \mathbf{U}\, d\Omega + \oint_{\partial\Omega} \mathbf{F}\, d(\partial\Omega) = 0 \quad ,$$

where the conservative quantities are condensed in the vector $\mathbf{U} = [\varrho, \varrho v, \varrho\varepsilon] = [\varrho, \varrho u, \varrho v, \varrho w, \varrho\varepsilon]$ and the integration with respect to the control surface is summarized by a general flux vector

$$\mathbf{F} = \begin{bmatrix} \varrho(\mathbf{v}\mathbf{n}) \\ \varrho\mathbf{v}(\mathbf{v}\mathbf{n}) + p\mathbf{n} \\ \varrho h(\mathbf{v}\mathbf{n}) \end{bmatrix} \quad ,$$

with $h = \varepsilon + p/\varrho$ and with $\mathbf{n}$ denoting the unit normal vector to the control surface. In order to close the system, it is necessary to establish a relationship between the thermodynamic variables. Assuming a thermodynamically and calorically perfect gas the pressure is related to the conservative variables by the equation of state

$$\varrho\varepsilon = \frac{1}{2}\varrho\,(\mathbf{v}\mathbf{v}) + \frac{1}{\varkappa - 1}p \quad .$$

This set of equations, which is valid only in the inviscid portion of the flow field, is an approach to the more general Navier–Stokes equations. Because of less computational effort, however, it is strongly recommended when the flow is not dominated by viscous effects but moderate and complicated shock structures appear that require the conservation properties.

4. NUMERICAL METHOD

4.1. Unstructured Discretization Scheme

The computational space is discretized by finite volumes or finite elements. The definition of a control volume is established then by a union of meeting elements and the apertaining control surface is specified by a sum over the external faces of the polyhedron formed from the union of meeting tetrahedron elements (Fig.2). Therewith, the system of partial equations is transformed into a set of ordinary differential equations with respect to time, the so–called semidiscrete form:

$$\frac{d}{dt}\left(\mathbf{U}_i \sum_k \Omega_k\right) + \sum_k \left(\bar{\mathbf{F}}_k \cdot \partial\Omega_k\right) = \mathbf{0} \quad .$$

More details of the method, i.e. how the conservation of all quantities is ensured and how boundaries are treated, are described in the basic paper [1].

Moreover, in the same publication [1] the correspondance of the present procedure with a Galerkin method is illustrated. Assuming piecewise linear functions in the flux integral, it is demonstrated that the approximation of the surface integration can be written in a finite volume fashion. Though for the evaluation of the time derivative a weighted average of neighbours has to be evaluated cor-

responding to a lumped mass Galerkin approach. In a more recent examination of this relationship [17] it is also concluded that the spatial discretization produced by the Galerkin finite element scheme with linear elements has a finite volume equivalence on some special control volumes. It is also outlined [17] that the time integrals produce different complete mass matrices, where the volume matrix provides better temporal stability and the finite element matrix is more accurate.

4.2. Dissipation

The semidiscrete form represents so far a nondissipative approximation to the Euler equations. In an Euler solution scheme, however, dissipative terms are needed, first to eliminate the occurence of undamped or lightly damped modes and second to prevent oscillations near shock waves.

The simplest form of dissipation is to add a term generated from the difference between the value at a given node and its nearest neighbours [1]. These differences are expressed by the edges joining at a given mesh point (Fig.3), where the contribution $\varepsilon_{k0}^{(1)}(U_k-U_0)$ at node 0 is balanced by a corresponding contribution $\varepsilon_{k0}^{(1)}(U_0-U_k)$ at node k, maintaining the conservative properties of the scheme. The coefficients $\varepsilon_{k0}^{(1)}$ may incorporate metric information depending on local cell volumes and face areas, and they can also be adapted to gradients of the solution. This dissipation scheme is no better then first order accurate unless the coefficients are proportional to the mesh spacing. A more accurate scheme is obtained by accumulating terms defined by $E_0 = (U_k-U_0)$ at every mesh point and recycling this edge differencing procedure with $\varepsilon_{k0}^{(2)}(E_k-E_0)$, resulting in a higher order dissipation term. An effective scheme is produced by blending accumulated differences with $\varepsilon_{k0}^{(1)}$ and $\varepsilon_{k0}^{(2)}$, and adapting $\varepsilon_{k0}^{(1)}$ to the local pressure gradient. Formulas of this type have been found to have good shock capturing properties, and the required sums can be efficiently assembled by loops over the edges.

Already in [1] it has been mentioned that the addition of properly controlled differences along edges in connection with a flux splitting can be used to assure a positivity condition for a system of partial equations, which will prevent growth in the maximum norm and inhibit oscillations in the solution. Because of its excellent shock resolution properties such a dissipation scheme, similar to that of Arminjon and Dervieux [18], is under investigation [19]. Since the three dimensional extension is not fully tested in the shared memory version of the code, this scheme is not considered in the present parallel massively parallel implementation.

4.3. Integration to a Steady State

The discretization procedure leads to a set of coupled ordinary diefferential equations, which can be written in the form

$$\frac{dU_i}{dt} + R_i(U) = 0 \quad ,$$

where $R_i(U)$ is the vector of the residuals, consisting of the flow balances together with the added dissipative terms. These equations are integrated in an iterative procedure until a steady state is reached. In the AIRPLANE code, a multistage Runge–Kutta time stepping scheme [20,21] is used which has proved effective on rectilinear meshes.

Also the different convergence acceleration techniques, such as enthalpy damping and the use of variable time steps close to the stability limits at each mesh point are implemented [22]. The scheme is accelerated further by the introduction of an residual averaging enlarging the stability margins. For the solution of the system of equations resulting from the implicit weighted averaging of nearest neighbours, which is similar to an implicit dissipation term, it is sufficient to perform two steps of a Jacobi iteration. As another interesting path to enlarge the stability region of the scheme further, an elaboration of this initial approach can be considered by using flux matrices instead of some parameter to weight the implicit residuals.

4.4. Vectorization and Microtasking

Within the iterative time–stepping flow solution procedure of the discretized system of equations, most operations are expressed by accumulated local sums. These local sums represent the flux balance through a control volume of a specific cell vertex. Considering a loop over all faces or edges, the evaluation of these summations would produce recurrencies and data dependencies disabling any vectorization. Fortunately, these effects can be avoided by a convenient data reordering in the sense of performing the accumulation for a regarded point in different loops. Following this guideline, the data references are sorted into independent groups in a so–called colouring scheme, where the local operations are gathered from different groups. For providing a good vector performance the length of these groups is defined typically by 512 data elements or by 1024 data elements, depending on the target vector architecture.

For a computer with a shared memory design and with only a few concurrent processing elements (up to eight processors) a parallelization concept on the basis of these groups is expected to be an appropriate approach. Though the groups themselves are independent with respect to vectorization and parallelization but not with respect to data request. Therefore, such a microtasking can cause synchronization problems since the same storage may be requested by different groups at the same time.

Another point to consider is the data access. Since in an unstrucutured method the data is stored randomly, it is in general arbitrarily distributed in the shared memory. If, in the case of a cache machine, the operations in one group would require data on different pages, the overall performance would significantly be slowed down. For

that reason, a thoughtful prepartitioning of the data into pieces less than page size is also advised. This is carried out by some recursive coordinate bisection which is outlined later on.

4.5. Parallelization Strategy

For the mapping of unstructured computational methods to a MIMD architecture or to a loosely coupled platform (WS cluster) an appropriate partitioning of the unstructured grid is needed.

An efficient partition will have to fulfill two primary conditions. With reference to the load balance of a given set of p processing elements (PE), one would like to partition the computational mesh into p subdomains of about equal size. A second issue is the minimization of the communication amount between PEs, which is a function of both the length of the boundary of the subdomains, as well as of the number of neighbouring subdomains. With regard to an explicit algorithm, the load balancing is probably more important, whereas in connection with an implicit algorithm with corresponding higher communication requirements the situation might be different.

An investigation of three algorithms for the partitioning problem has been carried out by Simon [23]. All three algorithms considered are recursive, incorporating the division by some strategy into two subdomains and then applying the same strategy to the subdomains. After carrying out k of these recursive steps a partitioning into $p=2^k$ subdomains is obtained. The regarded divide and conquer algorithms differ only by the partition–strategy of a single domain into two subdomains.

- recursive coordinate bisection (RCB)
- recursive graph bisection (RGB)
- recursive spectral bisection (RSB)

The RSB method is a very recent development and it is reported that with respect to communication overhead it has significant advantages over the two other algorithms.

In the current massively parallel implementation of the AIRPLANE code, the RCB algorithm has been considered as the most straightforward technique. This very intuitive approach seems also to be convenient for our problem definition at hand consisting in general of a three dimensional geometry with a convex hull as the computational domain. By the additional introduction of a transformation to spherical coordinates with respect to the centroid of the computational grid, an optimum implementation of the RCB method is achieved for AIRPLANE applications. Though a comparison to the more sophisticated RSB method would be worthwile.

4.6. Communication and Synchronization

The above mentioned domain decomposition technique can be carried out, in principle, in two different ways leading to different techniques for the continuation of the numerical scheme across the internal boundaries.

If an adjoining boundary is used as depicted in Fig.4, the terms built with nearest neighbours as well as the higher order discretization terms have to be assembled with a collocation technique. Thereby the values are collected at points on the domain boundaries in a two stage procedure. In a first step the contribution to the sum on all sides of a multiple defined point is accumulated and in a second step with a gather–scatter scheme

```
gather:
do loop over all boundary pairs
   value(ib2)=value(ib1)+value(ib2)
enddo
scatter:
do loop over all boundary pairs
   value(ib1)=value(ib2)
enddo
   where (ib1,ib2) defines one pair in the
   assemblage of connected boundary points
```

all contributions are first accumulated into one point and then scattered back to all points of the assemblage by using an appropriate ordered pointer set (Fig.4). In the shared memory a unique numbering of all points is established and the synchronization is performed directly by the above described loop. In order to enable vectorization and parallelization respectively, the pointers have to be grouped such that data recurrencies for the gathering loop are avoided. For a distributed memory platform the synchronization takes place via message passing, where the processing elements communicate by sending and receiving messages. Here the point index numbers are not necessarily unique and also the domain number has to be stored together with the index of the boundary point. The communication then reads: value of ib1 is send to id2 and accumulated to the value of ib2 for the gather step, and value of ib2 is send to id1 and stored in the register of ib1 for the scatter loop.

The second choice is to connect the subdomains by overlapping cells (Fig.5) resulting in additional so–called halo points. Thereby, the domain boundary consisting of such halo points, has to receive its data from the corresponding inner points of the given neighbouring subdomain. The synchronization then has the following schematic description:

```
do loop over all halo pairs
   value(ih1)=value(ih2)
enddo
   where (ih1,ih2) defines one pair in the
   assemblage of connected halo points
```

The vectorization of such a loop in a shared memory is staightforward since no data recurrency occurs. The corresponding distributed memory version in the case of halo point communication reads: send value of ih2 to id1, and store it in the register of ih1. If the registers, refering to halo points, are up to date, all nearest neighbour operations can be performed without communication. These are related to the accumulation of the Euler fluxes, and to the second order filter terms, which degrades the solution to first order at shock waves. The higher order dissi-

pative terms are determined in a two stage procedure as outlined in chapter 4.2. After calculating the sum of nearest neighbour differences, however, the results have to be updated at the holo adresses first, and in a second step the higher order dissipation can be performed up to the nearest boundary–neighbour point for each subdomain. Since the evaluation of enthalpy damping terms needs no neighbour point, a special treatment for synchronization has not to be considered. Although the residual averaging is a pointwise operation as well, an exchange in between the two stage Jacobi iteration is necessary to fulfill the primary condition for the massively parallel implementation, namely that all features of the original unstructured scheme are maintained assuring exactly the same convergence properties.

Concerning the computational efficiency, the advantage of the first choice is that the synchronization would be defined by less communication points. On the other hand all type of spatial operations, including the residual averaging, must be constructed by the assembling technique, which results in more operation counts on a shared memory and in an increased message passing in a distributed memory environment. The additional send and receive requirements are expected to be more critical and therefore the second choice is advised for massively parallel platforms. Though the first choice may be interesting for other kind of algorithms. Indeed, it is more straightforward to formulate a domain decomposition via overlapping elements especially in three dimensions. (Fig.6, Fig.7). The splitting itself is included in the preprocessor step, which provides the data structure for the flow solver, including the values of faces, edges and metric, and the coloured groups and the information on data which has to be transmitted between boundaries.

4.7. Distributed Memory Solver

According to the condition that the implementation should result in a software which is runnable on different MIMD and SIMD architectures, the programming model has to be machine independent. As a portable standard message–passing interface the PARMACS library [3,4,5] is chosen. Also, it is promised by the distributor [14] that a transformation to the emerging message passing standard [15] will be provided.

PARMACS follows the host/node concept, i.e. the programme has to be split into a host module and into a node module. For the mapping of the application onto the specific hardware, special functions are provided where the host copies the node load–module to a defined number of processing elements (PE) and starts the node processes. The process identification numbers, which are strongly machine dependent, are evaluated automatically. On hostless environments, one PE may serve as the host processor or one PE may have loaded the host process additionally. In the current version of the AIRPLANE–PAR code, the host programme is also responsible for IO and for passing the data to the node, which may result in a bottleneck. Although it is not implied by PARMACS to pass the data through the host process, no convenient solution has been found so far for a scalable parallel IO. Furthermore additional useful features, such as global sum and broadcast, will be included in future PARMACS versions which is under permanent development at GMD (Gesellschaft für Mathematik und Datenverarbeitung).

The synchronization of the time integration is performed by message–passing via synchron send and receive statements, meaning that the process stops until the message is received by the recipient. The corresponding iteration loop must be introduced in the host programme as well as in the node programme. The host controls the iteration and calculates for each iteration step the global convergence control parameters. A schematic description of this loop is outlined with the help of the following pseudo–code:

```
Host Iteration Loop:
begin do
   – increment the iteration count number
   ncyc = ncyc + 1
   – send the iteration number to the node processes
   do iproc=1,nproc
      – recipient of the message ≡ procid(iproc)
      – procid(iproc) evaluated via PARMACS
      – specify a label(iproc) for the message
      SENDR(procid,ncyc,msglen,label)
   enddo
   – calculation of new solution vector on nodes
   – gather of local control parameters from nodes
   – calculate global convergence parameters
   – output of convergence control parameters
if (ncyc.ge.nend) enddo
Node Iteration Loop:
begin do
   – determine the label(MYPROC) of the
     message to receive
   – receive the iteration number
   RECVR( ncyc,msglen,length,sender,type,
          MATCH_ID_AND_TYP(HOSTID,label) )
   – calculation
   – synchronize

   – calculation
   •
   •
   •
if (ncyc.ge.nend) enddo
```

MYPROC and HOSTID are the process identification numbers of an own process and a corresponding host process respectively.
Input to the function is the required length of the message 'msglen'. On return, the function gives the message together with its length, sender and type [5]

The synchronization of the different node processes is carried out by sending data from the nearest boundary–neighbour point of the internal domain boundaries to the corresponding boundary point. Inside the node coding, the values are exchanged in two successive loops, where the nearest neighbour data which is sent by the first loop matches to the halo data which is received by the second

loop. In order to avoid a deadlock caused by impossible synchronization this communication must be performed by asynchronous send and receive statements, meaning that the programme continues execution just after successful sending or receiving the message. According to that, each message must possess a unique identification number. A schematic outlining of this communication part within the node programme–coding clarifies the required asynchronity:

```
Sending of Nearest Boundary–Neighbour Points:
– neidom ≡ total number of neighbour domains
 do nei=1,neidom
    – determination of 'procid(nei)'
    – put message into a 'buffer' field
    – determine message length 'msglen'
    – specify a 'label(MYPROC)' as a function of
      own procid for message identification
    SEND(procid,buffer,msglen,label)
 enddo
Receiving of Data for Boundary Points:
 do nei=1,neidom
    – determination of 'procid(nei)'
    – determine the 'label(procid(nei))' of the
      message to receive
    – determine message length 'msglen'
    RECV(buffer,msglen,length,sender,type,
         MATCH_ID_AND_TYP(procid,label) )
 enddo

 the buffer field is defined by 4096 bytes providing
 a good ratio between startup time and message
 lengths to be transmitted
```

These synchronization loops are introduced at several places in the node programme, since the internal boundaries have to be updated not only at the end but also inside of a multistage step, assuring a proper evaluation of the higher order dissipation terms and a proper Jacobi iteration with respect to the global domain.

5. RESULTS

5.1. Test of Implementation

The development of error free software, especially for massively parallel computational platforms, is a major task which needs to be adressed with great care. For debugging reasons, a shared memory version of the parallel software is maintained, including all features of the massively parallel data structure but performing the communication by an ordinary data exchange instead of message passing.

Additionally, a very simple computational mesh about a cylinder has been defined consisting of a discretization of the surrounding cylindrical hull on the basis of cylinder coordinates. This test configuration can easily be scaled from very few mesh points to a considerable number of mesh points, and following such a strategy has proven to be sensible and helpful during debugging of the communication structure. An example for such a mesh is given in Fig.7, including a decomposition into four domains and depicting the overlapped cells by solid surfaces. In this figure, the compactness of the domains is exhibited as well, indicating the convenience of our present decomposition approach.

More complex examples for testing the parallel software have been considered by using the Onera–M6 wing and the S3a full aircraft configuration. A surface mesh of the S3a splitted into eight domains is presented in Fig.6. By the way, compared to the original version all test runs have demonstrated exactly the same results and convergence properties.

5.2. Performance Analyzing

For a better understanding of the behaviour of a parallel implementation, analyzing tools are essential for different reasons: for verification, for diagnostics if the programme fails to behave as expected, for measuring or estimating loads and latencies of different components and for tuning an application to the hardware. Such a functionality is provided by PA–Tools which is distributed together with PARMACS [5]. It generates some trace file during programme execution, assuring minimum overhead, which can be postprocessed by visual interfaces based on XWindow. Three aspects are covered by a choice of corresponding tools: the dynamic behaviour by 'DynamicMap', the overall time schedule by 'TimeMap' and the performance statistics by 'StatisticMap'.

The application of PA–TOOLS to our parallel application code was very useful with respect to code improvement. In the TimeMap representation of a run on two PEs some interrupts in programme execution were detected, which looked to be obscure because of no communication patterns at that points. This strange behaviour could only be associated with some barrier synchronization introduced during the code development for debugging reasons. By searching for useless barriers these execution interrupts have been eliminated and as a side effect some improvement of the code performance has been obtained by this analysis.

A TimeMap example for a run on 8 PEs is presented in Fig.8 showing a part of the iteration step with asynchronous communication. Reading the TimeMap figure, one should note that a cut (horizontal line) is introduced, in order to shorten the interval of constant colour.

5.3. Bench Suite

The whole considered bench suite for testing the performance on different platforms consists of a set of computational meshes ranging from 4,352 mesh points up to 93,632 mesh points. These meshes are splitted each into 2^n domains with n ranging from $1 \leq n \leq 4$ (2 to 16 domains). The resulting data is stored in unformatted IEEE representation for disk space reasons. Special IO routines are provided in the application program taking into account the different implementations (word/byte count, little/high endian) of the standard. Hardware tests have

been carried out on the in–House DEC workstation (WS) cluster at Dornier, on an IBM RS/6000 530H WS cluster connected via token ring and alternately via fibre (Fig.9) and on an iPSC/860. The WS clusters have been tested with up to 8 processing elements; on the iPSC, 16 PEs were available.

These preliminary results are presented in (Fig.11 to Fig.13) in the context of speedup, efficiency and some estimation of MFlops calibrated to a CONVEX C220 where the AIRPLANE code performs with about 15 MFlops. The figures demonstrate that for a standard WS network environment the limits for speedup and efficiency are reached very soon. Though some test runs with two fibre connected workstations at the IBM cluster indicate that this behaviour probably is related more to the communication bandwidth than to the communication startup time. Unfortunately, this fibre network consists of a point to point connection, where each PE has to generate protocols for passing communication data streams and therefore the throughput degrades heavily for more than two PEs. By regarding the above mentioned Time-Map (Fig.8) of the communication, introduced at several places within the iteration loop, this behaveour becomes obvious. Although the iPSC hypercube consists of a well suited interconnect–topology for up to 16 PEs (Fig.10), the bandwidth and startup time technology of this hardware is outdated and no more than a shift from 8 to 16 PEs of the WS–cluster results is achieved.

By this means, as a primary result, these performance tests indicate that an efficient interconnection network is most crucial for parallel computing. The communication speed will be more important for modern parallel systems, which are based on advanced local parallel technologies such as latest RISC processors and VECTOR facilities. For that reason, a well balanced massively parallel computer envisaging the TeraFlop power must provide high computational efficiency on the PE together with a very sophisticated interconnection hardware. This statement, however, refers to more or less communication intense algorithms well–known in CFD and may be totally different for other kind of scientific application.

6. CONCLUSION

A portable massively parallel version of the Aircraft Euler Method has been introduced. The implementation has been carried out on the basis of the PARMACS library as a portable message passing interface. Since this library is supported on different parallel platforms ranging from supercomputers such as CRAY to loosely coupled workstation cluster, a broad installation base of the production code, AIRPLANE–PAR, has been realized.

As the flow solver module is the most time consuming part of the programme system, it is the only one that has been parallelized so far. Unfortunately, a big bulk of data is produced by the preprocessor if the computational meshes are prepared for a multitude of different number of processing elements. According to that, one of the next development steps will include the realization of a massive parallel partitioning formulation, enabling the recombination of the preprocessor module with the flow solver module. On the basis of the current implementation, providing access to the rising concurrent computational systems with TeraFlop power, more sophisticated physical models like Reynolds averaged Navier–Stokes with appropriate turbulence models can be attacked efficiently. Future variants of this code will support such viscous modelling together with suitable grid adaption and refinement procedures, resulting in a very competitive CFD application package.

Performance results of the method have been carried out considering different massively parallel computational platforms, which demonstrate the portability of the current implementation. The results indicate that an efficient communication network is most crucial for parallel computing, especially with respect to scalability. Regarding these results as preliminary, much prospect of very high preformance is expected for the modern parallel machines, which are based on advanced local parallel technologies such as RISC processors and VECTOR facilities, and on highly sophisticated interconnection hardware.

7. REFERENCES

Ref. 1: A. Jameson, T.J. Baker and N.P. Weatherill; Calculation of Inviscid Transonic Flow over a Complete Aircraft; AIAA Paper 86–0103, AIAA 24th Aerospace Sciences Meeting, January 6–9, 1986/Reno, Nevada

Ref. 2: A. Jameson and T.J. Baker; Improvements to the Aircraft Euler Method; AIAA Paper 87–0452, AIAA 25th Aerospace Sciences Meeting, January 12–15, 1987/Reno, Nevada

Ref. 3: V. Sunderam; a framework for parallel distributed computing. Concurrency: Practice and Experience. 2(4): 315–339, 1990

Ref. 4: R. Hempel; The ANL/GMD macros (PARMACS) in fortran for portable parallel programming using the message passing programming model – user's guide and reference manual. Technical report. GMD. Postfach 1316. D–5205 Sankt Augustin 1. Germany, November 1991

Ref. 5: PARMACS and PA–Tools Reference Manuals; PALLAS GmbH, Hermühlheimer Str.10, D–W5040 Brühl; Distributor of PARMACS Software

Ref. 6: A. Jameson, S. Leicher and J. Dawson; Remarks on the Development of a Multiblock Three–Dimensional Euler–Code for Out–Of–Core and Multiprocessor Calculations, in: Progress and Supercomputing in Computational Fluid Dynamics, Birkhäuser Verlag, 1985

Ref. 7: W. Seibert; An Approach to the Interactive Generation of Blockstructured Volue Grids Using Computer Graphics Devices; First International Conference on Nu-

merical Grid Generation in Computational Fluid Dynamics, Landshut, W. Germany, 14th–17th July, 1986

Ref. 8: W. Seibert; A Graphic–Interactive Program–System to Generate Composite Grids for General Configurations; Second International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Miami Beach, Florida, USA, 5th–8th December, 1988

Ref. 9: W. Seibert, W. Fritz, S. Leicher; On the Way to an Integrated Mesh Generation System for Industrial Applications; AGARD Fluid Dynamics Panel Specialists' Meeting on "Applications of Mesh Generation to Complex 3–D Configurations", Leon, Norway, May 89

Ref. 10: George S. Almasi, Allan Gottlieb; Highly Parallel Computing; The Benjamin/Cummings Publishing Company Inc., (ISBN 0–8053–0177–1)

Ref. 11: G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, D.W. Walker; Solving Problems on Concurrent Processors, Prentice–Hall International, Inc., 1988, ISBN 0–13–823469–8

Ref. 12: H. Zima, H.–J. Bast, and H.M. Gerndt. SUPERB – a tool for semi–automatic MIMD/SIMD parallelization. Parallel Computing, 6, 1–18,1988

Ref. 13: H. Zima and B. Chapman. Supercompilers for Parallel and Vector Computers. ACM Press Frontier Series, Addison–Wesley, 1990

Ref. 14: Jack J. Dongarra, Rolf Hempel, Anthony J.G. Hey, David W. Walker; a proposal for a user–level message–passing interface in a distributed memory environment; ORNL/TM–12231, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831

Ref. 15: Karl Solchenbach; private communication. PALLAS GmbH, Hermühlheimer Str.10, D–W5040 Brühl; Distributor of PARMACS Software

Ref. 16: Dale A. Anderson, John C. Tannehill, Richard H. Pletcher; Computational Fluid Dynamics and Heat Transfer; McGraw–Hill Book Company, 1984 (ISBN 0 07 050328 1), pp. 181–257

Ref. 17: T. J. Barth; "On unstructured grids and solvers", in Computational Fluid Dynamics, Lecture Series 1990–03, Von Karman Institute, Belgium, March 1990

Ref. 18: P. Arminjon and A. Dervieux; Construction of TVD–like artificial viscosities on 2–dimensional arbitrary FEM grids; INRIA Report 1111, October 1989

Ref. 19: A. Jameson; Computational Algorithms for Aerodynamic Analysis and Design; Contribution to the 25th Anniversary Conference on Computer Science and Control

Ref. 20: A. Jameson, W. Schmidt and E. Turkel; Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge–Kutta Time Stepping Schemes; AIAA Paper 81–1259, AIAA 14th Fluid Dynamics and Plasma Dynamics Conference, Palo Alto, 1981

Ref. 21: A. Jameson; A Vertex Based Multigrid Algorithm for Three Dimensional Flow Calculations; ASME Symposium on Numerical Methods, Annaheim, December 1986

Ref. 22: A. Jameson; Transonic Flow Calculations, Princeton University Report, MAE 1651, 1984, included in Lecture Notes in Mathematics, 1127, edited by F.Brezzi, Springer Verlag, 1985, pp.156–242

Ref. 23: H. D. Simon; Partitioning of Unstructured Problems for Parallel Processing; Computing Systems in Engineering, Vol. 2, No.2/3, pp. 135–148, 1991

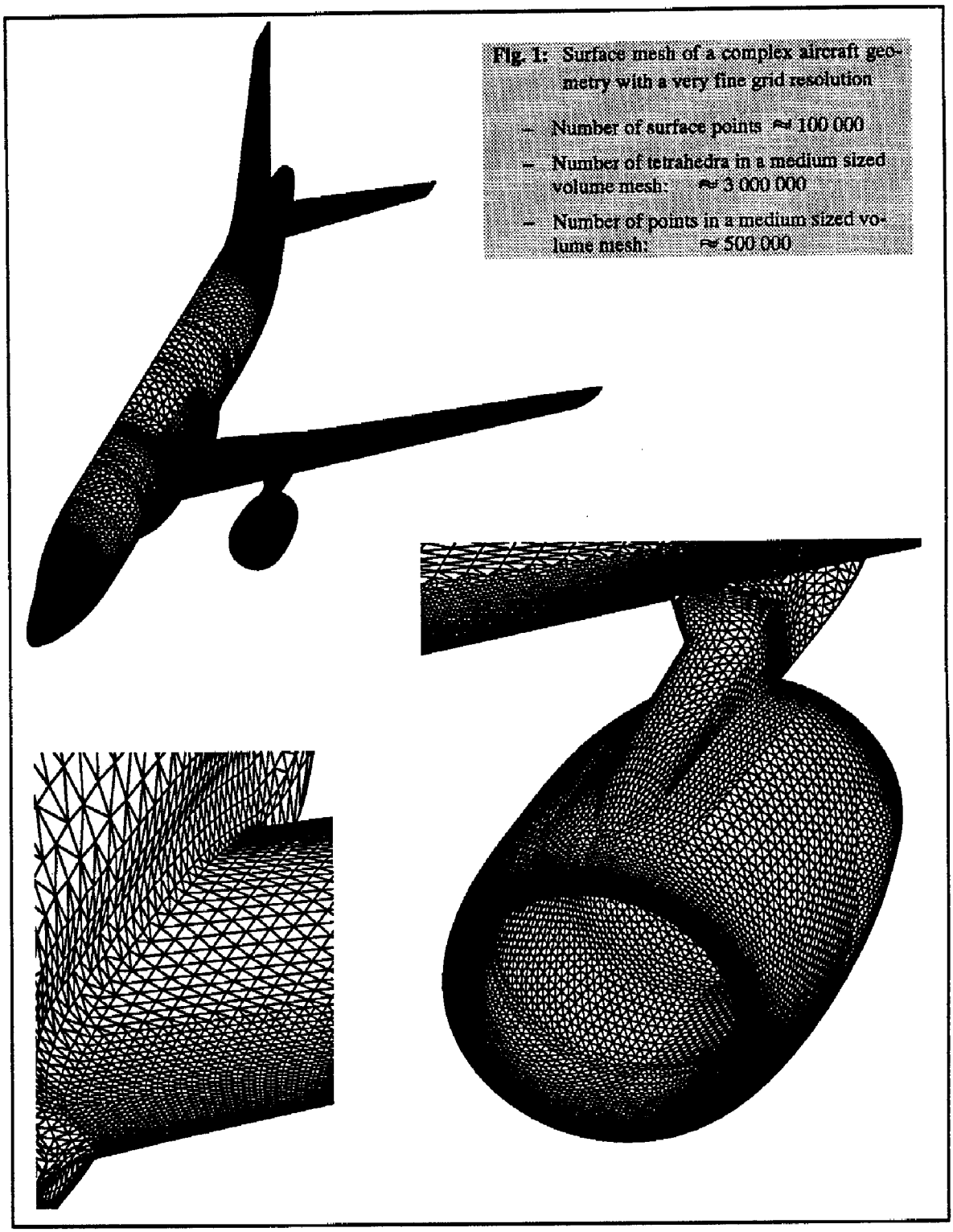


Fig. 1: Surface mesh of a complex aircraft geometry with a very fine grid resolution

- Number of surface points ≈ 100 000
- Number of tetrahedra in a medium sized volume mesh: ≈ 3 000 000
- Number of points in a medium sized volume mesh: ≈ 500 000

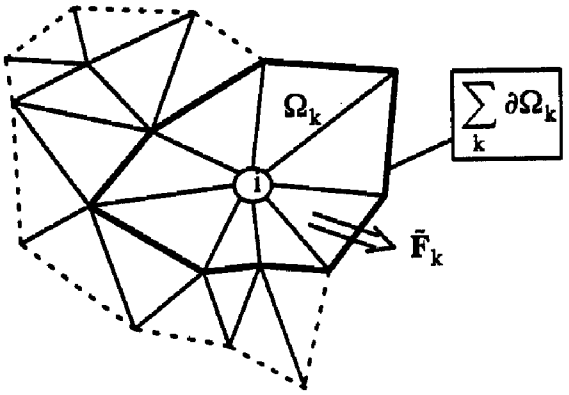


Fig. 2: Union of Meeting Elements

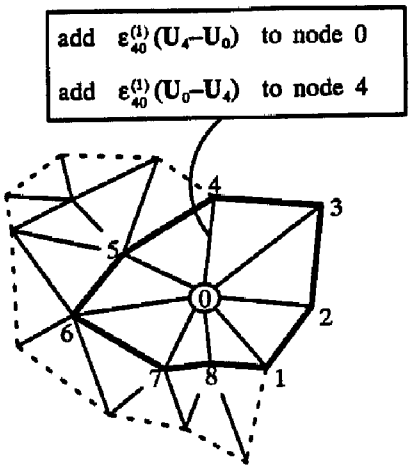
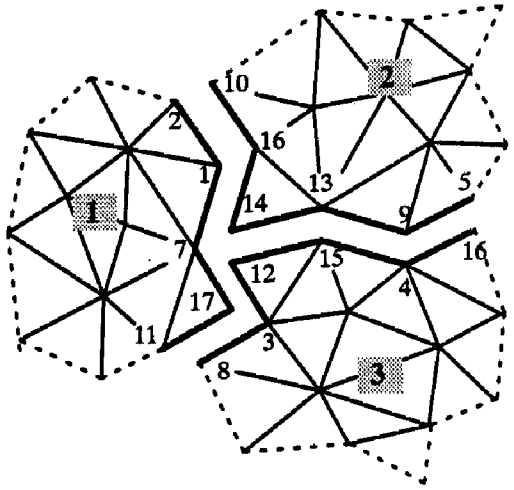
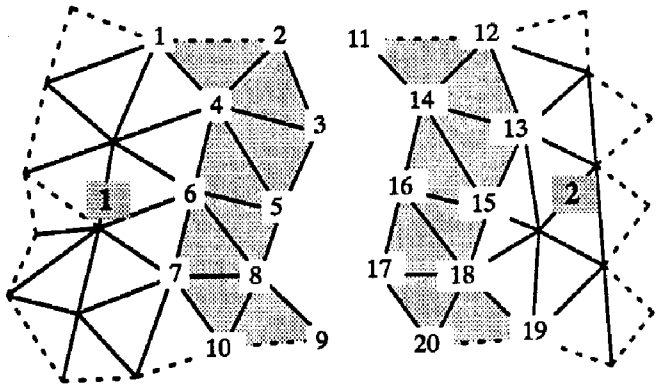


Fig. 3: Construction of dissipation from differences along edges in two dimensions



(ib1 , id1 , ib2 , id2)
(7 , 1 , 12 , 3)
(14 , 2 , 12 , 3)
(1 , 1 , 6 , 2)
(2 , 1 , 10 , 2)
(13 , 2 , 15 , 3)
(9 , 2 , 4 , 3)
(5 , 2 , 16 , 3)
(17 , 1 , 3 , 3)
(11 , 1 , 3 , 3)

Fig. 4: unstructured domain decomposed via adjoining boundaries, including an appropriate ordered pointer set for gather–scatter communication



(ih1 , id1 , ih2 , id2)
(11 , 2 , 1 , 1)
(14 , 2 , 4 , 1)
(16 , 2 , 6 , 1)
(17 , 2 , 7 , 1)
(20 , 2 , 10 , 1)
(2 , 1 , 12 , 2)
(3 , 1 , 13 , 2)
(5 , 1 , 15 , 2)
(8 , 1 , 18 , 2)
(9 , 1 , 19 , 2)

Fig. 5: unstructured domain decomposed via overlapping boundaries, including an example for a pointer field connecting the corresponding points

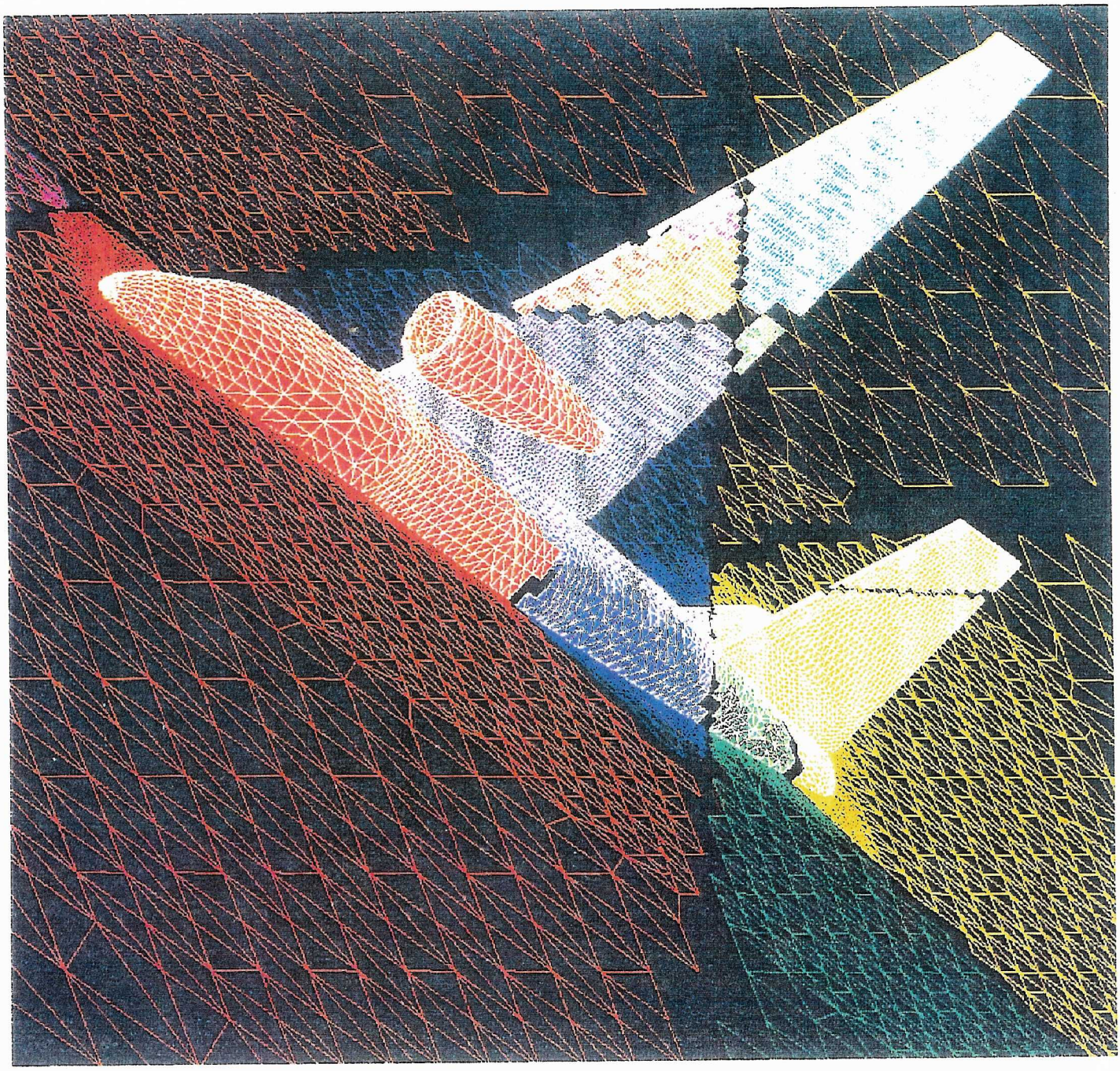


Fig. 6: unstructured mesh about a complete aircraft configuration (170 498 grid points) splitted into eight domains, depicting the overlaps by gaps

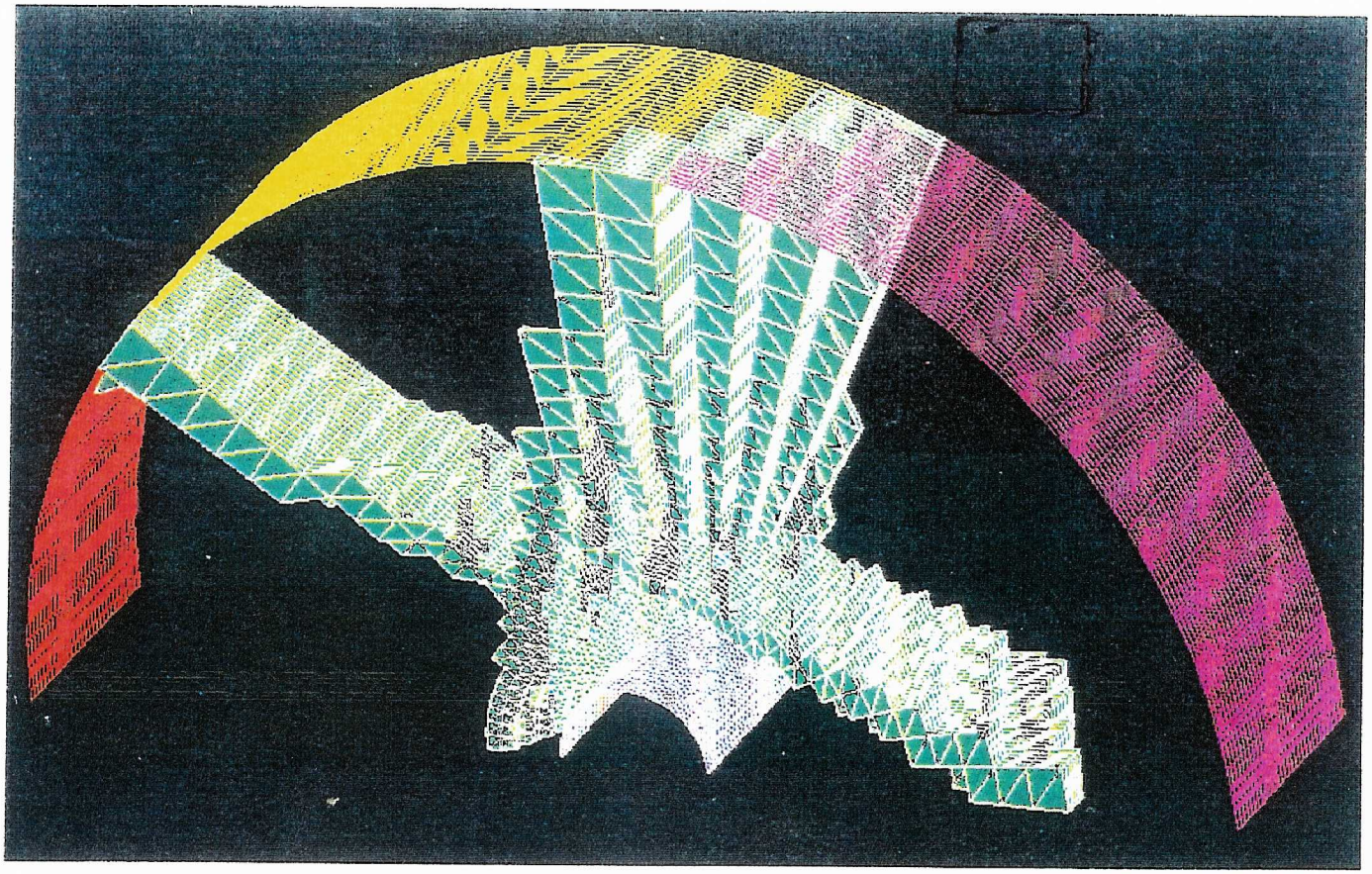


Fig. 7: unstructured mesh about a cylinder (24 567 grid points) splitted into four domains, including overlapped cells depicted by green surfaces

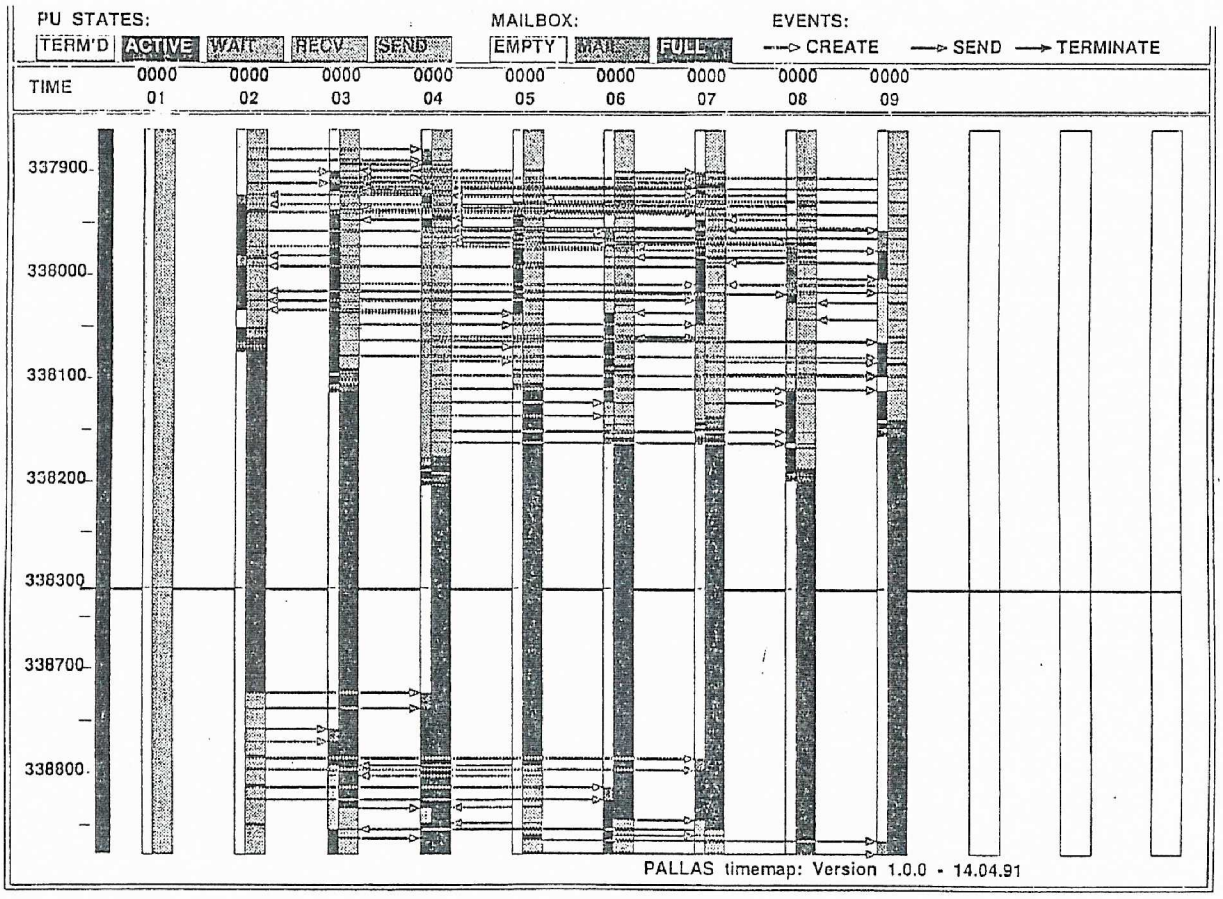


Fig. 8: TimeMap example for a run on nine PEs showing a part of the iteration step with asynchronous communication

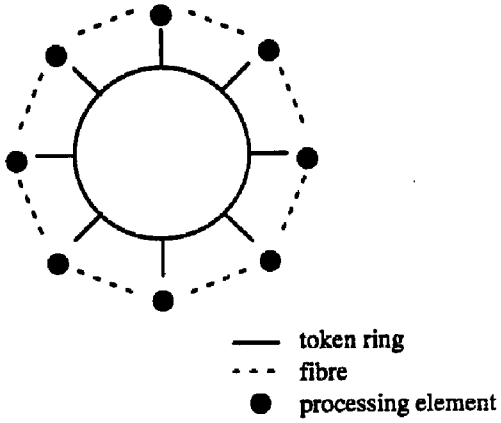


Fig. 9: Network Topology of the considered IBM RS/6000 530H workstation cluster

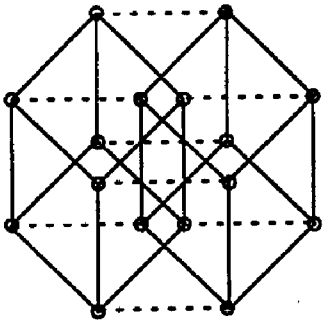
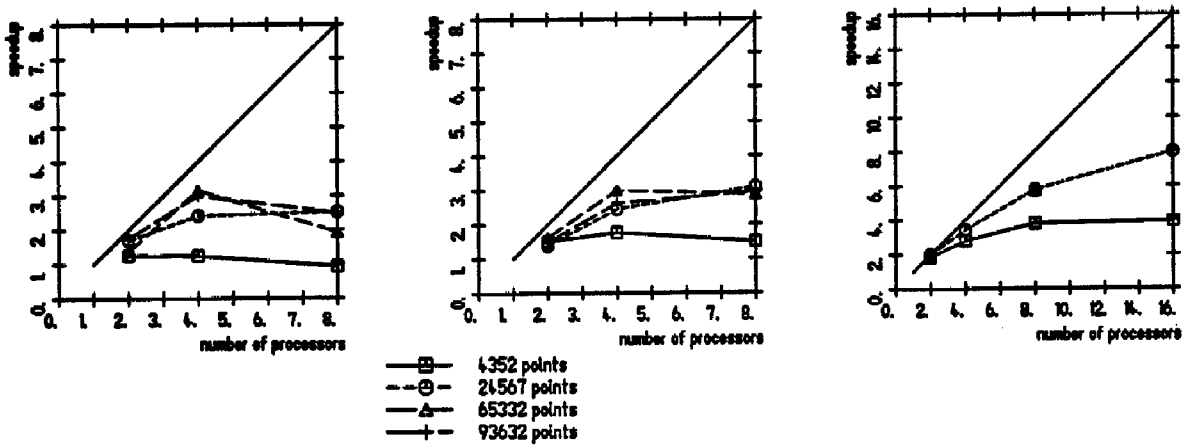


Fig. 10: Network Topology of an iPSC hypercube consisting of 16 processing elements



IBM RS/6000 530H
WS–cluster via token ring

DEC 5000/200
WS–cluster via ethernet

INTEL iPSC/860
Hypercube

Fig. 11: Speedup $S(n) = \frac{T(1)}{T(n)}$ for the considered computational platforms

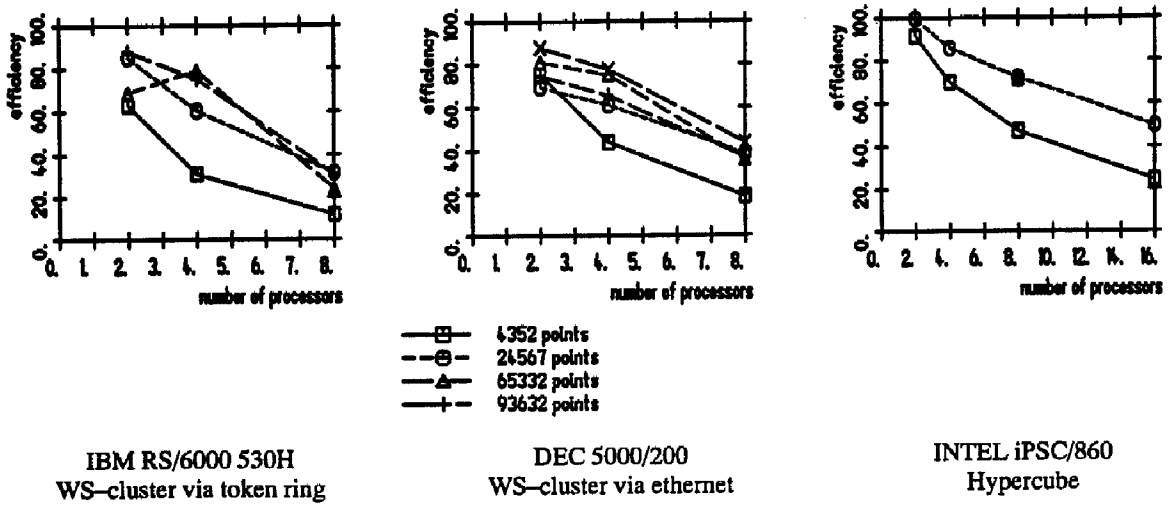


Fig. 12: Efficiency $E(n) = \frac{S(n)}{n}$ for the considered computational platforms

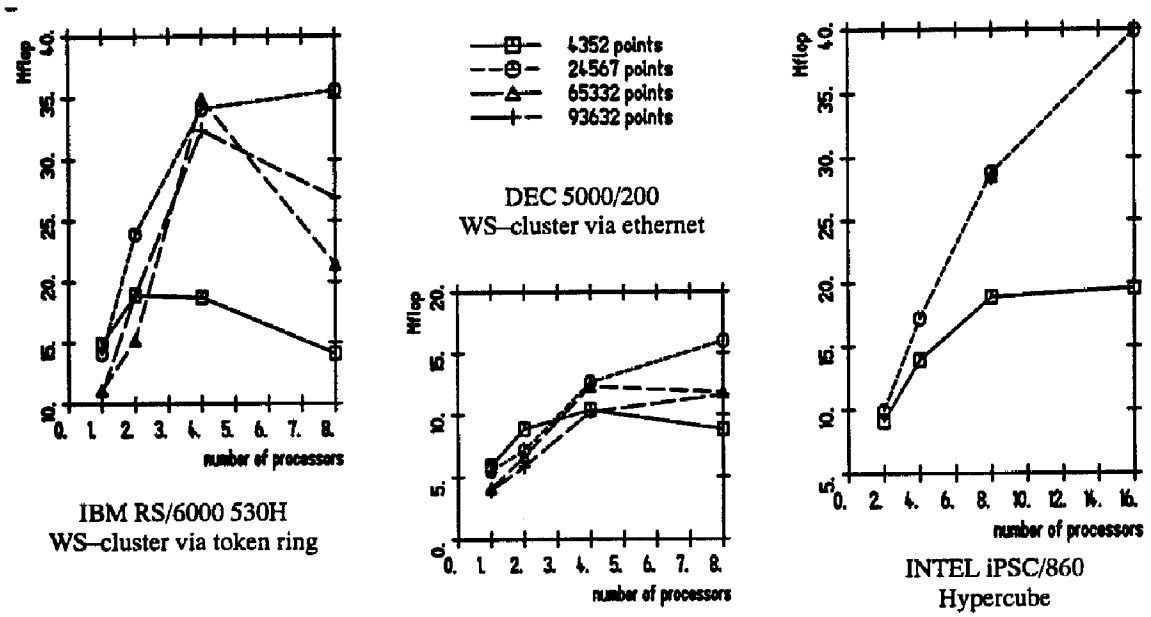


Fig. 13: Estimation of MFLOP for the considered computational platforms